Kernel Methods in Computational Biology

Computational Molecular Biology

Sorin Istrail, Pavel Pevzner, and Michael Waterman, editors

Computational molecular biology is a new discipline, bringing together computational, statistical, experimental, and technological methods, which is energizing and dramatically accelerating the discovery of new technologies and tools for molecular biology. The MIT Press Series on Computational Molecular Biology is intended to provide a unique and effective venue for the rapid publication of monographs, textbooks, edited collections, reference works, and lecture notes of the highest quality.

Computational Molecular Biology: An Algorithmic Approach, Pavel A. Pevzner, 2000

Computational Methods for Modeling Biochemical Networks, James M. Bower and Hamid Bolouri, editors, 2001

Current Topics in Computational Molecular Biology, Tao Jiang, Ying Xu, and Michael Q. Zhang, editors, 2002

Gene Regulation and Metabolism: Postgenomic Computation Approaches, Julio Collado-Vides, editor, 2002

Microarrays for Integrative Genomics, Isaac S. Kohane, Alvin Aho, and Atul J. Butte, 2002

Kernel Methods in Computational Biology, Bernhard Schölkopf, Koji Tsuda, and Jean-Philippe Vert, editors, 2004

An Introduction to Bioinformatics Algorithms, Neil C. Jones and Pavel A. Pevzner, 2004

Kernel Methods in Computational Biology

edited by
Bernhard Schölkopf
Koji Tsuda
Jean-Philippe Vert

A Bradford Book
The MIT Press
Cambridge, Massachusetts
London, England

This book was set in LaTex by the authors and was printed and bound in the United States of America

Library of Congress Cataloging-in-Publication Data

Kernel methods in computational biology / edited by Bernhard Schölkopf, Koji Tsuda, Jean-Philippe Vert.
p. cm.—(Computational molecular biology)
"A Bradford book."
Includes bibliographical references and index.
ISBN 0-262-19509-7 (alk. paper)
1. Computational biology. 2. Kernel functions. I. Schölkopf, Bernhard. II. Tsuda, Koji. III. Vert, Jean-Philippe. IV. Series.
QH324.2.K47 2004
570'.285—dc22 2003068640

10 9 8 7 6 5 4 3 2 1

Contents

Preface

Recent years have seen impressive progress in computational biology. To an increasing extent, this is owed to the use of modern machine learning techniques for the analysis of high-dimensional or structured data. In the early days of machine learning in computational biology, a substantial number of relatively straightforward applications of existing techniques to interesting data analysis problems were carried out. However, because the problems that can be dealt with in this way are gradually running out, it has become increasingly important to actively develop learning algorithms that can deal with the difficult aspects of biological data, such as their high dimensionality (e.g., in the case of microarray measurements), their representation as discrete and structured data (e.g., DNA and amino acid sequences), and the need to combine heterogeneous sources of information.

A recent branch of machine learning, called kernel methods, lends itself particularly well to the study of these aspects, making it rather suitable for problems of computational biology. A prominent example of a kernel method is the *support vector machine* (*SVM*). Its basic philosophy, which is shared by other kernel methods, is that with the use of a certain type of similarity measure (called a kernel), the data are implicitly embedded in a high-dimensional feature space, in which linear methods are used for learning and estimation problems. With the construction of various types of kernels, one can take into account particular aspects of problems of computational biology, while the choice of the linear learning method which is carried out in the feature spaces allows one to solve a variety of learning tasks such as pattern recognition, regression estimation, and principal component analysis.

This book provides an in-depth overview of current research in the field of kernel methods and their applications to computational biology. In order to help readers from different backgrounds follow the motivations and technicalities of the research chapters, the first part is made up of two tutorial and one survey chapter. The first chapter, by Alexander Zien, provides a compressed introduction to molecular and computational biology. Mainly directed toward computer scientists and mathematicians willing to get involved in computational biology, it may also provide a useful reference for bioinformaticians. The second chapter, by Jean-Philippe Vert, Koji Tsuda, and Bernhard Schölkopf, is a short introduction to kernel methods. Focusing more on intuitive concepts than on technical details, it is meant to provide a self-contained introduction for the reader new to this field. The third chapter, by William S. Noble, is an in-depth survey of recent applications of

kernel methods in computational biology, apart from the ones covered later in the book.

Following these three introductory chapters, the book is divided into three parts, which roughly correspond to three general trends in current research: kernel design, data integration, and advanced applications of SVMs to computational biology. While the chapters are self-contained and may be read independently of each other, this organization might help the reader compare different approaches focused on related issues and highlight the diversity of applications of kernel methods.

Part II is made up of six contributions that present different ideas or implementations for the design of kernel functions specifically adapted to various biological data. In chapter 4, Christina Leslie, Rui Kuang, and Eleazar Eskin present a family of kernels for strings based on the detection of similar short subsequences that have fast implementations and prove to be useful as kernels for protein sequences for the detection of remote homologies. A fast implementation of some of these kernels is detailed in chapter 5 by S.V.N. Vishwanathan and Alexander J. Smola, using suffix trees and suffix links to speed up the computation. Jean-Philippe Vert, Hiroto Saigo, and Tatsuya Akutsu present in chapter 6 a different kernel for protein sequences derived from measures of sequence similarities based on the detection of local alignments, also tested on a benchmark experiment of remote homology detection. Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi introduce in chapter 7 kernels for graphs, based on the detection of similar paths between graphs, with applications to the classification of chemical compounds. Kernels between nodes of a graph, called diffusion kernels, are then presented in chapter 8 by Risi Kondor and Jean-Philippe Vert, with applications to the comparison of gene expression and metabolic pathways. Finally, a kernel between short amino acid sequences is introduced in chapter 9 by Yann Guermeur, Alain Lifchitz, and Régis Vert, with application to protein secondary structure prediction.

Part III covers different approaches based on kernel methods to learn from heterogeneous information. In chapter 10, Yoshihiro Yamanishi, Jean-Philippe Vert and Minoru Kanehisa propose to detect correlations between heterogeneous data using a generalization of canonical correlation analysis that involves the kernel trick, and illustrate their approaches by the automatic detection of operons in bacterial genomes. A formalism based on semidefinite programming to combine different kernels representing heterogeneous data is presented in chapter 11 by Gert R. G. Lanckriet, Nello Cristianini, Michael I. Jordan, and William S. Noble, with applications in functional genomics. As a third kernel method to learn from heterogeneous data, Taishin Kin, Tsuyoshi Kato, and Koji Tsuda propose in chapter 12 a formalism based on the information geometry of positive semidefinite matrices to integrate several kernels, with applications in structural genomics.

Part IV contains several examples where SVMs are successfully applied to difficult problems in computational biology. Gunnar Rätsch and Sören Sonnenburg focus in chapter 13 on the problem of splice site prediction in genomic sequences, and develop a state-of-the-art algorithm based on SVMs. Chapter 14, by Balaji Krishnapuram, Lawrence Carin, and Alexander Hartemink, and chapter 15, by

Sepp Hochreiter and Klaus Obermayer, both focus on the classification of tissues based on gene profiling experiments and on the problem of gene selection in this context. They come up with two different variants of the SVM algorithm that perform gene selection and tissue classification simultaneously, with very promising experimental results.

The impetus for this book was a workshop entitled "Kernel Methods in Computational Biology" which was held in the Harnack-Haus of the Max Planck Society in Berlin, on April 14, 2003. The one-day workshop brought together the leading proponents of this emerging field, providing a snapshot of the state of the art. Held at the same time as the RECOMB conference, it attracted an audience of 135 registered participants. The program consisted of nine invited talks, three contributed talks, and five posters. The articles in this book are partly based on presentations at the workshop, augmented with several invited papers. All chapters have been carefully peer-reviewed and edited to produce, we hope, a useful vehicle for helping people getting up to speed on an exciting and promising direction in basic research.

We thank everybody who helped make the workshop and this book possible, in particular Sabrina Nielebock for administrative help with the workshop, Karin Bierig for help with the figures, and Arthur Gretton for proofreading.

Bernhard Schölkopf, Max Planck Institute for Biological Cybernetics, Tübingen, Germany

Koji Tsuda, Max Planck Institute for Biological Cybernetics, Tübingen, Germany, and AIST Computational Biology Research Center, Tokyo, Japan

Jean-Philippe Vert, Ecoles des Mines, Paris, France

I INTRODUCTION

1 A Primer on Molecular Biology

Alexander Zien

Modern molecular biology provides a rich source of challenging machine learning problems. This tutorial chapter aims to provide the necessary biological background knowledge required to communicate with biologists and to understand and properly formalize a number of most interesting problems in this application domain.

The largest part of the chapter (its first section) is devoted to the cell as the basic unit of life. Four aspects of cells are reviewed in sequence: (1) the molecules that cells make use of (above all, proteins, RNA, and DNA); (2) the spatial organization of cells ("compartmentalization"); (3) the way cells produce proteins ("protein expression"); and (4) cellular communication and evolution (of cells and organisms). In the second section, an overview is provided of the most frequent measurement technologies, data types, and data sources. Finally, important open problems in the analysis of these data (bioinformatics challenges) are briefly outlined.

1.1 The Cell

The basic unit of all (biological) life is the cell. A *cell* is basically a watery solution of certain molecules, surrounded by a lipid (fat) membrane. Typical sizes of cells range from 1 μm (bacteria) to 100 μm (plant cells). The most important properties of a living cell (and, in fact, of life itself) are the following:

Life

- It consists of a set of molecules that is separated from the exterior (as a human being is separated from his or her surroundings).
- It has a metabolism, that is, it can take up nutrients and convert them into other molecules and usable energy. The cell uses nutrients to renew its constituents, to grow, and to drive its actions (just like a human does).
- It is able to (approximately) replicate, that is, produce offspring that resemble itself.

- It can react to its environment in a way that tends to prolong its own existence and the existence of a (preferably high) number of offspring.

Viruses, which are simpler than cells, also satisfy some definitions that characterize life: they can, for example, reproduce. But because they depend so strongly on the help of host cells and they do not have their own metabolism, viruses are usually not considered to be alive.

Eukarya, prokarya

Two types of living organisms can be distinguished: *prokarya* (further subdivided into *eubacteria* and *archaea*), which are always single cells, and *eukarya* (which include all animals, plants, and fungi). Eukaryotic cells are more complex than prokarya in that their interior is more organized: the eukaryote is divided into so-called compartments. For instance, the *nucleus* contains hereditary information, and a number of *mitochondria* serve to supply the cell with certain energy-rich molecules.

The incredibly complex machinery of cells cannot be decently described in this short chapter. An excellent and detailed overview can be found in the textbook by Alberts et al. (2002), or, in a shortened version, in Alberts et al. (1998). Here, we try to provide some rough impressions of the absolute basics.

1.1.1 Important Molecules of the Cell

Cells are defined by the molecules they are composed of. Especially important for the integrity of cells are three kinds of macromolecules, which are now introduced. These molecules are *polymers*, which means that they are composed of a large number of covalently[1] linked *monomers*, small molecular building blocks. The set of different monomers and the way they are linked determine the type of polymer.

Nucleotides

DNA The major part of the heritable information of a cell is stored in the form of DNA molecules. They are called the cell's *genome*. *DNA (deoxyribonucleic acid)* is a chain molecule that is composed of linearly linked nucleotides. *Nucleotides* are small chemical compounds. There are essentially four different nucleotides that occur in cellular DNA, which are usually called A (adenine), C (cytosine), G (guanine), and T (thymine).[2] The chain of nucleotides has a direction, because its two ends are chemically different. Consequently, each DNA molecule can be described by a text over a four-letter alphabet. Chemists denote its beginning as the *5′-end* and its end as the *3′-end*. The two directions are denoted by *upstream*, for "towards" the

1. Among the different types of bonds that are possible between atoms, covalent bonds are the strongest. Molecules are defined as the smallest covalently connected sets of atoms; they are often represented by graphs of covalent connections.
2. The restriction to four nucleotides is a simplification that is sufficient for most bioinformatics analysis. In reality, in genomic DNA cytosines may be methylated. This modification can be biologically significant, but it is usually not revealed in the available data.

beginning, and *downstream*, for "towards" the end. Molecular chains of only a few nucleotides are called *oligonucleotides*.

DNA is a good carrier of information that is supposed to be retained for a long time (in fact, usually for the lifetime of a cell, which can be years). DNA can form very stable structures due to the following properties. The nucleotides A and T can bind to each other by forming two hydrogen bonds; therefore, A and T are said to be *complementary*. G and C are also complementary: they form three hydrogen bonds. Importantly, the ability to bind in this way holds for chains of nucleotides, that is, for DNA molecules. The *complement* of a DNA sequence is the sequence of the complements of its bases, but read in the reverse direction; complements are often called *complementary DNA (cDNA)*. Complementary strands can bind to each other tightly by forming a double helix structure, which enables all the hydrogen bonds between the pairs of complementary bases. The binding of two complementary DNA molecules is often referred to as *hybridization*.

Complementarity, hybridization

In cells, the genomic DNA is indeed present in the form of a double helix of two complementary strands, as illustrated in figure 1.1. Apart from the increased stability, this provides redundancy, which serves the cell in two ways. First, erroneous changes from one nucleotide to another, termed *point mutations*, can thereby be detected and corrected. Second, there is a natural way to duplicate the genome, which is necessary when the cell divides to produce two daughter cells. The double helix is separated into two single strands of DNA, each of which then serves as a template for synthesizing its complement. Since the complement of a complement of a DNA sequence is again the primary sequence, the above procedure results in two faithful copies of the original double-stranded DNA.

The size of genomes can be enormous; for instance, the human genome consists of more than 3 billion nucleotides. Although the human genome is separated into 23 separate DNA molecules, each part still has an average length of about 5 cm —about 5000 times longer than the diameter of a human cell! Consequently, the DNA in cells is kept in a highly packaged form. In regular intervals, assemblies of proteins (called *histones*) bind to the DNA. The DNA double helix winds about one and a half times around each histone complex to form a *nucleosome*; the nucleosomes resemble beads on a string (of DNA). The nucleosomes themselves are usually packed on top of one another to form a more compact fibrous form called *chromatin*. An even higher level of packing is achieved by introducing loops into the chromatin fiber. The resulting structures, one for each genomic DNA molecule, are known as *chromosomes*. They do not flow around freely in the nucleus, but are anchored to nuclear structures at sites called *matrix attachment regions (MARs)*.

Genome packing, chromosomes

In many organisms, two or more versions of the genome may be present in a cell. This is called a *diploid* or *polyploid* genome. In contrast, a single set of chromosomes is said to be *haploid*. In sexual organisms, most cells contain a diploid genome, where one version is inherited from each parent. The germ cells giving rise to offspring contain a haploid genome: for each chromosome, they randomly contain either the maternal or the paternal version (or a mixture thereof).

Ploidy

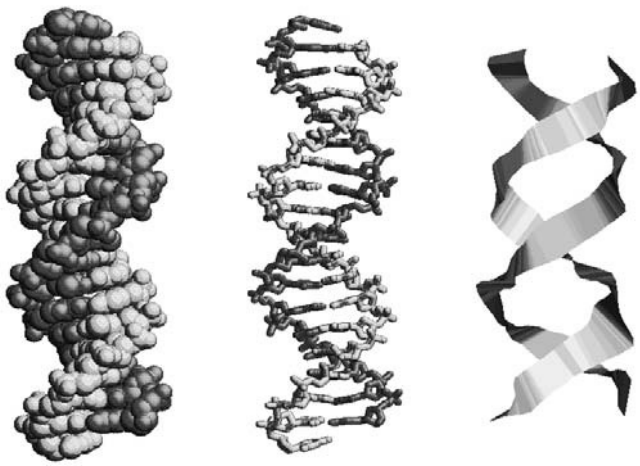


Figure 1.1 The double helix structure of genomic DNA. The same piece of DNA is visualized (using the program RasMol) in three different ways of increasing abstraction. *Left*, spacefill: each atom is shown as a ball. *Middle*, covalent bonds between heavy atoms are shown as sticks. *Right*, each strand of the double helix is shown as a ribbon. (DNA part of PDB entry `1hcq`.)

RNA *RNA (ribonucleic acid)* is very similar to DNA: again, it consists of nucleotides linked in a chain. In contrast to DNA, the nucleotide U (for uracil) is used instead of T, and the chemical details of the nucleotides differ slightly. Due to these difference RNA molecules are usually single-stranded, which allows them to form a variety of structures in three-dimensional (3D) space that can perform complex tasks (such RNAs are called *ribozymes*).

Genes

The importance of the genome is that it typically contains many genes. Although there is still debate about the exact definition, a *gene* can be thought of as a substring of the genome that is responsible for the production of one or a couple of types of RNA molecules. In the process of *gene expression*, the RNA is synthesized to be complementary to a part of the DNA template. As a result, each gene can control one or more properties of the organism, although often quite indirectly, as will become apparent below.

Note that genes also include parts of DNA that are not copied into RNA. Most important, each gene contains a sequence called a promoter, which specifies the conditions under which RNA copies of certain parts of the gene are produced. Although ribozymes are responsible for a few very important tasks in cells, the purpose of the vast majority of genes in a cell is to encode building instructions for

mRNA

proteins (certain macromolecules; see the next paragraph). The RNA molecules

Figure 1.2 Flow of genetic information.

involved in this process are called *messenger RNAs*, or *mRNAs*. In figure 1.2, the flow of information from the DNA to the proteins is illustrated.

Amino acids

Proteins *Proteins* are polymers composed of amino acids. Cells use 20 different types of amino acids for protein synthesis. Common to each amino acid are two chemical groups (an amino [N] group and a carboxyl [C] group) which form *peptide bonds* (a special kind of covalent bond) to link two amino acids. Since a water molecule is split off during the formation of such a bond, a protein is actually composed of *amino acid residues* (often, just *residues*). Proteins are also sometimes called *polypeptides* (most commonly in contexts where their 3D structures are not important); molecules consisting of only a few amino acids are called *oligopeptides*, or simply *peptides*. Due to their chemistry, the beginning and the end of a protein are called its *N-terminus* and its *C-terminus*, respectively. The chain of peptide links forms the *backbone* of a protein. Importantly, each amino acid also has a third group, the *side chain*. The side chains of the 20 natural amino acids show very different chemical properties.

Tertiary structure, fold

Each polypeptide *folds* into an elaborate spatial structure, called its *tertiary structure* or, sloppily, its *fold*. Figure 1.3 should convey an impression of the typical complexity of a fold by showing this structure for an arbitrary protein (in two common graphical representations). The tertiary structure depends on the particular sequence of amino acids (which is also sometimes called the *primary structure* and can be represented by a text over a 20-letter alphabet). The 3D structure of natural proteins[3] is usually assumed to be uniquely determined by the sequence (given cellular conditions such as acidity, the concentrations of ions, etc.).[4] However, sometimes the cell must help to achieve this uniqueness. In these cases, other proteins, named *chaperones*, guide the folding process.

Secondary structure

Among the *structural motifs*, i.e., spatial structures of subpeptides occurring in proteins, two are of exceptional importance: the α helix and the β strand. In an α *helix*, consecutive amino acids assume the shape of a spiral with 3.6 amino acids per turn. This motif is especially stable due to a regular pattern of weak bonds between any amino acid and the fourth next amino acid. In a β strand,

3. That is, proteins that occur in some natural life form.
4. Some studies suggest that the majority of possible sequences do not fold into a unique structure. However, nature appears to prefer sequences that do so; presumably, well-defined structures can contribute more to the cell's survival.

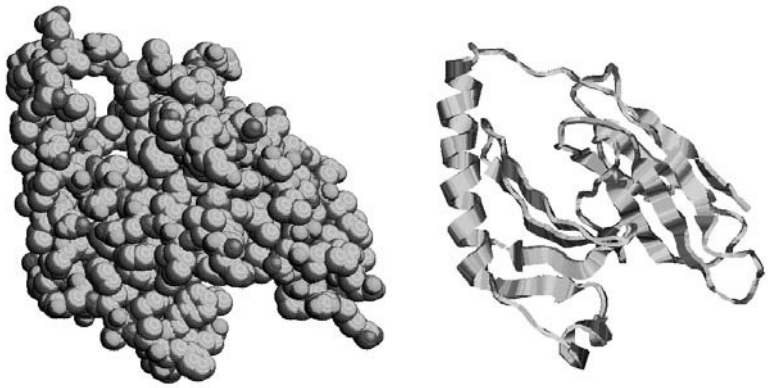


Figure 1.3 Two visual representations of the same protein. *Left*, full atom view. *Right*, the more useful cartoon view. (From PDB entry `1A6A`, drawn by RasMol.)

the backbone is extended and can gain stability from neighboring β strands. The resulting structure, a β *sheet*, again shows a regular pattern of weak bonds, this time between the strands. Together with the *coils*, which subsume the remainder of the protein, α helices and β strands are the elements of *secondary structure*. This is often exploited in schematic representations of proteins, as illustrated in figure 1.4. The secondary structure already determines in large part the complete protein fold, the *tertiary structure*.

Domains

A *domain* is a subunit of a protein which can fold separately into its native structure. Especially in higher organisms (multicellular eukaryotes), many multidomain proteins have evolved, supposedly because recombining domains is an efficient way of creating new proteins that perform useful functions.

Binding

The structure of the backbone of a folded protein determines its overall shape, and also which amino acids are exposed on the surface. Due to the diversity of the side chains, this allows for the generation of a huge variety of patterns of physicochemical properties on protein surfaces. The surface properties determine which other molecules the protein can bind to. The (cellular) function of a protein can most immediately be defined by its set of binding partners and the chemical reactions induced by the binding. For example, many proteins that bind small molecules have cavities, called *binding pockets*, into which the *ligand* (the specific small molecule) fits like a key into a lock. Frequently, the function of a protein requires it to bind to two or more other molecules. This is often achieved through a separate domain for each binding partner.

Functions

The functions of proteins in cells are as diverse as the tasks that cells have to perform. Functional categories include (but are not limited to) the following:

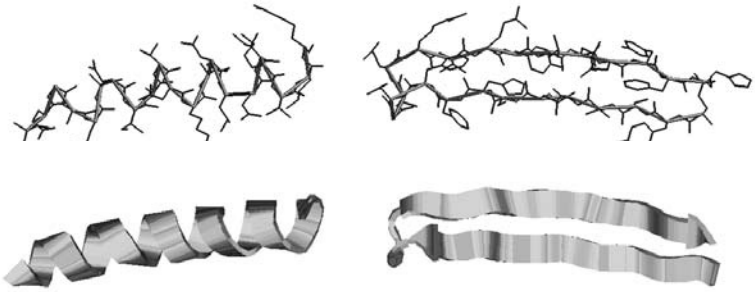


Figure 1.4 Secondary structure elements. *Left*, an α helix. *Right*, a β sheet with two strands. *Top*, stick model of the covalent bonds between heavy atoms; the backbone is emphasized by the thicker sticks. *Bottom*, cartoon view. (From PDB entry `1A6A`, visualized with RasMol.)

- *Metabolism.* Proteins called *enzymes* bind small molecules called *metabolites* to catalyze reactions yielding other small molecules. In this way, nucleotides for DNA and RNA, amino acids for proteins, lipids for membranes, and many other essential compounds are produced. Cells may be viewed as tiny but highly complex and competent chemical factories.
- *Energy.* This can be seen as a special case of metabolism, because cells produce a few types of small molecules as energy carriers.
- *Transcription, protein synthesis, and protein processing.* The huge machinery required to produce proper proteins from DNA is, to a great extent, run by proteins (although ribozymes play a crucial role, too).
- *Transport and motor proteins.* Cells can be more efficient due to a nonrandom spatial distribution of molecules. In particular, compartmentalized cells contain elaborate transport mechanisms to achieve and maintain appropriate local concentrations. Molecular motion can even become visible on a macroscopic scale: muscle contractions are driven by the motion of myosin proteins on actin filaments (longish intracellular structures built from actin proteins).
- *Communication (intra- or intercellular).* Communication is most important for multicellular organisms. While signaling molecules are usually much smaller than proteins, they are received and recognized by proteins. The processing of signals allows computations to be performed; this may be most obvious for the human brain (involving $\sim 10^{11}$ cells), but also underlies the directed motion of unicellular organisms.
- *Cell cycle.* Most cells (be they alone or part of a multicellular organism) recurrently divide into two daughter cells to reproduce. This complex process is orchestrated and carried out by proteins.

A complete list of protein functions is far beyond the scope of this chapter. In summary, proteins are major building blocks of the cell and, above all, the machines that keep cells running.

Macromolecules We have now met the three most important types of macromolecules in the cell (DNA, RNA, and protein) and their relation (the genetic flow of information). A fourth type of macromolecule which also occurs in cells shall only briefly be mentioned here: the polysaccharide. *Polysaccharides* are polymers composed of covalently linked *monosaccharides* (sugars, such as glucose, fructose, galactose). In contrast to the macromolecules discussed earlier, their bonding pattern is not necessarily linear, but often rather treelike. Examples illustrating the relevance of polysaccharides are starch, which is the principal food reserve of plants; glycogen, the counterpart of starch in animals; cellulose, a major constituent of the cell walls of plants; and chitin, which makes up most of the exoskeleton of insects.

Saccharides

In table 1.1, all four types of macromolecules and their most important properties and functions are summarized. Table 1.2 shows their contributions to the total mass of a cell, also in comparison to smaller types of molecules to be described below; not surprisingly, proteins dominate.

Proteins, RNA, and DNA can be parts of even more intricate *assemblies* or, synonymously, *complexes*. For example, as described above, histone proteins are used to pack DNA into chromatin. The *ribosome*, which performs the translation of mRNAs to proteins, is a huge assembly of several proteins and ribosomal RNA (rRNA). The individual molecules in an assembly (which are not connected by covalent bonds) are referred to as *subunits*. Just to make things more confusing, (stable) complexes of proteins (in the sense of individual translation products, as introduced above) are sometimes also called *proteins*; the subunits are then also called *(protein) chains*.

Complexes

Membrane *Membrane* is another huge assembly of smaller units. It mainly consists of a bilayer of lipids (of several different types). A membrane is not a macromolecule, because the lipids are not covalently connected (i.e., they remain separate molecules). Instead, the lipids stick together because they are largely *hydrophobic*, which means that they repel water. By forming a bilayer, all hydrophobic parts contact other hydrophobic surfaces (of other lipid molecules). Only the *hydrophilic* (water-loving) heads of the longish lipids face the water.

Hydrophobicity

The hydrophobicity of the membrane interior prevents water and molecules dissolved in water (which are hydrophilic) from penetrating the membrane. Thus, a membrane is used to separate the cell from its exterior: no large or hydrophilic compounds can pass it directly. In eukaryotes, membranes also serve to enclose compartments (which are subspaces of the cell with distinct chemical properties). To admit the controlled exchange of molecules and also of information, membranes also contain many proteins (often in the sense of protein complexes) that stick out on both sides. The surface of such *membrane-proteins* typically features a hydrophobic ring where it is embedded into the membrane.

Table 1.1 Important macromolecules of the cell. They are composed of small molecules, covalently connected to form linear chains.

Macromolecule	**DNA**	**RNA**
Building blocks	nucleotides (A,C,G,T)	nucleotides (A,C,G,U)
Typical length	1000s to 10^9s	100s to 1000s
Structure	double helix, tightly packed and organized in several levels	complex 3D structure, with structural motifs (secondary structure)
Function	storage of (most of) the hereditary information of an organism: the genome, which contains the genes as subsequences	■ *messenger RNA (mRNA)*: serves as the blueprint for protein production ■ *transfer RNA (tRNA)*: connects codons to amino acids (implementing the genetic code); used by the ribosome ■ *ribosomal RNA (rRNA)*: forms part of the ribosome (amounting to ∼90% of the total RNA)
Location	nucleus, mitochondria, chloroplasts	nucleus, cytosol, mitochondria, chloroplasts

Macromolecule	**Protein**	**Polysaccharides**
Building blocks	amino acids (20 different types)	monosaccharides (several types)
Typical length	10s to 1000s	up to 10^9 (e.g., starch)
Structure	complex and versatile, with structural motifs (secondary structure, domains, etc.)	often not linearly bonded but tree-like
Function	Extremely diverse. For example, ■ *enzymes* catalyze reactions of other molecules; ■ *structural proteins* build and stabilize the structure of the cell; ■ *receptors*, *kinases*, and other proteins receive, transport, and process signals from the exterior; ■ *transcription factors (TF)* regulate the production of all proteins.	■ modification of proteins and their properties ■ storage of energy (e.g., in starch) ■ structural stability (e.g., in chitin) ■ storage of water (e.g., in extracellular matrix in cartilage)
Location	everywhere in- and outside cell; dissolved in water or embedded in a membrane	everywhere in- and outside cell; often bound to proteins

Table 1.2 Approximate fractions of different classes of molecules of the total weight of a typical cell.

Molecule type	Cell Mass in Bacteria	Cell Mass in Mammals
H_2O (water)	70%	70%
DNA	1%	0.25%
RNA	6%	1%
proteins	15%	18%
lipids (fat)	2%	5%
polysaccharides (sugar)	2%	2%
metabolites and inorganic ions	4%	4%

Metabolites Of course, small molecules are vital for cells, too. Here we give just a few selected examples:

- Adenosine triphosphate (ATP) and NADPH (both derived from the nucleotide A) serve as ubiquitous ready-to-use sources of energy.
- Monosaccharides (sugars) and lipids (fats) can be converted into ATP, and therefore serve as a long-term source of energy. Saccharides are also often attached to proteins to modify their properties.
- *Signaling molecules* convey information by docking to their respective receptor proteins and triggering their action. For example, steroids (which include many sex hormones) can diffuse into a cell's nucleus and induce the activation of some genes.

Small molecules are more generally called *compounds*.

1.1.2 Compartmentalization of the Eukaryotic Cell

As mentioned earlier, eukaryotic cells contain many *compartments*, which are also called *organelles*. They are subspaces that are enclosed by single or double membranes. Figure 1.5 provides an overview of the major compartments in the cell, and table 1.3 summarizes some of their properties.

In each compartment, a cell maintains different concentrations of relevant molecules. This way, the compartmentalization allows the cell to perform diverse tasks and chemical reactions that require different environments (e.g., a certain acidity) efficiently. As an example, the bulk of a cell's hereditary information is stored as DNA molecules (condensed into chromatin) in the nucleus, where the transcription machinery (which produces mRNA copies from genes) can more easily find it than if the DNA were allowed to reside anywhere in the cell.

Subcellular localization

Since each type of compartment is devoted to different tasks in the cell, each requires a distinct set of proteins to perform the subtasks. In order to save resources, proteins are specifically delivered to the organelles that require them. Consequently, many proteins contain signals that specify their destination. These signals can either

Table 1.3 Important compartments of the eukaryotic cell. Chloroplasts (which occur in plants, but not in animals) and mitochondria contain their own (small) genomes and produce a part of the proteins they require themselves; they have probably evolved from enclosed prokaryotic cells.

Compartment	**Function(s)**	**Membrane**
Cytosol	protein synthesis, general metabolism, etc.	single
Nucleus	■ storage of main genome (DNA molecules) ■ RNA synthesis ■ ribosome synthesis (in the nucleolus)	double
Endoplasmatic reticulum (ER) (inner space of nuclear membrane, extending throughout the cell)	■ synthesis of most lipids (membrane) ■ synthesis of proteins for single-membrane organelles (rough ER) ■ post-translational processing of those proteins	single
Golgi apparatus	■ post-translational processing of proteins ■ distribution of proteins and lipids to single-membrane organelles	single
Vesicles (mobile bubbles)	transport of proteins and membrane between single-membrane organelles and to/from cell exterior	single
Endosomes	■ contain material taken up from the exterior; or ■ secrete contents (mainly proteins) to cell exterior	single
Lysosomes/vacuoles (plants, fungi)	digest of molecules, organelles, etc. / store waste and nutrients, control cell size	single
Peroxisomes	carry out oxidative (dangerous) reactions	single
Cell exterior / extracellular matrix	■ extracellular matrix connects cells, stabilizes the organism, contains nutrients, etc. ■ in polarized cells (e.g., nerve cells), the exterior is divided into basolateral and apical parts	single
Mitochondria	generate ATP by oxidizing nutrients	double
Chloroplasts (in plants)	generate energy-rich molecules from sunlight	double

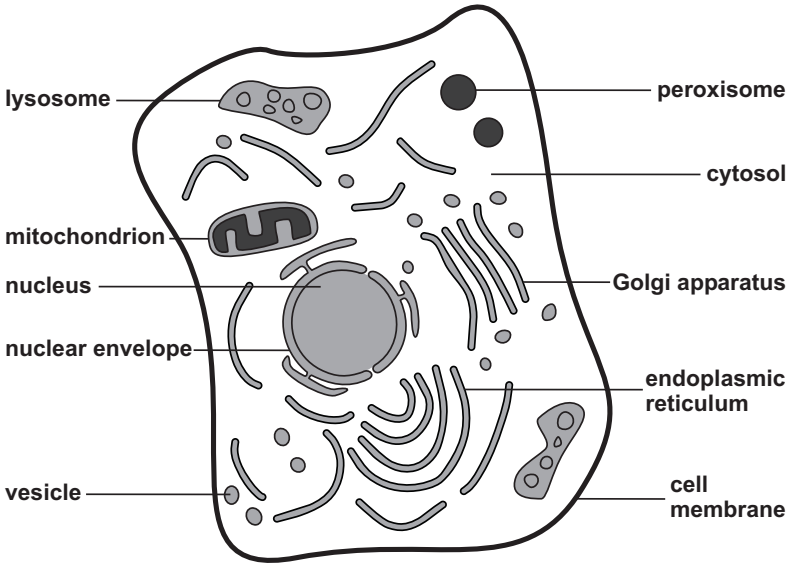


Figure 1.5 Compartments in a eukaryotic cell. All lines represent membranes. The interior of all compartments is shaded in gray; the cytosol is white. Inspired by a figure in Alberts et al. (1998) and crafted by Karin Bierig.

be entire peptides (e.g., hydrophobic stretches for transfer into the endoplasmatic reticulum [ER]) or characteristic surface patches of the folded protein. There are also default destinations when signals are absent: proteins showing no signal at all stay in the cytosol. The *subcellular localization* is obviously closely related to the function of the protein.

It should be noted that cells are in general not spatially symmetric. For example, the surface of many cells in multicellular organisms is divided into two domains: the apical and the basolateral. An extreme case is provided by nerve cells: their apical part consists of *axons*, thin extensions (that can be as long as 2 m in the human) which connect a neuron to other neurons. The *exocytotic* pathway, which transports proteins to the cell exterior, can distinguish between the two regions.

1.1.3 Expression of Genes and Proteins

One of the most fundamental processes in the cell is the production (and disposal) of proteins. Below, the life cycle of proteins is outlined for eukaryotic cells.

1. *Transcription. Messenger RNA (mRNA)* copies of a gene are produced. The products, called *pre-mRNAs* (since they are not yet spliced; see step 2), are complementary to the DNA sequence.

   (a) Initiation: Certain proteins, called *transcription factors (TFs)*, bind to *TF binding sites* in the gene promoters in the DNA.

   (b) Elongation: The mRNA copy of the gene is synthesized by a special protein (RNA polymerase II). It moves along the DNA and thereby sequentially extends the pre-mRNA by linking a nucleotide complementary to that found in the DNA.

   (c) Termination: A signal in the DNA causes the transcription to end and the mRNA to be released.

2. *Splicing.* Parts of the pre-mRNA, which are called *introns*, are removed. The remaining parts, called *exons*, are reconnected to form the mature mRNA. The spliced mRNAs travel from the nucleus (through huge, selective pores in its double membrane) into the cytosol. To increase the chemical stability of the mRNA, a chemical cap is formed at the $5'$-end and a *poly(A)* sequence (built from many A nucleotides) is appended to the $3'$-end.

3. *Translation.* In the cytosol, ribosomes await the mRNAs. Ribosomes synthesize proteins as specified by *codons* —triplets of consecutive nucleotides—in the mRNA.

   (a) Initiation: The ribosome finds a *start codon* (usually, the first AUG subsequence that has favorable neighboring nucleotides) in the mRNA.

   (b) Elongation: One by one, the ribosome attaches amino acids to the growing polypeptide (protein) chain. In each step, the ribosome translates the current codon into an amino acid according to the *genetic code*. The ribosome then moves to the next codon in the same *reading frame*, that is, to the next adjacent nonoverlapping codon.

   (c) Termination: Translation is stopped by any of three different *stop codons* encountered in the current reading frame.

4. *(Posttranslational) modification* (not for all proteins). The protein may be chemically modified, if it contains the relevant signals and if it resides in a compartment where these signals are recognized.

   (a) Additional chemical groups can be covalently attached to proteins (glycosylation (sugars), phosphorylation, methylation, etc).

   (b) Covalent bonds can be formed between amino acids.

   (c) Proteins can be covalently bound to each other.

   (d) Proteins can be cleaved, that is, cut into parts.

5. *Translocation* (not for all proteins). Proteins are delivered to the appropriate compartment, which is specified by signals in the amino acid sequence. The signal can either be a typical short segment of a sequence, or a structural motif on the surface of the protein (which may be composed of amino acids that are not neighbors in the sequence). In the absence of signals, the protein stays in the cytosol.

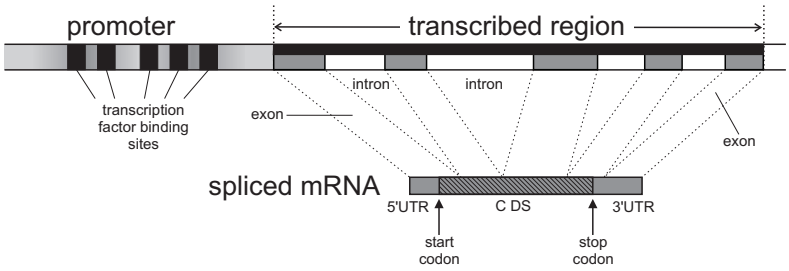


Figure 1.6 Typical gene structure in eukaryotes. At the top of the figure, a section of a genome is shown that contains a gene. Important features of the gene are the promoter containing several TF binding sites and the transcribed region, which is partitioned into exons and introns. After transcription, the pre-mRNA (not shown) is spliced: the introns are cut out; thus, the mature mRNA is the concatenation of the exons. Only a part of the mRNA encodes a protein (CDS, for coding sequence); the other parts are called UTRs (untranslated regions). Artwork by Karin Bierig.

6. *Degradation.* Almost all proteins are eventually destroyed by digestion into their individual amino acids.

In prokaryotes, the entire process is a bit less complex because splicing is uncommon and the translocation has only three different targets (cytosol, membrane, exterior) due to the lack of compartments.

Alternative splicing

The process of splicing implies complex *gene structures* composed of alternating introns and exons; an illustration is given in figure 1.6. However, it allows for increased flexibility by a mechanism known as *alternative splicing*: certain proteins can cause certain exons to be lengthened, shortened, or even skipped completely. Thus, the same gene can give rise to the production of different proteins. This is an important way for cells to adapt to the circumstances, including their cell type and extracellular signals. It is estimated that a human gene on average encodes for eight or nine different proteins. More detailed information on the process of splicing and its biological implications can be found in chapter 13, section 13.2.

Expression levels

Steps 1 and 2 of the scheme described above are called *gene expression*, while steps 1 through 5 are called *protein expression*. The term *expression level* of a molecule type is (a bit imprecisely) used to refer to either its current abundance in the cell, or to the rate of synthesis of new molecules. This difference is often neglected for gene expression, which may or may not be justified by the fact that mRNAs are degraded relatively quickly after having been translated several times. However, for proteins the distinction is crucial, because their lifetimes may be very long and differ vastly.

Regulation

The cellular concentration of any type of protein can be influenced by changing the efficiencies of the above steps. This is called *regulation* of expression. While cells in fact regulate each of the above steps, the main point for the quantitative

control of protein expression is certainly transcription initiation. In addition to the general TFs, which are always required for initiation, there are additional TFs which modify the probability or speed of transcription. They bind to short DNA motifs, for obvious reasons called *enhancers* and *silencers*, in the promoter. The effects of TF binding sites can extend over huge distances in the DNA sequence; therefore *insulators* (certain DNA signals) may be required to separate genes from each other and prevent mutual regulatory interference.

The steps of protein expression have a natural temporal ordering, where each step operates on the result of the preceding step. However, there are at least three types of deviation from a clear, serial manufacturing process: (1) Some of the steps may occur concurrently, or can be performed before the preceding step is finished. For example, much of the splicing is carried out while the gene is still being transcribed. Also, the translocation from the cytosol into the ER and some modifications take place during translation. (2) There is no compulsory ordering of translocation and modification. In fact, many proteins are modified in the ER and the Golgi apparatus, which are intermediate stations on the journey to their destination compartment (cf. section 1.1.2). (3) Degradation may occur even before the protein is finished and delivered.

In many cases, the mentioned exceptions relate to the *folding* of the newly synthesized protein into a 3D structure. A protein can already start to fold while it is still growing out of the ribosome, and modifications by other proteins at that time can have an impact on the way it folds. Some proteins are aided in finding the desired structure by helper proteins (*chaperones*), which, for instance, unfold incorrectly folded proteins. In case a protein repeatedly misfolds (i.e., does not assume the intended structure despite the help of chaperones), it can also be degraded.

1.1.4 Beyond the Cell

Cell Communication Cells, especially those in the same multicellular organism, can communicate by the exchange of extracellular *signal molecules*. This way, the coordinated action of many (in the case of a grown human, on the order of 10^{10} or 10^{11}) cells can be achieved. Even to perform no action requires signaling; in animals, cells that do not get a constant supply of certain signals from their neighbors commit *apoptosis*, that is, self-destruction. This is a safety provision used to eliminate malfunctioning cells; if the mechanism gets broken itself, uncontrolled proliferation (cancer) may result.

Depending on the properties of the emitted signal molecules, the signaling can affect neighboring cells only (*contact-dependent*); it can be locally restricted to a small cluster of cells (*paracrine*); or it can rely on distribution through the blood system (*endocrine*). A special case is the *synaptic signaling*, in which the electric signal transmitted by a neuron causes neurotransmitters to be released that induce an electric potential in the receiving neuron. As in endocrine signaling, the signal is carried over large distances (as electrical potential through the long axons of the neurons). But in contrast to endocrine signaling, the signaling molecules travel only

a very short distance extracellularly and are transmitted to a very specific set of target cells.

Extracellular signals can take one of two routes into the cell: through *cell surface receptors* or directly to *intracellular receptors*. For the latter, the signaling molecules have to traverse the cell membrane. This is possible for small hydrophobic molecules like steroid hormones (which include, e.g., the sex hormone testosterone). When such a molecule binds to the corresponding receptor protein, the protein usually travels to the nucleus (or it may already have been there) and activates or inhibits the transcription of one or several genes.

Signaling molecules that cannot permeate the cell membrane are recognized by cell surface receptors, to which they bind extracellularly. These receptors reside in the membrane and also have an intracellular part. The extracellular binding of the signaling molecule induces an action by the intracellular part, for example, a change of the 3D structure. In response to this change, a series of downstream actions begins: cytosolic proteins modify each other in a chain, until finally a TF is activated or deactivated or the reaction rate of an enzyme is altered.

Evolution The complexity of cells and organisms has evolved over several billion years of interplay of mutation and selection. Here, *mutation* means any kind of modification of the heritable information (basically the genome) of reproductive cells. The totality of heritable information giving rise to an organism is called its *genotype*, as opposed to *phenotype*, which subsumes the observable physical properties of the organism. Differences in the genotype sometimes manifest in different phenotypes; otherwise, the corresponding mutations are said to be *silent*.

Selection

Selection refers to the fact that the phenotypic changes may lead to differential reproductive success (e.g., some mutations are directly lethal); this may correlate with the organism's ability to survive in its environment. Often, however, mutations have no or a negligible impact on survival and reproduction (even if they are not silent). Several different genotypes (and possibly phenotypes) may then coexist in a population. In this case, their genetic differences are called *polymorphisms*.

Mutations

There are several different types of mutations. The simplest is the *point mutation* or *substitution*; here, a single nucleotide in the genome is changed. In the case of polymorphisms, they are called *single nucleotide polymorphisms (SNPs)*. Other types of mutations include the following:

- *Insertion.* A piece of DNA is inserted into the genome at a certain position.
- *Deletion.* A piece of DNA is cut from the genome at a certain position.
- *Inversion.* A piece of DNA is cut, flipped around and then re-inserted, thereby converting it into its complement.
- *Translocation.* A piece of DNA is moved to a different position.
- *Duplication.* A copy of a piece of DNA is inserted into the genome.

The term *rearrangement* subsumes inversion and translocation.

Genetic diversity

While mutations can be detrimental to the affected individual, they can also in rare cases be beneficial; or, much more frequently, just neutral (under the actual circumstances). Thereby mutations can increase the *genetic diversity* of a population, that is, the number of present polymorphisms. In combination with selection, this allow a species to adapt to changing environmental conditions and to survive in the long term. For example, many viruses (such as HIV) have imprecise replication mechanisms to produce a large fraction of mutants among the huge number of descendants. This way, subpopulations are created that do not match the patterns that the immune system of their host is looking for.

Many bacteria have evolved a strategy to achieve more complex mutations: by *horizontal gene transfer*, genetic material is not received from parental cells, but from other cells which may even belong to a different species. *Transposons*—mobile segments of DNA that can move around or copy themselves within the genome—presumably also serve to generate (certain kinds of) mutations. Sexual reproduction can be viewed as a sophisticated (and presumably more efficient) alternative to mutations for the purpose of maintaining genetic diversity. In sexual species, each individual owns a diploid genome (consisting of two different copies). During reproduction, the parental genomes are recombined on the basis of entire chromosomes and fragments of chromosomes ("crossing over"). In contrast to mutations, this almost always leads to offspring that can survive.

Sexual reproduction

In the course of evolution, populations of organisms often separate (e.g., spatially) and develop over time into distinct species. While the differences are the result of accumulating mutations, the genomes of the descendant species still share significant similarity. In particular, many encoded proteins remain similar; such proteins are said to be *orthologs*. If proteins within the same genome are similar due to a common origin (as the result of duplications), they are called *paralogs*. *Homology* refers to any kind of evolutionary relatedness, be it orthologous or paralogous. Homologous proteins must be distinguished from *analogous* proteins, which have the same function but have evolved independently (*convergent evolution*).

1.2 Molecular Biology Measurement Data

Modern molecular biology is characterized by the (usually highly automated) collection of large volumes of data. A large number of existing measurement technologies serve to produce data on various aspects of cells and organisms. Table 1.4 provides an overview of the most common data types.

For many molecular biology data types, more than one measurement technology exists. Serious analysis must be performed bearing this in mind. For example, protein structures can be resolved either by NMR (nuclear magnetic resonance) or by x-ray crystallography. (Both methods are sciences in themselves and are hard to apply, if they work at all for a particular protein.) For some analyses, the source of the data can make a difference: apart from the lower resolution of the NMR structures, the structures may show systematic differences in the amino acids on

Table 1.4 Common genomics data types and their representation for computational analysis.

Data Type and Details	Representation
Sequences	
■ **DNA**: genome (hereditary information)	string over nucleotides {A,C,G,T}
■ **full-length mRNAs**: spliced gene copies	string over ribonucleotides {A,C,G,U}
■ **ESTs** (expressed sequence tags): partial mRNAs	string over ribonucleotides {A,C,G,U}
■ **proteins**	string over amino acids (size 20)
Structures	
■ **metabolites**: positions and bonds of atoms	labeled graph embedded in 3D space
■ **macromolecules** (proteins, RNAs, DNA)	labeled graph embedded in 3D space
Interactions	
■ **proteins with metabolites**: receptors or enzymes binding ligands	real vectors (binding energies)
■ **proteins with DNA**: transcription factors, etc.	binary (bipartite graph)
■ **proteins with proteins**: complexes, etc.	binary (graph); Petri-net
Expression / localization data	
■ **gene expression**: abundances of mRNAs	real vectors or matrices
■ **protein expression**: abundances of proteins	real vectors or matrices
■ **metabolite (small molecule) "expression"**: concentrations of metabolites	real vectors or matrices
■ **protein localization**: compartment of presence	categorical
Cell / organism data	
■ **genotype**: single nucleotide polymorphisms	vector of nucleotides {A,C,G,T}
■ **phenotype**: cell type, size, gender, eye color, etc.	vector of real and categorical attributes
■ **state/clinical data**: disease, blood sugar, etc.	vector of real and categorical attributes
■ **environment**: nutrients, temperature, etc.	vector of real and categorical attributes
Population data	
■ **linkage disequilibrium**: LOD scores	real numbers
■ **pedigrees**	certain (treelike) graphs
■ **phylogenies**: "pedigree of species"	trees or generalizations of trees
Scientific texts	
■ **texts**: articles, abstracts, webpages	natural language texts (in English)

the protein surface, because they are in contact with water for NMR whereas they are in contact with a neighboring protein in the crystal used for x-ray diffraction. The problems become worse for gene expression data, where the preprocessing is also crucial, as stressed below.

A few data types are so fundamental and frequent that we discuss them in the following sections.

1.2.1 Sequence Data

Sequencing

The classic molecular biology data type is the sequence (more precisely, the DNA sequence). The process of "measuring" the sequence of nucleotides in a piece of DNA is called *sequencing* and is presently highly automated. Still, it is far from trivial. First, the sequencing process requires a huge number of identical DNA molecules. These can be gained from a small sample (or even a single molecule) by *amplification* through the *polymerase chain reaction (PCR)*. A more severe shortcoming is that only a few hundred up to about one thousand consecutive nucleotides of a piece of DNA can be determined in one run.

Nevertheless, is has become almost routine to sequence entire genomes. To that end, the DNA is first split into parts which are sequenced separately. The resulting set of sequences must be computationally assembled into the contiguous genome. Although techniques for the determination of protein sequences exist, it is nowadays common to sequence mRNAs (after first converting them to cDNA) or complete genomes, and then compute the translation products.

Sequence databases

Table 1.5 provides an overview of major sequence databases and portals on the Internet. Care is required when using these databases with machine learning methods: a major assumption of many algorithms, namely, that the data are *iid* (independent and identically distributed), is violated in most databases. The reason is that the proteins considered most interesting have been studied in great detail (i.e., in many species and versions) by biologists, and are therefore overrepresented. A common solution to this is redundancy reduction (the elimination of similar sequences), as provided, for instance, by the ASTRAL server at `http://astral.stanford.edu/`.

Apart from the big general sequence databases there exist a large number of more specialized databases, which often provide additional information that may be linked to the sequences. They cannot be listed here, but the journal *Nucleic Acids Research* provides reports on many of them in the first issue of each year. In addition, many of these databases are accessible via SRS.

Alignments A most basic and most important bioinformatics task is to find the set of homologs for a given sequence. Since sequences are a very important data type (not only for bioinformatics but also in other areas), new methods for sequence comparison are being developed. The new string kernels presented in chapters 4 and 5 are examples of such techniques. Nevertheless, the established methods continue to be widely used, and often serve as a crucial ingredient to

Table 1.5 Databases of molecular biological sequences.

Database	URL (`http://...`)	Remarks
Nucleotide sequence databases		
■ DDBJ	`www.ddbj.nig.ac.jp`	these three databases
■ EMBL	`www.ebi.ac.uk/embl/`	synchronize their
■ GenBank	`www.ncbi.nlm.nih.gov`	contents daily
Protein sequence databases		
■ SwissProt	`www.expasy.org/sprot/`	curated
■ TrEMBL	`www.expasy.org/sprot/`	not curated
(Some) Sequence motif databases		
■ eMotif	`motif.stanford.edu/emotif/`	protein regular expression patterns
■ SMART	`smart.embl-heidelberg.de/`	protein domain HMMs
■ TRANSFAC	`transfac.gbf.de/TRANSFAC/`	genomic TF binding sites
General portals		
■ EBI	`www.ebi.ac.uk`	European Bioinformatics Institute
■ Entrez	`www.ncbi.nlm.nih.gov/Entrez/`	U.S. National Bioinformatics Institute
■ ExPASy	`www.expasy.org`	Expert Protein Analysis System
■ SRS	`srs.ebi.ac.uk`	Sequence Retrieval System

machine learning approaches (cf. chapters 3 and 6). Here we briefly introduce the the most fundamental sequence analysis technique: the alignment.

Alignment

In a *global alignment* of two sequences $s = s_1 \dots s_{|s|}$ and $t = t_1 \dots t_{|t|}$, each sequence may be elongated by inserting copies of a special symbol (the dash, "-") at any position, yielding two stuffed sequences s' and t'. The first requirement is that the stuffed sequences have the same length. This allows them to be written on top of each other, so that each symbol of s is either mapped to a symbol of t (*substitution*), or mapped to a dash (*gap*), and vice versa. The second requirement for a valid alignment is that no dash be mapped to a dash, which restricts the length of any global alignment to a maximum of $|s| + |t|$. In a *local alignment*, a substring of s is globally aligned to a substring of t.

Optimal alignment

For aligning biological sequences, scores reflecting the probabilities of insertions/deletions and of mutations are assigned to gaps and to all different possible substitutions. The score of an entire alignment is defined as the sum of the individual scores. The similarity of s and t is often defined as the score of an optimal local alignment of s and t, where optimal means maximizing the score. Although there are exponentially many possible alignments (whether local or global), the optimal cost and an optimal alignment (of either mode) can be computed in time $\mathcal{O}(|s||t|)$ using dynamic programming (Needleman and Wunsch, 1970; Smith and Waterman, 1981).

Fast heuristics

Since quadratic time is still too slow for searching large databases, fast heuristics have been developed. FASTA and BLAST are much faster than dynamic programming, but achieve results almost as good. However, a much better measure of similarity can be computed by taking into account the distribution of closely related sequences. PSI-BLAST (Altschul et al., 1997) constructs a *multiple alignment* for each query sequence that consists of all similar sequences found in the database. From this a *position-specific scoring matrix (PSSM)* is constructed with which the database is searched again, thereby increasing the sensitivity of the search. This has become the most widely used method for making use of unlabeled data in supervised problems like protein classification.

Probabilistic similarity measures

With either alignment method, the obtained score depends on the length of the two sequences. For local alignments, this is compensated for by the use of so-called *p-values* or *E-values*, which quantify the chance of finding a random similarity in terms of probabilities or expected numbers of hits, respectively. Other methods of obtaining probabilistic similarity measures are based on *hidden Markov models (HMMs)* and Bayesian reasoning. An excellent textbook on alignments and related topics is Durbin et al. (1998).

1.2.2 Gene Expression Data

Gene expression data usually come in the form of a matrix of expression levels for a number of genes in a range of cell samples. There are quite a few technologies available to measure the level of expression of a large number of genes simultaneously. We focus on microarrays, which are presently the most popular technology for large-scale gene expression measurement. Then we outline two competing techniques that are based on a different approach. Finally, we discuss some implications for data analysis.

Microarrays Microarrays, sometimes also called DNA chips, employ hybridization to distinguish different genes, and therefore require that the sequences of genes to be measured be known in advance. In fact, a *microarray* is essentially a surface with a known location (called *spot*) for each gene to be measured. Present-day microarrays can bear a couple of thousand spots, so that the entire human genome can be covered with four chips. At each spot, oligonucleotides or cDNA fragments are fixed which are complementary to a (transcribed) subsequence of a gene. Ideally, the subsequences are determined in such a way that they are specific to the corresponding gene, that is, they are not similar to the complement of any other mRNA that is expected to occur in the sample.

Microarrays

Measurement (hybridization)

The measurement of a sample with a microarray (jargon: *hybridization*) begins by reverse-transcribing the mRNAs of a cell sample to cDNA. The cDNAs are labeled to make them detectable, for instance, by incorporating fluorescing or radioactive tags. Then, the sample is administered onto the microarray, and a number of cDNAs from the sample hybridize to the corresponding spot. This number is approximately proportional to the respective mRNA concentration in the sample. After washing

the array the concentration can be determined by measuring, at the corresponding spot, the intensity of the signal emitted by the molecular labels. If two different molecular labelings are used for two different samples, two measurements can be carried out at the same time on the same array.

Noise

A couple of facts are essential to the proper analysis of microarray data. First, *background noise* is present due to incomplete washing and nonspecific hybridization. If the mean background is estimated and subtracted, the resulting expression levels may become negative for some genes. Although true expression levels cannot be negative, statistical work seems to suggest that it is not a good idea to censor such values by setting them to zero or a small positive value; instead, variance-stabilizing transformations may be used. Second, due to varying efficiencies of the intermediary steps of the measurement, and to varying amounts of mRNA per cell, the results obtained with different microarrays or for different samples are not

Normalization

likely to be on the same scale. *Normalization* should therefore be applied. Moreover, systematic differences that arise from fluctuations between production batches should be compensated for. Finally, even normalization cannot make comparable data gathered with microarrays that are equipped with different oligonucleotides (e.g., chips of different brands). This is because the oligonucleotides have different hybridization energies which introduce a scaling constant for every spot (gene).

Other technologies A few methods for measuring gene expression levels are based on sequencing, clustering, and counting mRNA molecules, as detailed in the following three-step strategy:

1. *Sequencing.* This involves randomly picking mRNA molecules from the sample, reverse-transcribing them to cDNA, amplifying them, and then determining (parts of) their sequences.
2. *Clustering.* Sequences corresponding to the same genes must be identified.
3. *Counting.* The cluster sizes are estimates of the expression levels.

ESTs, SAGE

Two examples from this group of methods are *serial analysis of gene expression (SAGE)* and *expressed sequence tag (EST)* analysis. An EST results from partial single-pass sequencing of a transcript; it represents an error-prone substring of a (usually spliced) mRNA.

In contrast to microarray measurements, counting sequence tags (as in SAGE or EST analysis) yields natural numbers as outputs. The difficulty of analysis arises from three facts: (1) The number of sampled molecules is low (ESTs) to medium (SAGE) due to the effort (and cost) required. Thus, the counts are bad estimates of the true frequencies, especially for the low copy genes, which may not be detected at all. (2) Sequencing errors may lead to inclusion of a sequence in the wrong cluster. (3) Clusters may erroneously be merged (e.g., for very homologous genes) or split (e.g., if sequenced parts do not overlap). The importance of sequencing-based methods is that they do not require prior knowledge of the genes; therefore, they can in principle be applied to any genome.

Table 1.6 Major databases of gene expression data sets.

Database	URL (`http://...`)	Remarks
General databases		
■ ArrayExpress	`www.ebi.ac.uk/arrayexpress/`	by the EBI
■ GEO	`www.ncbi.nlm.nih.gov/geo/`	by the NCBI
Organism-specific databases		
■ MGI GXD	`www.informatics.jax.org`	mouse
■ TAIR Microarray	`www.arabidopsis.org`	*Arabidopsis*
■ WormBase	`www.wormbase.org`	*Caenorhabditis elegans*
Laboratory-specific databases		
■ SMD	`genome-www.stanford.edu/microarray/`	Stanford
■ YMD	`info.med.yale.edu/microarray/`	Yale

Northern blotting, quantitative PCR

Two other methods deserve mention, because they are frequently used by biologists. *Northern blotting* is the oldest approach to (semiquantitative) measurement of gene expression. In contrast, *quantitative polymerase chain reaction (qPCR)* is a very modern method, and is currently considered to allow for the most precise determination of expression levels. Both methods require prior knowledge of the sequence. In addition, they are not sufficiently automated for mass measurements; instead, they are used to confirm findings made with other technologies.

Databases Unfortunately, the databases for gene expression data are not yet as established as the sequence databases are. It is still common for microarray data sets to be available only from webpages that accompany a publication. Nevertheless, there are two databases that try to be very general. These and a few databases with more specialized domains are listed in table 1.6.

One reason for the slow establishment of general gene expression databases may be that it is surprisingly difficult to properly design its scheme. To be really useful, there is no point in just storing the matrices of measurement values. Instead, a description of the preprocessing of the data, the measurement technology (including details of the biochemical steps carried out in the laboratory), and above all the properties of the samples, must be given. For samples taken from hospital patients, for example, a complete description would ideally include the entire clinical data.

1.2.3 Protein Data

Reflecting their importance for cells, many aspects of proteins other than their sequences are wellstudied, including (tertiary) structures, interactions, functions, expression, and localization. Many of these data may be found in databases; see table 1.7.

The unique worldwide database of protein structures is the PDB (protein database). However, detailed structure comparisons are tedious, and thankfully

Table 1.7 Important databases on protein properties other than sequence.

Database	URL (`http://...`)	Remarks
Protein structures		
■ PDB	`www.rcsb.org/pdb/`	3D structures
■ SCOP	`scop.mrc-lmb.cam.ac.uk/scop/`	structural classification
■ CATH	`www.biochem.ucl.ac.uk/bsm/cath/`	structural classification
Molecular interactions and networks		
■ BIND	`www.bind.ca`	interaction network
■ KEGG	`www.genome.ad.jp/kegg/`	metabolic pathways
■ DIP	`dip.doe-mbi.ucla.edu`	interacting proteins
Protein functions		
■ GO	`www.geneontology.org`	controlled vocabulary
■ EC	`www.chem.qmul.ac.uk/iubmb/enzyme/`	enzyme numbers
■ MIPS	`mips.gsf.de/proj/yeast/catalogs/funcat/`	yeast gene functions
Protein expression		
■ 2DPAGE	`us.expasy.org/ch2d/`	2D gel electrophoresis data

databases exist that provide hierarchical classifications of the PDB proteins (or the domains therein) according to their structures. The most popular may be SCOP ("structural classification of proteins", which is largely manually constructed by experts) and CATH ("Class, Architecture, Topology and Homologous superfamily", which relies more heavily on automatic classification).

Protein *interactions* can be defined on several levels: *molecular interactions* refer to the binding partners of proteins, while the broader notion of *regulatory interactions* also includes indirect influences like up- or downregulation through signaling pathways. Proteins can also be linked in a *metabolic pathway* if they catalyze successive steps in a series of metabolic reactions. Any such interactions, also with other molecules like DNA, can be used as edges in a *biological network* graph. A couple of databases provide interactions measured or inferred by different methods.

There are also databases providing functional classifications of proteins (or of the respective genes). Protein function is related to interactions, localization, and expression. There are two popular ways to measure protein expression: (1) by *2D gel electrophoresis*, in which proteins are spatially separated in a gel according to mass and electric charge; and (2) by *mass spectrometry (MS)*, in which the masses of protein fragments are very precisely inferred by measuring their time of flight after a defined acceleration. However, to the best of our knowledge, no general databases containing MS protein expression levels exist yet; nor do such databases exist for protein localization.

Table 1.8 Databases on data types not covered in the previous tables.

Database	URL (http://...)	Remarks
dbSNP	www.ncbi.nlm.nih.gov/SNP/	single nucleotide polymorphisms
PubMed	www.ncbi.nlm.nih.gov/PubMed/	publication abstracts
NCI	cactus.nci.nih.gov	small molecule structures

Table 1.9 Collections of links to databases on the web.

URL (http://...)
bip.weizmann.ac.il/mb/molecular_biol_databases.html
molbio.info.nih.gov/molbio/db.html

1.2.4 Other Data Types

A number of data types additional to those discussed above also deserve mention. (1) Chemical compounds (small molecules) are interesting as metabolites, signaling molecules, and potential drugs. (2) SNPs account for most phenotypic differences between individuals, including susceptibility to many diseases. (3) The abstracts of the scientific molecular biology publications form a huge reservoir of badly structured data. These are the targets of text mining, which attempts to automatically extract information. Sources of these three types of data are listed in table 1.8; in addition, the URLs of two database directories are given in table 1.9.

Model organisms

Model organisms are organisms chosen by biologists to be representative of some class or property and, at the same time, to be simple and accessible. For example, the fruit fly *Drosophila melanogaster* shares many genes and somatic functions with humans, but is much easier to investigate (no ethical problems, short reproduction cycle, etc.). The value of model organisms for bioinformatics lies in the fact that not only are (almost) complete genomes available for them (this is now the case for hundreds of organisms) but also plenty of other data which can be set into relation with the genes. Some model organisms are compiled in table 1.10.

We conclude this section with a small disclaimer: our intention here is to provide pointers to the largest and most general mainstream databases of bioinformatics. However, there exist a large number of additional databases on various (often rather specialized) molecular biological problems. Once again, we refer the interested reader to the annual database issue of *Nucleic Acids Research*.

1.3 Bioinformatics Challenges

As described in section 1.2, modern molecular biologists measure huge amounts of data of various types. The intention is to use these data to (1) reconstruct the

Table 1.10 A selection of model organisms. We were unable to obtain information on the number of cells in the last three organisms; however, it is estimated that an adult human body consists of about $6 \cdot 10^{10}$ cells. The numbers of genes are estimates (except for HIV). The genome size is given as the number of nucleotides in a single strand of the haploid genome (Mb, for 10^6 basepairs).

Organism (Common Name)	**Level**	**Cells**	**Genes**	**Genome**
Human immunodeficiency virus (HIV)	virus	0	9	0.01 Mb
Methanococcus jannaschii	archaea	1	1750	1.66 Mb
Escherichia coli (human gut bacteria)	eubacteria	1	4300	4.6 Mb
Saccharomyces cerevisiae (brewer's yeast)	eukaryote	1	6000	12 Mb
Caenorhabditis elegans (nematode worm)	animal	959	19,500	100 Mb
Drosophila melanogaster (fruit fly)	animal	?	13,700	165 Mb
Arabidopsis thaliana (thale cress)	plant	?	25,498	125 Mb
Mus musculus (mouse)	mammal	?	35,000	3000 Mb

past (e.g., infer the evolution of species); (2) predict the future (e.g., predict how someone will respond to a certain drug); (3) guide biological engineering (such as improving the efficiency of brewer's yeast). Some of the concrete tasks are so complex that intermediate steps are already regarded as problems in their own right. For example, while the sequence of a protein in principle determines its function (in the particular environment provided by the cell that produces the protein), one of the grand challenges in bioinformatics is to predict its structure (which can then serve as a basis for investigating functional aspects like interactions). This can also be seen as an auxiliary goal complementing the three listed above: to replace difficult, laborious, time-consuming, and expensive measurements (x-ray crystallography) with more affordable ones (sequencing).

The rather vague goals described above manifest in a jungle of concrete computational problems. Some of them are very specific, for instance, related to a certain species or a particular protein; their solution can aid in a particular application. Many other problems are more fundamental, but often can be seen as building blocks for the solution of larger tasks. In the following subsections, we try to organize some of the more fundamental challenges into a small number of general schemes. To some of the problems described below, existing approaches are reviewed in chapter 3.

The following sections are organized along the drug development process of pharmaceutical companies, which is one of the main applications:

1. Understanding the biological system, especially the mechanism of the disease
2. Identifying target proteins which are pivotal for the disease
3. Characterizing those proteins; most importantly, their tertiary structure
4. Finding small molecules which bind to those proteins and which qualify as drugs

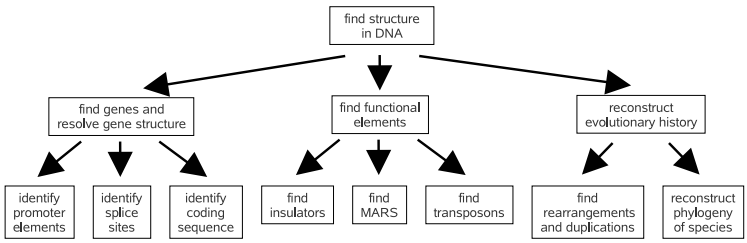


Figure 1.7 Finding structure in genomic DNA sequences: a small hierarchy of problems.

1.3.1 Genome Structure Analysis

The analysis of DNA sequences (partial or complete genomes) can be organized into a small tree, as depicted in figure 1.7. It contains at least three "grand challenge" problems:

- *Genome comparison.* The goal of this discipline is to reconstruct the evolutionary history, that is, the series of genome rearrangements, that led to different species. A difficulty is that the phylogeny and the common ancestors must be inferred on the basis of genomes of present-day species. While pairwise whole genome comparisons are already a challenge due to the sheer size of a genome, only comparison of multiple genomes will unleash the full power of this approach. Here, most efficient algorithms are asked for.
- *Gene finding.* This includes the identification of the gene structure, that is, the arrangement of the gene's elements (introns, exons, promoter, etc.). In computer science terms, the problem is to label substrings of the DNA. In large genomes with low gene content, like the human genome, especially the false positives can be a problem. An accurate solution to a large subproblem of gene structure identification, the prediction of splice sites, is described in chapter 13.
- *Understanding transcriptional regulation.* Here the goal is to quantitatively predict the expression levels of genes from the details of their promoters and the present quantities of TFs. In its broadest sense this problem would also include *modeling the 3D structure of DNA*. The packing of DNA is believed to have a big impact on gene expression, since genes must be unpacked before they can be transcribed. However, the available data may not yet be sufficient for such modeling.

All tasks are complicated by the fact that (presumably) by far not all functional DNA motifs are known by now. In fact, the understanding of the huge part of DNA which cannot yet be assigned a function can also be seen as a grand challenge, albeit possibly in molecular biology rather than in bioinformatics.

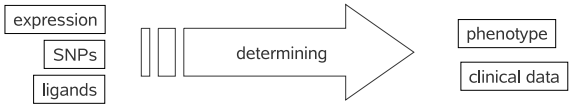


Figure 1.8 Macroscopic states (*right*) are caused by molecules (*left*); thus, molecular measurement data contain information predictive of the macroscopic states.

1.3.2 Relation of Molecular to Macroscopic Data

It is most interesting to identify the molecular causes of macroscopic events or states, because this understanding allows for a directed search for ways to cause or prevent such events and to maintain or change such states. Figure 1.8 provides examples of the molecular and macroscopic data types to be causally related. Three classes of tasks emerge from this general problem statement:

- *Population genetics.* The strategy in population genetics is to find chromosomal regions that are inherited along with (completely or partially) heritable traits; such regions can be supposed to contain genes responsible for those traits. The basic data for this are pedigrees of families which are annotated with both phenotypic and genotypic information on the individuals. The genotypic part consists of so-called *genetic markers* (such as SNPs) that relate genetic content to chromosomal location.
- *Diagnosis.* As an example, it is desirable to be able to base the diagnosis of certain diseases on gene expression patterns. For diseases that are hard to recognize or distinguish by classic means (e.g., histology), this can potentially be less subjective and ambiguous. Genetic diseases may also be diagnosed based on SNPs, or both SNPs and expression data. Sometimes unsupervised analysis of molecular data can even lead to refined definitions of diseases (Golub et al., 1999).
- *Therapy optimization.* Here, the idea is that every individual is different and that optimal treatment may be derived from molecular data: the efficacy of drugs may be predicted on the basis of the genotype of a pathogen (Beerenwinkel et al., 2003). Optimally, the interplay of the genotype (SNPs) of the patient with that of the pathogen should be taken into account.
- *Target finding.* This essentially amounts to applying feature selection to a successful prediction of a disease (diagnosis); see chapter 15. Ideally, the relevant features are related to the cause of the disease, which can then be selected as the target of drug development.
- *Systems biology.* This is the most ambitious challenge under this rubric, and is likely to keep bioinformaticians busy in coming years: the goal is nothing less than to quantitatively simulate entire cells (or large subsystems). This would (among many other things) allow replacement of animal experiments by computational

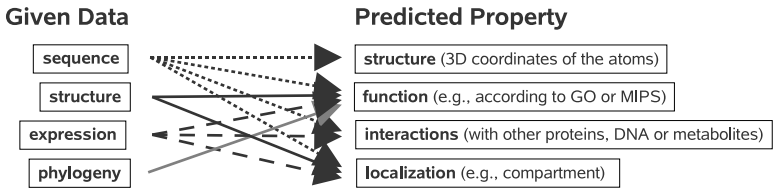


Figure 1.9 The prediction of different properties of proteins can be based on different (combinations of) data types.

simulations. A step in that direction has already been taken by E-CELL (Tomita, 2001).

1.3.3 Protein Property Prediction

Trying to predict properties of proteins is alluring, because proteins are so important for the cell (and for the biologist), and difficult, because proteins are so complex and versatile. There is a whole family of problems which are distinguished by the predicted property and by the data on which the prediction is based; this is sketched in figure 1.9.

There are also a number of prediction problems that are not explicitly shown, because they can be seen as intermediate steps. They include the prediction of structural motifs (most important, secondary structure) and of solvent accessibility of amino acids, which can provide valuable hints for a structure prediction. Another analytical task is the prediction of modifications (such as phosphorylation) from sequence. Modifications can affect all four types of properties shown on the right side of figure 1.9.

At least four grand challenge problems are instances of the family illustrated in the figure:

- *Structure prediction.* Here the amino acid sequence is given, and the 3D structure of the folded protein (in a cell) is to be computed. The fact that the cellular environment is so important in reaching the correct structure (cf. section 1.1.3) renders the idea of simulating its molecular motion in watery solution unappealing (although this is tried with vigor). Indeed, the most successful structure prediction methods are knowledge-based (i.e., at least in a way, machine learning) methods.[5] Chapters 4 and 12 describe applications of kernel methods to structure prediction.

5. This is demonstrated in the evaluation of the CASP competition available from `http://predictioncenter.llnl.gov/casp5/`.

- *Function prediction.* Here the problem starts with finding an appropriate definition of function. While hierarchical classifications of functions now exist, it is not clear whether they are well suited to serve as classes to be predicted. (At least, the idea of cutting the hierarchy at a fixed depth to define the classes seems to be questionable.) It has been shown that the best performance can be achieved by making use of multiple data types, although the proper combination is not trivial (Pavlidis et al., 2002, see also chapters 10 and 11).
- *Genetic network reconstruction.* The term *genetic network* refers to a graph specifying interactions between molecules, especially of regulatory type. Although a few experimental methods exist to find such interactions, there is also great interest in predicting them to obtain a more complete picture at less cost. A genetic network can allow deduction of hypotheses about, say, the effects of inhibiting certain proteins, which may suggest drug target candidates. Several models for computational treatment have been suggested: probably the most prominent classes are Boolean, linear, and Bayesian networks; a recent trend is the use of graph kernels (cf. chapter 8).
- *Docking.* This is the computational prediction of molecular binding events involving proteins. There are two flavors of protein docking: protein-protein docking and protein-ligand docking. The goal of protein-protein docking is basically to predict whether two proteins can bind at all; this can contribute edges for biological networks. While the backbones are usually taken to be fixed, it is important to model the flexibility of the involved side chains. In protein-ligand docking, the flexibility of the *ligand* (the small molecule) is essential; often, it is also crucial on the side of the protein (*induced fit*). Here, the goal includes predicting the strength of the binding (*affinity*), which must be high for inhibitors.

1.3.4 Small Molecule Problems

Chemoinformatics, cheminformatics

Computational work with chemical compounds is sometimes regarded as a subdiscipline of bioinformatics and sometimes viewed as a separate field termed *chemoinformatics* or *cheminformatics*. For completeness, we briefly introduce the main problems in this area that bear biological relevance. Usually they are closely related to the task of drug development.

- *Virtual HTS (high-throughput screening).*
This is essentially protein-ligand docking (see above) viewed from a different perspective. Simulating the experimental HTS, large databases of compounds are tested against a receptor protein to identify potential ligands.
- *Lead identification.* This means proposing a novel skeleton of a suitable drug molecule, called a *lead structure*, based on a collection of data related to a disease. The data may be the product of virtual or real HTS. While virtual HTS relies on the atomic structure of the binding pocket of the protein of interest, in practice it often is not known. Then techniques like *QSAR* (quantitative structure-activity relationships) are used to infer important properties from molecules with known

activity. Such properties can be summarized in a *pharmacophore*, an abstract characterization of the set of molecules of interest. In *lead hopping*, the task is to find new lead structures for a given pharmacophore (with known leads), for example, to evade patent problems.

■ *Predictive toxicology.* Not being (too) toxic is an important prerequisite for a drug. Depending on the time scale of the effect, toxicity may be acute or chronic. While acute cytostatic (inhibiting cell growth) toxicity is quite amenable to experimental investigations, lab screenings for chronic effects like carcinogenicity (causing cancer) are very time-consuming and hard to standardize. Thus, reliable in silico predictions would be of high value. Of course, there are further properties of small molecules that are relevant to drug candidates, often subsumed by the term *ADME* (*a*bsorption, *d*istribution, *m*etabolism, and *e*xcretion). Much effort has been spent on developing good representations of molecules for solving such prediction tasks; modern methods allow working directly with the natural representation by a labeled graph (cf. chapter 7).

1.4 Summary

It is hard to summarize the first two sections of this chapter. The section on the basic biology of the cell (section 1.1) is already a summary in itself. Please be warned that there is much, much more to cellular biology and that it really matters for successful bioinformatics. With respect to the molecular biology measurement data (section 1.2), we would like to add that biotechnology is a very active field and new technology is constantly being developed. Thus, new exciting types of data can be expected to become available and popular in the near future.

What we do want to explicitly point out here are a couple of things that may have already become apparent from section 1.3. In several examples it could be seen that many bioinformatics problems

1. can be posed as machine learning problems;
2. concern a structured data type (as opposed to "simple" real vectors);
3. concern a combination of different data types.

Corresponding to that, the subsequent chapters of this book are concerned with the following questions:

1. How to properly model bioinformatics problems in a machine learning framework.
2. How to operate on structured data types with the use of kernel functions.
3. How to combine different data types with clever kernels or algorithms.

While there already is some tradition of machine learning approaches to the analysis of molecular biology data, it has so far been mostly concerned with relatively

simple data structures (usually, fixed-length vectors of real values and categorical attributes). By demonstrating new ways to deal with complex data this book may contribute to accelerating the progress and success of machine learning in bioinformatics, and possibly also in other application areas.

2 A Primer on Kernel Methods

Jean-Philippe Vert
Koji Tsuda
Bernhard Schölkopf

Kernel methods in general, and support vector machines (SVMs) in particular, are increasingly used to solve various problems in computational biology. They offer versatile tools to process, analyze, and compare many types of data, and offer state-of-the-art performance in many cases. This self-contained introduction to positive definite kernels and kernel methods aims at providing the very basic knowledge and intuition that the reader might find useful in order to fully grasp the technical content of this book.

2.1 Introduction

Kernel methods in general and SVMs in particular have been successfully applied to a number of real-world problems and are now considered state-of-the-art in various domains, although it was only fairly recently that they became part of the mainstream in machine learning and empirical inference. The history of methods employing positive definite kernels, however, can be traced back at least a few decades. Aronszajn (1950) and Parzen (1962) were some of the first to employ these methods in statistics. Subsequently, Aizerman et al. (1964) used positive definite kernels in a way which was already closer to what people now call the *kernel trick*. They employed radial basis function kernels to reduce a convergence proof for the *potential function classifier* to the linear perceptron case. To do this, they had to argue that a positive definite kernel is identical to a dot product in another space (sometimes called the *feature space*), in which their algorithm reduced to the perceptron algorithm. They did not, however, use the feature space view to design new algorithms.

Kernel methods

The latter was done some thirty years later by Boser et al. (1992), to construct the SVMs, a generalization of the so-called *optimal hyperplane* algorithm. Initially, it

was thought that the main strength of SVMs compared to the optimal hyperplane algorithm was that they allowed the use of a larger class of similarity measures. Just as optimal hyperplanes, however, they were only used on vectorial data. But soon it was noted (Schölkopf, 1997) that kernels not only increase the flexibility by increasing the class of allowed similarity measures but also make it possible to work with nonvectorial data. This is due to the fact that kernels automatically provide a vectorial representation of the data in the feature space. The first examples of nontrivial kernels defined on nonvectorial data were those of Haussler (1999) and Watkins (2000) (see also Cristianini and Shawe-Taylor, 2000). Moreover, it was pointed out (Schölkopf et al., 1998) that kernels can be used to construct generalizations of any algorithm that can be carried out in terms of dot products, and the last 5 years have seen a large number of "kernelizations" of various algorithms (Graepel and Obermayer, 1998; Weston et al., 1999; Tsuda, 1999; Ruján and Marchand, 2000; Herbrich et al., 2000; Fyfe and Lai, 2000; Rosipal and Trejo, 2001; Akaho, 2001; Harmeling et al., 2001; Girolami, 2002; Suykens et al., 2002; Weston et al., 2003a; Vert and Kanehisa, 2003b; Kuss and Graepel, 2003).

Further threads of kernel work can be identified in approximation theory and statistics (Berg et al., 1984; Micchelli, 1986; Wahba, 2002; Poggio and Girosi, 1990), as well as in the area of Gaussian process prediction and related fields such as kriging, where kernels play the role of *covariance functions* (see, e.g., Weinert, 1982; Williams, 1998; MacKay, 1998).

This chapter is structured as follows. Section 2.2 is devoted to the presentation of kernels and some of their basic properties. Kernel methods are then introduced in section 2.3, SVMs being treated in more detail in section 2.4. We then discuss more advanced kernel topics relevant to this book, including the presentation of several families of kernels in section 2.6, and an introduction to the emergent field of kernel design in section 2.7.

2.2 Kernels

Kernels are the basic ingredient shared by all kernel methods. They provide a general framework to represent data, and must satisfy some mathematical conditions. These conditions give them a number of properties useful to bear in mind when it comes to understanding the intuition behind kernel methods and kernel design.

2.2.1 The Issue of Data Representation

Let us denote by $\mathcal{S} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)$ a set of n objects to be analyzed. We suppose that each object $\mathbf{x}_i$ is an element of a set $\mathcal{X}$, which may, for example, be the set of all possible images if one wants to analyze a set of images, or the set of all possible molecules in a biological context. In order to design data analysis methods, the first question to be addressed is how to represent the data set $\mathcal{S}$ for further processing.

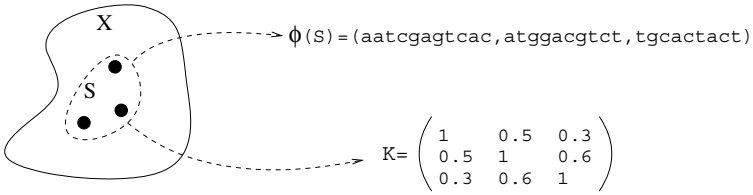


Figure 2.1 Two different representations of the same dataset. $\mathcal{X}$ is supposed to be the set of all oligonucleotides, and $\mathcal{S}$ is a data set of three particular oligonucleotides. The classic way to represent $\mathcal{S}$ is first to define a representation $\phi(\mathbf{x})$ for each element of $\mathbf{x} \in \mathcal{X}$, for example, as a sequence of letters to represent the succession of nucleotides, and then to represent $\mathcal{S}$ as the set $\phi(\mathcal{S})$ of representations of its elements (*upper part*). Kernel methods are based on a different representation of $\mathcal{S}$, as a matrix of pairwise similarity between its elements (*lower part*).

The vast majority of data analysis methods, outside kernel methods, have a natural answer to this question: first define a representation for each object, and then represent the set of objects by the set of their representations. Formally, this means that a representation $\phi(\mathbf{x}) \in \mathcal{F}$ is defined for each possible object $\mathbf{x} \in \mathcal{X}$, where the representation can, for example, be a real-valued vector ($\mathcal{F} = \mathbb{R}^p$), a finite-length string ($\mathcal{F}$ is then the set of all finite-length strings), or a more complex representation that can be processed by an algorithm. The data set $\mathcal{S}$ is then represented as the set of individual object representations, $\phi(\mathcal{S}) = (\phi(\mathbf{x}_1), \cdots, \phi(\mathbf{x}_n))$, and the algorithm is designed to process such data. As an example, if a protein is represented by a sequence of letters that corresponds to its primary structure, then a set of proteins can be represented by a set of sequences.

Kernel representation

Kernel methods are based on a radically different answer to the question of data representation. Data are not represented individually anymore, but only through a set of pairwise comparisons. In other words, instead of using a mapping $\phi : \mathcal{X} \to \mathcal{F}$ to represent each object $\mathbf{x} \in \mathcal{X}$ by $\phi(\mathbf{x}) \in \mathcal{F}$, a real-valued "comparison function" $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is used, and the data set $\mathcal{S}$ is represented by the $n \times n$ matrix of pairwise comparisons $k_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. All kernel methods are designed to process such square matrices. The difference between both approaches is represented in figure 2.1.

Several comments can already be made at this point. First, the representation as a square matrix does not depend on the nature of the objects to be analyzed. They can be images, molecules, or sequences, and the representation of a data set is always a real-valued square matrix. This suggests that an algorithm developed to process such a matrix can analyze images as well as molecules or sequences, as long as valid functions k can be defined. This also suggests that a complete modularity exists between the design of a function k to represent data on the one hand, and the design of an algorithm to process the data representations on the other hand. These properties turn out to be of utmost importance in fields like computational

biology, where data of different nature need to be integrated and analyzed in a unified framework.

Second, the size of the matrix used to represent a dataset of n objects is always $n \times n$, whatever the nature or the complexity of the objects. For example, a set of ten tissues, each characterized by thousands of gene expression levels, is represented by a 10×10 matrix, whatever the number of genes. Computationally, this is very attractive in the case when a small number of complex objects are to be processed.

Third, there are many cases where comparing objects is an easier task than finding an explicit representation for each object that a given algorithm can process. As an example, many data analysis algorithms, such as least squares regression or neural networks, require an explicit representation of each object $\mathbf{x}$ as a vector $\phi(\mathbf{x}) \in \mathbb{R}^p$. There is no obvious way to represent protein sequences as vectors in a biologically relevant way, however, while meaningful pairwise sequence comparison methods exist.

2.2.2 General Definition

As the reader might guess, the comparison function k is a critical component of any kernel method, because it defines how the algorithm "sees" the data. Most kernel methods described below can only process square matrices, which are symmetric *positive definite*. This means that if k is an $n \times n$ matrix of pairwise comparisons, it should satisfy $k_{i,j} = k_{j,i}$ for any $1 \leq i, j \leq n$, and $\boldsymbol{c}^\top k \boldsymbol{c} \geq 0$ for any $\boldsymbol{c} \in \mathbb{R}^n$.[1] This motivates the following definition:

Positive definite kernel

Definition 2.1 *A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a* positive definite kernel *iff it is symmetric, that is, $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$ for any two objects $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, and positive definite, that is,*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

for any $n > 0$, any choice of n objects $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathcal{X}$, and any choice of real numbers $c_1, \cdots, c_n \in \mathbb{R}$.

From now on, we only focus on positive definite kernels, and simply call them *kernels*. Definition 2.1 ensures that if k is a kernel, then any pairwise similarity matrix built from k is symmetric positive definite.

Imposing the condition that a comparison function be a kernel clearly restricts the class of functions one can use. For example, the local alignment scores widely used in computational biology to assess the similarity between sequences are not in general kernels (see chapter 6). However, this restriction is often worth the cost

1. In mathematics, such a matrix is usually called positive semidefinite, because $\mathbf{c}^\top k \mathbf{c}$ can be zero and not strictly positive. However, in this chapter, we follow the notation in approximation theory (Wahba, 2002), which omits "semi" for simplicity.

because it opens the door to the use of kernel methods (see section 2.3). Moreover, an increasing number of "tricks" are being developed to derive kernels from virtually any comparison function (see section 2.7).

Before we describe kernel methods in more detail, however, let us try to get a better intuition about kernels themselves. Kernels have several properties which one should bear in mind in order to fully understand kernel methods, and to understand the motivations behind the kernels developed in this book.

2.2.3 Kernels as Inner Product

Let us start with a simple example that leads to a fundamental property of kernels. Suppose the data to be analyzed are real vectors, that is, $\mathcal{X} = \mathbb{R}^p$ and any object is written as $\mathbf{x} = (x_1, \cdots, x_p)^\top$. One is tempted to compare such vectors using their inner product: for any $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$,

Linear kernel

$$k_L(\mathbf{x}, \mathbf{x}') := \mathbf{x}^\top \mathbf{x}' = \sum_{i=1}^{p} x_i x'_i. \tag{2.1}$$

This function is a kernel. Indeed, it is symmetric ($\mathbf{x}^\top \mathbf{x}' = \mathbf{x}'^\top \mathbf{x}$), and the positive definiteness results from the following simple calculation, valid for any $n > 0$, $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^p$, and $c_1, \cdots, c_n \in \mathbb{R}$:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k_L(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \mathbf{x}_i^\top \mathbf{x}_j = ||\sum_{i=1}^{n} c_i \mathbf{x}_i||^2 \geq 0. \tag{2.2}$$

The inner product between vectors is the first kernel we encounter. It is usually called the *linear kernel*. An obvious limitation of this kernel is that it is only defined when the data to be analyzed are vectors. For more general objects $\mathbf{x} \in \mathcal{X}$, however, this suggests a way to define kernels in a very systematic manner, by first representing each object $\mathbf{x} \in \mathcal{X}$ as a vector $\phi(\mathbf{x}) \in \mathbb{R}^p$, and then defining a kernel for any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ by

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}'). \tag{2.3}$$

Following the same line of computation as in (2.2), the reader can easily check that the function k defined in (2.3) is a valid kernel on the space $\mathcal{X}$, which does not need to be a vector space.

Any mapping $\phi : \mathcal{X} \to \mathbb{R}^p$ for some $p \geq 0$ results in a valid kernel through (2.3). Conversely, one might wonder whether there exist more general kernels than these. As the following classic result of Aronszajn (1950) shows, the answer is negative, at

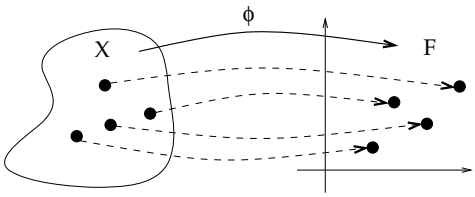


Figure 2.2 Any kernel on a space $\mathcal{X}$ can be represented as an inner product after the space $\mathcal{X}$ is mapped to a Hilbert space $\mathcal{F}$, called the feature space.

least if one allows $\mathbb{R}^p$ to be replaced by an eventually infinite-dimensional Hilbert space[2]:

Theorem 2.2 *For any kernel k on a space $\mathcal{X}$, there exists a Hilbert space $\mathcal{F}$ and a mapping $\phi : \mathcal{X} \to \mathcal{F}$ such that*

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle, \quad \textit{for any } \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \tag{2.4}$$

where $\langle u, v \rangle$ represents the dot product in the Hilbert space between any two points $u, v \in \mathcal{F}$.

Feature space

This result, illustrated in figure 2.2, provides a first useful intuition about kernels: they can all be thought of as dot products in some space $\mathcal{F}$, usually called the *feature space*. Hence, using a kernel boils down to representing each object $\mathbf{x} \in \mathcal{X}$ as a vector $\phi(\mathbf{x}) \in \mathcal{F}$, and computing dot products. There is, however, an important difference with respect to the explicit representation of objects as vectors, discussed in subsection 2.2.1: here the representation $\phi(\mathbf{x})$ does not need to be computed explicitly for each point in the data set $\mathcal{S}$, since only the pairwise dot products are necessary. In fact, there are many cases where the feature space associated with a simple kernel is infinite-dimensional, and the image $\phi(\mathbf{x})$ of a point $\mathbf{x}$ is tricky to represent even though the kernel is simple to compute.

This intuition of kernels as dot products is useful in order to provide a geometric interpretation of kernel methods. Indeed, most kernel methods possess such an interpretation when the points $\mathbf{x} \in \mathcal{X}$ are viewed as points $\phi(\mathbf{x})$ in the feature space.

2.2.4 Kernels as Measures of Similarity

In this book, as well as in the kernel methods community, kernels are often presented as measures of similarity, in the sense that $k(\mathbf{x}, \mathbf{x}')$ is "large" when $\mathbf{x}$ and $\mathbf{x}'$

2. A Hilbert space is a vector space endowed with a dot product (a strictly positive and symmetric bilinear form), that is complete for the norm induced. $\mathbb{R}^p$ with the classic inner product is an example of a finite-dimensional Hilbert space.

are "similar." This motivates the design of kernels for particular types of data or applications, because particular prior knowledge might suggest a relevant measure of similarity in a given context. As an example, the string and graph kernels presented in chapters 6 and 7 are motivated by a prior intuition of relevant notions of similarity: the fact that two biological sequences are similar when there exist good alignments between them, on the one hand, and the fact that two graphs are similar when they share many common paths, on the other.

The justification for this intuition of kernels as measures of similarity is not always obvious, however. From subsection 2.2.3 we know that kernels are dot products in a feature space. Yet the notion of dot product does not always fit one's intuition of similarity, which is more related to a notion of distance. There are cases where these notions coincide. Consider, for example, the following kernel on $\mathcal{X} = \mathbb{R}^p$, called the Gaussian radial basis function (RBF) kernel:

RBF kernel

$$k_G(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{d\left(\mathbf{x}, \mathbf{x}'\right)^2}{2\sigma^2}\right), \tag{2.5}$$

where σ is a parameter and d is the Euclidean distance. This is a valid kernel (see subsection 2.7.2), which can be written as a dot product $k_G\left(\mathbf{x}, \mathbf{x}'\right) = \left\langle \phi\left(\mathbf{x}\right), \phi\left(\mathbf{x}'\right)\right\rangle$ by theorem 2.2. The feature space is a functional space, and an explicit form of the map ϕ is not obvious. By (2.5), we see that this kernel is a decreasing function of the Euclidean distance between points, and therefore has a relevant interpretation as a measure of similarity: the larger the kernel $k_G(\mathbf{x}, \mathbf{x}')$, the closer the points $\mathbf{x}$ and $\mathbf{x}'$ in $\mathcal{X}$.

For more general kernels k on a space $\mathcal{X}$, basic linear algebra in the feature space associated with k by theorem 2.2 shows that the following holds for any two objects $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$:

$$k(\mathbf{x}, \mathbf{x}') = \frac{||\phi\left(\mathbf{x}\right)||^2 + ||\phi\left(\mathbf{x}'\right)||^2 - d\left(\phi\left(\mathbf{x}'\right), \phi\left(\mathbf{x}'\right)\right)^2}{2}, \tag{2.6}$$

where d is the Hilbert distance defined by $d(u, v)^2 = \left\langle (u - v), (u - v)\right\rangle$ and $||.||$ is the Hilbert norm ($||u||^2 = \langle u, u\rangle$). Equation (2.6) shows that the kernel $k\left(\mathbf{x}, \mathbf{x}'\right)$ measures the similarity between $\mathbf{x}$ and $\mathbf{x}'$ as the opposite of the square distance $d\left(\phi\left(\mathbf{x}\right), \phi\left(\mathbf{x}'\right)\right)^2$ between their images in the feature space, up to the terms $||\phi\left(\mathbf{x}\right)||^2$ and $||\phi\left(\mathbf{x}'\right)||^2$. If all points have the same length in the feature space, meaning $||\phi\left(\mathbf{x}\right)||^2 = k\left(\mathbf{x}, \mathbf{x}\right) = \textit{constant}$ for all $\mathbf{x} \in \mathcal{X}$, then the kernel is simply a decreasing measure of the distance in the feature space. This is, for example, the case for all translation-invariant kernels of the form $k\left(\mathbf{x}, \mathbf{x}'\right) = \psi\left(\phi\left(\mathbf{x}\right) - \phi\left(\mathbf{x}'\right)\right)$ such as the Gaussian RBF kernel (2.5), because in this case $k\left(\mathbf{x}, \mathbf{x}\right) = \psi(0)$ for any $\mathbf{x} \in \mathcal{X}$. For more general kernels, one should keep in mind the slight gap between the notion of dot product and similarity.

The conclusion of this section is that it is generally relevant to think of a kernel as a measure of similarity, in particular when it is constant on the diagonal. This intuition is useful in designing kernels and in understanding kernel methods.

For example, methods like SVMs (see section 2.4), which predict the values of a function at a given point from the observation of its values at different points, base their prediction on the hypothesis that "similar" points are likely to have similar values. The "similarity" between points mentioned here is precisely the similarity determined by the kernel.

2.2.5 Kernels as Measures of Function Regularity

Let k be a kernel on a space $\mathcal{X}$. In this section we show that k is associated with a set of real-valued functions on $\mathcal{X}$, $\mathcal{H}_k \subset \{f : \mathcal{X} \to \mathbb{R}\}$, endowed with a structure of Hilbert space (in particular, with a dot product and a norm). Understanding the functional space $\mathcal{H}_k$ and the norm associated with a kernel often helps in understanding kernel methods and in designing new kernels, as we illustrate in section 2.3.

Let us start with two examples of kernels and their associated functional spaces. Consider first the linear kernel (2.1) on a vector space $\mathcal{X} = \mathbb{R}^p$. The corresponding functional space is the space of linear function $f : \mathbb{R}^d \to \mathbb{R}$:

$$\mathcal{H}_k = \left\{ f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} \quad : \mathbf{w} \in \mathbb{R}^p \right\}, \tag{2.7}$$

and the associated norm is just the slope of the linear function,

$$||f||_{\mathcal{H}_k} = ||\mathbf{w}|| \quad \text{for } f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}. \tag{2.8}$$

As a second example, consider the Gaussian RBF kernel (2.5) on the same vector space $\mathcal{X} = \mathbb{R}^p$. The associated functional space is the set of functions $f : \mathbb{R}^d \to \mathbb{R}$ with Fourier transform $\hat{f}$ that satisfies

$$N(f) = \frac{1}{(2\pi\sigma^2)^{\frac{p}{2}}} \int_{\mathbb{R}^p} \left| \hat{f}(\omega) \right|^2 e^{\frac{\sigma^2}{2}||\omega||^2} d\omega < +\infty,$$

and the norm in $\mathcal{H}_k$ is precisely this functional: $||f||_{\mathcal{H}_k} = N(f)$. Hence $\mathcal{H}_k$ is a set of functions with Fourier transforms that decay rapidly, and the norm $||.||_{\mathcal{H}_k}$ quantifies how fast this decay is.

In both examples, the norm $||f||_{\mathcal{H}_k}$ decreases if the "smoothness" of f increases, where the definition of smoothness depends on the kernel. For the linear kernel, the smoothness is related to the slope of the function: a smooth function is a flat function. For the Gaussian RBF kernel, the smoothness of a function is measured by its Fourier spectrum: a smooth function has little energy at high frequencies. These examples of smoothness turn out to be very general, the precise definition of smoothness depending on the kernel considered. In fact, the notion of smoothness is dual to the notion of similarity discussed in subsection 2.2.4: a function is "smooth" when it varies slowly between "similar" points.

Let us now sketch the systematic construction of the functional space $\mathcal{H}_k$ from the kernel k. The set $\mathcal{H}_k$ is defined as the set of function $f : \mathcal{X} \to \mathbb{R}$ of the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}), \tag{2.9}$$

for $n > 0$, a finite number of points $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathcal{X}$, and a finite number of weights $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$, together with their limits under the norm:

$$||f||^2_{\mathcal{H}_k} := \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k\left(\mathbf{x}_i, \mathbf{x}_j\right). \tag{2.10}$$

It can be checked that this norm is independent of the representation of f in (2.9). $\mathcal{H}_k$ is in fact a Hilbert space, with a dot product defined for two elements $f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})$ and $g(\mathbf{x}) = \sum_{j=1}^{m} \alpha'_i k(\mathbf{x}'_i, \mathbf{x})$ by

Reproducing kernel Hilbert space

$$\langle f, g \rangle = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \alpha'_j k\left(\mathbf{x}_i, \mathbf{x}'_j\right).$$

An interesting property of this construction is that the value $f\left(\mathbf{x}\right)$ of a function $f \in \mathcal{H}_k$ at a point $\mathbf{x} \in \mathcal{X}$ can be expressed as a dot product in $\mathcal{H}_k$,

$$f\left(\mathbf{x}\right) = \langle f, k\left(\mathbf{x}, .\right) \rangle. \tag{2.11}$$

In particular, taking $f\left(.\right) = k\left(\mathbf{x}', .\right)$, we derive the following reproducing property valid for any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$:

$$k\left(\mathbf{x}, \mathbf{x}'\right) = \langle k\left(\mathbf{x}, .\right), k\left(\mathbf{x}', .\right) \rangle. \tag{2.12}$$

For this reason, the functional space $\mathcal{H}_k$ is usually called the reproducing kernel Hilbert space (RKHS) associated with k. The equality in (2.12) also shows that the Hilbert space $\mathcal{H}_k$ is one possible feature space associated with the kernel k, when we consider the mapping $\phi : \mathcal{X} \to \mathcal{H}_k$ defined by $\phi\left(\mathbf{x}\right) := k\left(\mathbf{x}, .\right)$. Indeed, (2.4) is exactly equivalent to (2.12) in this case. The construction of $\mathcal{H}_k$ therefore provides a proof of theorem 2.2.

Regularization

Aside from the technicalities of this section, the reader should keep in mind the connection between kernels and norms on functional spaces. Most kernel methods have an interpretation in terms of functional analysis. As an example, we show in the next sections that many kernel methods, including SVMs, can be defined as algorithms that, given a set of objects $\mathcal{S}$, return a function that solves the equation

$$\min_{f \in \mathcal{H}_k} R(f, \mathcal{S}) + c||f||_{\mathcal{H}_k}, \tag{2.13}$$

where $R(f, \mathcal{S})$ is small when f "fits" the data well, and the term $||f||_{\mathcal{H}_k}$ ensures that the solution of (2.13) is "smooth." In fact, besides fitting the data well and being smooth, the solution to (2.13) turns out to have special properties that are useful for computational reasons, which are discussed in theorem 2.3.3 below.

2.3 Some Kernel Methods

Having discussed the notions of data representation and kernels in section 2.2, let us now turn our attention to the algorithms that process the data to perform some particular tasks, such as clustering, computing various properties, inferring a regression or classification function from its observation on a finite set of points, and so on. We focus here on a class of algorithms called kernel methods, which can roughly be defined as those for which the data to be analyzed only enter the algorithm through the kernel function; in other words, algorithms that take as input the similarity matrix defined by a kernel.

Recent years have witnessed the development of a number of kernel methods, which we do not have the ambition to survey in full generality in this short introduction. Historically, the first kernel method recognized as such is the SVM (Boser et al., 1992), which has found many applications in computational biology (see survey in chapter 3), and which we describe in detail in section 2.4. Before this, let us try briefly to give a flavor of the two concepts that underlie most kernel methods: the *kernel trick* and the *representer theorem*.

2.3.1 The Kernel Trick

The kernel trick is a simple and general principle based on the property of kernels discussed in subsection 2.2.3, namely that they can be thought of as inner product. It can be stated as follows.

Kernel trick

Proposition 2.3 *Any algorithm for vectorial data that can be expressed only in terms of dot products between vectors can be performed implicitly in the feature space associated with any kernel, by replacing each dot product by a kernel evaluation.*

The kernel trick is obvious but has huge practical consequences that were only recently exploited. It is first a very convenient trick to transform linear methods, such as linear discriminant analysis (Hastie et al., 2001) or principal component analysis (PCA; Jolliffe, 1986), into nonlinear methods, by simply replacing the classic dot product by a more general kernel, such as the Gaussian RBF kernel (2.5). Nonlinearity is then obtained at no computational cost, as the algorithm remains exactly the same. The operation that transforms a linear algorithm into a more general kernel method is often called *kernelization*.

Second, the combination of the kernel trick with kernels defined on nonvectorial data permits the application of many classic algorithms on vectors to virtually any type of data, as long as a kernel can be defined. As an example, it becomes natural to perform PCA on a set of sequences, thanks to the availability of kernels for sequences such as those discussed in chapter 4 or chapter 6, or to search for structure-activity relationships on chemical compounds using least squares regression without computing an explicit representation of molecules as vectors, thanks to the graph kernel presented in chapter 7.

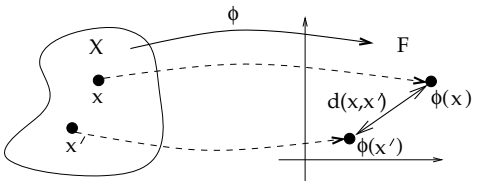


Figure 2.3 Given a space $\mathcal{X}$ endowed with a kernel, a distance can be defined between points of $\mathcal{X}$ mapped to the feature space $\mathcal{F}$ associated with the kernel. This distance can be computed without explicitly knowing the mapping ϕ thanks to the kernel trick.

2.3.2 Example: Computing Distances between Objects

Let us illustrate the kernel trick on the very simple problems of computing distances between points and clouds of points, which we attempt in general sets $\mathcal{X}$ endowed with a kernel k. Recall from theorem 2.2 that the kernel can be expressed as a dot product $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ in a dot product space $\mathcal{F}$ for some mapping $\phi : \mathcal{X} \to \mathcal{F}$.

As a starter consider two objects $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, such as two sequences or two molecules. These points are mapped to two vectors $\phi(\mathbf{x}_1)$ and $\phi(\mathbf{x}_2)$ in $\mathcal{F}$, so it is natural to define a distance $d(\mathbf{x}_1, \mathbf{x}_2)$ between the objects as the Hilbert distance between their images,

$$d(\mathbf{x}_1, \mathbf{x}_2) := ||\phi(\mathbf{x}_1) - \phi(\mathbf{x}_2)||. \tag{2.14}$$

This definition is illustrated in figure 2.3. At first sight, it seems necessary to explicitly compute the images $\phi(\mathbf{x}_1)$ and $\phi(\mathbf{x}_2)$ before computing this distance. However, the following simple equality shows that the distance (2.14) can be expressed in terms of dot products in $\mathcal{F}$:

$$||\phi(\mathbf{x}_1) - \phi(\mathbf{x}_2)||^2 = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_1) \rangle + \langle \phi(\mathbf{x}_2), \phi(\mathbf{x}_2) \rangle - 2\langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle. \tag{2.15}$$

Applying the kernel trick in (2.15) and plugging the result into (2.14) shows that the distance can be computed only in terms of the kernel,

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{k(\mathbf{x}_1, \mathbf{x}_1) + k(\mathbf{x}_2, \mathbf{x}_2) - 2k(\mathbf{x}_1, \mathbf{x}_2)}. \tag{2.16}$$

The effect of the kernel trick is easily understood in this example: it is possible to perform operations implicitly in the feature space. This is of utmost importance for kernels that are easy to calculate directly, but correspond to complex feature spaces, such as the Gaussian RBF kernel (2.5).

Let us now consider the following slightly more general problem. Let $\mathcal{S} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)$ be a fixed finite set of objects, and $\mathbf{x} \in \mathcal{X}$ a generic object. Is it possible to assess how "close" the object $\mathbf{x}$ is to the set of objects $\mathcal{S}$?

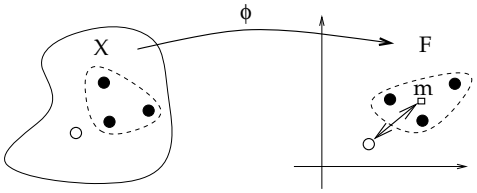


Figure 2.4 The distance between the white circle and the set of three black circles in the space $\mathcal{X}$ endowed with a kernel (*on the left*) is defined as the distance in the feature space between the image of the white circle and the centroid m of the images of the black circles. The centroid m might have no preimage in $\mathcal{X}$. This distance can nevertheless be computed implicitly with the kernel trick.

This question might be of interest in different contexts. For example, in binary classification, one observes two sets of objects $\mathcal{S}_1$ and $\mathcal{S}_2$ having two different properties, and one is asked to predict the property of a new object $\mathbf{x}$. A natural way to achieve this is to predict that $\mathbf{x}$ has the property of the objects in $\mathcal{S}_1$ if it is closer to $\mathcal{S}_1$ than $\mathcal{S}_2$, and the other property otherwise. A second example was recently proposed by Gorodkin et al. (2001) in the context of multiple sequence alignment. Given a set of biological sequences to align jointly, the authors proposed a pairwise alignment score as a kernel from which they derived a ranking of the sequences, from the most central to the most peripheral with respect to the whole set. This ranking can then be used to improve a greedy multiple alignment.

Having mapped the data set $\mathcal{S}$ and the object $\mathbf{x}$ in the feature space with the function ϕ, a natural way to measure the distance from $\mathbf{x}$ to $\mathcal{S}$ is to define it as the Euclidean distance between $\phi(\mathbf{x})$ and the centroid of $\mathcal{S}$ in the feature space, where the centroid is defined as

$$m = \frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i).$$

In general, there is no reason why m should be the image of an object $\mathbf{x} \in \mathcal{X}$ by ϕ, but still it is well defined as an element of $\mathcal{F}$. As illustrated in figure 2.4, we can now define the distance from $\mathbf{x}$ to $\mathcal{S}$ as follows:

$$\mathrm{dist}(\mathbf{x}, \mathcal{S}) = ||\phi(\mathbf{x}) - m|| = ||\phi(\mathbf{x}) - \frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)||. \tag{2.17}$$

Expanding the square distance (2.17) in terms of dot products in the feature space as we did in (2.15), and using the kernel trick, we obtain

$$\mathrm{dist}(\mathbf{x}, \mathcal{S}) = \sqrt{k(\mathbf{x},\mathbf{x}) - \frac{2}{n}\sum_{i=1}^{n}k(\mathbf{x},\mathbf{x}_i) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}k(\mathbf{x}_i,\mathbf{x}_j)}. \tag{2.18}$$

This shows that the distance from $\mathbf{x}$ to $\mathcal{S}$ can be computed entirely from the values of the kernels between pairs of points in $\{\mathbf{x}\} \cup \mathcal{S}$, even though it is defined as a distance in the feature space between $\phi(\mathbf{x})$ and a point m that does not necessarily even have a preimage $\phi^{-1}(m)$ in $\mathcal{X}$.

As a slight generalization to (2.18), interested readers can now easily verify that the kernel trick allows them to define the following functional as a distance between two sets of points $\mathcal{S}_1$ and $\mathcal{S}_2$:

$$\sqrt{\frac{1}{|\mathcal{S}_1|^2}\sum_{\mathbf{x},\mathbf{x}'\in\mathcal{S}_1} k(\mathbf{x},\mathbf{x}') + \frac{1}{|\mathcal{S}_2|^2}\sum_{\mathbf{x},\mathbf{x}'\in\mathcal{S}_2} k(\mathbf{x},\mathbf{x}') - \frac{2}{|\mathcal{S}_1||\mathcal{S}_2|}\sum_{\mathbf{x}\in\mathcal{S}_1,\mathbf{x}'\in\mathcal{S}_2} k(\mathbf{x},\mathbf{x}')}.$$

2.3.3 The Representer Theorem

The kernel trick is straightforward when one thinks of kernels as inner products (see subsection 2.2.3), and is a convenient guideline to deriving kernel methods from linear algorithms. When one thinks of kernels as regularization operators (see subsection 2.2.5), a simple but deep theorem can help understand many kernel methods in a different light. This theorem, called the representer theorem, was first stated less generally by Kimeldorf and Wahba (1971):

Representer theorem

Theorem 2.4 *Let $\mathcal{X}$ be a set endowed with a kernel k, and $\mathcal{S} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\} \subset \mathcal{X}$ a finite set of objects. Let $\Psi : \mathbb{R}^{n+1} \to \mathbb{R}$ be a function of $n+1$ arguments, strictly monotonic increasing in its last argument. Then any solution of the problem*

$$\min_{f\in\mathcal{H}_k} \Psi\left(f(\mathbf{x}_1), \cdots, f(\mathbf{x}_n), ||f||_{\mathcal{H}_k}\right), \tag{2.19}$$

where $(\mathcal{H}_k, ||.||_{\mathcal{H}_k})$ is the RKHS associated with k, admits a representation of the form

$$\forall \mathbf{x} \in \mathcal{X}, \quad f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}). \tag{2.20}$$

Proof With the notations of theorem 2.4, let us call $\xi(f, \mathcal{S})$ the function to be minimized in (2.19), and let

$$\mathcal{H}_k^{\mathcal{S}} = \left\{ f \in \mathcal{H}_k : f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}), (\alpha_1, \cdots, \alpha_n) \in \mathbb{R}^n \right\} \subset \mathcal{H}_k.$$

Any function $f \in \mathcal{H}_k$ can be decomposed as $f = f_{\mathcal{S}} + f_\perp$, where $f_{\mathcal{S}} \in \mathcal{H}_k^{\mathcal{S}}$ is the orthogonal projection of f onto the subspace $\mathcal{H}_k^{\mathcal{S}}$ and $f_\perp \perp \mathcal{H}_k^{\mathcal{S}}$. By (2.11) it follows that $f_\perp(\mathbf{x}_i) = \langle f_\perp, k(\mathbf{x}_i, .)\rangle = 0$ for $i = 1, \cdots, n$, because each function $k(\mathbf{x}_i, .)$ is an element of $\mathcal{H}_k^{\mathcal{S}}$ and $f_\perp$ is orthogonal to each element of $\mathcal{H}_k^{\mathcal{S}}$ by definition. Therefore $f(\mathbf{x}_i) = f_{\mathcal{S}}(\mathbf{x}_i)$ for each $i = 1, \cdots, n$. Moreover, Pythagoras theorem in $\mathcal{H}_k$ states that $||f||_{\mathcal{H}_k}^2 = ||f_{\mathcal{S}}||_{\mathcal{H}_k}^2 + ||f_\perp||_{\mathcal{H}_k}$. This shows that $\xi(f, \mathcal{S}) \geq \xi(f_{\mathcal{S}}, \mathcal{S})$, with equality iff $||f_\perp||_{\mathcal{H}_k} = 0$, or $f_\perp = 0$, because Ψ is strictly monotonic increasing

in its last argument. As a result, any minimum f of $\xi(f, \mathcal{S})$ must belong to $\mathcal{H}_k^{\mathcal{S}}$, which concludes the proof. ∎

Theorem 2.4 shows the dramatic effect of regularizing a problem by including a dependency in $||f||_{\mathcal{H}_k}$ in the function to optimize. As pointed out in subsection 2.2.5, this penalization makes sense because it forces the solution to be smooth, which is usually a powerful protection against overfitting of the data. The representer theorem shows that this penalization also has substantial computational advantages: any solution to (2.19) is known to belong to a subspace of $\mathcal{H}_k$ of dimension at most n, the number of points in $\mathcal{S}$, even though the optimization is carried out over a possibly infinite-dimensional space $\mathcal{H}_k$. A practical consequence is that (2.19) can be reformulated as an n-dimensional optimization problem, by plugging (2.20) into (2.19) and optimizing over $(\alpha_1, \cdots, \alpha_n) \in \mathbb{R}^n$.

Most kernel methods can be seen in light of the representer theorem. Indeed, as we show in the next examples, they often output a function of the subspace $\mathcal{H}_k^{\mathcal{S}}$; indeed one can often explicitly write the functional that is minimized, which involves a norm in $\mathcal{H}_k$. This observation can serve as a guide to choosing a kernel for practical applications, if one has some prior knowledge about the function the algorithm should output: it is in fact possible to design a kernel such that a priori desirable functions have a small norm.

2.3.4 Example: Kernel Principal Component Analysis

PCA

PCA (Jolliffe, 1986) is a powerful method to extract features from a set of vectors and to visualize them. Let us first suppose that $\mathcal{X} = \mathbb{R}^p$ and $\mathcal{S} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is a set of centered vectors,

$$\sum_{i=1}^{n} \mathbf{x}_i = 0.$$

The orthogonal projection onto a direction $\boldsymbol{w} \in \mathbb{R}^p$ is the function $h_{\boldsymbol{w}} : \mathcal{X} \to \mathbb{R}$ defined by

$$h_{\boldsymbol{w}}(\mathbf{x}) = \mathbf{x}^\top \frac{\boldsymbol{w}}{||\boldsymbol{w}||}. \tag{2.21}$$

As illustrated in figure 2.5, PCA finds successive directions $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_p$ for which the projections $h_{\boldsymbol{w}_i}$ have maximum empirical variance and $\boldsymbol{w}_i$ is orthogonal to $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{i-1}$, for $i = 1, \ldots, p$. Here the empirical variance of a projection $h_{\boldsymbol{w}}$ is defined by

$$\hat{var}(h_{\boldsymbol{w}}) := \frac{1}{n} \sum_{i=1}^{n} h_{\boldsymbol{w}}(\mathbf{x}_i)^2 = \frac{1}{n} \sum_{i=1}^{n} \frac{\left(\mathbf{x}_i^\top \boldsymbol{w}\right)^2}{||\boldsymbol{w}||^2}. \tag{2.22}$$

There may be ambiguity in this definition if two different directions have the same empirical variance, which we will not discuss further in the hope of keeping the attention of the reader on the kernelization of PCA.

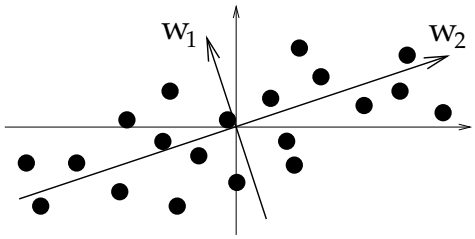


Figure 2.5 For centered vectorial data, principal component analysis (PCA) finds the orthogonal directions of largest variations.

Let us now rephrase PCA in terms of functional optimization (Schölkopf and Smola, 2002). Let k_L be the linear kernel (2.1), and $\mathcal{H}_k$ the RKHS (2.7) associated with k_L. Given any direction $\boldsymbol{w} \in \mathbb{R}^d$, we use (2.8) to associate the function $f_{\boldsymbol{w}} \in \mathcal{H}_k$ defined by $f_{\boldsymbol{w}}(\mathbf{x}) = \boldsymbol{w}^\top \mathbf{x}$, with norm $||f_{\boldsymbol{w}}|| = ||\boldsymbol{w}||$. Here, $||f_{\boldsymbol{w}}||$ is understood as the norm of f in $\mathcal{H}_k$, while $||\boldsymbol{w}||$ is the Euclidean norm of w in $\mathbb{R}^d$. The empirical variance (2.22) of the projection onto $\boldsymbol{w}$ can therefore be expressed in terms of $f_{\boldsymbol{w}}$;

$$\forall \boldsymbol{w} \in \mathbb{R}^p, \quad \hat{var}\left(h_{\boldsymbol{w}}\right) = \frac{1}{n||f_{\boldsymbol{w}}||^2} \sum_{i=1}^{n} f_{\boldsymbol{w}}(\mathbf{x}_i)^2.$$

Moreover, orthogonality of two directions $\boldsymbol{w}, \boldsymbol{w}' \in \mathbb{R}^p$ is equivalent to the orthogonality of the corresponding functions $f_w, f'_w \in \mathcal{H}_k$ with respect to the dot product of $\mathcal{H}_k$. Linear PCA can therefore be rephrased as finding successive functions $f_1, \ldots, f_p \in \mathcal{H}_k$ defined recursively as follows: for $i = 1, \ldots, p$, f_i maximizes the functional

$$\forall f \in \mathcal{H}_k, \quad \Psi(f) := \frac{1}{n||f||^2} \sum_{j=1}^{n} f(\mathbf{x}_j)^2, \tag{2.23}$$

under the constraints of orthogonality with respect to $f_1, \ldots, f_{i-1}$. Here again, the definition has some ambiguity if several functions have the same value.

The functional (2.23) satisfies the conditions of theorem 2.4: it is a strictly decreasing function of $||f||$, and only depends on the values that f takes on the points $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Theorem 2.4 shows that the successive directions f_i, for $i = 1, \ldots, p$, admit a representation

$$\forall \mathbf{x} \in \mathcal{X}, \quad f_i(\mathbf{x}) = \sum_{j=1}^{n} \alpha_{i,j} k(\mathbf{x}_j, \mathbf{x}), \tag{2.24}$$

for some $\boldsymbol{\alpha}_i = (\alpha_{i,1}, \ldots, \alpha_{i,n})^\top \in \mathbb{R}^n$ [indeed, the proof of theorem 2.4 remains valid when the optimization (2.19) is performed on a subspace of $\mathcal{H}_k$, such as a subspace

defined by orthogonality conditions]. Combining (2.24) and (2.10), we can express the norm $||f_i||$ in terms of $\boldsymbol{\alpha}_i$ using matrix notations,

$$||f_i||^2 = \boldsymbol{\alpha}_i^\top k \boldsymbol{\alpha}_i, \tag{2.25}$$

which with (2.24) yields

$$\sum_{i=1}^{n} f_i(\mathbf{x}_i)^2 = \boldsymbol{\alpha}_i^\top k^2 \boldsymbol{\alpha}_i. \tag{2.26}$$

Plugging (2.25) and (2.26) into (2.23), we obtain a dual formulation of PCA which consists in finding $\alpha_1, \ldots, \alpha_p \in \mathbb{R}^n$ defined recursively: for $i = 1, \ldots, p$, α_i maximizes the function

$$\frac{\boldsymbol{\alpha}^\top k^2 \boldsymbol{\alpha}}{n \boldsymbol{\alpha}^\top k \boldsymbol{\alpha}}, \tag{2.27}$$

under the constraints $\boldsymbol{\alpha}_i k \boldsymbol{\alpha}_j = 0$, for $j = 1, \ldots, i-1$. The principal components are then recovered by (2.24).

Classic linear algebra (Schölkopf et al., 1998) shows that the solutions $\boldsymbol{\alpha}_i$ of this problem are precisely the eigenvectors of k. In order to recover the projections (2.21) onto the principal directions, the eigenvector $\boldsymbol{\alpha}_i$ of k with eigenvalue λ_i must be scaled to ensure $1 = ||\boldsymbol{w}_i|| = \boldsymbol{\alpha}_i^\top k \boldsymbol{\alpha}_i = \lambda_i ||\boldsymbol{\alpha}_i||^2$, or $||\boldsymbol{\alpha}_i|| = 1/\sqrt{\lambda_i}$. Of course this conputation is only valid if the data are centered in the feature space. This is not a restriction, however, because any kernel matrix k can be transformed into a matrix $\tilde{k}$ corresponding to the inner products of the same points after centering in the feature space, using the kernel trick. The reader can check that $\tilde{k}$ is obtained by the formula $\tilde{k} = (I - e/n)k(I - e/n)$, where I is the identity matrix and e is the singular matrix with all entries equal to 1.

This representation of PCA only involves the diagonalization of the $n \times n$ matrix of pairwise comparisons with the linear kernel. It can therefore be kernelized by simply replacing this matrix with the same $n \times n$ matrix of pairwise comparisons obtained from a different kernel. In that case, linear PCA is performed implicitly in the corresponding feature space. The resulting method, called *kernel PCA*, is a useful tool to extract features from a set of objects in a space endowed with a kernel. For example, PCA involving a kernel defined on strings can be a useful visualization tool for sets of biological sequences. By projecting the sequences onto the first two or three principal components, one can observe the structure of the set of points, such as the presence of clusters or outliers, as shown in figure 2.6.

Kernel PCA

2.4 Support Vector Machines

Suppose that the data set $\mathcal{S}$ consists of a series of objects $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$, together with a series of labels $y_1, \ldots, y_n \in \mathcal{Y}$ associated with the objects. SVMs are kernel

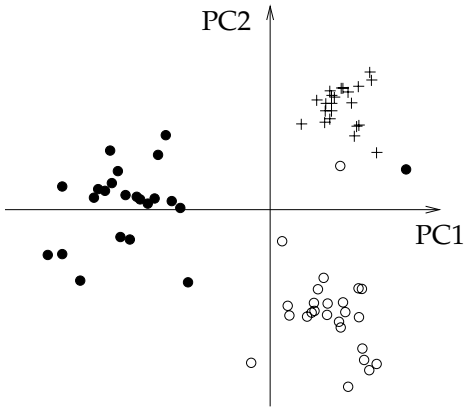


Figure 2.6 An example of kernel PCA. A set of 74 human tRNA sequences is analyzed using a kernel for sequences (the second-order marginalized kernel based on SCFG (Kin et al., 2002)). This set of tRNAs contains three classes, called Ala-AGC (*white circles*), Asn-GTT (*black circles*) and Cys-GCA (*plus symbols*). This plot shows the 74 sequences projected onto the first two principal components. By visual inspection, the three classes appear to be well separated in the feature space associated with the kernel. See also Vert (2002b) for another example of kernel PCA application to biological data.

methods to learn a function $f : \mathcal{X} \to \mathcal{Y}$ from $\mathcal{S}$, which can be used to predict the label of any new object $\mathbf{x} \in \mathcal{X}$ by $f(\mathbf{x})$.

Pattern recognition

In this tutorial we only consider the simple case where each object is classified into one of two classes, indicated by the label $y \in \{-1, +1\}$. This simple problem, called *binary classification* or *pattern recognition* in the machine learning community, turns out to be very useful in practice. Examples of pattern recognition problems in computational biology include predicting whether a protein is secreted or not from its amino acid sequence, predicting whether a tissue is healthy from a gene profiling experiment, or predicting whether a chemical compound can bind a given target or not from its structure. In each case, a positive prediction is associated with the label $+1$, and a negative prediction with the label -1. In order to perform pattern recognition, one needs a data set of objects with known tags, such as a database of proteins known to be secreted or not, in order to learn a prediction function that can then be applied to proteins without annotation.

As usual with kernel methods, we begin with a description of the algorithm when objects are vectors, $\mathcal{X} = \mathbb{R}^p$. In this case, the SVM tries to separate the two classes of points using a linear function of the form $f(\mathbf{x}) = \boldsymbol{w}^\top \mathbf{x} + b$, with $\boldsymbol{w} \in \mathbb{R}^p$ and $b \in \mathbb{R}$. Such a function assigns a label $+1$ to the points $\mathbf{x} \in \mathcal{X}$ with $f(\mathbf{x}) \geq 0$, and a label -1 to the points $\mathbf{x} \in \mathcal{X}$ with $f(\mathbf{x}) < 0$. The problem is therefore to learn such a function f from a data set of observations $\mathcal{S}$.

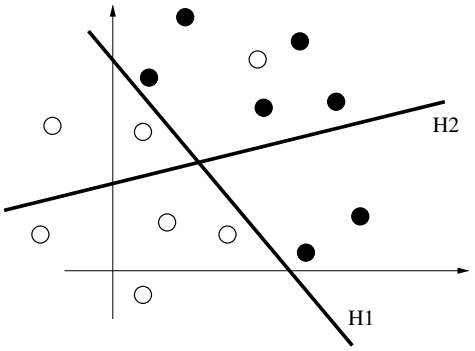


Figure 2.7 The hyperplane $H1$ discriminates the white circles from the black ones with 1 mistake. The hyperplane $H2$ separates these points with 5 mistakes. The empirical risk minimization principle states that one should choose a hyperplane with a minimal number of errors on the training set, which is $H1$ in this case

For a candidate function $f(\mathbf{x}) = \boldsymbol{w}^\top \mathbf{x} + b$, one can check for each observation $(\mathbf{x}_i, y_i)$ whether it is correctly classified by f, that is, whether $y_i f(\mathbf{x}_i) \geq 0$ or not. A natural criterion to choose f might be to minimize the number of classification errors on $\mathcal{S}$; the number of indices $i \in [1, n]$ such that $y_i f(\mathbf{x}_i) \geq 0$. This general principle, called *empirical risk minimization*, is illustrated in figure 2.7. This can, for example, be accomplished using the linear perceptron. As shown in figure 2.8, however, this usually does not define a unique solution, even when it is possible to perfectly separate the points.

Empirical risk minimization

SVMs are unique in that they focus more on the confidence of the classifications than on the number of misclassifications. This emphasis stems from general results on learning theory, developed in particular by Vapnik and Chervonenkis since the late 1960s (Vapnik and Chervonenkis, 1968, 1971, 1974). One way to formalize it with linear classifiers is shown in figure 2.9. The linear function $f(\mathbf{x}) = \boldsymbol{w}^\top \mathbf{x} + b$ defines two half-spaces of points classified positively and negatively with large confidence, namely the sets $h^+ = \{\mathbf{x} : f(\mathbf{x}) \geq 1\}$ and $h^- = \{\mathbf{x} : f(\mathbf{x}) \leq -1\}$. The distance between these two half-spaces, called the *margin*, is exactly equal to $2/||\boldsymbol{w}||$. If possible, one might require all points in the training set $\mathcal{S}$ to be correctly classified with strong confidence by a linear function f with largest possible margin. This would correspond to the problem of maximizing $2/||\boldsymbol{w}||$ under the constraints $y_i\left(\boldsymbol{w}^\top \mathbf{x} + b\right) \geq 1$ for $i = 1, \ldots, n$. In order to accommodate the cases when the training set cannot be correctly separated by a linear hyperplane, SVMs slightly modify this problem by softening the constraints using the continuous hinge loss function shown in figure 2.10,

Margin

Hinge loss

$$c(f, \mathbf{x}, y) = \max(0, 1 - y f(\mathbf{x})). \tag{2.28}$$

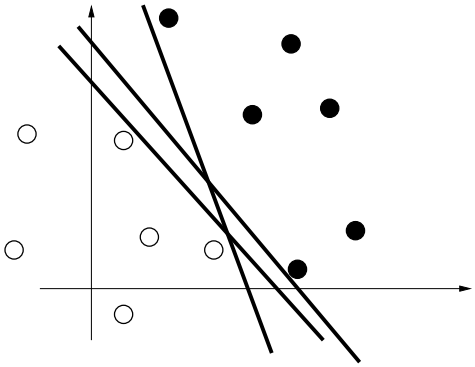


Figure 2.8 Even when the training data are linearly separable, the empirical risk minimization principle does not define a unique solution.

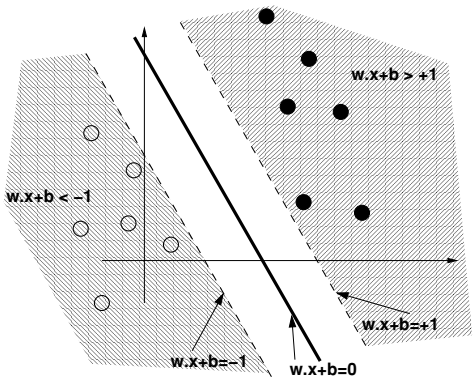


Figure 2.9 An affine function $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ defines two half-spaces where points are classified with large confidence: $h^+ = \{\mathbf{x} : f(\mathbf{x}) \geq 1\}$ for the positive points (*black circles*) and $h^- = \{\mathbf{x} : f(\mathbf{x}) \leq -1\}$ (*white circles*). The distance between the half-spaces is equal to $1/\|\mathbf{w}\|$.

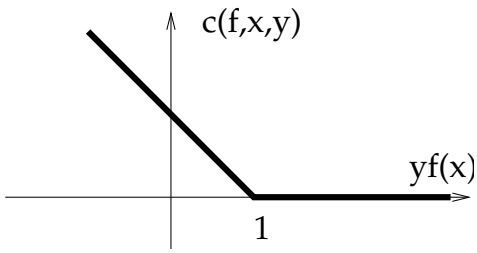


Figure 2.10 The hinge loss function. As long as $yf(\mathbf{x}) \geq 1$, the point $\mathbf{x}$ is correctly classified by the function f with large confidence and the hinge loss is null. When $yf(\mathbf{x}) < 1$, $\mathbf{x}$ is either correctly classified with small confidence ($0 \leq yf(\mathbf{x}) < 1$), or misclassified ($yf(\mathbf{x}) < 0$). In these cases the hinge loss is positive, and increases as $1 - yf(\mathbf{x})$. SVMs find a linear separating function with a large margin and small average hinge loss on the training set.

If a point $(\mathbf{x}, y)$ is correctly classified by f with large confidence, then $c(f, \mathbf{x}, y) = 0$. If this is not the case, then $c(f, \mathbf{x}, y)$ increases with the distance from $\mathbf{x}$ to the correct half-space of large confidence.

SVMs combine the requirements of large margin (i.e., small $||\boldsymbol{w}||$), and few misclassifications or classifications with little confidence on the training set, by solving the problem

$$\underset{f(\mathbf{x})=\boldsymbol{w}^\top \mathbf{x}+b}{\operatorname{argmin}} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n} c(f, \mathbf{x}_i, y_i), \tag{2.29}$$

where C is a parameter that controls the tradeoff between the two requirements. Larger values of C might lead to linear functions with smaller margin but more examples correctly classified with strong confidence (the choice of this parameter is discussed in subsection 2.5.3).

Stated as (2.29), the reader might observe that this problem is very close to the minimization of a functional on a RKHS satisfying the hypothesis of theorem 2.4, with the slight difference that we consider here affine functions f, and not only linear functions of the form (2.7). It turns out that the representer theorem can be adapted to this case (see, e.g., theorem 4.3 in Schölkopf and Smola, 2002), and any $\boldsymbol{w}$ solution of (2.29) has an expansion as a linear combination of $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Let us now directly demonstrate this property and highlight several interesting properties of the solution of (2.29).

2.4.1 Solving the Optimization Problem

The hinge loss function (2.28) is not differentiable, so direct minimization of (2.29) is not straightforward. To overcome this issue let us introduce n new variables $\xi_1, \ldots, \xi_n$, called slack variables, and rewrite (2.29) as the problem of minimizing:

$$\underset{\boldsymbol{w},b,\xi_1,\ldots,\xi_n}{\operatorname{argmin}} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\xi_i, \tag{2.30}$$

under the constraints $\xi_i \geq c(f, \mathbf{x}_i, , y_i)$ for $i = 1, \ldots, n$. These two problems are equivalent, because the minimization of (2.30) with respect to ξ_i is obtained when ξ_i takes its minimal value, namely $c(f, \mathbf{x}_i, y_i)$. By definition of the hinge loss (2.28), the constraint $\xi_i \geq c(f, \mathbf{x}_i, y_i)$ is equivalent to the two constraints $\xi_i \geq 0$ and $\xi_i \geq 1 - y_i(\boldsymbol{w}^\top \mathbf{x}_i + b)$. We have therefore shown that (2.29) is equivalent to the quadratic programming problem

$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\xi_i, \tag{2.31}$$

under the constraints

$$\text{for } i = 1, \ldots, n, \quad \begin{cases} \xi_i \geq 0, \\ \xi_i - 1 + y_i(\boldsymbol{w}^\top \mathbf{x}_i + b) \geq 0. \end{cases} \tag{2.32}$$

Lagrange multipliers

This constrained optimization problem can be processed using *Lagrange multipliers*, which are written as $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \geq 0$ for each of the constraints $y_i\,(\mathbf{w}.\mathbf{x}_i + b) \geq 1 - \xi_i$, and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n) \geq 0$ for each of the constraints $\xi_i \geq 0$. We can then introduce the Lagrangian,

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left[\xi_i - 1 + y_i\left(\boldsymbol{w}^\top \mathbf{x}_i + b\right)\right] - \sum_{i=1}^{n}\beta_i\xi_i. \tag{2.33}$$

In order to solve (2.31) we need to find the unique saddle point of L, which is a minimum with respect to $(\mathbf{w}, b, \boldsymbol{\xi})$ and a maximum with respect to $(\boldsymbol{\alpha}, \boldsymbol{\beta}) \geq 0$.

For fixed $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ let us first minimize the Lagrangian as a function of $(\mathbf{w}, b, \xi)$. This is done by setting the partial derivatives to 0;

$$\frac{\partial L}{\partial \mathbf{w}}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{w} - \sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i = 0, \tag{2.34}$$

$$\frac{\partial L}{\partial b}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{i=1}^{n} y_i\alpha_i = 0, \tag{2.35}$$

$$\frac{\partial L}{\partial \xi_i}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = C - \alpha_i - \beta_i = 0, \quad \text{for } i = 1, \ldots, n. \tag{2.36}$$

With (2.34) we recover the representer theorem that states $\boldsymbol{w}$ is a linear combination of the $\mathbf{x}_1, \ldots, \mathbf{x}_n$. More precisely, we get

$$\boldsymbol{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i. \tag{2.37}$$

Plugging (2.37) into (2.33) and using (2.35), we obtain the value of the Lagrangian when minimized with respect to $(\mathbf{w}, b, \mathbf{x})$,

$$\forall (\boldsymbol{\alpha}, \boldsymbol{\beta}) \geq 0, \quad \inf_{\mathbf{w}, b, \boldsymbol{\xi}} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i, \tag{2.38}$$

under the constraints (2.35) and (2.36) on $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ (if these constraints are not fulfilled, the infimum is equal to $-\infty$).

The function (2.38) has to be maximized with respect to $\boldsymbol{\alpha} \geq 0$ and $\boldsymbol{\beta} \geq 0$. But $\boldsymbol{\beta}$ does not appear in this function, so we just need to maximize (2.38) as a function of $\boldsymbol{\alpha}$ and to check that there exists some $\boldsymbol{\beta} \geq 0$ for which (2.36) holds. This is the case iff $\alpha_i \leq C$ for $i = 1, \ldots, N$, because only in this case can we find $\beta_i \geq 0$ such that $\beta_i + \alpha_i = C$.

As a result, the initial problem (2.31) is equivalent to the following *dual problem*:

Dual problem

find $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ which minimizes

$$W(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i, \tag{2.39}$$

under the constraints

$$\begin{cases} \sum_{i=0}^{n} y_i \alpha_i = 0, \\ 0 \leq \alpha_i \leq C \quad \text{for } i = 1, \ldots, n. \end{cases}$$

Once $\boldsymbol{\alpha}$ is found one recovers the other dual vector $\boldsymbol{\beta}$ with the constraint

$$\beta_i = C - \alpha_i, \quad \text{for } i = 1, \ldots, N.$$

The vector $\boldsymbol{w}$ is then obtained from (2.37). In order to recover b, we can use the Karush-Kuhn-Tucker conditions which state that the constraints corresponding to non-zero Lagrange multipliers are met at the saddle point of the Lagrangian. As a result, for any $0 \leq i \leq n$ with $0 < \alpha_i < C$ (which implies $\boldsymbol{\beta} > 0$), the constraints $\xi_i = 0$ and $\xi_i - 1 + y_i (\boldsymbol{w}.\mathbf{x}_i + b)$ hold. We thus obtain

$$b = y_i - \boldsymbol{w}^\top \boldsymbol{x}_i = y_i - \sum_{j=1}^{n} y_j \alpha_j \mathbf{x}_j^\top \mathbf{x}_i. \tag{2.40}$$

Figure 2.11 shows a typical linear function learned by an SVM, together with the Lagrange multipliers α_i and β_i associated to each point. The constraint $\alpha_i < C$ implies $\beta_i > 0$, and therefore $\xi_i = 0$. These points are are correctly classified with large confidence. The constraints $0 < \alpha_i < C$ imply $\beta_i > 0$, and therefore

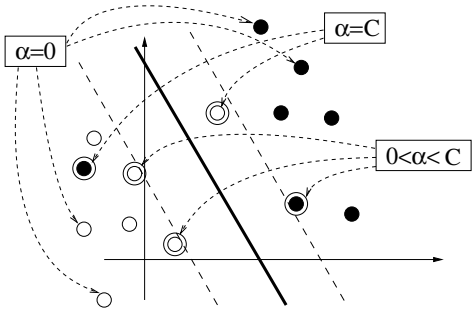


Figure 2.11 A linear function learned by an SVM to discriminate black from white circles, and the corresponding Lagrange multipliers $\boldsymbol{\alpha}$. Each point correctly classified with large confidence ($yf(\mathbf{x}) > 1$) has a null multiplier. Other points are called support vector. They can be on the boundary, in which case the multiplier satisfies $0 \leq \alpha \leq C$, or on the wrong side of this boundary, in which case $\alpha = C$.

$\boldsymbol{w}^\top \boldsymbol{x}_i + b = \boldsymbol{y}_i$. These points are correctly classified, but at the limit of the half-space of large confidence. Points not correctly classified with large confidence correspond to $\xi_i > 0$, and therefore $\beta_i = 0$ and $\alpha_i = C$.

Support vectors

The points with positive Lagrange multiplier $\alpha_i = 0$ are called *support vectors*. From (2.37) we see that $\boldsymbol{w}$ is a linear combination of the support vectors alone. Moreover, the solution found by the SVM does not change when non-support vectors are removed from the training set. Thus the set of support vectors contains all the information about the data set used by SVM to learn a discrimination function. This can easily be seen when it comes to predicting the class of a new object $\mathbf{x} \in \mathcal{X}$. Indeed we must then form the linear function

$$f(\mathbf{x}) = \boldsymbol{w}^\top \mathbf{x} + b = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i^\top \mathbf{x} + b, \tag{2.41}$$

and predict that the class of $\mathbf{x}$ is -1 or $+1$ depending on the sign of this function. The sum in (2.41) only involves support vectors.

2.4.2 General SVMs

From (2.39), (2.40), and (2.41), we see that learning a linear classifier and predicting the class of a new point only involves the points in the training set through their dot products. The kernel trick can therefore be applied to perform the SVM algorithm in the feature space associated with a general kernel. It can be stated as follows: find $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ which minimizes

$$W(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{n} \alpha_i, \tag{2.42}$$

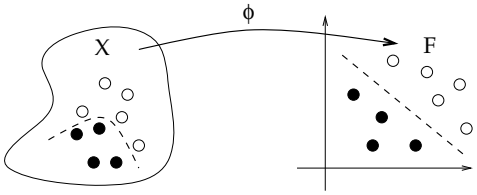


Figure 2.12 SVMs perform a linear discrimination of a training set of labeled points in the feature space associated with a kernel. The resulting separation can be nonlinear in the original space.

under the constraints

$$\begin{cases} \sum_{i=0}^{n} y_i \alpha_i = 0, \\ 0 \leq \alpha_i \leq C \quad \text{for } i = 1, \ldots, n. \end{cases}$$

Next, find an index i with $0 < \alpha_i < C$, and set:

$$b = y_i - \sum_{j=1}^{n} y_j \alpha_j k\left(\mathbf{x}_j, \mathbf{x}_i\right).$$

The classification of a new object $\mathbf{x} \in \mathcal{X}$ is then based on the sign of the function

$$f\left(\mathbf{x}\right) = \sum_{i=1}^{n} y_i \alpha_i k\left(\mathbf{x}_i, \mathbf{x}\right) + b. \tag{2.43}$$

The resulting function, although linear in the feature space associated with the kernel, can of course be nonlinear if the initial space is a vector space and the kernel is nonlinear. An example of nonlinear separation is illustrated in figure 2.12.

2.4.3 Variants and Extensions

Many variants of the basic SVM algorithm presented in the preceding sections have been proposed. Among the many variants surveyed in Schölkopf and Smola (2002), let us mention here a few directions to generalize the basic SVM algorithm. First, in the case of binary classification, several interesting modifications are obtained by changing the function (2.29) being optimized. Different norms on $\boldsymbol{w}$ together with different cost functions lead to interesting variants, which can be more or less difficult to solve. Lack of space prevents us from being exhaustive, so we will have to skip interesting approaches such as the *leave-one-out machine* (Weston, 1999; Weston and Herbrich, 2000) and the *Bayes point machines* (Ruján and Marchand, 2000; Herbrich et al., 2001; Rychetsky et al., 2000), as well as algorithms for tasks which are different from pattern recognition, such as regression estimation (Vapnik, 1995) and novelty detection (Schölkopf et al., 2001).

2.4.4 ν-SVMs

An alternative realization of a soft margin SVM (2.29) uses the ν-parametrization (Schölkopf et al., 2000). In this approach, the parameter C is replaced by $\nu \in [0, 1]$, which can be shown to lower- and upper-bound the number of examples that will be support vectors and that will come to lie on the wrong side of the hyperplane, respectively. In many situations, this provides a more natural parameterization than that using the somewhat unintuitive parameter C. The so-called ν-SVM uses a primal objective function with the error term $\frac{1}{\nu m}\sum_i \xi_i - \rho$, and separation constraints

ν-SVM

$$y_i \cdot ((\boldsymbol{w} \cdot \boldsymbol{x}_i) + b) \geq \rho - \xi_i, \quad i = 1, \ldots, m. \tag{2.44}$$

The margin parameter ρ is a variable of the optimization problem. The dual can be shown to consist of maximizing the quadratic part of (2.39), subject to $0 \leq \alpha_i \leq 1/(\nu m)$, $\sum_i \alpha_i y_i = 0$ and the additional constraint $\sum_i \alpha_i = 1$.

2.4.5 Linear Programming Machines

The idea of linear programming (LP) machines is to use the kernel expansion $f(x) = \sum_{i=1}^m \upsilon_i k(x, x_i) + b$ [cf. (2.43)] as an ansatz for the solution, but to use a different regularizer, namely the ℓ_1 norm of the coefficient vector (Mangasarian, 1965; Frieß and Harrison, 1998; Mattera et al., 1999; Bennett, 1999; Weston et al., 1999). The main motivation for this is that this regularizer is known to induce sparse solutions. This amounts to the objective function

$$R_{\mathrm{reg}}[g] := \frac{1}{m}\|\upsilon\|_1 + C\, R_{\mathrm{emp}}[g], \tag{2.45}$$

where $\|\upsilon\|_1 = \sum_{i=1}^m |\upsilon_i|$ denotes the ℓ_1 norm in coefficient space, using the soft margin empirical risk,

$$R_{\mathrm{emp}}[g] = \frac{1}{m}\sum_i \xi_i, \tag{2.46}$$

with slack terms

$$\xi_i = \max\{1 - y_i f(x_i), 0\}. \tag{2.47}$$

We thus obtain the LP problem

$$\min_{\boldsymbol{\alpha}, \boldsymbol{\xi} \in \mathbb{R}^m, b \in \mathbb{R}} \frac{1}{m}\sum_{i=1}^m (\alpha_i + \alpha_i^*) + C\sum_{i=1}^m \xi_i, \tag{2.48}$$

subject to

$$\begin{cases} y_i f(x_i) \geq 1 - \xi_i, \\ \alpha_i, \alpha_i^*, \xi_i \geq 0. \end{cases}$$

Here, the ℓ_1-norm has been componentwise split into positive and negative parts, that is, $v_i = \alpha_i - \alpha_i^*$. The solution differs from (2.43) in that each expansion pattern no longer necessarily has a weight $\alpha_i y_i$ with a sign equal to its class label; nor do the expansion patterns lie on or beyond the margin — in LP machines they can basically be anywhere, a feature which is reminiscent of the *relevance vector machine* (Tipping, 2001).

LP machines can also benefit from the ν-trick. In this case, the programming problem can be shown to take the following form (Graepel et al., 1999b):

$$\min_{\boldsymbol{\alpha},\boldsymbol{\xi}\in\mathbb{R}^m,b,\rho\in\mathbb{R}} \frac{1}{m}\sum_{i=1}^{m}\xi_i - \nu\rho, \tag{2.49}$$

subject to

$$\begin{cases} \frac{1}{m}\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) = 1, \\ y_i f(\boldsymbol{x}_i) \geq \rho - \xi_i, \\ \alpha_i, \alpha_i^*, \xi_i, \rho \geq 0. \end{cases}$$

2.5 SVMs in Practice

SVM implementations

A number of SVM implementations are freely or commercially available: For instance, SVM light,[3] LIBSVM, [4] and mySVM [5] are popular in the machine learning community, and a more complete and up-to-date list is available on the kernel method community website `http://www.kernel-machines.org`. While newcomers may feel that these programs can solve the learning tasks automatically, it in fact remains challenging to apply SVMs in a fully automatic manner. Questions regarding the choice of kernel, of parameters, of data representation, or of different flavors of SVMs, remain largely empirical in real-world applications. While default setting and parameters are generally useful as a starting point, big improvements can result from careful tuning of the algorithm. As an example, Hsu et al. (2003) report an accuracy improvement from 36% to 85.2% by an appropriate tuning.

2.5.1 Multiclass Problems

The basic SVM algorithm for pattern recognition is designed for classification of objects into two classes, but many real-world applications deal with more than two classes. This is, for example, the case when one wants to assign a function or a structure to a protein sequence, or a disease family to a tissue from gene expression

3. Available from `http://svmlight.joachims.org/`
4. Available from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`
5. Available from `http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/`

experiments. One way to use SVMs in this context is to apply an implementation that specifically solves multiclass problems (see, e.g., chapter 9). However, these implementations remain rare and cannot handle more than a few classes.

The most widely used method for multiclass problems involves reformulating them as a number of binary classification problems, and solving these problems with binary SVMs. The resulting SVMs must then be combined to form a multiclass prediction algorithm. The most common way to perform this split and combination is called the *One-against-all* scheme. It consists in first finding a discrimination between each class and all the others, thus transforming a problem with N classes into N binary problems. The scores output by each SVM are then combined by a *max rule*: an object is assigned to the class corresponding to the SVM that outputs the largest score. As an example, let us consider a three-class classification problem with the following training set labels:

One-against-all

$$\boldsymbol{y} = (1, 1, 1, 2, 2, 2, 3, 3, 3).$$

In the one-against-all scheme this problem is decomposed as three binary problems with the following class assignments:

$$\begin{aligned} \boldsymbol{y}_1 &= (1, 1, 1, -1, -1, -1, -1, -1, -1) \\ \boldsymbol{y}_2 &= (-1, -1, -1, 1, 1, 1, -1, -1, -1) \\ \boldsymbol{y}_3 &= (-1, -1, -1, -1, -1, -1, 1, 1, 1) \end{aligned} \tag{2.50}$$

Three SVMs are trained on the three class labels respectively. When an unknown sample is classified, the outputs of SVMs are compared and the sample is assigned to the class with the largest output.

We conclude this subsection by noting that several other methods for multiclass problems have been proposed.

- In *pairwise classification*, one classifier is learned for each possible pair of classes (see Friedman, 1996; Schmidt and Gish, 1996; Kreßel, 1999).
- In *error-correcting output codes*, a set of classifiers is trained, each one solving the task of separating the union of certain classes from the complement. By cleverly choosing the classes, the outputs of several such classifiers code the class membership of a given test point rather robustly (Allwein et al., 2000)
- *Multiclass objective functions* capture a multiclass problem by defining an objective function that simultaneously trains all the classifiers involved (Weston and Watkins, 1999). While this may be the most elegant approach, it tends to be too expensive to train all classifiers simultaneously, if the problem size is large.

2.5.2 Kernel Normalization

When the data set is a set of vectors, it is often effective to linearly scale each attribute to zero mean and unit variance, and then apply the Gaussian RBF kernel

or polynomial kernel (Hsu et al., 2003). The main advantage of this normalization is to avoid attributes in larger numeric ranges dominating those in smaller ranges.

For more general kernels such as string or graph kernels, the kernel matrix is often directly obtained without feature vectors. In this case, it is considered effective to normalize the kernel matrix such that all the diagonal elements are 1. If k_{ij} denotes the (i, j)th element of a kernel matrix k, this normalization is

$$k'_{ij} = \frac{k_{ij}}{\sqrt{k_{ii}k_{jj}}}.$$

More advanced methods for kernel normalization are described by Schölkopf et al. (2002)

2.5.3 Parameter Setting

In order to use a basic SVM for binary classification, two kinds of parameters have to be determined:

- The regularization parameter C of the SVM
- The kernel and its parameters

A proper choice of these parameters is crucial to the good performance of the algorithm. A temptation to be avoided is to set the parameters based on the performance of the SVM on the training set, because this is likely to lead to overfitting: the performance increases on the training set used, but decreases on new samples.

Cross-validation

A standard way to fix parameters is to use cross-validation. Let us denote by γ a parameter of a kernel to be set, for instance, the width of the Gaussian RBF kernel, and C the parameter of the algorithm. Given specific values of C and γ, the k-fold cross-validation error is calculated as follows: first of all, the training set $\mathcal{Z} = \{x_i, y_i\}_{i=1}^n$ is randomly divided into k subsets $\mathcal{Z}_1, \cdots, \mathcal{Z}_k$ of approximately equal size. The SVM is trained on $k-1$ subsets and its error rate on the remaining subset is computed. Repeating this process k times such that each subset is tested once, the cross-validation error is determined by the average of the test errors. When $k = n$, the cross-validation error is especially called the *leave-one-out error*.

Grid search

In this scheme C and γ are determined so as to minimize the cross-validation error. This goal is approximately achieved by a *Grid search*. A set of candidate values are chosen both for C and γ, and the cross-validation error is computed for every possible combination of them. If n_c and n_γ are the number of candidate values, then the cross-validation error is computed $n_c n_\gamma$ times, which means that the SVM is trained $k n_c n_\gamma$ times in total. Typically, users do not have any idea about the optimal values for C and γ, so the candidate values must cover a very large domain. Hsu et al. (2003) suggest the candidate values be determined as an exponentially growing sequence (e.g., $C = 2^{-5}, 2^{-3}, \cdots, 2^{15}, \gamma = 2^{-15}, \cdots, 2^3$). When there are more than two parameters, the grid search becomes difficult as the

number of grid points grows exponentially. In such cases, one can use a gradient search to minimize an upper bound on the leave-one-out error (Chapelle et al., 2002).

Model selection can lead to costly computations. For example, when $k = n_c = n_\gamma = 10$, the SVM must be trained 1000 times to choose C and γ, which might be prohibitive for large data sets. However, this process can be easily parallelized, which alleviates the burden of cross-validation.

2.6 More Kernels

In section 2.3, we presented some of the possibilities for data analysis offered by kernel methods. They are completely modular, in the sense that each method can be applied to any kernel. In this section we present classic or recently developed kernels that the reader might find it useful to be aware of.

2.6.1 Kernels for Vectors

A few kernels for vectors have gained considerable attention in the SVM community. The *linear kernel* which we already met,

$$k_L(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}',$$

Polynomial kernels

is a particular instance of the *polynomial kernels* defined for $d \geq 0$ by

$$k_{Poly1}(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^\top \mathbf{x}'\right)^d,$$

or

$$k_{Poly2}(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^\top \mathbf{x}' + c\right)^d,$$

where d is the degree of the polynomial and c is a constant in the second kernel. The polynomial kernel k_{Poly1} of degree 2 corresponds to a feature space spanned by all products of 2 variables, that is, $\left\{x_1^2, x_1 x_2, x_2^2\right\}$. It is easy to see that the kernel k_{Poly2} of degree 2 corresponds to a feature space spanned by all products of at most 2 variables, that is, $\left\{1, x_1, x_2, x_1^2, x_1 x_2, x_2^2\right\}$. More generally the kernel k_{Poly1} corresponds to a feature space spanned by all products of exactly d variables, while the kernel k_{Poly2} corresponds to a feature space spanned by all products of at most d variables.

Gaussian RBF kernel

The *Gaussian RBF kernel*

$$k_G(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right),$$

where σ is a parameter, is one of the most frequently used kernels in practice, thanks to its capacity to generate nonparametric classification functions. Indeed, the discriminant function learned by an SVM has the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} y_i \alpha_i \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}||^2}{2\sigma^2}\right)$$

and is therefore a sum of Gaussian centered on the support vectors. Almost any decision boundary can be obtained with this kernel. Observe that the smaller the parameter σ, the more peaked the Gaussians are around the support vectors, and therefore the more complex the decision boundary can be. Larger σ corresponds to a smoother decision boundary.

Sigmoid kernel

The *sigmoid kernel* is defined by

$$k(\mathbf{x}, \mathbf{x}') = \tanh\left(\kappa \mathbf{x}^\top \mathbf{x}' + \theta\right),$$

where $\kappa > 0$ and $\theta < 0$ are parameters respectively called *gain* and *threshold*. The main motivation behind the use of this kernel is that the decision function learned by an SVM,

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i \tanh\left(\kappa \mathbf{x}_i^\top \mathbf{x} + \theta\right),$$

is a particular type of two-layer sigmoidal neural network. In fact the sigmoid kernel is not always positive definite, but has still been successfully used in practice.

2.6.2 Kernels for Strings

Computational biology is a field rich in strings, such as peptide or nucleotide strings. As a result, much work has been devoted recently to the problem of making kernels for strings, as illustrated in chapters 4, 5, and 6.

String kernels differ in the information about strings they encode, their implementation, and their complexity. In order to give a flavor of string kernels, we present a particular string kernel proposed by Lodhi et al. (2002) in the context of natural language processing, which is one of the earliest string kernels. The basic idea is to count the number of subsequences up to length n in a sequence, and compose a high-dimensional feature vector by these counts. The string kernel is defined as a dot product between such feature vectors.

More precisely, let Σ be the set of symbols. A string s of length $|s|$ is defined as $s = s_1, \ldots, s_{|s|} \in \Sigma^{|s|}$. The set of all strings is $\mathcal{X} = \bigcup_{i=0}^{\infty} \Sigma^i$. An index set $\boldsymbol{i}$ of length l is an l-tuple of positions in s,

$$\boldsymbol{i} = (i_1, \ldots, i_l), \quad 1 \leq i_1 < \ldots < i_l \leq |s|,$$

and we denote by $s[\boldsymbol{i}] = s_{i_1}, \ldots, s_{i_l}$ the subsequence of s corresponding to the indices in $\boldsymbol{i}$. Let us define the weight of the index set $\boldsymbol{i}$ by

$$\lambda^{l(\boldsymbol{i})}, \text{ where } \quad l(\boldsymbol{i}) = i_l - i_1 + 1,$$

where $\lambda < 1$ is a predetermined constant. Thus, for a given subsequence length l, the weight decreases exponentially with the number of gaps in the subsequence.

For each sequence $u \in \Sigma^k$, where k is fixed, let us now define a feature $\Phi_u : \mathcal{X} \to \mathbb{R}$ as

$$\forall s \in \mathcal{X}, \quad \Phi_u(s) = \sum_{\boldsymbol{i}: s[\boldsymbol{i}]=u} \lambda^{l(\boldsymbol{i})}.$$

Considering all sequences u of length n, we can map each sequence $s \in \mathcal{X}$ to a $|\Sigma|^n$-dimensional feature space by the mapping $s \to (\Phi_u(s))_{u \in \Sigma^n}$. We can then define a kernel for strings as the dot product between these representations,

$$\begin{aligned} \forall s, t \in \mathcal{X}, \quad k_n(s,t) &= \sum_{u \in \Sigma^n} \sum_{\boldsymbol{i}: u=s[\boldsymbol{i}]} \lambda^{l(\boldsymbol{i})} \sum_{\boldsymbol{j}: u=t[\boldsymbol{j}]} \lambda^{l(\boldsymbol{j})} \\ &= \sum_{u \in \Sigma^n} \sum_{\boldsymbol{i}: u=s[\boldsymbol{i}]} \sum_{\boldsymbol{j}: u=t[\boldsymbol{j}]} \lambda^{l(\boldsymbol{i})+l(\boldsymbol{j})}. \end{aligned} \tag{2.51}$$

Calculating each feature is hopeless because of the high dimensionality. However, it has been shown that a recursive algorithm can calculate k_n efficiently with a time complexity $O(n|s||t|)$ (Lodhi et al., 2002), using dynamic programming.

2.6.3 The Fisher Kernel

Probabilistic models are convenient to represent families of complex objects that arise in computational biology. Typically, such models are useful when one wants to characterize a family of objects $\mathbf{x}$ that belong to a big set $\mathcal{X}$, but only span a very small subset of $\mathcal{X}$. The models can then be used to infer a probability distribution on $\mathcal{X}$ concentrated on the objects observed or likely to be observed. For example, hidden Markov models (HMMs) are a central tool for modeling protein families or finding genes from DNA sequences (Durbin et al., 1998). More complicated models called stochastic context-free grammars (SCFGs) are useful for modeling RNA sequences (Sakakibara et al., 1994).

The success of a particular probabilistic model requires that the distribution of actual objects be well characterized by that model. The Fisher kernel (Jaakkola and Haussler, 1999) provides a general principle to design a kernel for objects well modeled by a probabilistic distribution, or more precisely a parametric statistical model. Denote by $p(\mathbf{x}|\boldsymbol{\theta})$, $\mathbf{x} \in \mathcal{X}, \boldsymbol{\theta} \in \Re^p$ a parametric statistical model with a p-dimensional parameter $\boldsymbol{\theta}$ on the measurable space;

$$\forall \theta \in \Theta, \quad \int_{\mathcal{X}} p(\mathbf{x}|\theta) d\mathbf{x} = 1.$$

Moreover, $(\mathbf{x}, \theta) \rightarrow p(\mathbf{x}|\theta)$ is required to be smooth enough for all following computations to make sense.

Given a sample $\mathcal{S} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, suppose a parameter $\hat{\boldsymbol{\theta}}$ is estimated to model $\mathcal{S}$, for example, by maximum likelihood. The Fisher kernel is then defined as

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad k(\mathbf{x}, \mathbf{x}') = \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\hat{\boldsymbol{\theta}})^\top J^{-1} \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}'|\hat{\boldsymbol{\theta}}),$$

where

$$\nabla_{\boldsymbol{\theta}} = \left(\frac{\partial}{\partial \theta_1}, \cdots, \frac{\partial}{\partial \theta_p} \right)^\top$$

is a gradient vector with respect to $\boldsymbol{\theta}$ and J is the Fisher information matrix;

$$J = \int_{\mathbf{x} \in \mathcal{X}} \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\hat{\boldsymbol{\theta}}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\hat{\boldsymbol{\theta}})^\top p(\mathbf{x}|\hat{\boldsymbol{\theta}}) dx.$$

The Fisher kernel can be understood intuitively when the parametric model is an exponential family. An exponential family of densities is written as

$$p(\mathbf{x}|\boldsymbol{\theta}) = \exp(\boldsymbol{\theta}^\top \boldsymbol{s}(\mathbf{x}) + \phi(\boldsymbol{\theta})),$$

where $\boldsymbol{s} : \mathcal{X} \rightarrow \Re^p$ is a vector-valued function and ϕ is a normalization factor to ensure that $\sum_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}|\boldsymbol{\theta}) = 1$. The function $\boldsymbol{s}$, commonly called "sufficient statistics," plays a role of feature extraction from x, because $p(x|\boldsymbol{\theta})$ depends on x solely through $\boldsymbol{s}$. The Fisher kernel can recover this "hidden" feature of x because

$$\nabla_{\boldsymbol{\theta}} \log p(x|\hat{\theta}) = \boldsymbol{s}(x) + \nabla_{\boldsymbol{\theta}} \phi(\hat{\boldsymbol{\theta}}),$$

and the second term is a constant independent of x. Usually the Fisher kernel is applied to complicated probability distributions which do not belong to the exponential family (e.g., HMMs). However, the Fisher kernel can still effectively reveal the features implicitly used in a probabilistic model (see Tsuda et al., 2004, for details).

The first application of the Fisher kernel was in the context of the protein remote homology detection, in combination with SVMs, where it outperformed all other state-of-the-art methods (Jaakkola et al., 2000). Extensions to the Fisher kernel idea can be seen, for example, in Tsuda et al. (2002a,b), Sonnenburg et al. (2002), and Seeger (2002).

2.7 Designing Kernels

As suggested in the previous section, a wide choice of kernels already exists. Many data or applications may still benefit from the design of particular kernels, adapted specifically to a given task. In this section, we review several useful results and principles when one wants to design a new kernel, or even "learn" a kernel from the observed data.

2.7.1 Operations on Kernels

The class of kernel functions on a set $\mathcal{X}$ has several useful closure properties. It is a convex cone, which means that if k_1 and k_2 are two kernels, then any linear combination,

$$\lambda_1 k_1 + \lambda_2 k_2,$$

with $\lambda_1, \lambda_2 \geq 0$ is a kernel.

The set of kernels is also closed under the topology of pointwise convergence, which means that if one has a family of kernel $(k_i)_{i\in\mathbb{N}}$ that converges in a pointwise fashion to a function,

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad \lim_{n\to\infty} k_n(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}'),$$

then k is a kernel.

Other useful properties include closure under the pointwise multiplication, also called the Schur product (Schur, 1911): if k_1 and k_2 are two kernels, then

$$k(\mathbf{x}, \mathbf{x}') := k_1(\mathbf{x}, \mathbf{x}')\, k_2(\mathbf{x}, \mathbf{x}')$$

is also a kernel. From this and the closure under pointwise limit we can deduce a useful corollary: if $f(z) = \sum_{i=0}^{\infty} a_i z_i$ is holomorphic in $\{z \in \mathbb{C} : |z| < \rho\}$, and if k is a kernel such that $|k(\mathbf{x}, \mathbf{x}')| < \rho$ for any $\mathbf{x}, \mathbf{x}'$, then $f \circ k$ is a valid kernel. As an example, for any kernel k, $\exp(k)$ is a valid kernel, and for any bounded kernel $|k| < \rho$, $(\rho - \mathbf{x})^{-1}$ is a valid kernel.

On the other hand, other operations on kernels are in general forbidden. For example, if k is a kernel, then $\log(k)$ is not positive definite in general, and neither is k^β for $0 < \beta < 1$. In fact these two operations are linked by the following result: k^β is positive definite for any $\beta > 0$ iff $\log(k)$ is *conditionally positive definite*:

Conditionally positive definite

$$\sum_{i,j=1}^{n} c_i c_j \log(k(\mathbf{x}_i, \mathbf{x}_j)) \geq 0$$

for any $n > 0$, $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$ and $c_1, \ldots, c_n \in \mathbb{R}$ with the additional constraint that $\sum_{i=1}^{n} c_i = 0$. Such a kernel is called *infinitely divisible*. These considerations are, for example, discussed in chapter 6.

2.7.2 Translation-Invariant Kernels and Kernels on Semi-Groups

When $\mathcal{X} = \mathbb{R}^p$, the class of translation-invariant kernels is defined as the class of kernels of the form

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad k(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x} - \mathbf{x}'),$$

for some function $\psi : \mathbb{R}^p \to \mathbb{R}$. The Gaussian kernel (2.5) is an example of a translation-invariant kernel. These kernels are particular examples of *group kernels*:

Group kernel

if $(\mathcal{X}, .)$ is a group,[6] then group kernels are defined as functions of the form $k(\mathbf{x}, \mathbf{x}') = \psi\left(\mathbf{x}^{-1}\mathbf{x}'\right)$, with $\psi : \mathcal{X} \to \mathbb{R}$. Conditions on ψ to ensure that k is a symmetric positive definite kernel have been studied in relation to harmonic analysis on groups and semigroups; the interested reader should consult Berg et al. (1984) for a complete treatment of these conditions for Abelian groups and semigroups. In the case $(\mathcal{X}, .) = (\mathbb{R}^p, +)$, the classic Bochner theorem states that if ψ is continuous, then k is a valid kernel iff ψ is the Fourier transform of a nonnegative finite measure. This is, for example, the case for the Gaussian RBF kernel. If $(\mathcal{X}, .)$ is a discrete Abelian semi-group, then k is a kernel iff ψ is the Fourier transform of a nonnegative Radon measure. Such results can be extended to more general groups and semi-groups, and suggest principled ways to design kernels on sets with a group structure, such as the set of permutations, or on sets endowed with a group action. These kernels are related to the diffusion kernel presented in chapter 8, which can be considered an efficient way to compute a group kernel if the graph is considered as the Cayley graph of a group.

2.7.3 Combining Kernels

Rather than design a kernel from scratch, one might be tempted to generate a kernel from a family of available kernels. In such cases, multiple kernel matrices $k_1, k_2, \cdots, k_c$ for the same set of objects are available. We might then wish to use kernel methods to combine this heterogeneous information; in other words, we would like to design a single kernel k from several basic kernels $k_1, \ldots, k_c$. A simple way to achieve this is to take the sum of the kernels:

$$k = \sum_{i=1}^{c} k_i.$$

This is clearly a valid kernel that can be interpreted as taking the direct product of the feature spaces of the basic kernels as a feature space. This approach was proposed in Pavlidis et al. (2002) in the context of functional genomics, and validated as a useful way to integrate heterogeneous data.

A slight generalization of this approach is to take a weighted sum,

$$k = \sum_{i=1}^{c} \mu_i k_i.$$

A nontrivial question is how to chose the weights automatically. Several approaches have been pioneered recently and are presented in forthcoming chapters: semidefinite programming in chapter 11, kernel canonical correlation analysis in chapter 10, and an information geometry-based approach in chapter 12.

6. A group is a set with an associative operation, a neutral element, and such that any element has an inverse. If the operation is commutative, the group is called Abelian.

2.7.4 From Similarity Scores to Kernels

Another typical situation which arises when designing a kernel in computational biology, as well as in other fields, is when one has a "good" function to measure the similarity between objects, but which is unfortunately not a kernel. Such an example is, for instance, treated in detail in chapter 6, where a kernel for biological sequences is built to mimic well-known measures of similarity between sequences. Other examples include the design of a kernel for molecular 3D structures from measures of structural similarity (see chapter 12).

Empirical kernel map

Again, there is no single answer but rather a number of approaches that have been proposed and tested recently. Let $\mathcal{X}$ be a set and $s : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a measure of similarity. One principled way to convert s into a valid kernel is called the *empirical kernel map* (Tsuda, 1999). It consists in first choosing a finite set of objects $t_1, \cdots, t_r \in \mathcal{X}$ called *templates*. An object $\mathbf{x} \in \mathcal{X}$ is then represented by a vector of similarity with respect to the template samples:

$$\mathbf{x} \in \mathcal{X} \to \phi(\mathbf{x}) = (s(x, t_1), \ldots, s(x, t_r))^\top \in \mathbb{R}^p.$$

The kernel is then defined as the dot product between two similarity vectors:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}) = \sum_{i=1}^{r} s(\mathbf{x}, t_i)\, s(\mathbf{x}', t_i).$$

Liao and Noble (2002) successfully applied this technique to transform an alignment score between protein sequences into a powerful kernel for remote homology detection However, one drawback of this method is that the results depend on the choice of template samples, as well as the fact that it can be computationally prohibitive.

In some cases, all objects to be processed by kernel methods are known in advance. This is the case, for example, when kernel PCA is performed on a finite set of objects, or when an SVM is trained in a transductive framework, that is, when the unannotated objects to be classified are known in advance. A good example of a transductive problem is in functional genomics on an organism: given the knowledge we have about the functions of some genes of an organism, can we predict the functions of the unannotated genes, which we know in advance because we know the whole genome.

Removal of negative eigenvalues

In such cases, the problem boils down to making a symmetric positive definite matrix kernel matrix out of a pairwise similarity matrix. A natural way to perform this is by eigendecomposition of the similarity matrix (which is supposed to be symmetric), and removal of negative eigenvalues (Graepel et al., 1999a; Roth et al., 2003). Recently Roth et al. (2003) pointed out that this method preserves clusters in data, and showed promising experimental results in classifying protein sequences based on the FASTA scores.

2.8 Conclusion

This short introductory tour of positive definite kernels and kernel methods only contains a very brief and partial summary of a field that has a long history, but was only recently investigated in depth in the context of empirical inference and machine learning. As highlighted by the different chapters of this book, this field is very active nowadays, with promising applications in computational biology.

3 Support Vector Machine Applications in Computational Biology

William Stafford Noble[1]

During the past 3 years, the support vector machine (SVM) learning algorithm has been extensively applied within the field of computational biology. The algorithm has been used to detect patterns within and among biological sequences, to classify genes and patients based upon gene expression profiles, and has recently been applied to several new biological problems. This chapter reviews the state of the art with respect to SVM applications in computational biology.

3.1 Introduction

The SVM algorithm (Boser et al., 1992; Vapnik, 1998) is a classification algorithm that provides state-of-the-art performance in a wide variety of application domains, including handwriting recognition, object recognition, speaker identification, face detection, and text categorization (Cristianini and Shawe-Taylor, 2000). During the past 3 years, SVMs have been applied very broadly within the field of computational biology, to pattern recognition problems, including protein remote homology detection, microarray gene expression analysis, recognition of translation start sites, functional classification of promoter regions, prediction of protein-protein interactions, and peptide identification from mass spectrometry data. The purpose of this chapter is to review these applications, summarizing the state of the art.

Two main motivations suggest the use of SVMs in computational biology. First, many biological problems involve high-dimensional, noisy data, for which SVMs are known to behave well compared to other statistical or machine learning methods. Second, in contrast to most machine learning methods, kernel methods like the SVM

1. Formerly William Noble Grundy. See `www.gs.washington.edu/noble/name-change.html`.

can easily handle nonvector inputs, such as variable length sequences or graphs. These types of data are common in biology applications, and often require the engineering of knowledge-based kernel functions. Much of this review consists in explaining these kernels and relating them to one another.

This review assumes that the reader has a basic familiarity with SVMs, including the notion of a kernel function and the mapping from input space to feature space. Background information can be found in Cristianini and Shawe-Taylor (2000), Burges (1998), and at `www.kernel-machines.org`. The chapter is organized by application domain, beginning in section 3.2 with perhaps the most intensively studied application, the recognition of subtle similarities among protein sequences. Section 3.3 reviews other protein and gene classification tasks, and section 3.4 looks at problems that involve recognizing patterns within a protein or DNA sequence. Section 3.5 reviews the many applications of SVMs to the analysis of DNA microarray expression data. Section 3.6 describes three approaches to learning from heterogeneous biological data. Finally, the chapter closes with a description of several applications that do not fit neatly into the previous categories, followed by a brief discussion.

3.2 Protein Remote Homology Detection

Over the past 25 years, researchers have developed a battery of successively more powerful methods for detecting protein sequence similarities. This development can be broken into four stages. Early methods looked for pairwise similarities between proteins. Among such algorithms, the Smith-Waterman dynamic programming algorithm (Smith and Waterman, 1981) is among the most accurate, whereas heuristic algorithms such as BLAST (Altschul et al., 1990) and FASTA (Pearson, 1990) trade reduced accuracy for improved efficiency.

In the second stage, further accuracy was achieved by collecting aggregate statistics from a set of similar sequences and comparing the resulting statistics to a single, unlabeled protein of interest. Profiles (Gribskov et al., 1990) and hidden Markov models (HMMs) (Krogh et al., 1994; Baldi et al., 1994) are two methods for representing these aggregate statistics. For a given false-positive rate, these family-based methods allow the computational biologist to infer nearly three times as many homologies as a simple pairwise alignment algorithm (Park et al., 1998).

In stage 3, additional accuracy was gleaned by leveraging the information in large databases of unlabeled protein sequences. Iterative methods such as PSI-BLAST (Altschul et al., 1997) and SAM-T98 (Karplus et al., 1998) improve upon profile-based methods by iteratively collecting homologous sequences from a large database and incorporating the resulting statistics into a single model. All of the resulting statistics, however, are generated from positive examples, that is, from sequences that are known or posited to be evolutionarily related to one another.

The Fisher kernel

In 1999, Jaakkola et al. ushered in stage 4 of the development of homology detection algorithms with a paper that garnered the "Best Paper" award at the

annual Intelligent Systems for Molecular Biology conference. Their primary insight was that additional accuracy can be obtained by modeling the difference between positive and negative examples. Because the homology task requires discriminating between related and unrelated sequences, explicitly modeling the difference between these two sets of sequences yields an extremely powerful method. The algorithm described in that paper is called *SVM-Fisher*.

The SVM-Fisher method (Jaakkola et al., 1999, 2000) couples an iterative HMM training scheme with the SVM. For any given family of related proteins, the HMM provides a kernel function. First, the HMM is trained on positive members of the training set using the standard Baum-Welch training routine. The training is iterated, adding to the training set at each round similar sequences from a large unlabelled database. After training, the gradient vector of any sequence—positive, negative, or unlabeled—can be computed with respect to the trained model. As in the Baum-Welch training algorithm for HMMs, the forward and backward matrices are combined to yield a count of observations for each parameter in the HMM. As shown in Jaakkola et al. (1999), the counts can be converted into components of a gradient vector $\tilde{\mathbf{U}}$ via the following equation:

$$\tilde{\mathbf{U}}_{ij} = \frac{E_j(i)}{e_j(i)} - \sum_k E_j(k), \tag{3.1}$$

where $E_j(i)$ is the number of times that amino acid i is observed in state j, and $e_j(i)$ is the emission probability for amino acid i in state j. Although these gradients can be computed for every HMM parameter, the SVM-Fisher method uses only the gradient components that correspond to emission probabilities in the match states. Furthermore, a more compact gradient vector can be derived using a mixture decomposition of the emission probabilities. Each sequence vector summarizes how different the given sequence is from a typical member of the given protein family. Finally, an SVM is trained on a collection of positively and negatively labeled protein gradient vectors. By combining HMMs and SVMs, SVM-Fisher offers an interpretable model, a means of incorporating prior knowledge and missing data, and excellent recognition performance.

Indeed, the SVM-Fisher method yields results that improve significantly upon the previous state of the art. The standard benchmark for this classification task comes from the Structural Classification of Proteins (SCOP) (Murzin et al., 1995), which provides protein superfamily labels based upon human interpretation of three-dimensional protein structures (see fig. 3.1). The original experiment compared SVM-Fisher to BLAST and to the SAM-T98 iterative HMM method.

Hughey and Krogh (1996), and a subsequent experiment included a comparison to PSI-BLAST (Leslie et al., 2002). In each case, SVM-Fisher performed significantly better than previous methods. Subsequent work by Karchin et al. (2002) demonstrated the successful application of the SVM-Fisher methodology to the recognition of a large, pharmaceutically important class of proteins, the G protein-coupled receptors.

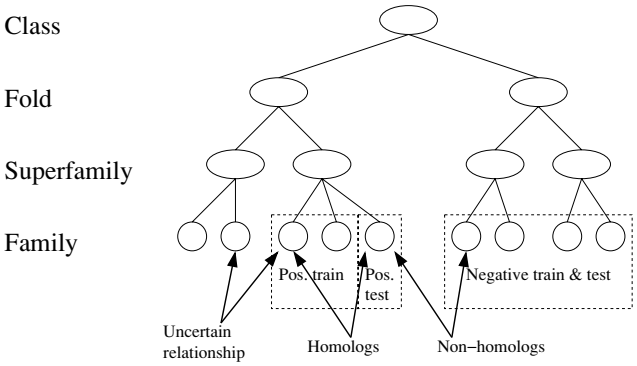


Figure 3.1 The SCOP hiearchy of protein domains. SCOP is a hand-curated database that is arranged hierarchically according to protein three-dimensional structure. The three primary levels of the hierarchy—family, superfamily, and fold—correspond to varying degrees of similarity. Proteins within a single family show clear evolutionary relationships, typically evidenced by more than 30% pairwise identities at the sequence level, while members of a superfamily may have low sequence identity, but have structural and functional features that suggest a common evolutionary origin. Finally, proteins belong to the same fold if they have the same major secondary structures in the same arrangement and with the same topological connections. Proteins placed together in the same fold category may not have a common evolutionary origin. The figure illustrates how a SCOP-based benchmark is created. All but one family within a given superfamily constitute the positive training set, and the held-out family constitutes the positive test set. Negative examples are drawn from outside of the training set fold.

Recently, the Fisher kernel framework was elegantly generalized by Tsuda et al. (2002b). They describe a general method for deriving a kernel from any latent variable model, such as an HMM. The kernel assumes the availability of the hidden variables, which are estimated probabilistically. The resulting *joint kernel* can be converted to a *marginalized kernel* by taking its expectation with respect to the hidden variables. The Fisher kernel, it turns out, is a special case of marginalized kernels. The framework is demonstrated by using a small HMM-based marginalized kernel to characterize a single family of bacterial proteins.

Composition kernels

Subsequent to the introduction of the Fisher kernel, many different kernels have been applied to the problem of protein remote homology. Ding and Dubchak (2001) define one of the simplest such kernels, a composition-based kernel function that characterizes a given protein via the frequency with which various amino acids occur therein. In this work, each protein is characterized by a simple vector of letter frequencies. Each protein sequence is represented via six different alphabets, corresponding to amino acids, predicted secondary structure, hydrophobicity, normalized van der Waals volume, polarity, and polarizability. A single protein is represented by the letter frequencies across each of these alphabets, for a total of 125 features.

The focus of this work is not the kernel function but the machinery for making multiclass predictions. The most common means of training an SVM for an n-class problem is the *one-vs.-others method*: n SVMs are trained, one per class, using members of all other classes as negative examples. The final classification of a test example is the class corresponding to the SVM that yields the discriminant with largest absolute value. Ding and Dubchak introduce a method called the *unique one-vs.-others* method, which performs additional SVM optimizations to sort out disagreements among SVMs trained using the standard, one-vs-others method, and they show that their method leads to significant improvement in test set accuracy. The work also shows that an SVM outperforms a similarly trained neural network on this task.

A similar composition kernel is used by Cai et al. (2001) to recognize broad structural classes of proteins (all-α, all-β, α/β, and $\alpha + \beta$). On this task, the SVM yields better discrimination performance than a neural network method and a method previously developed by the same authors.

Motif kernels

A significant drawback to the composition kernel is the simplicity of the protein representation. Logan et al. (2001) propose a richer representational scheme, in which features correspond to motifs in a pre-existing database. The BLOCKS database (Henikoff and Henikoff, 1991) contains weight matrix motifs derived from protein multiple alignments. Because these motifs occur in regions that are highly conserved, they tend to correspond to functionally important regions of the proteins. This observation motivates using motifs as features for an SVM. Logan et al. use the BLIMPS tool (Wallace and Henikoff, 1992) to compare 10,000 BLOCKS motifs to each protein in the SCOP database. The resulting scores are used to map each protein into a 10,000-dimensional space. On a small collection of SCOP families, this motif kernel performs better than an HMM method and comparably to the Fisher-SVM.

Recently, a different motif kernel was described by Ben-Hur and Brutlag (2003). This kernel uses the eBLOCKS database (`motif.stanford.edu/eblocks`), which contains close to 500,000 motifs. Rather than represent each motif via a weight matrix, eBLOCKS uses discrete sequence motifs. For example, the 6-mer motif `[AS].DKF[FILMV]` contains three types of sites: the first position matches either `A` or `S`, the second position matches any amino acid, and the third position matches only the amino acid `D`. Thus, this motif would match the following example sequences: `ACDKFF`, `SRDKFI`, and `SADKFV`. Because the motif database is so large, a simple vector representation is computationally infeasible. Ben-Hur and Brutlag therefore demonstrate how to compute the corresonding kernel values efficiently using a trie data structure. Tested on a SCOP benchmark (Liao and Noble, 2002), the motif kernel provides a significant improvement in performance over previously described kernels.

Pairwise comparison kernels

One appealing characteristic of the Fisher kernel is its ability to incorporate prior knowledge that is built into the profile HMM framework, including a simple model of molecular evolution. An alternative evolutionary model is implicit in pairwise sequence comparison algorithms, such as the Smith-Waterman (Smith and

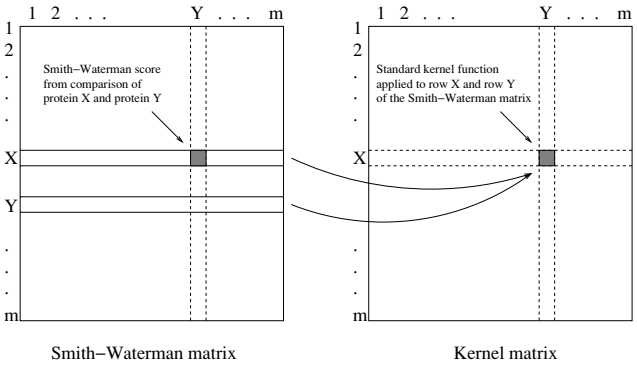


Figure 3.2 An empirical kernel map derived from the Smith-Waterman sequence comparison algorithm. Each matrix contains m rows and columns, corresponding to the proteins in the training set. Each entry in the matrix on the left is the Smith-Waterman score of the corresponding proteins. Each entry on the right is the result of applying a standard kernel function (e.g., dot product, polynomial, or radial basis) to the two corresponding rows from the Smith-Waterman matrix.

Waterman, 1981) dynamic programming algorithm and its heuristic approximation, BLAST (Altschul et al., 1990). Like the HMM, these algorithms assume that molecular evolution primarily proceeds via mutations and small-scale insertions and deletions. Furthermore, through extensive application over more than two decades of research, pairwise sequence comparison algorithms have been exhaustively analyzed and optimized. For example, the distribution of scores produced by these algorithms can be well characterized and used to compute a p-value or E-value associated with each observed score.

Liao and Noble (2002, 2003) describe a simple method for generating a kernel from the scores produced by a pairwise sequence comparison algorithm. These algorithms have the form of a kernel function, in the sense that they measure the similarity between a pair of objects being classified; however, the scores themselves are not positive definite and so cannot be used as kernels. Therefore, Liao and Noble employ the empirical kernel map (Tsuda, 1999) to convert the scores to a valid kernel. This procedure is illustrated in fig. 3.2. The matrix on the left is an $m \times m$ matrix of Smith-Waterman scores, corresponding to all pairs of proteins in a training set. Each row in this matrix can be used as a vector representation of the corresponding protein. A standard kernel function is then used to compute the similarity between these vectors. Thus, each entry in the matrix on the right in fig. 3.2 is simply the scalar product of two rows from the matrix on the left. Because the procedure uses a standard kernel function, the empirical kernel map guarantees a valid kernel matrix. Furthermore, the empirical kernel map offers an easy way to incorporate prior knowledge directly into the kernel. For example, a sequence kernel based on the Smith-Waterman or BLAST algorithm benefits from its implicit

model of molecular evolution as well as from two decades of empirical optimization of the algorithm's parameters. In conjunction with an SVM classifier, the Smith-Waterman empirical kernel map yields a powerful method—called SVM-pairwise—for detection of subtle protein sequence similarity, performing significantly better than the Fisher kernel on the data set used in that paper (Liao and Noble, 2002).

One drawback to the SVM-pairwise algorithm is its efficiency; however, several variants of the algorithm address this issue. The computation of the kernel matrix requires precomputation of all pairwise sequence comparison scores in the training set. For the Smith-Waterman algorithm, each such computation is $O(p^2)$, where p is the length of the protein sequences. This step can be sped up by a factor of p by using the heuristic BLAST algorithm instead, at a small loss in accuracy (Liao and Noble, 2002). The second step of the kernel computation—calculation of the empirical kernel map—is also expensive, requiring $O(m)$ time for each kernel value, where m is the number of proteins in the training set. For some families of proteins, the value of m can become quite large, on the order of 10,000. This step can be sped up by using a smaller *vectorization set* of proteins in the empirical kernel map, where the vectorization set defines the columns in the left-hand matrix in fig. 3.2. For example, using a vectorization set consisting only of the positive training examples leads to a significant time savings, again at a relatively small decrease in performance (Liao and Noble, 2003).

String kernels

String kernels comprise another class of kernels for protein remote homology detection. Like the BLAST and Smith-Waterman algorithms, string kernels operate directly on pairs of proteins; however, string kernels are positive definite functions and hence do not require the empirical feature map. The most general types of string kernels are pair HMM and convolution kernels (Watkins, 2000; Haussler, 1999; Lodhi et al., 2002). However, these kernels are expensive to compute and have not been applied to protein classification.

Leslie, Eskin, and Noble (2002) describe a simple string kernel—the *spectrum kernel*—that is more efficient to compute. This kernel is, in a sense, a generalization of the composition kernel mentioned earlier, in which the composition is computed with respect to length-k substrings, called k-mers. For example, for $k = 5$ and an alphabet of size 20, each vector consists of $5^{20} = 9.5 \times 10^{13}$ elements, each corresponding to a single 5-mer. The kernel can be computed efficiently using a trie data structure. On the SCOP benchmark used by Jaakkola et al. (1999), the spectrum kernel using $k = 3$ provides performance comparable to that of the HMM-based Fisher kernel. An alternative version of the spectrum kernel based upon suffix trees and suffix links was subsequently described by Vishwanathan and Smola (2003). For computing individual kernel values, the suffix tree implementation is faster by a factor of $O(k)$. However, this difference disappears for the computation of a complete matrix of m^2 kernel values: the trie-based spectrum kernel method allows for efficient construction of the full matrix in one pass of the algorithm, and this computation is as fast as calculating m^2 individual kernel values with the suffix tree method.

Table 3.1 Efficiency of kernels for protein sequence comparison. Each entry in the first table is the running time required to compute an $m \times m$ matrix of kernel values. Variables are defined in the second table. For simplicity, all proteins are assumed to be of approximately the same length p.

Kernel	Complexity	Reference
SVM-Fisher	$O(s^2mp + sm^2)$	(Jaakkola et al., 1999)
SVM-pairwise	$O(vmp^2 + vm^2)$	(Liao and Noble, 2003)
spectrum	$O(pm^2)$	(Leslie et al., 2002)
mismatch	$O(k^M \ell^M pm^2)$	(Leslie et al., 2003b)
gappy, substitution, wildcard	$O(c_K pm^2)$	(Leslie and Kuang, 2003)
weight matrix motif	$O(\ell pqm^2)$	(Logan et al., 2001)
discrete motif	$O(pqm^2)$	(Ben-Hur and Brutlag, 2003)

Variable definitions

p	length of one protein
m	number of proteins in training set
s	number of states in profile HMM
v	number of proteins in vectorization set
k	k-mer (substring) length
M	number of allowed mismatches
ℓ	size of alphabet
c_K	constant that is independent of alphabet size
q	number of motifs in database

The spectrum kernel has also been generalized to allow for a more accurate model of molecular evolution. Mutations in the protein sequence are modeled using a *mismatch kernel* (Leslie et al., 2003b), in which matches between k-mers are allowed to contain at most M mismatches. Thus, for $M = 1$, a feature corresponding to a k-mer such as `VTWTA` would match sequences such as `VTATA`, `VCWTA`, or `VTWTK`. Further flexibility, including deletions of amino acids and more accurate modeling of mutations, are modeled using a collection of string kernel functions introduced by Leslie and Kuang (2003). These generalizations also use the trie data structure, and have a running time that does not depend upon the size of the alphabet.

The efficiencies of the various kernel functions for protein remote homology detection are summarized in table 3.1. With respect to the quality of the results produced by these various kernels, conclusions are difficult to draw. There are two primary SCOP benchmarks, one that includes in the training set additional non-SCOP homologs identified via an HMM (Jaakkola et al., 1999) and one that uses only SCOP domains (Liao and Noble, 2002). The SVM-Fisher method performs well on its original benchmark (Jaakkola et al., 1999) but less well when non-SCOP homologs are removed from the training set (Liao and Noble, 2002), presumably

because the HMMs are consequently undertrained. The SVM-pairwise algorithm performs better than SVM-Fisher on the nonhomology benchmark (Liao and Noble, 2002); however, performing SVM-pairwise on the Jaakkola benchmark is not practical due to the $O(m^3)$ running time of the empirical kernel map. Published results indicate that the discrete motif method outperforms SVM-pairwise on the nonhomology benchmark (Ben-Hur and Brutlag, 2003); however, subsequent experiments using a larger E-value threshold show the two methods performing comparably. Finally, although the spectrum kernel does not perform as well as SVM-Fisher (Leslie et al., 2002), its variants (mismatch, gappy, substitution, and wildcard) are comparable to SVM-Fisher on the homology benchmark (Leslie et al., 2003b; Leslie and Kuang, 2003) and (for the mismatch kernel) comparable to SVM-pairwise on the nonhomology benchmark (Leslie et al., 2003a).

3.3 Classification of Genes and Proteins

The recognition of remote homology relationships among proteins is a multiclass classification problem, in which the classes are defined by similarities of protein 3D structure. There are, however, numerous other ways in which proteins and their corresponding genes can be placed into biologically interesting categories. SVMs have been applied to the recognition of several such types of categories.

Functional classification of promoter regions

In addition to the primary amino acid sequence, the functional role of a protein can sometimes be determined by analyzing the DNA sequence that occurs upstream of the corresponding gene. This region contains the switching mechanism that controls when the gene is turned on and off; that is, when and how frequently the gene is translated into a protein sequence. Pavlidis et al. (2001a) demonstrate the application of the Fisher kernel to the problem of classifying genes according to the characteristics of their switching mechanisms. This work thus assumes that genes with similar switching mechanisms are likely to operate in response to the same environmental stimulation and hence are likely to have similar or related functional roles. The Fisher kernel is derived from a motif-based HMM, constructed using Meta-MEME (Grundy et al., 1997). In this model, each motif corresponds to one transcription factor binding site. The method is used successfully to predict membership in two groups of coregulated genes in yeast.

Prediction of protein function from phylogenetic profiles

Protein function can also be determined via sequence comparison with other species. Vert (2002b) describes an elegant kernel function that operates on phylogenetic profiles. A phylogenetic profile is a bit string representation of a protein, in which each bit corresponds to one species for which the complete genome is available (Pellegrini et al., 1999). A bit is 1 if the protein has a close homolog in that species, and 0 otherwise. Thus, the phylogenetic profile captures (part of) the evolutionary history of a given protein. Two proteins that have similar phylogenetic profiles likely have similar functions, via a kind of guilt by association. Say that in every genome that protein A is observed, we also observe protein B, and vice versa.

Given enough complete genomes, the probability of such consistent co-occurrence happening by chance is extremely small.

Vert's phylogenetic profile kernel uses a simple Bayesian tree model to capture the evolutionary relationships among sequences. The tree defines a joint probability distribution, and the corresponding feature space contains one dimension for each possible evolutionary history. The tree kernel is a weighted sum over these histories. Vert demonstrates how to compute this kernel in linear time. For predicting yeast protein functional classes, an SVM trained using the tree kernel performs significantly better than an SVM trained using a simple dot product kernel from the same data set.

Prediction of subcellular localization

Hua and Sun (2001b) use SVMs to perform protein classification with respect to subcellular localization. Here, the label of each protein corresponds to the region of the cell in which it typically resides, including for prokaryotes the cytoplasm, the periplasm, and the exterior of the cell, and for eukaryotes the nucleus, cytoplasm, mitochondria, and the exterior of the cell. In this work, the kernel function is a simple, 20-feature composition kernel. The SVM is shown to produce more accurate classifications than competing methods, including a neural network, a Markov model, and an algorithm specifically designed for this task (Chou and Elrod, 1999).

Distinguishing between benign and pathologic human immunoglobulin light chains

Zavaljevski et al. (2002) describe the application of an SVM to a clinically important, binary protein classification problem. The class of human antibody light chain proteins is large and is implicated in several types of plasma cell diseases. In particular, Zavaljevski, Stevens, and Reifman use SVMs to classify the κ family of human antibody light chains into benign or pathogenic categories. The data set consists of 70 protein sequences. Significantly, these proteins are aligned to one another, in a multiple alignment of width 120. This alignment suggests a simple vectorization, in which each binary feature represents the occurrence of a particular amino acid at a particular position in the alignment. In order to reduce the size of the resulting feature vector, the authors compress the amino acid to an alphabet of size 7, based upon biochemical similarities.

In addition to making accurate predictions, the SVM is used in this context to identify positions in the alignment that are most discriminative with respect to the benign/pathogenic distinction. This identification is accomplished via *selective kernel scaling*, in which a scaling factor is computed for each alignment position and subsequently incorporated into the kernel computation. The scale factors are computed in two different fashions: first, by measuring the degree of conservation in a reference alignment of 14 prototypical human κ light chains, and second, by computing a normalized sensitivity index based upon the output of the SVM. The latter method is iterative and is related to the recursive feature elimination method described below (Guyon et al., 2002). The resulting classifier yields an accuracy of around 80%, measured using leave-one-out cross-validation, which compares favorably with the error rate of human experts. Furthermore, the kernel scaling technique confirms the importance of three previously identified positions in the alignment.

3.4 Prediction along the DNA or Protein Strand

In addition to classifying invidual gene or protein sequences, SVMs have been applied to a number of tasks that involve searching for a particular pattern within a single sequence.

Translation start sites

An early such application involved the recognition of translation start sites in DNA. These positions mark the beginnings of protein-coding genes; hence, an accurate recognizer for this task is an integral part of automatic gene-finding methods. Zien et al. (2000) compare SVMs to a previously described neural network approach to this problem. A fixed-length window of DNA is encoded in redundant binary form (4 bits per base), and the SVM and neural network are trained on the resulting vectors. Using a simple polynomial kernel function, the SVM improves upon the neural network's error rate (15.4% down to 13.2%). Furthermore, Zien et al. demonstrate how to encode prior knowledge about the importance of local interactions along the DNA strand. This locality-improved kernel reduces the error still further to 11.9%.

Splice sites

A similar application is described by Degroeve et al. (2002). Here, rather than recognizing the starts of genes, the SVM learns to recognize the starts of introns. Training and testing are performed on sequences from *Arabidopsis thaliana*. Once again, the data are collected in fixed-length windows and encoded in redundant binary form. The emphasis in this work is feature selection: the authors would like to determine which positions around the splice site provide the most information. They therefore propose a wrapper-based feature selection method, removing features one at a time using the following selection criterion:

$$\underset{m}{\text{argmax}} \left(\sum_{j=1}^{l} y_i \times \left(\sum_{i=1}^{l} \alpha_i y_i k({x_i}^m, {x_j}^m) + b \right) \right), \tag{3.2}$$

where y_j is the label ($+1$ or -1) of example j, b is the SVM bias term, and ${x_j}^m$ is instance x_j with feature m set to its mean value. Three SVM methods (using linear, polynomial, and radial basis function kernels) are compared to a similar method based upon a weight matrix, or naive Bayes classifier. The experiments do not show a clear superiority of any method. Indeed, in no case does feature selection improve performance relative to using the entire window of 100 bases. All methods, not surprisingly, indicate that the most important features are those closest to the splice site, though the methods do not agree on which specific sites are most relevant.

Signal peptide cleavage sites

Signal peptides are molecular bar codes at the end of a protein sequence that help to direct the protein to a particular location in the cell. Vert (2002a) describes an SVM approach to recognizing the position at which a signal peptide is cleaved from the main protein once it reaches its location. This application is thus similar to recognizing translation starts and splice sites, except that it is performed on proteins

rather than DNA sequences. The recognition of signal peptides is important for the development of new drugs.

However, the emphasis in Vert's paper is not the signal peptide application per se, but the description of a general class of kernels derived from probabilistic models. The primary aim is to describe a kernel that defines two objects as "close" when they share rare common substructures. Here, "rarity" is defined with respect to a particular naive Bayes probabilistic model. In general, for any probability density p on X and any set of substructures $V \subset P(S)$, the kernel $k_{p,V}$ is defined as follows:

$$k_{p,V}(x,y) = \frac{p(x)p(y)}{|V|} \sum_{T \epsilon V} \frac{\delta(x_T, y_T)}{p(x_T)}, \tag{3.3}$$

for any two realizations $(x,y) \epsilon A^{2S}$, where $\delta(x_T, y_T)$ is 1 if $x_T = y_T$, 0 otherwise.

Previous research has successfully applied a simple weight matrix model to the recognition of signal peptide cleavage sites (von Heijne, 1986). Accordingly, Vert demonstrates how to derive from a weight matrix a kernel based upon co-occurrences of rare substrings. The resulting SVM yields dramatically better recognition performance than the simple weight matrix approach. For example, at a false-positive rate of 3%, the weight matrix method retrieves 46% of true positives, whereas the SVM method retrieves 68%.

Functional RNAs in prokaryotes

The three previous methods aim at recognizing specific sites in a DNA or protein sequence. In contrast, Carter et al. (2001) have demonstrated the application of SVMs to the problem of recognizing functional RNAs in genomic DNA. With respect to a typical protein-coding gene, RNA is an intermediate between the repository of genetic information (the DNA strand) and the functional product (the protein). Functional RNAs (fRNAs), in contrast, are RNA molecules that have a functional role in the cell and do not code for a protein molecule. Recognizing these RNAs in the DNA strand is difficult because they are typically short and lack the many constraints imposed upon genes that encode proteins. However, because the genes are so short, they can be recognized effectively using a fixed-width sliding window. This is the approach used by Carter, Dubchak, and Holbrook (2001). Each window is encoded using two types of features: compositional features (frequencies of nucleotides and dinucleotides) and structural features (occurrences of six structural motifs associated with fRNAs). The SVM performs well, with leave-one-out error rates of approximately 0.7% to 16.8%, depending upon the organism. However, the SVM is compared to a neural network, which performs slightly better. The comparison is somewhat unfair because the neural network employs a structured network that builds in prior knowledge about the two different classes of inputs, whereas the SVM kernel treats all the inputs uniformly. Thus, this application provides a clear opportunity for engineering an SVM kernel.

Secondary structure

Finally, Hua and Sun (2001a) have demonstrated how to predict the secondary structure at each location along a protein strand. Secondary structure elements fall into three categories: helix, sheet, or coil. Accordingly, this is a multiclass recognition problem, which Hua and Sun address in a straightforward fashion. The

protein sequence is encoded in redundant binary fashion, using an 11-amino acid sliding window. An RBF kernel is used, and three separate SVMs are trained, one per secondary structure element. The final classification of a given amino acid is the label associated with the SVM that assigns the discriminant score that is farthest from zero. The resulting classifier achieves a per-residue accuracy of 73.5% on a standard data set, which is comparable to existing methods based upon neural networks.

3.5 Microarray Gene Expression Analysis

All of the SVM applications described thus far have involved the analysis of biosequences. There is, however, an entirely different type of data, the analysis of which has received considerable attention recently (see Knudsen, 2002 for a useful overview). A *microarray* measures the number of copies of messenger RNA (mRNA) in a given sample of cells. The technology comes in two primary forms. The first technique involves affixing known DNA strands (called *probes*) to a 1 cm^2 glass slide. A fluorescently labeled sample of mRNA is then washed over the slide, and mRNAs that match the probes on the slide bind there. Subsequently, the dye is fluoresced under a microscope, and the intensity at each spot is measured. Each spot on the slide corresponds to a known gene; hence, each spot intensity indirectly indicates how many copies of that gene's mRNA exist in the sample. The second technique is similar to the first, except that the substrate is a silicon chip, and the probes are synthesized photolithographically on the surface of the silicon. Because synthesizing long sequences is expensive, many (between 20 and 40) spots are created for each gene, each spot containing copies of a relatively short (25-nucleotide) probe sequence. Again, the spot intensities are measured via fluorescence. The overall signal for a given gene is computed by combining the measurements from the corresponding spots. Using either technology, the end result is a collection of on the order of 10,000 measurements of gene activity per experiment. The microarray is appealing because of its ability to produce data in a high-throughput fashion. However, the data themselves are quite noisy. Consequently, many research groups have resorted to the use of clustering and pattern recognition techniques to interpret their microarray data.

3.5.1 Gene Classification

The first application of SVMs to microarray data involved the classification of yeast genes into functional categories (Brown et al., 2000). The microarray data were collected from several previous studies (DeRisi et al., 1997; Spellman et al., 1998; Chu et al., 1998) and had previously been analyzed using hierarchical clustering (Eisen et al., 1998). The data set consisted of 79 glass slide microarray experiments, each measuring the activity of approximately 6000 yeast genes. Based upon the previously published analysis, Brown et al. selected five functional classes from

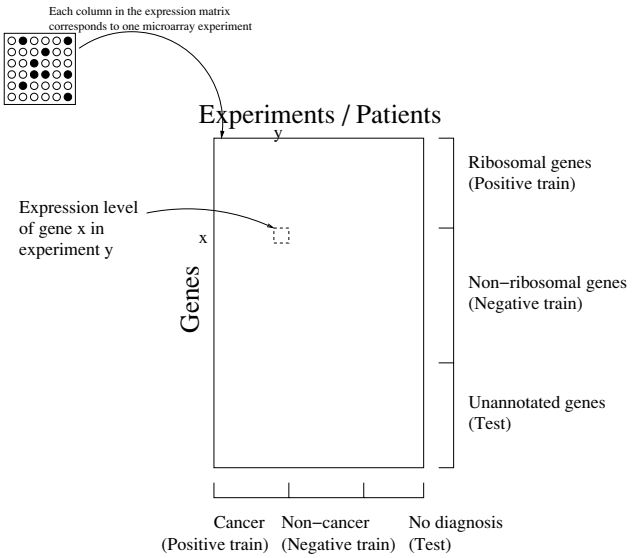


Figure 3.3 Classification tasks with microarray gene expression data. Data from many separate microarray experiments are collected into a single matrix, indexed by gene (row) and experiment (column). Classification can be performed along either dimension of this matrix: gene functional classification along the row dimension or diagnostic or prognostic patient classification along the column dimension.

the MIPS yeast genome database (Mewes et al., 2000)—tricarboxylic acid (TCA) pathway, respiration chain complexes, cytoplasmic ribosomal proteins, proteasome, and histones—and measured the ability of the SVM to recognize members of each of these classes.

The SVM yielded very good performance on this task. In comparison with a collection of traditional machine learning techniques, including Fisher's linear discriminant, C4.5, Parzen windows, and MOC1, the SVM using either an RBF or third-degree polynomial kernel was always the best performing method. Furthermore, the study demonstrated that the SVM can be used both to make predictions for previously unannotated genes and to identify genes in the training set that have been mislabeled. Finally, an analysis of the mistakes made by the SVM shows that the learning algorithm's behavior is in many cases explainable due to noise or known biological anomalies. For example, some of the false-negative examples in the TCA class turn out to be post-translationally modified, meaning that the regulation of these genes occurs after the mRNA has been translated into a protein. In such cases, microarray data cannot be expected to provide useful insights.

3.5.2 Tissue Classification

A more popular application of SVMs to the analysis of microarray data involves transposing the matrix of expression values. Rather than classify each gene according to its profile across multiple experiments, the SVM learns to classify experiments. In this type of study, one experiment typically corresponds to one patient, and the classification label corresponds to a diagnosis. As such, the dimensionality of the problem is unusual: typically, a data set contains tens of experiments (examples) and thousands of genes (features).

Acute myeloid and acute lymphoblastic leukemia

The first application of a supervised learning algorithm to a tissue classification task was performed by Golub et al. (1999). They used a collection of 38 training samples and 34 test samples to train a simple learning algorithm called "weighted voting" to recognize the distinction between two forms of leukemia: Acute myeloid (ALM) and acute lymphoblastic leukemia (ALL). This algorithm uses a feature selection metric, the signal-to-noise ratio $P(j)$, defined as follows:

$$P(j) = \left| \frac{\mu_1(j) - \mu_{-1}(j)}{\sigma_1(j) + \sigma_{-1}(j)} \right|, \tag{3.4}$$

where j is the gene index, μ_i is the mean of class 1 for gene j, μ_{-1} is the mean of class -1 for gene j, and σ_1 and σ_{-1} are the corresponding per-class standard deviations. This metric is closely related to the Fisher criterion score used in Fisher's linear discriminant (Duda and Hart, 1973).

Subsequently, Mukherjee et al. (1999) demonstrated the application of the SVM to this learning task. Because of the high dimensionality of the examples, a linear kernel is applied. Using the signal-to-noise ratio as a feature selection method, Mukherjee et al. improved upon the accuracy of the weighted voting method, reducing the error rate from 6% (2 errrors out of 34) to 0%. Note, however, that the method lacks a principled means of setting a priori the number of selected features. Without feature selection, the SVM makes 1 error, and with the number of features set too low (49 genes out of 7129), the number of errors is again 2.

Mukherjee et al. (1999) also describe a technique for assigning confidence values to the SVM predictions. The method assumes that the probability of a particular class, given a particular example, is approximately equal to the probability of the class given the corresponding SVM discriminant value. Discriminant values are estimated using leave-one-out cross-validation, and their distribution is estimated using an SVM-based, nonparametric density estimation algorithm (Mukherjee and Vapnik, 1999). Introducing confidence levels results in 100% accuracy and between 0 and 4 rejects, depending upon the number of features selected.

Colon cancer

In work carried out concurrently, Moler et al. (2000) describe the application of SVMs to the recognition of colon cancer tissues. The data set consists of 40 colon cancer tumor and 22 normal colon tissues (Alon et al., 1999). This work describes a general, modular framework for the analysis of gene expression data, including generative, Bayesian methods for unsupervised and supervised learning, and the SVM for discriminative supervised learning.

The SVM is used in two ways, first to identify outlier or mislabeled training examples. An unsupervised naive Bayes class discovery method identifies four classes in the entire data set, and a multiclass (one-vs.-all) linear SVM is trained and tested on all 1988 genes via leave-one-out cross-validation on these four classes. The authors claim that examples that are always support vectors are of particular interest: if these examples are consistently assigned to their labeled class, then they are considered unambiguous; if the examples are inconsistently assigned, then they may be mislabeled. Overall, the results suggest that the data can be divided into three subtypes (clearly tumor, mainly nontumor and heterogeneous), which the authors claim may be of clinical significance.

The second SVM application involves recognition of tumor vs. nontumor tissues. A feature selection metric, the naive Bayes relevance (NBR) score, is proposed, which is based on the probability of a class given the observed value of the feature, under a Gaussian model. The performance of the SVM using various numbers of selected genes is compared to the performance of a naive Bayes classifier using the same genes. In every case, the SVM performs better than the naive Bayes.

Ovarian cancer

In a similar set of experiments, Furey et al. (2000) apply linear SVMs with feature selection to three cancer data sets. The first data set consists of 31 tissue samples, including cancerous ovarian, normal ovarian, and normal nonovarian tissue. The other sets are the AML/ALL and colon cancer sets mentioned above. Following Golub et al. (1999), the signal-to-noise ratio is used to select genes for input to the classifier. The SVM successfully identifies a mislabeled sample in the ovarian set, and is able to produce a perfect classification. However, this classification is fragile with respect to the SVM parameter settings (softness of the margin and number of genes selected for input). Overall, the SVM provides reasonably good performance across multiple data sets, although the experiments also demonstrate that several perceptron-based algorithms perform similarly.

Soft tissue sarcoma

Segal et al. (2003b) use the SVM to develop a genome-based classification scheme for clear cell sarcoma. This type of tumor displays characteristics of both soft tissue sarcoma and melanoma. A linear SVM is trained to recognize the distinction between melanoma and soft tissue sarcoma, using 256 genes selected via a t-test. In a leave-one-out setting, the classifier correctly classifies 75 out of 76 examples. Subsequently, the trained classifier is applied to five previously unseen clear cell sarcoma examples, and places all five within the melanoma class. Thus, SVM analysis of gene expression profiles supports the classification of clear cell sarcoma as a distinct genomic subtype of melanoma.

In related work, Segal et al. (2003a) use SVMs to investigate the complex histopathology of adult soft tissue sarcomas. Here, the data set consists of 51 samples that have been classified by pathologists into nine histologic subtypes. The SVM, again using a t-test for feature selection, successfully recognizes the four subtypes for which molecular phenotypes are already known. Among the remaining samples, a combination of SVMs and hierarchical clustering uncovers a well-separated subset of the malignant fibrous hystiocytoma subtype, which is a particularly controversial subtype.

Recursive feature elimination

All of the methods described thus far for cancer classification rely upon a score (either the signal-to-noise ratio, NBR score, or t-test) for selecting which genes to give to the SVM classifier. A significant drawback to these scores is that they treat each gene independently, thereby ignoring any significant gene-gene correlations that may occur in the data. Guyon et al. (2002) propose an SVM-based learning method, called SVM recursive feature elimination (SVM-RFE) that addresses this issue. The motivating idea is that the orientation of the separating hyperplane found by the SVM can be used to select informative features: if the plane is orthogonal to a particular feature dimension, then that feature is informative, and vice versa. Specifically, given an SVM with weight vector $\vec{w} = \sum_k \alpha_k y_k \vec{x}_k$, the ranking criterion for feature i is $c_i = (w_i)^2$. This criterion suggests the following wrapper-based learning method:

1. Initialize the data set to contain all features.
2. Train an SVM on the data set.
3. Rank features according to the criterion c.
4. Eliminate the lowest-ranked feature.
5. If more than one feature remains, return to step 2.

In practice, the algorithm is sped up by removing half of the features in step 4.

The SVM-RFE algorithm is tested on the AML/ALL and colon cancer data sets. For the leukemia data sets, SVM-RFE identifies two genes that together yield zero leave-one-out error. In addition, several other classification algorithms, including the weighted voting algorithm, are applied to the data using the genes selected by SVM-RFE. The results show that the selection of genes is more important than the particular learning algorithm employed.

Gene selection

SVM-RFE has the dual goal of producing a good discriminator and reducing the number of genes to a manageable number. If we eliminate the first goal, then we are left with the problem of gene ranking. Identifying genes that exhibit predictive power in discriminating between two classes of samples is often the primary goal of a microarray study. Su et al. (2003) describe a tool called RankGene that produces gene rankings. One of the ranking metrics available in RankGene is the discriminant of a one-dimensional SVM trained on a given gene.

Multi-class classification

Many tissue classification analyses have been hampered somewhat by the dearth of useful, publically available gene expression data sets. Yeang et al. (2001) addressed this issue by producing a data set of 190 samples from 14 tumor classes. This collection was later expanded to include 308 samples, including 90 normal tissue samples (Ramaswamy et al., 2001). The initial study compares six different supervised learning methods: weighted voting, k-nearest neighbor, and the SVM, each trained for multiclass classification using both a one-vs.-all and an all-pairs approach. The signal-to-noise ratio is used for feature selection for the weighted voting and k-nearest neighbor, but feature selection is not applied to the SVM algorithm. Nonetheless, the one-vs.-all SVM algorithm trained using all genes performs better than the all-pairs SVM and better than any of the other classifiers trained using 20,

40, 50, 100, or 200 genes. The second, larger study does apply SVM-RFE, but the best performance is again obtained by the one-vs.-all SVM trained using all genes.

At this stage, the diagnosis and prognosis of cancer using microarray assays is still the subject of both hype and controversy. For example, an important and occasionally overlooked characteristic of these studies is the risk of introducing selection bias by choosing discriminative genes prior to performing cross-validation. Ambroise and McLachlan (2002) demonstrate that this bias occurs in several published studies, including the SVM-RFE analysis performed by Guyon et al. (2002). A reanalysis of the colon cancer and leukemia data sets, taking into account the selection bias, shows that feature selection does not actually improve discrimination performance relative to an SVM trained from all of the genes. This result agrees with the results reported by Ramaswamy et al. (2001). Despite the controversy, a microarray assay is already in clinical trial in the Netherlands for determining whether breast cancer patients will receive adjuvant treatment (chemotherapy, tamoxifen, or radiation) after surgery (Schubert, 2003), and at least five additional clinical trials are set to begin soon (Branca, 2003). Ironically, the Dutch microarray screen is based, in part, on a (non-SVM-based) microarray analysis (van't Veer et al., 2002) that has been demonstrated independently to suffer from selection bias (Tibshirani and Efron, 2002).

3.6 Data Fusion

Now that the human genome is more or less completely sequenced, more interest is being paid to the problem of data fusion, of integrating heterogeneous biological data. For example, for a given gene we might know the protein it encodes, that protein's similarity to other proteins, the mRNA expression levels associated with the given gene under hundreds of experimental conditions, the occurrences of known or inferred transcription factor binding sites in the upstream region of that gene, and the identities of many of the proteins that interact with the given gene's protein product. Each of these distinct data types provides one view of the molecular machinery of the cell.

Summing kernel matrices

Several efforts have been made to perform biological data fusion in the context of SVM learning. Pavlidis et al. (2001b, 2002) trained SVMs to recognize functional categories of yeast genes, using a combination of microarray gene expression data and phylogenetic profiles. In this case, both types of data are fixed-length, real-valued vectors, so a standard third-degree polynomial kernel is employed. Pavlidis et al. compare three different techniques for combining these two types of data (see figure 3.4): early integration, in which the two vectors are simply concatenated; intermediate integration, in which two kernels are computed separately and then added; and late integration, in which two SVMs are trained and their discriminant scores are added. Intermediate integration provides the best results, presumably because it trades off making too many independence assumptions (in late integration) vs. allowing too many dependencies (in early integration). The authors also

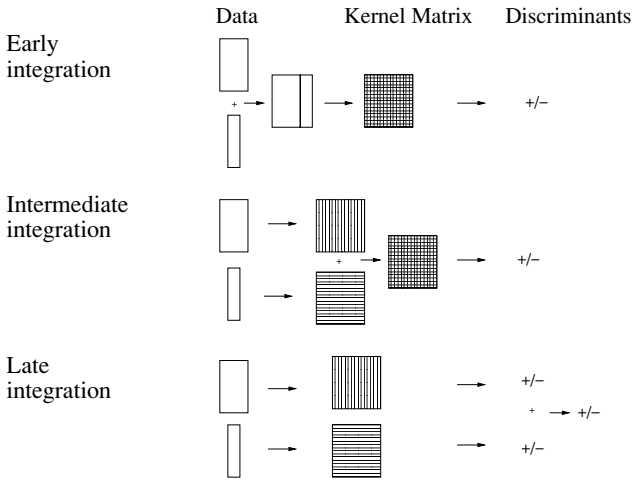


Figure 3.4 Three methods for learning from heterogeneous data with a support vector machine. In early integration, the two types of data are concatenated to form a single set of input vectors. In intermediate integration, the kernel values are computed separately for each data set and then summed. In late integration, one SVM is trained on each data type, and the resulting discriminant values are summed.

present some heuristic techniques for choosing scaling factors to be applied to each kernel function.

Kernel canonical correlation analysis

Another form of data fusion was performed by Vert and Kanehisa (2003b). This approach integrates gene expression profiles with prior knowledge of a metabolic network. The network represents pathways of proteins that operate upon one another in the cell. Vert and Kanehisa hypothesize that gene expression patterns that are well measured (i.e., that correspond to actual biological events, such as the activation or inhibition of a particular pathway) are more likely to be shared by genes that are close to one another in the metabolic network. Accordingly, the expression data and the metabolic network are encoded into kernel functions, and these functions are combined in feature space using canonical correlation analysis (Bach and Jordan, 2002). Using yeast functional categories, an SVM trained from the combined kernel performs significantly better than an SVM trained only on expression data.

Semidefinite programming

Recently, Lanckriet et al. (2004) have described a new method for integrating heterogeneous genomic data. Similar to the work of Pavlidis et al. (2001b, 2002), the method involves summing a collection of kernel matrices, one per data set. In this case, however, each matrix is weighted, and Lanckriet et al. demonstrate how to optimize simultaneously the hyperplane selection and the selection of kernel weights. The result is a convex optimization problem that can be solved with semidefinite

programming techniques. The paper demonstrates the utility of these techniques by solving the problem of predicting membrane proteins from heterogeneous data, including amino acid sequences, hydropathy profiles, gene expression data, and known protein-protein interactions. An SVM algorithm trained from all of these data performs significantly better than the SVM trained on any single type of data and better than existing algorithms for membrane protein classification. Furthermore, the algorithm is robust to noise: when a randomly generated data set is included in the mix, the corresponding kernel function receives a weight close to zero, and the overall performance of the discriminator is essentially unchanged.

Expectation-maximization for missing data

Finally, Tsuda et al. (2003) describe a different type of data fusion algorithm. This approach applies a variant of the expectation-maximization algorithm (Dempster et al., 1977) to the problem of infering missing entries in a kernel matrix by using a second kernel matrix from an alternative data source. The method is demonstrated using two kernel matrices derived from two different types of bacterial protein sequences (16S rRNA and gyrase subunit B). The quality of the resulting matrix is evaluated by using the matrix to perform unsupervised learning. The results suggest that this approach may prove useful in a supervised context as well.

3.7 Other Applications

Cancer classification from methylation data

Model et al. (2001) describe a classification task very similar to the cancer classification tasks described above. The primary difference is that, in this case, the data come from a methylation assay, rather than a microarray gene expression profile. Methylation is a molecular modification of DNA, in which a methyl group is added to the nucleotide cytosine. Methylation patterns in the upstream regions of genes are thought to be a major factor in gene regulation. Model et al. have developed a high-throughput method for collecting methylation data, and have used it to collect data from leukemia patients, 17 with AML and 8 with ALL. Each methylation pattern contains measurements from 81 positions along the DNA strand. The computational experiment consists of training a linear SVM to differentiate between AML and ALL. Many feature selection methods are employed, including principle component analysis, the signal-to-noise ratio, the Fisher criterion score, the t-test, and a method called backward elimination. The last is essentially identical to the SVM-RFE algorithm of Guyon et al. (2002) and appears to have been invented independently. For this task, SVM-RFE does not outperform the linear feature selection methods. Instead, feature selection via the Fisher criterion score provides the best results.

Prediction of developmental age of *Drosophila* embryos

Perhaps one of the most unusual learning tasks is described by Myasnikova et al. (2002). They are interested in characterizing gene expression changes in *Drosophila* during development, and they measure these changes in a gene-specific fashion using fluorescent dyes and light microscopy of *Drosophila* embryos. In order to precisely and efficiently analyze the resulting data, they need an automatic method for

determining the developmental age of a *Drosophila* embryo. To solve this problem, they use support vector regression (Drucker et al., 1997).

The data set consists of 103 embyros for which the precise developmental age is known. A microphotograph of each embryo is reduced, using previously developed techniques, to a table of values in which each row corresponds to a single cell, and columns represent the x and y coordinates of the nucleus and the expression levels of three genes in that cell. The resulting regression estimator appears to perform well, though no comparison to other algorithms is performed. The authors also demonstrate how factor analysis, performed on a data set of labeled and unlabeled examples, can be used to reduce the number of features to 3, thereby significantly increasing the speed of the regression estimation with no accompanying loss in accuracy.

Prediction of protein-protein interactions

Bock and Gough (2001) apply SVMs to the very important problem of predicting protein-protein interactions. This task fits cleanly into a binary discrimination framework: given a pair of proteins, the SVM predicts whether or not they interact. A critical question is how best to represent the protein pairs, and Bock and Gough derive a set of features characterizing the charge, hydrophobicity, and surface tension at each amino acid in a given protein. Protein pairs are represented simply as the concatenation of the corresponding vectors. The SVM performs impressively, achieving an accuracy better than 80% in a cross-validated test. However, the experiment suffers from a significant flaw: the negative examples are generated randomly. Therefore, it is not clear whether the SVM is learning to differentiate between interating and noninteracting protein pairs, or to differentiate between real and simulated protein pairs. Further experiments are needed.

Indeed, a subsequent experiment addressing this same problem shows the SVM performing comparably to a simple Bayesian technique (Gomez et al., 2003). The SVM's drawback, in this work, is that the training set is extremely large, and the SVM is consequently quite slow relative to the simpler method.

Peptide identification from mass spectrometry data

In tandem mass spectrometry, a sample of unknown proteins is enzymatically digested into relatively short strings of amino acids, called peptides. These peptides are size-selected via mass spectrometry, fragmented via ionization, and the fragments are measured by a second mass spectrometer. The final spectrum contains peaks corresponding to all or most of the substrings in a single peptide. It is possible to infer the original peptide from the spectrum, using only the known masses of the amino acids. In practice, however, performing this task de novo is too difficult, and successful algorithms like SEQUEST (Eng et al., 1994) use an auxiliary database of known proteins. SEQUEST performs a simulation of tandem mass spectrometry on each peptide in the database, searching for a theoretical spectrum that matches the observed spectrum.

Anderson et al. (2003) apply the SVM to the problem of interpreting SEQUEST output. The algorithm produces a large number of false positives, and the SVM's task is to learn to differentiate true from false positives. Thus, the input to the classifier is a pair of spectra—observed and theoretical—and the output is a prediction—true positive or false positive. The input spectra are represented by

a collection of 13 parameters, reflecting the quality of the observed spectrum, the similarity of the observed and theoretical spectrum, and the difference between this match and the next-best match found by SEQUEST. The SVM uses a quadratic kernel function, and achieves error rates of 7% to 14%, depending upon the quality of the instrument used to generate the data. This compares favorably with QScore, a previously published, non learning-based probabilistic algorithm that addresses the same task (Moore et al., 2002). The same SVM has been subsequently used to construct an assay of the ubiquitin system (Gururaja et al., 2003), which is responsible for targeting proteins for degradation.

3.8 Discussion

Clearly, the application of SVM learning in computational biology is a popular and successful undertaking. The appeal of this approach is due in part to the power of the SVM algorithm, and in part to the flexibility of the kernel approach to representing data. In particular, the kernel framework accommodates in a straightforward fashion many different types of data—vectors, strings, trees, graphs, and so on—that are common in biology. Also, kernels provide an easy way to incorporate biological knowledge and unlabeled data into the learning algorithm. A kernel matrix derived from a particular experiment can thus summarize the relevant features of the primary data, encapsulate biological knowledge, and serve as input to a wide variety of subsequent data analyses.

Finally, as an avenue for future research, the kernel approach to learning allows for a principled way to perform *transduction* (Gammerman et al., 1998). A transductive learning task is one in which the (unlabeled) test data are available to the algorithm a priori. In the post-genomic era, many computational biology tasks are transductive because the entire complement of genes or proteins in a given organism is known. Exploiting the finite nature of these learning tasks may lead to improved recognition performance in many biological domains.

II KERNELS FOR BIOLOGICAL DATA

4 Inexact Matching String Kernels for Protein Classification

Christina Leslie
Rui Kuang
Eleazar Eskin

We review several families of string kernels designed in particular for use with support vector machines (SVMs) for classification of protein sequence data, mismatch kernels, and three newer related models: restricted gappy kernels, substitution kernels, and wildcard kernels. These kernels are based on feature spaces indexed by l-length subsequences or "l-mers" from the string alphabet Σ (or the alphabet augmented by a wildcard character) and incorporate various notions of inexact string matching. All the kernels can be computed efficiently with a recursive function based on a trie-based data structure, with computation time that scales linearly with sequence length: the kernel value $k(x, y)$ can be computed in $O(c_k(|x| + |y|))$ time, where the constant c_k depends on the parameters of the kernel. In particular, for the newer kernel models, the constant c_k is independent of the size $|\Sigma|$ of the alphabet, which significantly speeds up computation time. Moreover, when used with an SVM classifier, all the kernel models allow linear time prediction on test sequences. Finally, we report protein classification experiments on a benchmark SCOP (structural classification of proteins) data set, where we show that inexact matching kernels achieve SVM classification performance comparable to the best competing methods.

4.1 Introduction

A central problem in computational biology is the classification of protein sequences into functional and structural families based on sequence homology. Approaches based on pairwise alignment of sequences (Waterman et al., 1991; Altschul et al., 1990, 1997), profiles for protein families (Gribskov et al., 1987), consensus patterns

using motifs (Bairoch, 1995; Attwood et al., 1998), and hidden Markov models (HMM's; Krogh et al., 1994; Eddy, 1995; Baldi et al., 1994) have all been used for this problem. Recent research suggests that the best-performing methods are discriminative: protein sequences are seen as a set of labeled examples — positive if they are in the family and negative otherwise — and a learning algorithm attempts to *learn* a decision boundary between the different classes.

However, in order to use discriminative classifiers like SVMs (Vapnik, 1998) for this problem, one must choose a suitable vector representation of protein sequence data. Protein sequences can be viewed as variable-length strings from the alphabet of 20 amino acids, typically several hundred characters long; each sequence x must be represented in a vector space by a feature mapping $x \mapsto \Phi(x)$. If we use kernel methods such as SVMs, which only require inner products $k(x, y) = \langle \Phi(x), \Phi(y) \rangle$ for training and testing, then we can implicitly accomplish the above mapping using a kernel for sequence data.

In the recent Fisher kernel approach (Jaakkola et al., 2000), one derives the feature representation for an SVM classifier from a generative model for a protein family. Here, one first builds a profile HMM for the positive training sequences, defining a log likelihood function $\log P(x|\theta)$ for any protein sequence x. Then the gradient vector $\nabla_\theta \log P(x|\theta)|_{\theta=\theta_0}$, where θ_0 is the maximum likelihood estimate for model parameters, defines an explicit vector of features, called Fisher scores, for x. This representation gives strong classification results, but the Fisher scores must be computed by an $O(|x|^2)$ forward-backward algorithm, making the kernel evaluations somewhat slow.

The SVM-pairwise method (Liao and Noble, 2002) provides an alternative vectorization, based on pairwise alignment scores like Smith-Waterman (SW) (Waterman et al., 1991) or BLAST (Altschul et al., 1990). These alignment-based scores do not define a positive definite kernel; however, one can use a feature representation based on the empirical kernel map $\Phi(x) = \langle d(x_1, x), \ldots, d(x_m, x) \rangle$, where $d(x, y)$ is the pairwise score (or corresponding E-value) between x and y and x_i, $i = 1 \ldots m$, are the training sequences. Using SW E-values in this fashion gives strong classification performance (Liao and Noble, 2002). Note, however, that the method is slow, both because computing each SW score is $O(|x|^2)$ and because computing each empirically mapped kernel value is $O(m)$.

In this chapter, we discuss a family of representations and kernels based on the *spectrum* of a sequence. The l-spectrum of a sequence is the set of all l-length contiguous subsequences, or l-mers, that it contains. Here, we define kernels whose implicit feature map is indexed by the set of all l-mers from the alphabet Σ of amino acids: $\Phi(x) = (\phi_\alpha(x))_{\alpha \in \Sigma^l}$. If $\phi_\alpha(x)$ is simply a count of the number of exact occurrences of l-mer α in x, we obtain the spectrum feature map of Leslie et al. (2002). However, by incorporating some notion of inexact string matching, we obtain a much improved classification performance. In the (l, m)-mismatch feature map of Leslie et al. (2003b), the coordinate function $\phi_\alpha(x)$ represents a count of all occurrences of α, allowing up to m mismatches, in x. Using the same feature space indexed by l-mers from Σ (or Σ augmented with a wildcard character), one

can obtain other models for counting inexact matches (Leslie and Kuang, 2003), based on counts with restricted gaps, probabilistic substitutions, or wildcards. All these inexact string matching representations define true string kernels (rather than only giving feature maps): one does not have to explicitly compute and store the feature vectors, but rather one can directly and efficiently compute the kernel value $k(x, y)$ using a recursive function based on traversal of a trie data structure. The complexity of calculating $k(x, y)$ is linear in the length of the input sequences, that is, $O(c_k(|x| + |y|))$, where c_k is a parameter-dependent constant. For the mismatch kernel, $c_k = l^{m+1}|\Sigma|^m$, while the other inexact matching string kernels have the advantage that c_k is independent of alphabet size Σ, leading to faster compute times.

For useful kernel parameters, the mismatch kernel and its relatives – the gappy, substitution, and wildcard kernels – are fast, scalable, and give impressive performance. We summarize our experimental results on a benchmark SCOP data set for remote protein homology detection, where we show that both the mismatch kernel and the faster inexact matching variations kernels achieve SVM classification performance comparable to the Fisher-SVM approach, the best previous method for this data set. We do not include a comparison with the SVM-pairwise method here, since the kernel computations for this method are expensive for this large SCOP data set. However, in the longer version of Leslie et al. (2003b), we show that the mismatch kernel used with an SVM classifier is competitive with SVM-pairwise on the smaller SCOP benchmark presented in Liao and Noble (2002).

There has been much recent work on string kernels in applications like text categorization and speech recognition, as well as protein classification. Previous work includes convolution kernels (Haussler, 1999), dynamic alignment kernels (Watkins, 2000), and the gappy n-gram kernel developed for text classification (Lodhi et al., 2002). A practical disadvantage of these string kernels is their computational expense. Most of the kernels rely on dynamic programming algorithms for which the computation of each kernel value $k(x, y)$ is quadratic in the length of the input sequences x and y, that is, $O(|x||y|)$ with constant factor that depends on the parameters of the kernel. The exact matching spectrum kernel has been extended to compute the weighted sum of l-spectrum kernels for different l by using suffix trees and suffix links (Vishwanathan and Smola, 2003), allowing for a compute time of $O(|x|+|y|)$; this result eliminates the linear dependence on l in the spectrum kernel, whose complexity is $O(l(|x| + |y|))$. However, the kernels we describe in this paper – mismatch, gappy, substitution, and wildcard kernels – use notions of inexact string matching for improved representation and classification performance while maintaining linear time scaling.

4.2 Definitions of Feature Maps and String Kernels

Below, we give the definition of mismatch kernels (Leslie et al., 2003b) and three newer variations which incorporate other notions of inexact string matching: re-

stricted gappy kernels, substitution kernels, and wildcard kernels (Leslie and Kuang, 2003). In each case, the kernel is defined via an explicit feature map from the space of all finite sequences from an alphabet Σ to a vector space indexed by the set of l-length subsequences from alphabet Σ or, in the case of wildcard kernels, Σ augmented by a wildcard character.

4.2.1 Spectrum and Mismatch Kernels

Spectrum feature map

For a very simple feature map into the $|\Sigma|^l$-dimensional vector space indexed by l-mers, we can assign to a sequence x a vector given as follows: for each l-mer α, the coordinate indexed by α will be the number of times α occurs in x. This gives the l-spectrum feature map defined in Leslie et al. (2002):

$$\Phi_l^{\text{Spectrum}}(x) = (\phi_\alpha(x))_{\alpha \in \Sigma^l}$$

where $\phi_\alpha(x) = \#$ occurrences of α in x. Now the l-spectrum kernel $k_l^{\text{Spectrum}}(x, y)$ for two sequences x and y is obtained by taking the inner product in feature space. This kernel gives a simple notion of sequence similarity: two sequences will have a large l-spectrum kernel value if they share many of the same l-mers. One can extend this idea by taking weighted sums of l-spectrum kernels for different values of l, as described in Vishwanathan and Smola (2003).

For a more biologically reasonable representation, we want to allow some degree of inexact matching in our feature map – that is, we want the kernel to measure the number of *similar* l-mers shared by two sequences – which we accomplish with mismatches. In Leslie et al. (2003b), we define the (l, m)-mismatch kernel via a feature map $\Phi_{(l,m)}^{\text{Mismatch}}$, again mapping to the vector space indexed by the set of l-mers from Σ. For a fixed l-mer $\alpha = a_1 a_2 \dots a_l$, with each a_i a character in Σ, the (l, m)-neighborhood generated by α is the set of all l-length sequences β from Σ that differ from α by at most m mismatches. We denote this set by $N_{(l,m)}(\alpha)$ and call it the "mismatch neighborhood" around α.

Mismatch feature map

For an l-mer α, the feature map is defined as

$$\Phi_{(l,m)}^{\text{Mismatch}}(\alpha) = (\phi_\beta(\alpha))_{\beta \in \Sigma^l}$$

where $\phi_\beta(\alpha) = 1$ if β belongs to $N_{(l,m)}(\alpha)$, and $\phi_\beta(\alpha) = 0$ otherwise. For a sequence x of any length, we extend the map additively by summing the feature vectors for all the l-mers in x:

$$\Phi_{(l,m)}^{\text{Mismatch}}(x) = \sum_{l\text{-mers } \alpha \text{ in } x} \Phi_{(l,m)}^{\text{Mismatch}}(\alpha)$$

Each instance of an l-mer contributes to all coordinates in its mismatch neighborhood, and the β-coordinate of $\Phi_{(l,m)}^{\text{Mismatch}}(x)$ is just a count of all instances of the

l-mer β occurring with up to m mismatches in x. The (l, m)-mismatch kernel $k_{(l,m)}$ is then given by the inner product of feature vectors:

$$k^{\text{Mismatch}}_{(l,m)}(x, y) = \langle \Phi^{\text{Mismatch}}_{(l,m)}(x), \Phi^{\text{Mismatch}}_{(l,m)}(y) \rangle.$$

For $m = 0$, we obtain the l-spectrum (Leslie et al., 2002).

4.2.2 Restricted Gappy Kernels

For the (g, l)-gappy string kernel (Leslie and Kuang, 2003), we use the same $|\Sigma|^l$-dimensional feature space, indexed by the set of l-mers from Σ, but we define our feature map based on gappy matches of g-mers (with $g > l$) to l-mer features. For a fixed g-mer $\alpha = a_1 a_2 \dots a_g$ (each $a_i \in \Sigma$), let $G_{(g,l)}(\alpha)$ be the set of all l-mers that can be obtained from l-length subsequences occurring in α (with up to $g - l$ gaps). Here, an l-length subsequence refers to an ordered sequence of characters from α at positions $1 \le i_1 < i_2 < \ldots < i_l \le g$, and an l-mer is obtained by concatenating these characters: $a_{i_1} a_{i_2} \dots a_{i_l}$. Then we define the gappy feature map on α as

Gappy feature map

$$\Phi^{\text{Gap}}_{(g,l)}(\alpha) = (\phi_\beta(\alpha))_{\beta \in \Sigma^l}$$

where $\phi_\beta(\alpha) = 1$ if β belongs to $G_{(g,l)}(\alpha)$, and $\phi_\beta(\alpha) = 0$ otherwise. In other words, each instance g-mer contributes to the set of l-mer features that occur (in at least one way) as subsequences with up to $g - l$ gaps in the g-mer. Now we extend the feature map to arbitrary finite sequences x by summing the feature vectors for all the g-mers in x:

$$\Phi^{\text{Gap}}_{(g,l)}(x) = \sum_{\text{g-mers } \alpha \in x} \Phi^{\text{Gap}}_{g,l}(\alpha)$$

The kernel $k^{\text{Gap}}_{(g,l)}(x, y)$ is defined as before by taking the inner product of feature vectors for x and y.

Alternatively, given an instance g-mer, we may wish to count the number of occurrences of each l-length subsequence and weight each occurrence by the number of gaps. Following Lodhi et al. (2002), we can define for g-mer α and l-mer feature $\beta = b_1 b_2 \dots b_l$ the weighting

$$\phi^\lambda_\beta(\alpha) = \frac{1}{\lambda^l} \sum_{\substack{1 \le i_1 < i_2 < \ldots < i_l \le g \\ a_{i_j} = b_j \text{ for } j = 1 \ldots l}} \lambda^{i_l - i_1 + 1}$$

where the multiplicative factor satisfies $0 < \lambda \le 1$. We can then obtain a weighted version of the gappy kernel $k^{\text{Weighted Gap}}_{(g,l,\lambda)}$ from the feature map:

$$\Phi^{\text{Weighted Gap}}_{(g,l,\lambda)}(x) = \sum_{\text{g-mers } \alpha \in x} (\phi^\lambda_\beta(\alpha))_{\beta \in \Sigma^l}$$

This feature map is related to the gappy l-gram kernel defined in Lodhi et al. (2002) but enforces the restriction that only those l-character subsequences that

occur with at most $g - l$ gaps (rather than all gappy occurrences) contribute to the corresponding l-mer feature. Note that for our kernel, a restricted gappy l-mer instance is counted in all (overlapping) g-mers that contain it, whereas in Lodhi et al. (2002), a gappy l-mer instance is only counted once. If we wish to approximate the gappy l-gram kernel, we can define a small variation of our restricted gappy kernel where one only counts a gappy l-mer instance if its first character occurs in the first position of a g-mer window (Leslie and Kuang, 2003).This modified feature map now gives a "truncation" of the usual gappy l-gram kernel.

In section 4.3, we show that our restricted gappy kernel has $O(c(g,l)(|x| + |y|))$ computation time, where constant $c(g,l)$ depends on the size of g and l, while the original gappy l-gram kernel has complexity $O(l|x||y|)$. We will see that for reasonable choices of g and l, we obtain much faster computation time, while in experimental results reported in section 4.5, we still obtain good classification performance.

4.2.3 Substitution Kernels

The substitution kernel (Leslie and Kuang, 2003) is similar to the mismatch kernel, except that we replace the combinatorial definition of a mismatch neighborhood with a similarity neighborhood based on a probabilistic model of character substitutions. In computational biology, it is standard to use pairwise scores $s(a,b)$, derived from estimated evolutionary substitution probabilities, from a substitution matrix (Henikoff and Henikoff, 1992; Schwartz and Dayhoff, 1978; Altschul et al., 1990). One system derives the scores $s(a,b)$ from estimates of conditional substitution probabilities $P(a|b) = p(a,b)/q(b)$, where $p(a,b)$ is the probability that a and b co-occur in an alignment of closely related proteins, $q(a)$ is the background frequency of amino acid a, and $P(a|b)$ represents the probability of a mutation into a during a fixed evolutionary time interval given that the ancestor amino acid was b. We define the mutation neighborhood $M_{(l,\sigma)}(\alpha)$ of an l-mer $\alpha = a_1 a_2 \dots a_l$ as follows:

$$M_{(l,\sigma)}(\alpha) = \{\beta = b_1 b_2 \dots b_l \in \Sigma^l : -\sum_{i}^{l} \log P(a_i|b_i) < \sigma\}$$

We can choose $\sigma = \sigma(N)$ such that $\max_{\alpha \in \Sigma^l} |M_{(l,\sigma)}(\alpha)| < N$, so we have theoretical control over the maximum size of the mutation neighborhoods. In practice, choosing σ to allow an appropriate amount of mutation while restricting neighborhood size may require experimentation and cross-validation.

Substitution feature map

Now we define the substitution feature map analogously to the mismatch feature map:

$$\Phi^{\text{Sub}}_{(l,\sigma)}(x) = \sum_{l\text{-mers } \alpha \text{ in } x} (\phi_\beta(\alpha))_{\beta \in \Sigma^l}$$

where $\phi_\beta(\alpha) = 1$ if β belongs to the mutation neighborhood $M_{(l,\sigma)}(\alpha)$, and $\phi_\beta(\alpha) = 0$ otherwise.

4.2.4 Wildcard Kernels

Finally, we can define wildcard kernels (Leslie and Kuang, 2003) by augmenting the alphabet Σ with a wildcard character denoted by $*$. Here, we map to a feature space indexed by the set $\mathcal{W}$ of l-length subsequences from $\Sigma \cup \{*\}$ having at most m occurrences of the character $*$. The feature space has dimension $\sum_{i=0}^{m} \binom{l}{i} |\Sigma|^{l-i}$.

An l-mer α matches a subsequence β in $\mathcal{W}$ if all nonwildcard entries of β are equal to the corresponding entries of α (wildcards match all characters). The wildcard feature map is given by

Wildcard feature map

$$\Phi^{\text{Wildcard}}_{(l,m,\lambda)}(x) = \sum_{l\text{-mers } \alpha \text{ in } x} (\phi_\beta(\alpha))_{\beta \in \mathcal{W}}$$

where $\phi_\beta(\alpha) = \lambda^j$ if α matches pattern β containing j wildcard characters, $\phi_\beta(\alpha) = 0$ if α does not match β, and $0 < \lambda \leq 1$.

Other variations of the wildcard idea, including specialized weightings and use of groupings of related characters, are described in Eskin et al. (2003).

4.3 Efficient Computation

All the kernels we define above can be efficiently computed using a trie data structure, similar to the mismatch tree approach first presented in Leslie et al. (2003b). We review the mismatch kernel and gappy kernel (Leslie and Kuang, 2003) computation in most detail, since the other kernels are easier adaptations of the mismatch kernel computation. For simplicity, we explain how to compute a single kernel value $k(x, y)$ for a pair of input sequences; computation of the full kernel matrix in one traversal of the data structure is a straightforward extension.

4.3.1 (l, m)-Mismatch Kernel Computation

We use a trie-based data structure, called a mismatch tree, to represent the feature space (the set of all l-mers) and to organize a lexical traversal of all instances of l-mers that occur (with mismatches) in the data. The entire kernel matrix can be computed in one traversal of the tree, though for simplicity we discuss complexity for a single kernel evaluation $k(x, y)$. More details can be found in Leslie et al. (2003b) and the forthcoming longer version of that paper.

Mismatch tree

An (l, m)-mismatch tree is a rooted tree of depth l where each internal node has 20 (more generally, $|\Sigma|$) branches, each labeled with an amino acid (symbol from Σ). A leaf node represents a fixed l-mer in our feature space, obtained by

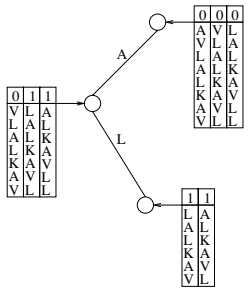


Figure 4.1 An $(8,1)$-mismatch tree for a sequence `AVLALKAVLL` used for computing the kernel matrix with l-mers of length 8 allowing 1 mismatch. The path from the root to the node is the "prefix" of a particular l-mer feature. The leaf node stores the number of mismatches between the prefix of an l-mer instance and the prefix of a feature and a pointer to the tail of the l-mer. The figure shows the tree after expanding the path AL.

concatenating the branch symbols along the path from root to leaf. An internal node represents the prefix for those l-mer features that are its descendants in the tree. We perform a depth-first traversal of the data structure and store, at a node of depth d, pointers to all substrings ("l-mer instances") from the sample data set whose d-length prefixes are within m mismatches from the d-length prefix represented by the path down from the root; this set of substrings represents the valid instances of the d-length prefix in the data. We also keep track, for each valid instance, of how many mismatches it has when compared to the prefix. Note that the set of valid instances for a node is a subset of the set of valid instances for the parent of the node; when we descend from a parent to a child, each instance is either passed down (with 0 or 1 additional mismatch), or it is eliminated because it has exceeded m mismatches. When we encounter a node with an empty list of pointers (no valid occurrences of the current prefix), we do not need to search below it in the tree.

When we reach a leaf node — corresponding to a particular feature l-mer α — we have pointers to all instance l-mers occurring in the source sequences that are up to m mismatches from α. Because for a source sequence x, the instances with mismatches of α in x — the l-mers in x belonging to $N_{(l,m)}(\alpha)$ — are exactly the ones that contribute to the α-coordinate of the feature vector $\Phi(x)$, we can now sum the contributions of all instances occurring in each source sequence and update the kernel value $k(x,y)$: if $n_\alpha(x)$ and $n_\alpha(y)$ are the number of instances (with mismatches) of l-mer α in x and y, we perform the update $k(x,y) \leftarrow k(x,y) + n_\alpha(x) \cdot n_\alpha(y)$. Figure 4.1 gives an example of the mismatch tree traversal.

During the kernel computation, we need only search down paths corresponding to l-mers that occur (with mismatches) in the data. The number of l-mers within m mismatches of any given fixed l-mer is

```
traverse(currInstances,currMismatches,currDepth)
   if currDepth == l
      process leaf
   else
      for each symbol a in alphabet {
         for each inst in currInstances {
            if inst[currDepth] != a
               if currMismatches{inst} < m {
                  add inst to newInstances
                  newMismatches{inst} = currMismatches{inst} + 1
               }
            else {
               add inst to newInstances
               newMismatches{inst} = currMismatches{inst}
            }
         }
         if newInstances not empty
            traverse(newInstances,newMismatches,currDepth+1)
      }
```

Figure 4.2 Depth-first traversal for mismatch kernel computation.

$$\sum_{i=0}^{m} \binom{l}{i} (|\Sigma| - 1)^i = O(l^m |\Sigma|^m).$$

Mismatch kernel complexity

To compute the kernel value $k(x, y)$, we make one kernel update per leaf node that we reach in the traversal, and each l-mer instance that is counted at a leaf node is processed l times as it is passed down the path from the root. Thus, to calculate a single kernel value, the complexity is $O(l^{m+1}|\Sigma|^m(|x| + |y|))$.

Another advantage of the mismatch algorithm is its efficient use of memory, which also leads to faster running time in practice. Because we perform a depth-first traversal, when we backtrack in the tree, we collapse the current node and expand the next node. Thus the only expanded nodes are along the current search path, and there is a maximum of l stored nodes (counting the root node) at any time. However, bookkeeping information must be stored with each node along the search path to guarantee the complexity given above. The kernel computation can in fact be achieved by a recursive function, without explicitly building and storing the full tree. Simplified pseudocode for such a recursive function is outlined in figure 4.2. However, in this simple version, we do extra work because we process each l-mer instance at an internal node Σ times in order to check which child nodes it should be passed down to, resulting in an additional factor of $|\Sigma|$ in the theoretical complexity. Instead, by storing more bookkeeping information, we can process the list of instances exactly once and set up all the lists of instances to be passed to the child nodes before descending to the first child node. We provide the simplified version of the pseudocode for greater readability.

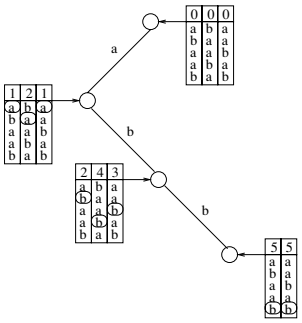


Figure 4.3 Trie traversal for gappy kernel. Expansion along a path from root to leaf during traversal of the trie for the $(5, 3)$-gappy kernel, showing only the instance 5-mers for a single sequence $x = abaabab$. Each node stores its valid 5-mer instances and the index to the last match for each instance. Instances at the leaf node contribute to the kernel for 3-mer feature abb.

4.3.2 (g, l)-Gappy Kernel Computation

For the (g, l)-gappy kernel, we again represent our feature space as a rooted tree of depth l where each internal node has $|\Sigma|$ branches, each branch is labeled with a symbol from Σ, and each leaf node represents a fixed l-mer in feature space.

Using a depth-first traversal of this tree, we now maintain at each visited node a set of pointers to all g-mer instances in the input sequences that contain a subsequence (with gaps) that matches the current prefix pattern; we also store, for each g-mer instance, an index pointing to the last position we have seen so far in the g-mer. At the root, we store pointers to all g-mer instances, and for each instance, the stored index is 0, indicating that we have not yet seen any characters in the g-mer.

As we pass from a parent node to a child node along a branch labeled with symbol a, a g-mer instance from the parent's list is passed down to the child only if the remainder of the g-mer sequence contains the symbol a, and if the first occurrence of this symbol would not include more than the allowable number of gaps. If such a character exists, we pass the g-mer to the child node along with its updated index (first occurrence of a from remainder of g-mer); otherwise, we drop the instance and do not pass it to the child. Thus at each node of depth d, we have effectively performed a greedy gapped alignment of g-mers from the input sequences to the current d-length prefix, allowing insertion of up to $g-l$ gaps into the prefix sequence to obtain each alignment. When we encounter a node with an empty list of pointers (no valid occurrences of the current prefix), we do not need to search below it in the tree; in fact, unless there is a valid g-mer instance from each of x and y, we do not have to process the subtree.

```
traverse(currInstances,currIndices,currDepth)
   if currDepth == l
      process leaf
   else
      for each symbol a in alphabet {
         for each inst in currInstances {
           nextHit = findNextLocation(a,inst,currIndices{inst})
           if nextHit <= (g - l + currDepth + 1) {
             add inst to newInstances
             newIndices{inst} = nextHit
           }
         }
         if newInstances not empty
            traverse(newInstances,newIndices,currDepth+1)
      }
```

Figure 4.4 Depth-first traversal for gappy kernel computation.

When we reach a leaf node, we sum the contributions of all instances occurring in each source sequence to obtain feature values for x and y corresponding to the current l-mer, and we update the kernel by adding the product of these feature values. Again, we can accomplish the depth-first traversal with a recursive function and do not have to store the full trie in memory. Figure 4.3 shows expansion down a path during the recursive traversal, and figure 4.4 gives simplified pseudocode for the algorithm. As with the mismatch kernel algorithm, in order to guarantee theoretical time complexity described below, we actually need to maintain more bookkeeping information: we should process the full list of g-mer instances once and set up all the child node lists before proceeding to the first recursive call. That is, to process a g-mer instance, instead of iterating through the alphabet and scanning for the first match of each symbol a, we should just add the g-mer instance to all child lists corresponding to first occurrences of symbols in the remainder of the sequence. The version above is given for understandability and ease of implementation.

The computation at the leaf node depends on which version of the gappy kernel one uses. For the unweighted feature map, we obtain the feature values of x and y corresponding to the current l-mer by counting the g-mer instances at the leaf coming from x and y, respectively; the product of these counts gives the contribution to the kernel for this l-mer feature. For the λ-weighted gappy feature map, we need a count of all alignments of each valid g-mer instance against the l-mer feature allowing up to $g - l$ gaps. This can be computed with a simple dynamic programming routine (similar to the Needleman-Wunsch algorithm), where we sum over a restricted set of paths, as shown in figure 4.5. The complexity is $O(l(g-l))$, since we fill a restricted trellis of $(l+1)(g-l+1)$ squares. Note that when we align a subsequence $b_{i_1}b_{i_2}\ldots b_{i_l}$ against an l-mer $a_1a_2\ldots a_l$, we only penalize interior gaps corresponding to nonconsecutive indices in $1 \le i_1 < i_2 \ldots < i_l \le g$. Therefore, the multiplicative gap cost is 1 in the 0th and last rows of the trellis and λ in the other rows.

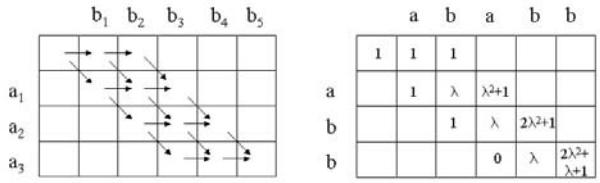


Figure 4.5 Dynamic programming at the leaf node. The left trellis shows the restricted paths for aligning a g-mer against an l-mer, with insertion of up to $g - l$ gaps in the l-mer, for $g = 5$ and $l = 3$. The basic recursion for summing path weights is $S(i, j) = m(a_i, b_j)S(i - 1, j - 1) + g(i)S(i, j - 1)$, where $m(a, b) = 1$ if a and b match, 0 if they are different, and the gap penalty $g(i) = 1$ for $i = 0, l$ and $g(i) = \lambda$ for other rows. The right trellis shows the example of aligning *ababb* against 3-mer *abb*.

Each g-mer instance in the input data can contribute to

$$\begin{pmatrix} g \\ l \end{pmatrix} = O(g^{g-l})$$

l-mer features (assuming that $g - l$ is smaller than l). Therefore, we visit at most $O(g^{g-l}(|x| + |y|)$ leaf nodes in the traversal. Since we iterate through at most g positions of each g-mer instance as we pass from root to leaf, the traversal time is $O(g^{g-l+1}(|x| + |y|))$. The total processing time at leaf nodes is $O(g^{g-l}(|x| + |y|))$ for the unweighted gappy kernel and $O(l(g - l)g^{g-l}(|x| + |y|))$ for the weighted gappy kernel. Therefore, in both cases, we have total complexity of the form $O(c(g, l)(|x| + |y|))$, with $c(g, l) = O((g - l)g^{g-l+1})$ for the more expensive kernel.

Gappy kernel complexity

4.3.3 (l, σ)-Substitution Kernel Computation

Similar to the mismatch kernel algorithm, for the substitution kernel we use a depth l trie to represent the feature space. We store, at each depth d node that we visit, a set of pointers to all l-mer instances α in the input data whose d-length prefixes have current mutation score $-\sum_{i=1}^{d} \log P(a_i|b_i) < \sigma$ of the current prefix pattern $b_1 b_2 \dots b_d$, and we store the current mutation score for each l-mer instance. As we pass from a parent node at depth d to a child node at depth $d + 1$ along a branch labeled with symbol b, we process each l-mer α by adding $-\log P(a_{d+1}|b)$ to the mutation score and pass it to the child iff the score is still less than σ. As before, we update the kernel at the leaf node by computing the contribution of the corresponding l-mer feature.

Substitution kernel complexity

The number of leaf nodes visited in the traversal is $O(N_\sigma(|x| + |y|))$, where $N_\sigma = \max_{\alpha \in \Sigma^l} |M_{(l,\sigma)}|$. We can choose σ sufficiently small to get any desired bound on N_σ. Total complexity for the kernel value computation is $O(lN_\sigma(|x| + |y|))$.

4.3.4 (l, m)-Wildcard Kernel Computation

Again, very similar to the mismatch kernel algorithm, the wildcard kernel uses a depth l trie with branches labeled by characters in $\Sigma \cup \{*\}$, where we prune (do not traverse) subtrees corresponding to prefix patterns with greater than m wildcard characters. At each node of depth d, we maintain pointers to all l-mers instances in the input sequences whose d-length prefixes match the current d-length prefix pattern (with wildcards) represented by the path down from the root.

Each l-mer instance in the data matches at most

$$\sum_{i=0}^{m} \binom{l}{i} = O(l^m)$$

l-length patterns having up to m wildcards. Thus the number of leaf nodes visited is in the traversal is $O(l^m(|x| + |y|))$, and total complexity for the kernel value computation is $O(l^{m+1}(|x| + |y|))$.

Wildcard kernel complexity

4.3.5 Comparison with Mismatch Kernel Complexity

As we have seen, for the (l, m) mismatch kernel, the size of the mismatch neighborhood of an instance l-mer is $O(l^m|\Sigma|^m)$, so total kernel value computation is $O(l^{m+1}|\Sigma|^m(|x|+|y|))$. The three newer kernels have running time $O(c_k(|x|+|y|))$, where constant c_k depends on the parameters of the kernel but not on the size of the alphabet Σ. The difference comes from the fact that new inexact matching neighborhoods we define – the gapped match set, the mutation neighborhood, and the wildcard neighborhood – all have cardinality that does not depend on Σ. Therefore, we have improved constant term for larger alphabets (such as the alphabet of 20 amino acids). In section 4.5, we show that these new, faster kernels have performance comparable to the mismatch kernel in protein classification experiments.

4.4 Fast Prediction

Our string kernels provide an additional advantage when used with SVMs: the particular form of the SVM solution combined with the definition of the feature map allows us to implement fast prediction on test sequences. The learned SVM classifier is given by $f(x) = \sum_{i=1}^{r} y_i a_i \langle \Phi(x_i), \Phi(x) \rangle + b$, where x_i are the training sequences that map to support vectors, y_i are labels (± 1), and a_i are weights obtained from the dual SVM optimization problem. Note that the classification function evaluated on the test sequence x is the sum of classification "scores" $f(\alpha)$ for the l-mers α it contains (g-mers in the case of the gappy kernel). We can therefore precompute and store all the non-zero l-mer scores. Then the prediction $f(x)$ can be calculated in linear time (i.e., $O(|x|)$) by scanning through the l-mers in x and looking up the precomputed l-mer scores.

Linear-time prediction

For the mismatch kernel, one way to compute the l-mer scores is to use two passes of the mismatch tree data structure. In the first pass, we compute the non-zero coordinates of the normal vector $w = \sum_{i=1}^{r} y_i a_i \Phi_{(l,m)}(x_i)$. We traverse the support sequences x_i, and at leaf node corresponding to l-mer β, we compute the weighted sum of valid instances to obtain the coordinate w_β of the normal vector. In the second pass, we use the normal coordinates w_β to obtain the l-mer scores. We traverse the set of l-mers β having non-zero w_β, and at leaf node corresponding to l-mer α, we compute the sum $\sum_{\beta \text{ in } N_{(l,m)}(\alpha)} w_\beta$ to obtain the score for α.

Similar computation of the normal vector coordinates and l-mer (or g-mer) scores can be performed for the other kernels.

4.5 Experiments

We tested the mismatch kernel and all the newer string kernels with SVM classifiers on a benchmark SCOP (version 1.37) data set from Jaakkola et al. (2000), which is designed for the remote protein homology detection problem, and reported results in Leslie et al. (2003b) and Leslie and Kuang (2003). In these experiments, remote homology is simulated by holding out all members of a target SCOP family from a given superfamily as a test set, while examples chosen from the remaining families in the same superfamily form the positive training set. The negative test and training examples are chosen from disjoint sets of folds outside the target family's fold, so that negative test and negative training sets are unrelated to each other and to the positive examples. More details of the experimental setup can be found in Jaakkola et al. (2000).

We summarize these experimental results here in two parts. We first give a method comparison of the mismatch kernel with the Fisher-SVM method and a profile HMM-based method, showing that the mismatch kernel performs competitively with the Fisher kernel. We then compare the SVM classification performance of the three newer string kernels with both the mismatch kernel and the Fisher kernel (Jaakkola et al., 2000). These results establish that the newer string kernels can accomplish performance similar to the mismatch kernel while improving the constant factor in the computational complexity.

All methods are evaluated using the receiver operating characteristic (ROC) score, which is the area under the graph of the rate of true positives as a function of the rate of false positives as the threshold for the classifier varies (Gribskov and Robinson, 1996). Perfect ranking of all positives above all negatives gives a ROC score of 1, while a random classifier has an expected score close to 0.5.

4.5.1 Mismatch Kernels

We use small values of l for the mismatch kernel because the test sets are designed for remote homology detection. Without mismatches, the only reasonable values are $l = 3$ and $l = 4$, since $l \geq 5$ results in a spectrum kernel matrix that is

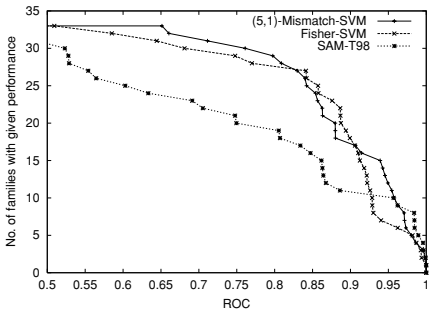


Figure 4.6 Comparison of three homology detection methods for the SCOP data set. The graph plots the total number of families for which a given method exceeds a ROC score threshold. Each series corresponds to one of the homology detection methods described in the text.

almost everywhere 0 off the diagonal (and $l < 3$ is not informative). When allowing mismatches, we were therefore interested in slightly longer l and a very small number of mismatches for efficiency in training. We tested $(l, m) = (5, 1)$ and $(6, 1)$, where we normalized the kernel by $k(x, y) \leftarrow \frac{k(x,y)}{\sqrt{k(x,x)}\sqrt{k(y,y)}}$. Our results show that $(l, m) = (5, 1)$ yields slightly better performance, though results for both choices were similar. [Data for $(l, m) = (6, 1)$ not shown.]

In figure 4.6, we compare mismatch-SVM results against the original experimental results from Jaakkola et al. (2000) for two methods, the SAM-T98 iterative HMM and the SVM-Fisher method. The figure includes results for all 33 SCOP families, and each series corresponds to one homology detection method. Qualitatively, the curves for SVM-Fisher and mismatch-SVM are quite similar. When we compare the overall performance of two methods using a two-tailed signed rank test with a p-value threshold of .05 and a Bonferroni correction for multiple comparisons, we find only the following significant differences: Fisher-SVM and mismatch-SVM perform better than SAM-T98. There is no statistically significant difference between the performance of Fisher-SVM and mismatch-SVM.

4.5.2 Other Inexact Matching Kernels

We tested the (g, l)-gappy kernel on the same SCOP experiments with parameter choices $(g, l) = (6, 4)$, $(7, 4)$, $(8, 5)$, $(8, 6)$, and $(9, 6)$. Among them $(g, l) = (6, 4)$ yielded the best results, though other choices of parameters had quite similar performance (data not shown). We also tested the alternative weighted gappy kernel, where the contribution of an instance g-mer to an l-mer feature is a weighted sum of all the possible matches of the l-mer to subsequences in the g-mer with multiplicative gap penalty λ $(0 < \lambda \leq 1)$. We used gap penalty $\lambda = 1.0$ and $\lambda = 0.5$ with the $(6, 4)$ weighted gappy kernel. We found that $\lambda = 0.5$ weighting slightly

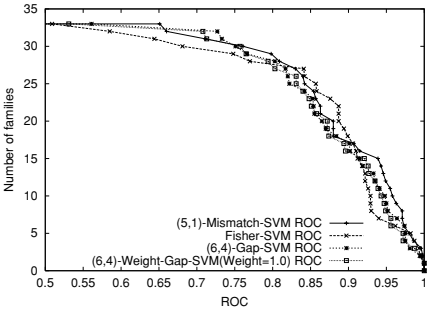


Figure 4.7 Comparison of mismatch-SVM, Fisher-SVM, and gap-SVM. The graph plots the total number of families for which a given method exceeds a ROC score threshold. The $(6, 4)$-Gap-SVM uses the unweighted gappy string kernel. The $(6, 4)$-Weight-Gap-SVM uses the weighted version of the gappy string kernel, which counts the total number alignments of an l-mer against a g-mer with multiplicative gap penalty of λ.

weakened performance (results not shown). In figure 4.7, we see that unweighted and weighted ($\lambda = 1.0$) gappy kernels have comparable results to $(5, 1)$-mismatch kernel and Fisher kernel.

We tested the substitution kernels with $(l, \sigma) = (4, 6.0)$. Here, $\sigma = 6.0$ was chosen so that the members of a mutation neighborhood of a particular 4-mer would typically have only one position with a substitution, and such substitutions would have fairly high probability. Therefore, the mutation neighborhoods were much smaller than, for example, $(4, 1)$-mismatch neighborhoods. The results are shown in Figure 4.8 (left). Again, the substitution kernel has comparable performance with mismatch-SVM and Fisher-SVM, though the results are perhaps slightly weaker for more difficult test families.

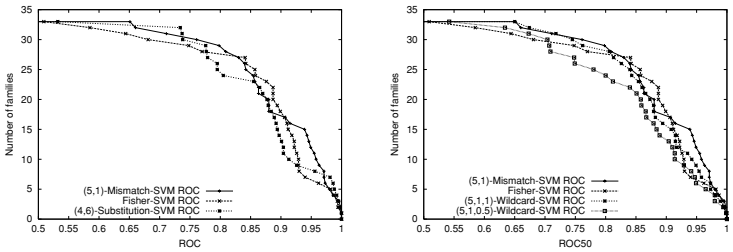


Figure 4.8 Comparison with substitution-SVM and wildcard-SVM. The graph plots the total number of families for which a given method exceeds a ROC score threshold. Results for the substitution kernel (*left*) and wildcard kernel (*right*) are shown.

For comparison with the $(5, 1)$-mismatch kernel, we tested wildcard kernels with parameters $(l, m, \lambda) = (5, 1, 1.0)$ and $(l, m, \lambda) = (5, 1, 0.5)$. Results are shown in figure 4.8 (right). The wildcard kernel with $\lambda = 1.0$ seems to perform as well or almost as well as the $(5, 1)$-mismatch kernel and Fisher kernel, whereas enforcing a penalty on wildcard characters of $\lambda = 0.5$ seems to weaken performance somewhat.

4.6 Conclusion

We have described a number of different kernels that capture a notion of inexact matching – the mismatch kernel, which counts l-mer occurrences with mismatches, and three newer kernels that use restricted gaps, probabilistic substitutions, and wildcards – but maintain fast computation time. Using a recursive function based on a trie data structure, we show that for all these kernels, the time to compute a kernel value $k(x, y)$ is $O(c_k(|x| + |y|))$, where the constant c_k depends on the parameters of the kernel. For the mismatch kernel, $|\Sigma|$ as well as l and m controls the size of the mismatch neighborhood and hence the constant c_k. For the three newer kernels, however, c_k is independent of the size of the alphabet Σ, which significantly improves on the constant factor involved in the mismatch kernel computation.

We present results on a benchmark SCOP data set for the remote protein homology detection problem and show both that the mismatch kernel is competitive with the best competing methods and that many of the new, faster kernels achieve performance comparable to the mismatch kernel.

In certain biological applications, the l-length subsequence features that are "most significant" for discrimination can themselves be of biological interest. In the forthcoming longer version of Leslie et al. (2003b), we show how to extract discriminative l-mers from an SVM classifier trained using the mismatch kernel. More generally, it would be interesting to investigate feature selection on the set of l-mer features used in these kernels, so that we identify a feature subset that both improves discrimination and gives biologically interpretable differences between positive and negative examples.

We also note that Cortes et al. (2003) have recently presented a framework based on transducers (finite state automata) that can generate many previously defined string kernels as rational kernels. In Leslie and Kuang (2003), we give transducers for some of our inexact matching string kernels. By understanding the connection between our trie-based data structures and computational methods used in transducers, such as failure functions, or by extending suffix tree methods (Vishwanathan and Smola, 2003) to these inexact matching kernels, we could hope for improved constant factors in the kernel complexity.

Acknowledgments

C.L. is supported by an Award in Informatics from the PhRMA Foundation and by NIH grant LM07276-02.

5 Fast Kernels for String and Tree Matching

S.V.N. Vishwanathan
Alexander Johannes Smola

In this chapter we present a new algorithm suitable for matching discrete objects such as strings and trees in linear time, thus obviating dynamic programming with quadratic time complexity.

This algorithm can be extended in various ways to provide linear time prediction cost in the length of the sequence to be classified. We demonstrate extensions in the case of position-dependent weights, sliding window classifiers for a long sequence, and efficient algorithms for dealing with weights given in the form of dictionaries. This improvement on the currently available algorithms makes string kernels a viable alternative for the practitioner.

5.1 Introduction

Many problems in machine learning require a data classification algorithm to work with a set of discrete objects. Common examples include biological sequence analysis where data are represented as strings (Durbin et al., 1998), and natural language processing (NLP), where the data are given in the form of a string combined with a parse tree (Collins and Duffy, 2002) or an annotated sequence (Altun et al., 2003).

In order to apply kernel methods one defines a measure of similarity between discrete structures via a feature map $\phi : \mathcal{X} \to \mathcal{F}$. Here $\mathcal{X}$ is the set of discrete structures (e.g., the set of all parse trees of a language) and $\mathcal{F}$ is a Hilbert space. Since $\phi(x) \in \mathcal{F}$ we can define a kernel by evaluating the scalar products

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \tag{5.1}$$

where $x, x' \in \mathcal{X}$. The success of a kernel method employing k depends both on the *faithful representation* of discrete data and an *efficient means of computing* k.

Recent research effort has focused on defining meaningful kernels on strings. Many ideas based on the use of substrings (Herbrich, 2002), gapped substrings (Lodhi et al., 2002), k-length substrings (Leslie et al., 2002), and mismatch penalties (Leslie et al., 2003b) have been proposed. This chapter presents a means of computing substring kernels on strings (Herbrich, 2002; Leslie et al., 2002; Joachims, 2002) and trees in *linear time* in the size of the arguments, independent of the weights associated with any of the matching subwords. We also present a linear time algorithm for prediction which is independent of the number of support vectors (SVs). This is a significant improvement, since the so-far fastest methods rely on dynamic programming which incurs a quadratic cost in the length of the argument (Herbrich, 2002) or are additionally linear in the length of the matching substring (Leslie et al., 2003b). Further extensions to finite-state machines, formal languages, automata, etc. can be found in Vishwanathan (2002), Smola and Vishwanathan (2003), and Cortes et al. (2003). Other means of generating kernels via the underlying correlation structure can be found in Takimoto and Warmuth (1999).

In a nutshell our idea works as follows: assume we have a kernel $k(x, x') := \sum_{i \in I} \phi_i(x)\phi_i(x')$, where the index set I may be large, yet the number of nonzero entries is small in comparison to $|I|$ or the terms $\phi_i(x)$ have special structure. Then an efficient way of computing k is to sort the set of non-zero entries of $\phi(x)$ and $\phi(x')$ beforehand and count only matching non-zeros.

Sparse vectors

This is similar to the dot product of sparse vectors in numerical analysis. As long as the sorting is done in an intelligent manner, the cost of computing k is linear in the sum of non-zero entries combined. In order to use this idea for matching strings (which have a quadratically increasing number of substrings) and trees (which can be transformed into strings), efficient sorting is realized by the compression of the set of all substrings into a suffix tree. Moreover, dictionary keeping allows us to use (almost) arbitrary weightings for each of the substrings and still compute the kernels in linear time.

Our results improve on the algorithm proposed by Leslie et al. (2003b) in the case of *exact* matches, as the algorithm is now independent of the length of the matches and furthermore we have complete freedom in choosing the weight parameters. For inexact matches, unfortunately, such modifications are (still) not possible and we suggest the online construction of Leslie et al. (2003b) for an efficient implementation.

The chapter is organized as follows. In section 5.2 we give the basic definition of the string and tree kernels used in this chapter. Section 5.3 contains the main result of the chapter, namely how suffix trees can be used to compute string kernels efficiently. Sections 5.4 and 5.5 deal with the issue of computing weights efficiently and how to use the newly established algorithms for prediction purposes while keeping the linear-time property. Experimental results in section 5.6 and a discussion (section 5.7) conclude the chapter.

5.2 Kernels

5.2.1 String Kernels

We begin by introducing some notation. Let $\mathcal{A}$ be a finite set which we call the *alphabet*, for example, $\mathcal{A} = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$. The elements of $\mathcal{A}$ are *characters*. Let \$ be a sentinel character such that $\$ \notin \mathcal{A}$. Any $x \in \mathcal{A}^k$ for $k = 0, 1, 2 \ldots$ is called a *string*. The empty string is denoted by ϵ and $\mathcal{A}^*$ represents the set of all nonempty strings—the Kleene closure (Hopcroft and Ullman, 1979)—defined over the alphabet $\mathcal{A}$.

In the following we use $s, t, u, v, w, x, y, z \in \mathcal{A}^*$ to denote strings and $a, b, c \in \mathcal{A}$ to denote characters. $|x|$ denotes the length of x, $uv \in \mathcal{A}^*$ the concatenation of two strings u and v, $au \in \mathcal{A}^*$ the concatenation of a character and a string. We use $x[i:j]$ with $1 \leq i \leq j \leq |x|$ to denote the substring of x between locations i and j (both inclusive). If $x = uvw$ for some (possibly empty) u, v, w, then u is called a *prefix* of x while v is called a substring (also denoted by $v \sqsubseteq x$) and w is called a *suffix* of x. Finally, $\mathrm{num}_y(x)$ denotes the number of occurrences of y in x (i.e., the number of times y occurs as a substring of x). The types of kernels we will be studying are defined by

String kernel definition

$$k(x, x') := \sum_{s \sqsubseteq x, s' \sqsubseteq x'} w_s \delta_{s,s'} = \sum_{s \in \mathcal{A}^*} \mathrm{num}_s(x)\, \mathrm{num}_s(x') w_s. \tag{5.2}$$

That is, we count the number of occurrences of every substring s in both x and x' and weight it by w_s, where the latter may be a weight chosen a priori or after seeing data, for example, inverse document frequency counting (Leopold and Kindermann, 2002). This includes a large number of special cases:

- Setting $w_s = 0$ for all $|s| > 1$ yields the bag-of-characters kernel, counting simply single characters (Joachims, 2002).
- The bag-of-words kernel is generated by requiring s to be bounded by whitespace, that is, $w_s \neq 0$ iff s is bounded by a whitespace character on either side (Joachims, 2002).
- Setting $w_s = 0$ for all $|s| > n$ yields limited range correlations of length n; that is, we only consider the contributions due to substrings of length n or less.
- The *k-spectrum* kernel takes into account substrings of length k (Leslie et al., 2002). It is achieved by setting $w_s = 0$ for all $|s| \neq k$.
- TFIDF weights (Salton, 1989) are achieved by first creating a (compressed) list of all s, including frequencies of occurrence, and subsequently rescaling w_s accordingly.

All these kernels can be computed efficiently via the construction of suffix trees, as we will see in the following sections.

However, before we go about computing $k(x, x')$, let us turn to trees. The latter are important for two reasons: first, since the *suffix tree* representation of a string will be used to compute kernels efficiently, and second, since we may wish to

compute kernels on trees, which will be carried out by reducing trees to strings and then applying a string kernel.

5.2.2 Tree Kernels

A tree is defined as a simple, directed, connected graph with no cycles. A *rooted tree* has a single special node called the root. An *internal node* has one or more child nodes and is called the *parent* of its children. The *root* is a node with no parent. A node with no children is referred to as a *leaf*. A sequence of nodes $n_1, n_2, \ldots, n_k$, such that n_i is the parent of n_{i+1} for $i = 1, 2, \ldots, k-1$ is called a *path*. Given two nodes a and d, if there is a path from node a to d, then a is called an *ancestor* of d and d is called a *descendent* of a. We define a *subtree* as a node in the tree together with all its descendents. A subtree rooted at node n is denoted as T_n and $t \models T$ is used to indicate that t is a subtree of T. An *ordered tree* is a rooted tree in which the order of the subtrees (hanging from every node) is significant. From this point on all trees (and subtrees) we consider are ordered trees. If a set of nodes in the tree along with the corresponding edges forms a tree, then we define it to be a *subset tree*.

If every node n of the tree contains a label, denoted by label(n), then the tree is called a *labeled* tree. If only the leaf nodes contain labels, then the tree is called a *leaf-labeled* tree. Kernels on trees can be defined by defining kernels on matching subset trees, as proposed by Collins and Duffy (2002), or (more restrictively) by defining kernels on matching subtrees. In the latter case we have

Tree kernel definition

$$k(T, T') = \sum_{t \models T, t' \models T'} w_t \delta_{t,t'}. \tag{5.3}$$

where $\delta_{t,t'} = 1$ iff the ordered subtrees t and t' are isomorphic and have the same order of child nodes at every level. We can compute our (more restrictive) tree kernels in linear time, but if we consider subset trees, as in Collins and Duffy (2002), quadratic-time algorithms are required.

If a tree $\bar{T}$ can be obtained from a tree T by swapping the order of child nodes, then it is said to be *equivalent*. Alternatively, equivalent trees are obtained by permuting the order of the leaf nodes without disturbing any parent-child relationship in the tree (see figure 5.1). The tree kernel, as defined in (5.3), does not take this equivalence into account. While this may be desirable in many applications (where ordered trees are naturally used), there are domains, for instance, phylogenetic trees in bioinformatics, where it is desirable to reduce trees to a canonical form before computing kernels. We achieve that by implicitly sorting (or ordering) the trees.

5.2.3 Ordering Trees

To order trees we assume that a lexicographic order is associated with the labels if they exist. Furthermore, we assume that the additional symbols '[', ']' satisfy '[' < ']',

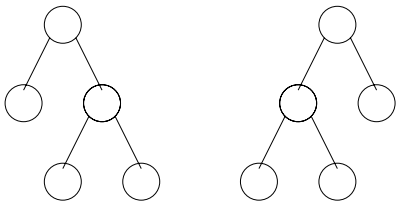


Figure 5.1 Two equivalent trees, which can be transformed into each other by swapping the children of the root node.

and that '$]'$, '$['$ < label(n) for all labels. We will use these symbols to define tags for each node as follows:

- For an unlabeled leaf n define tag$(n) :=$ `[]`.
- For a labeled leaf n define tag$(n) :=$ `[`label(n)`]`.
- For an unlabeled node n with children $n_1, \ldots, n_c$ define a lexicographically sorted permutation π of the child nodes such that tag$(n_{\pi(i)}) \leq$ tag$(n_{\pi(j)})$ if $\pi(i) < \pi(j)$ and define

$$\text{tag}(n) = \texttt{[}\,\text{tag}(n_{\pi(1)})\,\text{tag}(n_{\pi(2)})\ldots\text{tag}(n_{\pi(c)})\texttt{]}.$$

Tree to string conversion

- For a labeled node perform the same operations as above and set

$$\text{tag}(n) = \texttt{[}\,\text{label}(n)\,\text{tag}(n_{\pi(1)})\,\text{tag}(n_{\pi(2)})\ldots\text{tag}(n_{\pi(c)})\texttt{]}.$$

For instance, the root nodes of both trees depicted in figure 5.1 would be encoded as [[[][]][]]. We now prove that the tag of the root node, indeed, is a unique identifier and that it can be constructed in log linear time.

Theorem 5.1 *Denote by T a binary tree with l nodes and let λ be the maximum length of a label. Then the following properties hold for the tag of the root node:*

1. tag(root) *can be computed in $(\lambda + 2)(l \log_2 l)$ time and linear storage in l.*
2. *Every substring s of* tag(root) *starting with $'['$ and ending with a balanced $']'$ has a one-to-one mapping with a subtree T_n of T where s is the tag on node n.*
3. *If trees T and $\tilde{T}$ are equivalent, then their* tag(root) *is the same. Furthermore,* tag(root) *allows the reconstruction of a unique element of the equivalence class.*

Proof **Claim 1** We use induction. The tag of a leaf can be constructed in constant time by storing [,], and a pointer to the label of the leaf (if it exists), that is, in 3 operations. Next assume that we are at node n, with children n_1, n_2. Let T_n contain l_n nodes and T_{n_1} and T_{n_2} contain l_1, l_2 nodes respectively. By our induction assumption we can construct the tag for n_1 and n_2 in $(\lambda+2)(l_1 \log_2 l_1)$ and $(\lambda + 2)(l_2 \log_2 l_2)$ time respectively. Comparing the tags of n_1 and n_2 costs at most

$(\lambda+2)\min(l_1,l_2)$ operations and the tag itself can be constructed in constant time and linear space by manipulating pointers. Without loss of generality we assume that $l_1 \le l_2$. Thus, the time required to construct $\mathrm{tag}(n)$ (normalized by $\lambda+2$) is

$$l_1(\log_2 l_1 + 1) + l_2 \log_2(l_2) = l_1 \log_2(2l_1) + l_2 \log_2(l_2) \le l_n \log_2(l_n). \tag{5.4}$$

Claim 2 Without loss of generality we consider a labeled tree and use induction to show that every subtree of T is associated with a balanced substring of tag(root). For a leaf node n we assign $\mathrm{tag}(n) = [\mathrm{label}(n)]$ and it is clear that it corresponds to a balanced substring of tag(root). Suppose we are at an internal node n with two child nodes n_1 and n_2. Furthermore, assume that the subtrees T_{n_1} and T_{n_2} correspond to balanced substrings $\mathrm{tag}(n_1)$ and $\mathrm{tag}(n_2)$ respectively. Since we assign $\mathrm{tag}(n) = [\mathrm{label}(n)\,\mathrm{tag}(n_1)\,\mathrm{tag}(n_2)]$ it is clear that the substring corresponding to T_n is balanced.

Conversely, by our recursive definition, every balanced substring of tag(root) must be of the form $\texttt{[}\,\mathrm{label}(n)\,\mathrm{tag}(n_1)\,\mathrm{tag}(n_2)\texttt{]}$ for some node n with children n_1 and n_2. But this precisely corresponds to the tag on node n and hence to the subtree T_n. This proves claim 2.

Another way of visualizing our ordering is by imagining that we perform a DFS (depth first search) on the tree T and emit a $'['$ followed by label(n), when we visit a node n for the first time and a $']'$ when we leave a node for the last time. It is clear that a *balanced* substring s of tag(root) is emitted only when the corresponding DFS on T_n is completed.

Claim 3 Since leaf nodes do not have children their tag is clearly invariant under permutation. For an internal node we perform lexicographic sorting on the tags of its children. This sorting maps all the trees of the equivalence class into a canonical representative. We then define the tag(root) of this tree as the string corresponding to our given tree. This directly proves the first part of our claim.

Concerning the reconstruction, we proceed as follows: each tag of a subtree starts with '$[$' and ends in a balanced '$]$', hence we can strip the first [] pair from the tag, take whatever is left outside brackets as the label of the root node, and repeat the procedure with the balanced [...] entries for the children of the root node. This will construct a tree with the same tag as tag(root), thus proving our claim. ■

An extension to trees with d nodes is straightforward (the cost increases to $d \log_2 d$ of the original cost due to additional comparisons required in the sorting of the leaves), yet the proof, in particular, (5.4) becomes more technical without providing additional insight; hence we omit this generalization for brevity.

The above tree-to-string conversion algorithm along with claim 2 of theorem 5.1 yields the following straightforward corollary, whose proof is left as a simple exercise for the reader.

Corollary 5.2 *Given trees T, T' the subtree matching kernel defined in (5.3) can be computed via string kernels, if we use the strings* $\mathrm{tag}(T)$ *and* $\mathrm{tag}(T')$ *and require that only balanced substrings s of the form* [...] *have nonzero weight w_s.*

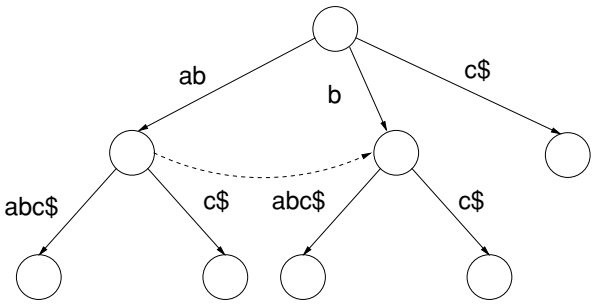


Figure 5.2 Suffix Tree of `ababc`. The leaf containing \$ hanging off the root node has been omitted for readability.

This reduces the problem of tree kernels to string kernels and all we need to show in the following is how the latter can be computed efficiently. For this purpose we need to introduce suffix trees.

5.3 Suffix Trees and Matching Statistics

5.3.1 Definition

The suffix tree is a compacted trie (Knuth, 1998) that stores all suffixes of a given text string (Weiner, 1973). We denote the suffix tree of the string x by $S(x)$. Moreover, let nodes($S(x)$) be the set of all nodes of $S(x)$ and let root($S(x)$) be the root of $S(x)$. If w denotes the path from the root to a node we label the node as $\overline{w}$. For a node $\overline{w}$, $T_{\overline{w}}$ denotes the subtree tree rooted at the node, lvs($\overline{w}$) denotes the number of leaves in $T_{\overline{w}}$ and par($\overline{w}$) denotes it parent node.

We denote by words($S(x)$) the set of all nonempty strings w such that $\overline{wu} \in$ nodes($S(x)$) for some (possibly empty) string u, which means that words($S(x)$) is the set of all possible substrings of x (Giegerich and Kurtz, 1997). For every $t \in$ words($S(x)$) we define ceil(t) as the node $\overline{w}$ such that $w = tu$ and u is the shortest (possibly empty) substring such that $\overline{w} \in$ nodes($S(x)$). That is, it is the immediate next node on the path leading up to t in $S(x)$. Finally, for every $t \in$ words($S(x)$) we define floor(t) as the node $\overline{w}$ such that $t = wu$ and u is the shortest non-empty substring such that $\overline{w} \in$ nodes($S(x)$). That is, it is the last node encountered on the path leading up to t in $S(x)$. For an internal node $\overline{w}$ it is clear that ceil(w) = $\overline{w}$ and floor(w) is the parent of $\overline{w}$. Given a string t and a suffix tree $S(x)$, we can decide if $t \in$ words($S(x)$) in $O(|t|)$ time by just walking down the corresponding edges of $S(x)$ (Weiner, 1973).

If the sentinel character \$ is added to the string x, then it can be shown that for any $t \in$ words($S(x)$), lvs(ceil(t)) gives us the number of occurrence of t in x

Table 5.1 Matching statistic of `bcbab` with respect to S(`ababc`) are shown here along with the matching substrings.

String	`b`	`c`	`b`	`a`	`b`
v_i	2	1	3	2	1
$x[i : \overline{v_i}]$	`bc`	`c`	`bab`	`ab`	`b`
c_i	`bc$`	`c$`	`babc$`	`ab`	`b`
c'_i	`b`	root	`b`	root	root

(Giegerich and Kurtz, 1997). The idea works as follows: all suffixes of x starting with t have to pass through ceil(t), hence we simply have to count the occurrences of the sentinel character, which can be found only in the leaves. Note that a simple DFS on $S(x)$ will enable us to calculate lvs($\overline{w}$) for each node $\overline{w}$ of $S(x)$ in $O(|x|)$ time and space.

Let $\overline{aw}$ be a node in $S(x)$, and v be the longest suffix of w such that $\overline{v} \in$ nodes($S(x)$). An unlabeled edge $\overline{aw} \rightarrow \overline{v}$ is called a suffix link in $S(x)$. A suffix link of the form $\overline{aw} \rightarrow \overline{w}$ is called *atomic*. It can be shown that all the suffix links in a suffix tree are atomic (Giegerich and Kurtz, 1997, proposition 2.9). We add suffix links to $S(x)$, to allow us to perform efficient string matching: suppose we found that aw is a substring of x by parsing the suffix tree $S(x)$. It is clear that w is also a substring of x. If we want to locate the node corresponding to w, it would be wasteful to parse the tree again. Suffix links can help us locate this node in constant time. The suffix tree building algorithms (e.g., Ukkonen [1995]) make use of this property of suffix links and construct the suffix tree and all its suffix links in *linear time*.

Note that suffix links are just a special case of so-called *failure functions* from automata theory: there they allow one to backtrack gracefully and reuse some of the information obtained in checking whether a string is accepted by an automaton (Hopcroft and Ullman, 1979; Cortes et al., 2003).

5.3.2 Matching Statistics

Given strings x, y with $|x| = n$ and $|y| = m$, the matching statistics of x with respect to y are given by vectors v, c, and c' with $v_i \in \mathbb{N}$ and $c_i, c'_i \in$ nodes($S(y)$) for $i = 1, 2, \ldots, n$. We define v_i as the length of the longest substring of y matching a prefix of $x[i : n]$, $\overline{v_i} := i + v_i - 1$, while c_i is the node corresponding to ceil($x[i : \overline{v_i}]$) and c'_i is the node corresponding to floor($x[i : \overline{v_i}]$) in $S(y)$. For an example, see table 5.1.

For a given x one can construct the matching statistics corresponding to y in linear time. The key observation is that $v_{i+1} \geq v_i - 1$, since if $x[i : \overline{v_i}]$ is a substring of y, then definitely $x[i + 1 : \overline{v_i}]$ is also a substring of y. Besides this, the matching substring in y that we find *must* have $x[i+1 : \overline{v_i}]$ as a prefix. The matching statistics algorithm of Chang and Lawler (1994) exploits this observation and uses it cleverly

to walk down the suffix links of $S(y)$ to compute the matching statistics in $O(|x|)$ time.

More specifically, given c'_i the algorithm finds the intermediate node $p'_{i+1} := \text{floor}(x[i+1 : \overline{v_i}])$ by first walking down the suffix link of c'_i and then walking down the edges corresponding to the remaining portion of $x[i+1 : \overline{v_i}]$ until it reaches p'_{i+1}. Now c_{i+1}, c'_{i+1} and v_{i+1} can be found easily by walking from p'_{i+1} along the edges of $S(y)$ that match $x[\overline{v_i}+1 : n]$, until we can go no further. The value of v_1 is found by simply walking down $S(y)$ to find the longest prefix of x which matches a substring of y.

5.3.3 Matching Substrings

Using the matching statistics we can read off the number of matching substrings in x and y. The useful observation here is that the only substrings which occur in both x and y are those which are prefixes of $x[i : \overline{v_i}]$. The number of occurrences of a substring in y can be found by lvs(ceil(w)), as discussed in section 5.3.1. The two lemmas below formalize this.

Lemma 5.3 *w is a substring of x iff there is an i such that w is a prefix of $x[i : n]$. The number of occurrences of w in x can be calculated by finding all such i.*

Proof Let w be a substring of x. For each occurrence of w in x we can write $w = x[i : j]$ for some unique indices i and j. Clearly $x[i : j]$ is a prefix of $x[i : n]$. Conversely, for every index i, every prefix of $x[i : n]$ is a substring of x. ■

Lemma 5.4 *The set of matching substrings of x and y is the set of all prefixes of $x[i : \overline{v_i}]$.*

Proof Let w be a substring of both x and y. By the above lemma there is an i such that w is a prefix of $x[i : n]$. Since v_i is the length of the maximal prefix of $x[i : n]$ which is a substring in y, it follows that $v_i \geq |w|$. Hence w must be a prefix of $x[i : \overline{v_i}]$.

Conversely, at any position i the longest prefix of $x[i : n]$ which matches some substring of y is $x[i : \overline{v_i}]$. Hence it follows that every prefix of $x[i : \overline{v_i}]$ is a substring of both x and y. ■

5.3.4 Efficient Kernel Computation

From the previous discussion we know how to determine the set of all longest prefixes $x[i : \overline{v_i}]$ of $x[i : n]$ in y in linear time. The following theorem uses this information to compute kernels efficiently.

Theorem 5.5 *Let x and y be strings such that v, c, and c' be the matching statistics of x with respect to y. Assume that*

$$W(y,t) = \sum_{p \in \text{prefix}(v)} w_{up} - w_u \text{ where } \overline{u} = \text{floor}(t) \text{ in } S(y) \text{ and } t = uv. \tag{5.5}$$

Fast kernel computation

can be computed in constant time for any t. Then $k(x,y)$ defined in (5.2) can be computed in $O(|x|+|y|)$ time as

$$k(x,y) = \sum_{i=1}^{|x|} [\text{val}(c'_i) + \text{lvs}(c_i) \cdot W(y, x[i : \overline{v_i}])] \tag{5.6}$$

where $\text{val}(\overline{t}) := \text{val}(\text{par}(t)) + \text{lvs}(\overline{t}) \cdot W(y,t)$ *and* $\text{val}(\text{root}) := 0$.

Proof We first show that (5.6) can indeed be computed in linear time. We know that for $S(y)$ the number of leaves per node can be computed in $O(|y|)$ time by performing a DFS. Also, v, c, and c' can be computed in $O(|x|)$ time by a invocation of the matching statistics algorithm. By assumption on $W(y,t)$ and by exploiting the recursive nature of $\text{val}(\overline{t})$ we can precompute $W(y,w)$ for all $\overline{w} \in \text{nodes}(S(y))$ by using a simple top-down procedure in $O(|y|)$ time.

Now we may compute each term in constant time by a simple lookup for $\text{val}(c'_i)$ and $\text{lvs}(c_i)$ and by computing $W(y, x[i : \overline{v_i}])$ in constant time. Since we have $|x|$ terms, the whole procedure takes $O(|x|)$ time, which proves the $O(|x|+|y|)$ time complexity.

Now we prove that (5.6) really computes the kernel. We know from lemma 5.4 that the sum in (5.2) can be decomposed into the sum over matches between y and each of the prefixes of $x[i : \overline{v_i}]$ (this takes care of all the substrings in x matching with y). This reduces the problem to showing that each term in the sum of (5.6) corresponds to the contribution of all prefixes of $x[i : \overline{v_i}]$.

The key observation here is that all substrings of y which share the same ceil node occur the same number of times in y. This allows us to bracket the contribution due to each of the prefixes of $x[i : \overline{v_i}]$ efficiently. Consider our previous example with $x =$ `bcbab` and $y =$ `ababc`. To compute val(`bab`) we need to consider the contributions due to `bab`, `ba` as well as `b`. Looking at $S(y)$ immediately tells us that `b` occurs twice in y (because $\text{lvs}(\text{ceil}(\texttt{b})) = 2$) and hence its contribution must be counted twice, while `ba` and `bab` occur only once in y and hence their contributions must be counted only once.

Because of its recursive definition and by exploiting the above observation it is clear that $\text{val}(\overline{w})$ for each $\overline{w} \in \text{nodes}(S(y))$ computes the contribution to the kernel due to w and *all* its prefixes. Given an arbitrary substring t such that $\overline{u} = \text{floor}(t)$ and $t = uv$ there are two components of t which contribute to the kernel. One is the contribution due to u and all the prefixes and the other is the contribution due to all strings of the form us where $s \in \text{prefix}(v)$. Since each such substring occurs exactly $\text{lvs}(\text{ceil}(t))$ times in y we can perform efficient bracketing and use $W(y,t)$ to compute the kernel. ∎

Clearly, once $S(y)$ and the annotations of $S(y)$ have been computed, evaluating another $k(x, y)$ costs only $O(|x|)$ rather than $O(|x| + |y|)$ time. This suggests that for prediction we can gain extra efficiency by precomputing a large part of the kernel expansion. This idea is made more explicit in section 5.5.1. Before we do so, however, we need to address the issue under which conditions $W(y, t)$ can really be computed in constant time.

5.4 Weights and Kernels

5.4.1 Length Dependent Weights

If the weights w_s depend only on $|s|$ we have $w_s = w_{|s|}$. Define $\omega_j := \sum_{i=1}^{j} w_i$ and compute its values beforehand up to ω_J where $J \geq |x|$ for all x. Then the sum in $W(y, t)$ telescopes and it follows that

$$W(y,t) = \left[\sum_{j=|\operatorname{floor}(t)|}^{|t|} w_j \right] - w_{|\operatorname{floor}(t)|} = \sum_{j=1}^{|t|} w_j - \sum_{j=1}^{|\operatorname{floor}(t)|} w_j = \omega_{|t|} - \omega_{|\operatorname{floor}(t)|} \quad (5.7)$$

which can be computed in constant time by looking up the precomputed values. Examples of such weighting schemes include decay factors $w_i = \lambda^{-i}$, or indicator functions $w_i = 1$, bounded range interactions $w_i = 1$ if $i \leq n$ and $w_i = 0$ otherwise, and character counts $w_i = \delta_{1i}$ (Joachims, 2002). Denote by $\tau := |t|$ and $\gamma := |\operatorname{floor}(t)|$ the string boundaries. In this case we can compute $W(y, t)$ as follows:

$$W(y,t) = \frac{(\lambda^{-\gamma} - \lambda^{-\tau})}{\lambda - 1} \qquad \text{Exponential decay} \qquad (5.8)$$

$$W(y,t) = \tau - \gamma \qquad \text{Constant weight} \qquad (5.9)$$

$$W(y,t) = \max(0, \min(\tau, n) - \gamma) \qquad \text{Bounded range} \qquad (5.10)$$

$$W(y,t) = \begin{cases} 1 & \text{if } \gamma = 0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{Bag of characters} \qquad (5.11)$$

5.4.2 Position-Dependent Weights

Next we study the case where the weights w_s depend on $|s|$ and the position of s relative to the beginning of the strings x, y. That is, $w_s = w_{|s|}\phi(|u_x|+1)\phi(|u_y|+1)$, where $x = u_x s v_x$ and $y = u_y s v_y$.

Note that all words of y beginning at the same position i share the same weight, namely $\phi(i)$. They also share the same terminal leaf in $S(y)$ (recall that there is exactly one leaf per suffix). Hence it is sufficient to enumerate the leaves and weigh them based on the starting position of the suffix they correspond to. Recall that in subsection 5.3.1 we defined $\operatorname{lvs}(\overline{w})$ as the number of leaves in $T_{\overline{w}}$. If we assume a unit weight placed on each of the leaves, then $\operatorname{lvs}(\overline{w})$ can be thought of as the

sum of the weights of the leaves in $T_{\overline{w}}$. In the case of position-dependent weights each leaf contains a possibly different weight. Hence we generalize the above notion to define $\mathrm{lvs}(\overline{w})$ as the sum of the weights of the leaves in $T_{\overline{w}}$. Furthermore, to compute the kernel, (5.6) needs to be rewritten as follows:

$$k(x,y) = \sum_{i=1}^{|x|} \phi(i) \left[\mathrm{val}(c'_i) + \mathrm{lvs}(c_i) \cdot W(y, x[i:\overline{v_i}])\right] \tag{5.12}$$

where $\mathrm{val}(c_i)$ is as defined in theorem 5.5.

5.4.3 Dictionary Weights

Next assume that we have a set of strings $D = \{x_1, x_2, \ldots, x_k\}$ with corresponding weights w_{x_i} and that all other string-weights vanish, that is, $w_d = 0$ for $d \notin D$. Such a method was recently put to good practical use by Ben-Hur and Brutlag (2003), who use a set of motifs as the features used in classification of remote homologies.

We now provide a fast version of the motif-kernel algorithm, which does not depend on the number of matching motifs or the number of SVs anymore, unlike Ben-Hur and Brutlag (2003, figure 2), however, with the assumption that the motifs *do not contain any wildcards*.

For the dictionary D we define $|D| = \sum_{i=1}^{k} |x_i|$. The suffix tree of D is denoted by $S(D)$ and is defined as the compacted trie obtained by inserting all the suffixes of all $x_i \in D$. By a clever modification of the construction algorithm of McCreight (1976) we can construct $S(D)$ in $O(|D|)$ time (Amir et al., 1994). Given a new string x_{k+1} we can insert it into $S(D)$ in $O(|x_{k+1}|)$ time. The suffix tree $S(D)$ contains a node corresponding to each of the strings in D. If none of the strings in D are substrings of each other, then $S(D)$ contains a unique leaf node corresponding to each x_i. To ensure that none of the strings in D are substrings of each other we might need to append a new symbol $\$_i$ to each string x_i in the dictionary.

Given a text and a static or dynamic dictionary of patterns the problem of reporting all occurrences of any pattern of the dictionary in the text has been well studied (Aho and Corasick, 1975; Amir et al., 1994). Here we concentrate on the static dictionary matching problem and sketch an algorithm which is close in spirit to our string kernel algorithm.

The basic idea of our algorithm is as follows: Given strings x and y we first construct $S(\{x, y\})$. We then show that to compute the kernel $k(x, y)$ it is sufficient to annotate the nodes of this joint suffix tree. We then show how the annotation can be performed by (conceptually) constructing the suffix tree $S(\{x, y\} \bigcup D)$.

We first prove the following technical lemma:

Lemma 5.6 *Let x and y be strings such that v, c, and c' be the matching statistics of x with respect to y. The suffix tree $S(\{x, y\})$ contains a node corresponding to $x[i:\overline{v_i}]$ for $i = 1 \ldots |x|$.*

Proof By definition of matching statistics, $x[i : \overline{v_i}]$ occurs as a substring of y while $x[i : \overline{v_i} + 1]$ does not. This implies that in $S(\{x, y\})$ we can share the path corresponding to $x[i : \overline{v_i}]$ but to represent the string $x[i : \overline{v_i}+1]$ and its superstrings we will need to introduce a node. ∎

Let $\overline{w} \in \text{nodes}(S(\{x, y\}))$, we define $\text{lvs}_x(\overline{w})$ and $\text{lvs}_y(\overline{w})$ as the number of times w occurs as a substring of x and y respectively. The following lemma provides an alternative view of our string kernel by characterizing it in terms of the nodes of $S(\{x, y\})$.

Lemma 5.7 *Given the strings x and y the string kernel defined in (5.2) can be computed as*

$$k(x, y) = \sum_{\overline{t} \in \text{nodes}(S(\{x,y\}))} \text{lvs}_x(\overline{t}) \cdot \text{lvs}_y(\overline{t}) \cdot W(\{x, y\}, t) \tag{5.13}$$

where $W(\{x, y\}, t)$ is defined as in theorem 5.5.

Proof We sketch the outline of the proof. We first observe that all strings which share a common ceil (say $\overline{t}$) in $S(\{x, y\})$ occur $\text{lvs}_x(\overline{t})$, $\text{lvs}_y(\overline{t})$ number of times in x and y respectively. From the previous lemma we know that $x[i : \overline{v_i}]$ occurs as a node in this joint suffix tree. These two observations allow us to bracket the terms efficiently to compute the kernel. ∎

In order to compute the kernel $k(x, y)$ all that remains is to compute $W(\{x, y\}, t)$ efficiently for all nodes $\overline{t} \in \text{nodes}(S(\{x, y\}))$. Let $\overline{t}$ be a node in $S(\{x, y\})$ and $\overline{w}$ be its parent. In order to compute $W(\{x, y\}, t)$ we just need to sum up the weights of all strings from the dictionary D which end on the edge connecting $\overline{w}$ to $\overline{t}$. This can be easily computed by keeping track of the weights on the path leading from string w to string t in the suffix tree $S(D)$. In fact this computation can be performed in constant time if we annotate each node in $S(D)$ with the sum of the weights on the path from the root to the node.

A conceptually easy way to think of the above procedure is as follows: Assume that we construct $S(\{x, y\} \bigcup D)$ (this can be done in $O(|x| + |y|)$ time if $S(D)$ is already given (Amir et al., 1994)). Each $\overline{t} \in \text{nodes}(S(\{x, y\}))$ is also a node in $S(\{x, y\} \bigcup D)$. For each such node t let $\overline{w}$ be its parent in $S(\{x, y\})$. We compute the weights on the path from $\overline{w}$ to $\overline{t}$ in $S(\{x, y\} \bigcup D)$. As before, this computation can be performed in constant time by annotating the nodes of $S(D)$ beforehand.

Wildcards Mismatches can be taken into account if we allow for a rather simple modification of the above algorithm (albeit at a somewhat higher runtime): treat wildcards as an additional (special) symbol and build the suffix tree based on the set of dictionary terms. When computing the matching statistics algorithm and parsing the suffix tree, allow for multiple paths, that is, if a vertex contains a wildcard `'*'` and the symbol `'A'` in its children, then when observing `'A'` both paths would need

to be followed. This means that the algorithm now is more expensive, exactly by the number of concurrent paths that need to be taken into account simultaneously.

Another approach to account for wildcards is to build compact finite state automata (FSA) to represent the dictionary patterns (Navarro and Raffinot, 1999). This can be viewed as a generalization of suffix trees with failure functions.

5.4.4 Generic Weights

In the case of generic weights, where individual subwords are assigned weights independent of each other, we proceed as in the dictionary approach. The main difference is that here the dictionary is only given implicitly. Hence denote by $X = \{x_1, \ldots, x_m\}$ the set of all arguments to be used for kernel evaluation purposes.

Compute w_s for all substrings of x_i and treat the result as a dictionary. This reduces the present case to the situation above, which may result in superlinear time to annotate $S(D)$, due to the significantly larger number of non-zero weights. However, this should not be surprising, since we need to read each of the weights used in computing the kernel at least once.

Our approach offers the option to assign weights according to the frequency of occurrence of strings in a text. For this purpose, build a suffix tree first, use the latter to obtain string frequency counts, then compute the according weights, and proceed by annotating the suffix tree. This allows one to implement kernels, such as the ones proposed by Leopold and Kindermann (2002), efficiently.

5.5 Optimization and Prediction

Beyond the actual evaluation of scalar products, we can extend our techniques to compute *linear combinations* of kernel functions efficiently. Crucial to this context is the fact that the computation of $k(x, y)$ is *asymmetric* in x and y insofar as a suffix tree for y is computed, whereas x is merely compared to $S(y)$.

In the following we extend this idea by manipulating the suffix tree of the set of SVs directly, which will allow us to obtain $O(|x|)$ time algorithms even for combinations of kernels.

5.5.1 Linear Time Prediction

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$ be the set of SVs. Recall that for prediction in a support vector machine, we need to compute

$$f(x) = \sum_{i=1}^{m} \alpha_i y_i k(x_i, x), \tag{5.14}$$

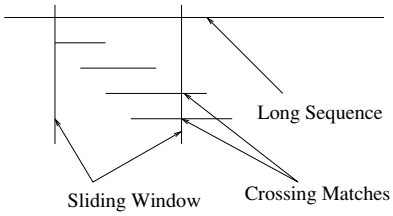


Figure 5.3 In a sliding window classifier we extend only those matches which cross the current window boundary.

which implies that we need to combine the contribution due to matching substrings from each one of the SVs. Using the definition of the string kernel we can expand f as

$$f(x) = \sum_{i=1}^{m} \sum_{s \sqsubseteq x_i} \sum_{s' \sqsubseteq x} \alpha_i y_i w_s \delta_{s,s'}. \tag{5.15}$$

Previously each substring match contributed a weight of w_s to the kernel. Now substring s from SV x_i contributes a weight of $\alpha_i y_i w_s$. This suggests a simple strategy akin to the one we used for position-dependent weights.

We first construct $S(\mathcal{X})$ in linear time by using the algorithm of Amir et al. (1994). Next, each leaf in $S(\mathcal{X})$ arising from x_i, is assigned the weight $\alpha_i y_i$ rather than 1. All we need to do now is define at each node $\overline{w}$ an updated weight, given by

Weighted leaf nodes

$$\text{weight}(\overline{w}) = \sum_{i=1}^{m} \alpha_i y_i \, \text{lvs}_{x_i}(\overline{w}) \tag{5.16}$$

where $\text{lvs}_{x_i}(\overline{w})$ denotes the number of leaves of $S(x_i)$ at node $\text{ceil}(\overline{w})$.

This allows us to take the contribution of all SVs into account simultaneously. A straightforward application of the matching statistics algorithm of Chang and Lawler (1994) shows that we can find the matching statistics of x with respect to all strings in $\mathcal{X}$ in $O(|x|)$ time. Now (5.6) can be applied verbatim to compute $f(x)$. See Vishwanathan (2002) for further details and a proof.

In summary, we can classify texts in linear time regardless of the size of the training set. This makes SVM for large-scale text categorization practically feasible. However, note that this ability comes at a cost: we have to store $S(\mathcal{X})$, which has memory requirements linear in the size of the SVs.

5.5.2 Sliding Windows

Now assume that we want to classify subsequences of a long string, for example, a DNA sequence d. The typical approach in this case is to take $d[i : i + n]$, where n

specifies the window size, and compute $f(d[i:i+n])$ for all $i \leq |d| - n$ *individually* without taking the correlation between the function values $f(d[i:i+n])$ into account.

This is highly wasteful, since the difference between two adjoint strings, that is, $d[i:i+n]$ and $d[i+1:i+n+1]$ is just given by $d[i]$ and $d[i+n+1]$. All other symbols and their order are *identical*. This means that a large part of what has been computed for $d[i:n]$ can be reused for its successor $d[i+1:n+1]$.

As before, we merge the suffix trees of all SVs into a master suffix tree denoted by $S(\mathcal{X})$. Next we compute the matching statistics of the entire long sequence x with respect to $S(\mathcal{X})$. Let c_i^j and v_i^j denote the matching statistics of the jth character in the ith window, $x[i:i+n]$. Since a match cannot extend beyond the boundary of the window it is clear that $v_i^j = \min(n-j, v_{i+j})$ while $c_i^j = \text{ceil}(x[i:\overline{v}_i^j])$. This means that we need not parse $S(\mathcal{X})$ again and again for computing the matching statistics of each window, which in turn will lead to implementation speedups.

An even more efficient strategy is to keep track of those strings whose matches end on the current window boundary. In the worst case there can be n of them, but in general the number will be typically smaller than n. When we perform estimation for the next window we need to extend only those matches which ended on the previous window boundary. If the number of matches that we need to extend is less than n, then we can perform estimation for the next window in sublinear time (that is less than $O(n)$ time). Figure 5.3 depicts this strategy.

In general, v_i denotes the length of the longest match at location i of string x. This implies that the match corresponding to $x[i:\overline{v_i}]$ will cross at most v_i window boundaries and hence will need to be extended exactly v_i times. Thus the total time required to compute the sliding window kernel is proportional to $\sum_i \min(v_i, n)$ which is typically much smaller than $n|x|$.

5.6 Experimental Results

For a proof of concept we tested our approach on a remote homology detection problem[1] (Jaakkola et al., 2000) using Stafford Noble's SVM package[2] as the training algorithm. A length weighted kernel was used and we assigned weights $w_s = \lambda^{|s|}$ for all substring matches of length greater than 3 regardless of triplet boundaries. To evaluate performance we computed the ROC_{50} scores.[3]

1. Details and data available from www.cse.ucsc.edu/research/compbio/discriminative.
2. Available from www.cs.columbia.edu/compbio/svm.
3. The ROC_{50} score (Gribskov and Robinson, 1996; Leslie et al., 2002) is the area under the receiver operating characteristic curve (the plot of true positives as a function of false positives) up to the first 50 false positives. A score of 1 indicates perfect separation of positives from negatives, whereas a score of 0 indicates that none of the top 50 sequences selected by the algorithm were positives.

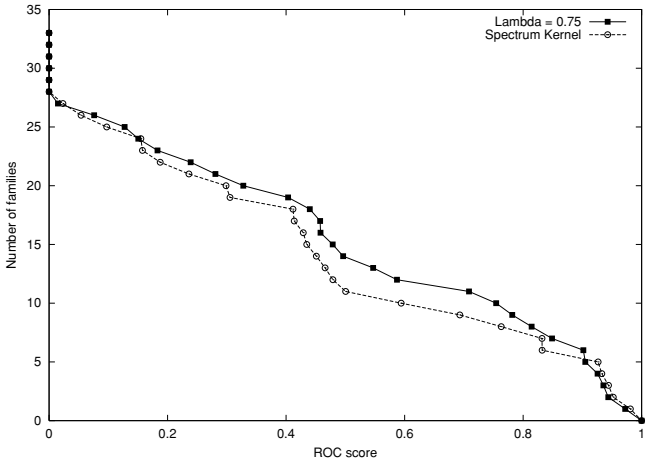


Figure 5.4 Total number of families for which an SVM classifier exceeds a ROC_{50} score threshold.

Being a proof of concept, we did not try to tune the soft margin SVM parameters (the main point of the chapter being the introduction of a novel means of evaluating string kernels efficiently rather than applications—a separate paper focusing on applications is in preparation).

Figure 5.4 contains the ROC_{50} scores for the spectrum kernel with $k = 3$ (Leslie et al., 2002) and our string kernel with $\lambda = 0.75$. We tested with $\lambda \in \{0.25, 0.5, 0.75, 0.9\}$ and report the best results here. As can be seen, our kernel outperforms the spectrum kernel based on the total number of families for which a classifier based on our kernel exceeds a ROC_{50} score threshold.

It should be noted that this is the first method to allow users to specify weights rather arbitrarily for all possible lengths of matching sequences and still be able to compute kernels at $O(|x| + |x'|)$ time, plus, to predict on new sequences at $O(|x|)$ time, once the set of SVs is established.[4]

5.7 Conclusion

We have shown that string kernels need not come at a superlinear cost in SVMs and that prediction can be carried out at cost linear only in the length of the argument,

4. Leslie et al. (2002) obtain an $O(k|x|)$ algorithm in the (somewhat more restrictive) case of $w_s = \delta_k(|s|)$.

thus providing optimal runtime behavior. Furthermore, the same algorithm can be applied to trees.

The methodology pointed out in this chapter has several immediate extensions: for instance, we may consider coarsening levels for trees by removing some of the leaves. For not too-unbalanced trees (we assume that the tree shrinks at least by a constant factor at each coarsening) computation of the kernel over all coarsening levels can then be carried out at cost, still linear in the overall size of the tree. The idea of coarsening can be extended to approximate string matching. If we remove characters, this amounts to the use of wildcards.

Likewise, we can consider the strings generated by finite-state machines and thereby compare the finite-state machines themselves. This leads to kernels on automata and other dynamical systems. More details and extensions can be found in Vishwanathan (2002).

Acknowledgments

We thank Asa Ben-Hur, Eleazar Eskin, Patrick Haffner, and Bob Williamson for comments and suggestions. We thank Jean-Philippe Vert and Koji Tsuda for carefully going over the draft and pointing out many errors and helping us improve the presentation significantly. This research was supported by a grant of the Australian Research Council. S.V.N.V. thanks Trivium India Software and Netscaler Inc. for their support.

6 Local Alignment Kernels for Biological Sequences

Jean-Philippe Vert
Hiroto Saigo
Tatsuya Akutsu

We present a family of kernels for strings based on the detection of local alignments, a widely used principle to compare biological sequences. These kernels belong to the family of convolution kernels introduced by Haussler (1999), and can be implemented with dynamic programming. We highlight the relations between these kernels and sequence alignment scores used in bioinformatics, and propose a solution to overcome the issue of diagonal dominance they suffer from. When tested on their ability to detect protein homology at the superfamily level on a benchmark data set, they outperform other state-of-the-art methods.

6.1 Introduction

With the advent of high-throughput sequencing technologies, biological sequences are accumulating at an unprecedented pace in databases. The need to analyze, compare, and annotate these sequences is more than ever a central problem in postgenomics. There has recently been a growing interest in the use of kernel methods to process sequences, particularly for classification of sequences with Support Vector Machines (SVMs). This trend has been accompanied by a growing interest in the development of kernel functions for strings, which form the basic ingredient that any kernel method is based on. Examples of string kernels include the Fisher kernel (Jaakkola et al., 2000), spectrum kernel (Leslie et al., 2002), mismatch kernel (Leslie et al., 2003b), pairwise kernel (Liao and Noble, 2002), and the string kernel proposed by Lodhi et al. (2002).

A kernel function can often be thought of as a measure of similarity. Different kernels correspond to different notions of similarity. For example, each string kernel

mentioned above corresponds to one notion of similarity between strings, such as the similarity of Fisher score vectors with respect to a hidden Markov model (HMM) for the Fisher kernel, or the similarity of the frequency of small blocks in the sequences for the spectrum kernel.

Independent of kernel methods, the problem of assessing the similarity between biological sequences has been the object of much investigation in computational biology over the last three decades. The notion of local alignments (which we review below) has emerged as a powerful approach to detecting evolutionary relationships between sequences, and the measure of the best local alignment with the Smith-Waterman algorithm (Smith and Waterman, 1981) or its fast heuristics gapped BLAST, PSI-BLAST (Altschul et al., 1997), and FASTA (Pearson, 1990) is nowadays the method of choice to measure the similarity between sequences.

The main motivation behind this work is the observation that the notions of similarity between sequences defined by the string kernels developed so far differ considerably from the notion of similarity based on local alignment. This suggests that biologically more relevant kernels might be defined if the notion of similarity they induce are more similar to those found useful by the computational biology community, namely the similarity based on local alignment scoring.

The most direct way to reconcile kernel methods and classic measures of similarity for biological sequences would be to consider a classic measure of similarity such as the Smith-Waterman score as a kernel. However, it turns out that this choice is not a valid kernel because it violates the condition of positive definiteness for certain choices of parameters, as we show in the experimental part. Still, following the avenue paved by Haussler (1999) and Watkins (2000), we propose a family of string kernels based on the scoring of local alignments. Two strings are similar with these kernels when they have many high-scoring local alignments, which is the property that motivates this work.

We present promising results on a benchmark problem of remote homology detection, where the new kernels outperform all other string kernels tested. However, this performance is only obtained at the price of performing an operation on the kernel values, which otherwise can be exponentially small even for sequences with meaningful similarities.

The chapter is organized as follows. In sections 6.2 and 6.3 we recall the two main ingredients we need, namely the classic notions of local alignment and Smith-Waterman score on the one hand, and the idea of convolution kernels on the other hand. In section 6.4 we introduce a family of convolution kernels called local alignment kernels and present some of their properties. Their implementation is discussed in section 6.5 and the issue of diagonal dominance is raised in section 6.6. We conclude with experimental results on the problem of detecting protein remote homology in sections 6.7 and 6.8.

6.2 Sequence Local Alignment

In this section we introduce basic notations and recall the classic notion of sequence alignment and Smith-Waterman local alignment score.

6.2.1 Basic Notations

Let $\mathcal{A}$ be a finite set called the *alphabet*, typically the set of 20 amino acids or of 4 nucleotides {A,C,G,T}. Elements of the alphabet are called *letters*. A *string* of length $n \geq 0$ is a concatenation of n letters, $\mathbf{x} = x_1 \dots x_n$. The empty string (of length 0) is denoted by ϵ, and the set of all strings is denoted by $\mathcal{X} = \{\epsilon\} \cup \bigcup_{n=1}^{\infty} \mathcal{A}^n$. The length of a string $\mathbf{x}$ is denoted $|\mathbf{x}|$. The concatenation of two strings $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2$ is written $\mathbf{xy}$.

6.2.2 Alignments and Smith-Waterman Score

With these basic notations we can introduce the notion of alignment that plays an essential role below:

Sequence alignment

Definition 6.1 *An* alignment (with gaps) π *of* $p \geq 0$ *positions between two sequences* $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2$ *is a pair of* p*-tuples:*

$$\pi = ((\pi_1(1), \dots, \pi_1(p)), (\pi_2(1), \dots, \pi_2(p))) \in \mathbb{N}^{2p}$$

that satisfies

$$1 \leq \pi_1(1) < \pi_1(2) < \dots < \pi_1(p) \leq |\mathbf{x}|,$$
$$1 \leq \pi_2(1) < \pi_2(2) < \dots < \pi_2(p) \leq |\mathbf{y}|.$$

The simple idea behind this formal definition is that the alignment encodes p positions in each sequence that are aligned to each other, that is, the $\pi_1(i)$th letter of $\mathbf{x}$ is aligned to the $\pi_2(i)$th letter of $\mathbf{y}$ for $i = 1, \dots, p$. A convenient way to represent an alignment is to write the two sequences one above the other, by placing the p aligned positions on the same columns, and inserting the symbol '$-$' in both sequences whenever necessary to ensure that only aligned letters are on the same column (this representation is unique up to the choice of the position where the symbol '-' is inserted between two consecutively aligned positions).

As an example, if $\mathbf{x}$ = `GAATCCG` and $\mathbf{y}$ = `GATTGC`, then the 4-letter alignment $\pi = ((1, 2, 4, 6), (1, 3, 4, 5))$ is represented as follows:

```
G-AATCCG-
GAT-T-G-C
```

q The notion of alignment stems from the observation that the transformation of sequences during evolution can roughly be described by substitutions of single

letters, deletions, or insertions. As a result, a natural way to compare homologous proteins is to align them in order to detect the conserved regions.

Let us denote by $\Pi(\mathbf{x}, \mathbf{y})$ the set of all possible alignments between two sequences $\mathbf{x}$ and $\mathbf{y}$, and by $|\pi|$ the number of positions aligned by the alignment $\pi \in \Pi(\mathbf{x}, \mathbf{y})$. In order to evaluate how "good" an alignment $\pi \in \Pi(\mathbf{x}, \mathbf{y})$ is, and eventually find the best one, various scoring schemes have been developed. We now present one of the most widely used scoring schemes, often referred to as *local alignment score*. It is defined in terms of a substitution matrix $S \in \mathbb{R}^{\mathcal{A} \times \mathcal{A}}$ and a gap penalty function $g : \mathbb{N} \to \mathbb{R}$ such that $g(0) = 0$ as follows:

Definition 6.2 *The local alignment score of an alignment $\pi \in \Pi(\mathbf{x}, \mathbf{y})$ is equal to*

$$s_{S,g}(\pi) := \sum_{i=1}^{|\pi|} S\left(x_{\pi_1(i)}, y_{\pi_2(i)}\right) - \sum_{i=1}^{|\pi|-1} \left[g(\pi_1(i+1) - \pi_1(i)) + g(\pi_2(i+1) - \pi_2(i))\right]. \tag{6.1}$$

In other words, the local alignment score of an alignment π is the sum of the substitution scores between the letters at the aligned positions, minus the sum of the gap penalty when '-' symbols must be inserted between aligned positions. The name "local alignment" comes from the fact that no gap penalty is counted before the first and after the last aligned letters.

We can now define the local alignment score between sequences:

Smith-Waterman score

Definition 6.3 *The local alignment score or Smith-Waterman score (denoted SW score below) between two sequences $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2$ is the local alignment score of their best alignment, that is:*

$$SW_{S,g}(\mathbf{x}, \mathbf{y}) := \max_{\pi \in \Pi(\mathbf{x}, \mathbf{y})} s_{S,g}(\pi). \tag{6.2}$$

The SW score is a widely used measure of similarity between DNA or protein sequences. It can be computed with a complexity $O(|\mathbf{x}||\mathbf{y}|)$ by dynamic programming with the Smith-Waterman algorithm (Smith and Waterman, 1981), and lower complexity heuristics are implemented in the widely used softwares gapped BLAST (Altschul et al., 1997) and FASTA (Pearson, 1990).

6.2.3 Is the SW Score a Valid String Kernel?

A natural question at this point is the following: is the SW score (6.2) a valid kernel for string? Remember that a function $k : \mathcal{X}^2 \to \mathbb{R}$ is a valid kernel if it is symmetric, that is, $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$ for any $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2$, and positive definite, that is:

$$\forall n \in \mathbb{N}, \forall (\mathbf{x}_1, \ldots, \mathbf{x}_n, a_1, \ldots, a_n) \in \mathcal{X}^n \times \mathbb{R}^n, \quad \sum_{i,j=1}^{n} a_i a_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0. \tag{6.3}$$

String kernels

We simply call such functions *string kernels* below. To the best of our knowledge, there is no general result that states for which parameters S and g the local

alignment score $SW_{S,g}$ is a string kernel. As the following toy example shows, there are cases where the SW score is a string kernel:

Proposition 6.4 *Let $g = 0$ (no gap penalty) and S be a substitution matrix null except for one letter $a \in \mathcal{A}$ on the diagonal, that is, $S(a, a) = 1$ and $S(u, v) = 0$ except if $u = v = a$. Then the score $SW_{S,g}$ is a string kernel.*

Proof Consider two sequences $\mathbf{x}$ and $\mathbf{y}$ with, respectively, m_x and m_y occurrences of the letter a. With no gap penalty and no contribution to the score of aligned letters except for pairs of a, the highest score is obtained when as many a as possible are aligned, that is, $\min(m_x, m_y)$. Each aligned pair of a adds 1 to the score, which shows that

$$SW_{S,g}(\mathbf{x}, \mathbf{y}) = \min(m_x, m_y).$$

This is clearly a valid kernel, as it can be written as an inner product $\phi(\mathbf{x}).\phi(\mathbf{y})$ between two vector representations of $\mathbf{x}$ and $\mathbf{y}$, for instance, by the mapping $\phi : \mathcal{X} \to \mathbb{R}^{\mathbb{N}}$ defined by $\phi(\mathbf{x})_i = 1$ if $i \leq m_x$, 0 otherwise. ■

On the other hand, for more classic parameter S and g, we observe empirically (see section 6.7) that the SW score can lack positive definiteness. This motivates the study of convolution kernels in the next section, and the development of string kernels based on local alignment scores in section 6.4.

6.3 Convolution Kernels

In this section we recall the notion of convolution kernels for strings introduced in a more general framework by Haussler (1999). Roughly speaking, convolution is a powerful operation to combine two string kernels into a single one, particularly suited to the detection of local similarities between sequences. More formally, they are defined as follows:

Convolution kernel

Definition 6.5 *Let k_1 and k_2 be two functions $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The convolution of k_1 and k_2, denoted by $k_1 \star k_2$, is the following function:*

$$\forall (\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2, \quad k_1 \star k_2(\mathbf{x}, \mathbf{y}) := \sum_{\mathbf{x}_1\mathbf{x}_2 = \mathbf{x}, \mathbf{y}_1\mathbf{y}_2 = \mathbf{y}} k_1(\mathbf{x}_1, \mathbf{y}_1) k_2(\mathbf{x}_2, \mathbf{y}_2). \tag{6.4}$$

Haussler (1999) proved the following important result:

Theorem 6.6 *The convolution of two string kernels is a string kernel.*

This theorem implies that we can build complex string kernels by convolving simple ones. Moreover, it can easily be checked that the convolution operation is

associative, and that the convolution of p kernels $k_1, \cdots, k_p$ is the following string kernel:

$$\forall (\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2, \quad k_1 \star \cdots \star k_p(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{x}_1 \cdots \mathbf{x}_p = \mathbf{x}, \mathbf{y}_1 \cdots \mathbf{y}_p = \mathbf{y}} k_1(\mathbf{x}_1, \mathbf{y}_1) \cdots k_p(\mathbf{x}_p, \mathbf{y}_p).$$

Convolution kernels were introduced by Haussler (1999) in a more general context than string kernels, and were, for instance, successfully applied to define kernels for trees with applications in natural language processing (Collins and Duffy, 2002). In the context of biological sequences, Haussler (1999) showed that the probability $P(\mathbf{x}, \mathbf{y})$ of a pair of sequences under a pair HMM (Durbin et al., 1998) is a convolution of basic string kernel and is therefore a valid string kernel, under some assumptions on the model parameters. This result, which was proved independently by Watkins (2000), makes a first link between string kernels and string alignment scores. Indeed the probability $P(\mathbf{x}, \mathbf{y})$ studied by Haussler (1999) and Watkins (2000) is related to the score obtained by global alignment of two strings with the Needleman-Wunsch algorithm (Needleman and Wunsch, 1970), at least if the probability is replaced by a log-odds score which takes the form

Pair HMM

$$s(\mathbf{x}, \mathbf{y}) = \log \frac{P(\mathbf{x}, \mathbf{y})}{q(\mathbf{x}) q(\mathbf{y})},$$

where q is a background model (Durbin et al., 1998).

The results, however, are of little practical interest for at least two reasons. First, it is commonly recognized in computational biology that local alignment scores like the SW score are more relevant than global alignment scores to assess the similarity between sequences. Second, the probabilities of two sequences under a pair HMM model are usually much too small (typically 10^{-100} to 10^{-1000}) to be of any interest as kernels. The objects of the next two sections are to overcome both these issues, first by proposing a convolution kernel based on local alignment score (section 6.4), and then on making it useful in real-world application (section 6.6).

6.4 Local Alignment Kernels

Following the approach pioneered by Haussler (1999), we define in this section a complex string kernel by convolution and sum of simple kernels. Let us therefore start by defining three basic string kernels, which will serve as "basic blocks" to derive a more complex kernel. To do so, we assume a substitution matrix S and a gap penalty function g are given, and we define the three kernels as functions of S and g.

The first basic kernel, denoted by k_0, is a trivial kernel constantly equal to 1, that is:

$$\forall (\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2, \quad k_0(\mathbf{x}, \mathbf{y}) := 1. \tag{6.5}$$

This kernel is useful to compare the parts of two sequences that don't contribute to the local alignment score, that is, the substrings before and after the alignment.

The second basic kernel, which we call k_a, is used to quantify the similarity of aligned letters. It is defined as follows:

$$\forall(\mathbf{x},\mathbf{y}) \in \mathcal{X}^2, \quad k_a^{(\beta)}(\mathbf{x},\mathbf{y}) := \begin{cases} 0 & \text{if } |\mathbf{x}| \neq 1 \text{ or } |\mathbf{y}| \neq 1, \\ \exp\left(\beta S(\mathbf{x},\mathbf{y})\right) & \text{otherwise,} \end{cases} \tag{6.6}$$

where $\beta \geq 0$ is a parameter. Observe that it is null as soon as one of the sequences is not exactly reduced to a single letter. Observe also that this is only a string kernel for the values of β which ensure that the matrix $\left(\exp\left(\beta S(a,b)\right)\right)_{a,b\in\mathcal{A}}$ is positive definite. This is the case whatever $\beta \geq 0$ iff the matrix $\left(S(a,b)\right)_{a,b\in\mathcal{A}}$ is conditionally positive definite (Berg et al., 1984), which can be checked case by case (this holds in particular if the matrix $\left(S(a,b)\right)_{a,b\in\mathcal{A}}$ is positive definite).

Finally, to translate the penalty gap model, we define the third basic string kernel, which we denote k_g, as follows:

$$\forall(\mathbf{x},\mathbf{y}) \in \mathcal{X}^2, \quad k_g^{(\beta)}(\mathbf{x},\mathbf{y}) := \exp\left[\beta\left(g(|\mathbf{x}|) + g(|\mathbf{y}|)\right)\right], \tag{6.7}$$

where $\beta \geq 0$ is the same parameter as in (6.6), and g is the gap penalty function. k_g is a valid string kernel, because it can be written as a scalar product $k_g^{(\beta)}(\mathbf{x},\mathbf{y}) = \phi_g^{(\beta)}(\mathbf{x}).\phi_g^{(\beta)}(\mathbf{y})$ between 1-dimensional vectors given by $\phi_g^{(\beta)}(\mathbf{x}) = \exp\left(\beta g(|\mathbf{x}|)\right)$.

Let us now combine these three basic kernels by convolution. For any fixed $n > 0$, consider first the following string kernel:

$$k_{(n)}^{(\beta)} := k_0 \star \left(k_a^{(\beta)} \star k_g^{(\beta)}\right)^{(n-1)} \star k_a^{(\beta)} \star k_0.$$

This kernel quantifies the similarity between two strings $\mathbf{x}$ and $\mathbf{y}$ based on local alignments of exactly n residues. Indeed the convolution operation sums up the contributions of all possible decompositions of $\mathbf{x}$ and $\mathbf{y}$ simultaneously into an initial part (whose similarity is measured by k_0), a succession of n aligned residues (whose similarity is measured by $k_a^{(\beta)}$) possibly separated by $n-1$ gaps (whose similarity is measured by $k_g^{(\beta)}$), and a terminal part (whose similarity is measured by k_0). For $n = 0$ (no residue aligned), we simply define the kernel $k_{(0)}^{(\beta)} = k_0$.

In order to compare two sequences through all possible local alignments, it is necessary to take into account alignments with different numbers n of aligned residues. A simple solution is to sum up the contributions of all kernels $k_{(n)}^{(\beta)}$ for $n \geq 0$, which results in the following *local alignment kernel* (which we call LA kernel below):

Local alignment kernel

$$k_{LA}^{(\beta)} := \sum_{i=0}^{\infty} k_{(i)}^{(\beta)}. \tag{6.8}$$

Note that for any pair of finite-length sequences $(\mathbf{x},\mathbf{y}) \in \mathcal{X}^2$ the right-hand term of (6.8) estimated at $(\mathbf{x},\mathbf{y})$ is a convergent series, because it has only a finite number of non-null terms. Therefore $k_{LA}^{(\beta)}$ is defined as a pointwise limit of string kernels,

and is therefore a string kernel by closure property of the class of positive definite kernels under pointwise limit (Berg et al., 1984).

The following theorem, whose proof is postponed to appendix A, highlights the relations between the LA string kernel, the local alignment scores of all possible alignments, and the SW score:

Theorem 6.7 *The LA kernel (6.8) is expressed in terms of local alignment scores as follows:*

$$\forall(\mathbf{x},\mathbf{y}) \in \mathcal{X}^2, \quad k_{LA}^{(\beta)}(\mathbf{x},\mathbf{y}) = \sum_{\pi \in \Pi(\mathbf{x},\mathbf{y})} \exp\left(\beta s_{S,g}(\mathbf{x},\mathbf{y},\pi)\right). \tag{6.9}$$

The SW score (6.2) is related to the LA kernel by the following equality:

$$\forall(\mathbf{x},\mathbf{y}) \in \mathcal{X}^2, \quad \lim_{\beta \to +\infty} \frac{1}{\beta} \ln k_{LA}^{(\beta)}(\mathbf{x},\mathbf{y}) = SW_{S,g}(\mathbf{x},\mathbf{y}). \tag{6.10}$$

These equations clarify the link between the LA kernel (6.8) and the SW score (6.2), and highlight why the SW score might fail to be a valid string kernel. First, the SW score only keeps the contribution of the best local alignment to quantify the similarity between two sequences, instead of summing up the contributions of all possible local alignments like the LA kernel does. Second, the SW score is the logarithm of this alignment score, and taking the logarithm is usually an operation which does not preserve the property of being positive definite (Berg et al., 1984).

6.5 Kernel Computation

A naive computation of the LA kernel using (6.9) would require a sum over $|\Pi(\mathbf{x},\mathbf{y})|$ alignments, resulting in a complexity exponential with respect to $|\mathbf{x}|$ and $|\mathbf{y}|$. In this section we show that this expression can be factorized and computed using dynamic programming by a slight modification of the Smith-Waterman algorithm, at least if the gap penalty function is affine. This efficient computation is summarized in the following theorem:

Theorem 6.8 *Let $(\mathbf{x},\mathbf{y}) \in \mathcal{X}^2$ be two strings, and g be an affine gap penalty function:*

$$\begin{cases} g(0) & = 0, \\ g(n) & = d + e(n-1) \text{ if } n \geq 1, \end{cases} \tag{6.11}$$

then the LA kernel $k_{LA}^{(\beta)}(\mathbf{x},\mathbf{y})$ between $\mathbf{x}$ and $\mathbf{y}$ is equal to

$$k_{LA}^{(\beta)}(\mathbf{x},\mathbf{y}) = 1 + X_2(|\mathbf{x}|,|\mathbf{y}|) + Y_2(|\mathbf{x}|,|\mathbf{y}|) + M(|\mathbf{x}|,|\mathbf{y}|),$$

where $M(i,j), X(i,j), Y(i,j), X_2(i,j)$, *and* $Y_2(i,j)$ *for* $0 \leq i \leq |\mathbf{x}|$, *and* $0 \leq j \leq |\mathbf{y}|$ *are recursively defined by the following equations:*

$$\begin{cases} M(i,0) = M(0,j) = 0, \\ X(i,0) = X(0,j) = 0, \\ Y(i,0) = Y(0,j) = 0, \\ X_2(i,0) = X_2(0,j) = 0, \\ Y_2(i,0) = Y_2(0,j) = 0, \end{cases}$$

Dynamic programming

and for $i = 1, \ldots, |\mathbf{x}|$ *and* $j = 1, \ldots, |\mathbf{y}|$*:*

$$\begin{cases} M(i,j) & = \exp(\beta S(x_i, y_j)) \\ & \qquad [1 + X(i-1,j-1) + Y(i-1,j-1) + M(i-1,j-1)], \\ X(i,j) & = \exp(\beta d) M(i-1,j) + \exp(\beta e) X(i-1,j), \\ Y(i,j) & = \exp(\beta d) [M(i,j-1) + X(i,j-1)] + \exp(\beta e) Y(i,j-1), \\ X_2(i,j) & = M(i-1,j) + X_2(i-1,j), \\ Y_2(i,j) & = M(i,j-1) + X_2(i,j-1) + Y_2(i,j-1). \end{cases}$$

The proof of theorem 6.8 is postponed to appendix B. One can observe that the Smith-Waterman algorithm (Smith and Waterman, 1981) to compute the SW score is obtained from these equations by replacing each addition by a max operation, and taking the logarithm of the result divided by β.

Rational kernels

An equivalent way to describe the implementation of the LA kernel is through the weighted finite-state transducer (WFST) shown in figure 6.1 (see the legend for an explanation). The implementation as a WFST is possible because each basic kernel we defined can be implemented using such automata, and because the addition and convolution of WFST can be computed by WFST. Such kernels are called rational kernels (Cortes et al., 2003).

6.6 Diagonal Dominance Issue

In many cases of practical interest the LA kernel defined in (6.8) suffers from the diagonal dominance problem, namely the fact that $k_{LA}^{(\beta)}(\mathbf{x}, \mathbf{x})$ is easily orders of magnitudes larger than $k_{LA}^{(\beta)}(\mathbf{x}, \mathbf{y})$ for two different sequences $\mathbf{x}$ and $\mathbf{y}$, even though $\mathbf{x}$ and $\mathbf{y}$ might share interesting similarities. This is particularly evident for increasing values of the parameter β, because from (6.10)

$$\frac{k_{LA}^{(\beta)}(\mathbf{x}, \mathbf{x})}{k_{LA}^{(\beta)}(\mathbf{x}, \mathbf{x})} \sim \exp \beta \left(SW_{S,g}(\mathbf{x}, xb) - SW_{S,g}(\mathbf{x}, \mathbf{y}) \right)$$

when $\beta \to \infty$. In practice, it has been observed that SVMs don't perform well in this situation. Indeed diagonal dominance implies that different objects are almost orthogonal in the feature space, and it can be shown that instead of learning a

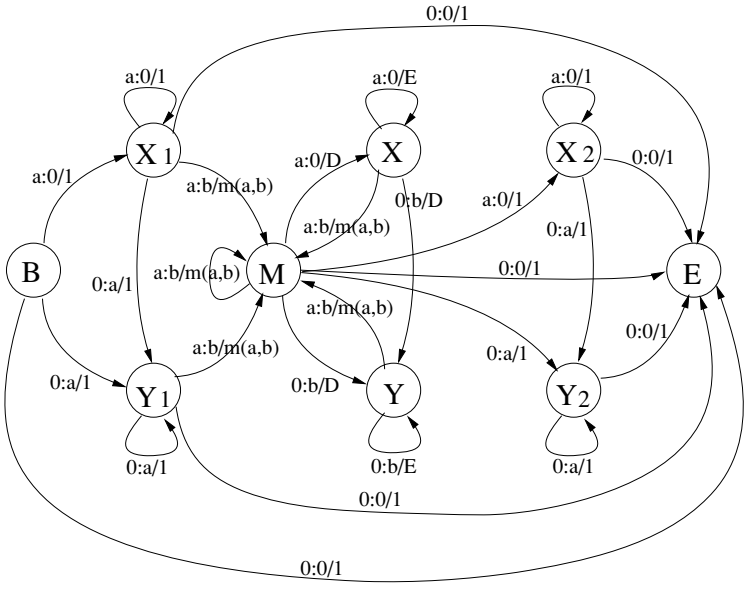


Figure 6.1 A weighted finite-state transducer is a directed graph with an initial state B and a terminal state E. To each edge is attached a label of the form $u : v/w$. The label corresponds to an action of processing an input sequence $\mathbf{x}$ and an output sequence $\mathbf{y}$, and a weight associated with the action. The action of a label $u : v/w$ is "read nothing (if $u = 0$) or one letter $a \in \mathcal{A}$ (if $u = a$) in $\mathbf{x}$, and output nothing (if $v = 0$) or one letter $b \in \mathcal{A}$ (if $v = b$) in $\mathbf{y}$". The weight associated with the action is w, with the conventions that $m(a, b) = \exp(\beta S(a, b))$, $D = \exp(\beta d)$, and $E = \exp(\beta e)$. A path from B to E is *compatible* with two sequences $\mathbf{x}$ and $\mathbf{y}$ if the successive actions performed on the edges along the path correspond to reading $\mathbf{x}$ and outputting $\mathbf{y}$. The score of a compatible path is the product of its edge weights. The kernel $k_{LA}^{(\beta)}(\mathbf{x}, \mathbf{y})$ is computed by the automaton shown on this picture as the sum of the scores of all compatible paths with $\mathbf{x}$ and $\mathbf{y}$.

classifier with good generalization performances, an SVM tends only to memorize the data in that case.

The exponential dependency of the LA kernel with respect to the β parameters suggests the following normalization, in order to remove the diagonal dominance when β increases:

$$\tilde{k}_{LA}^{(\beta)}(\mathbf{x}, \mathbf{y}) = \frac{1}{\beta} \ln k_{LA}^{(\beta)}(\mathbf{x}, \mathbf{y}). \tag{6.12}$$

An obvious problem with this operation is that the logarithm of a positive definite kernel is not a positive definite kernel in general (Berg et al., 1984). Still $\tilde{k}_{LA}^{(\beta)}$ has several interesting properties. It increases monotonically with $k_{LA}^{(\beta)}$, and by (6.9) it is always non-negative, because at least one local alignment between any two sequences has a non-negative score (at worst the score of aligning no residue is null). Besides, by (6.10), $\tilde{k}_{LA}^{(\beta)}$ behaves like the SW score for large β. Intuitively, $\tilde{k}_{LA}^{(\beta)}$ is therefore of the order of magnitude of the SW score but includes contributions from all possible local alignments.

Because $\tilde{k}_{LA}^{(\beta)}$ might not be a positive definite kernel, some care must be taken to ensure that the SVM converges to a large margin discrimination rule during learning. We tested two approaches to make the symmetric function $\tilde{k}_{LA}^{(\beta)}$ positive definite on a given training set of sequences, which we now describe.

Spectral translation

The first approach we propose is to add to the diagonal of the training Gram matrix a non-negative constant large enough to make it positive definite. In all experiments presented below we perform this operation by subtracting from the diagonal the smallest negative eigenvalue of the training Gram matrix, if there are negative eigenvalues. The resulting kernel, which we call LA-eig, is equal to $\tilde{k}_{LA}^{(\beta)}$ except eventually on the diagonal.

Empirical kernel map

We compare this approach to the method proposed in Schölkopf et al. (2002) which consists in working with the *empirical kernel map*. In this case, for a given training set $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of sequences, each possible sequence $\mathbf{x}$ is mapped to the n-dimensional vector $\left(\tilde{k}_{LA}^{(\beta)}(\mathbf{x}, \mathbf{x}_1), \ldots, \tilde{k}_{LA}^{(\beta)}(\mathbf{x}, \mathbf{x}_n)\right)^\top$. These vector representations are then used to train the SVM and predict the class of new sequences. The corresponding kernel between two sequences $\mathbf{x}$ and $\mathbf{y}$, which we call the LA-ekm kernel, is therefore equal to $\sum_{i=1}^{n} \tilde{k}_{LA}^{(\beta)}(\mathbf{x}, \mathbf{x}_i) \tilde{k}_{LA}^{(\beta)}(\mathbf{y}, \mathbf{x}_i)$. On the training set, this amounts to taking $\left(\tilde{k}_{LA}^{(\beta)}\right)^\top . \tilde{k}_{LA}^{(\beta)}$ as a new symmetric positive definite Gram matrix.

6.7 Experiments

The accuracy of an SVM combined with the LA-eig and LA-ekm kernels is evaluated with the benchmark procedure used in Liao and Noble (2002). Three other state-of-the-art methods were also run on this benchmark, in order to compare them with the method proposed in this chapter: SVM with a Fisher kernel (Jaakkola et al.,

2000), SVM-pairwise (Liao and Noble, 2002), and SVM with a mismatch kernel (Leslie et al., 2003b).

Remote homology detection

The problem is to classify protein domains into superfamilies in the structural classification of proteins (SCOP) (Murzin et al., 1995) version 1.53. The data, already used in Liao and Noble (2002),[1] consists of 4352 sequences extracted from the Astral database (similar sequences were removed with an E-value threshold of 10^{-25}), grouped into families and superfamilies. For each family, the protein domains within the family are considered positive test examples, and protein domains within the superfamily but outside the family are considered positive training examples. This yields 54 families with at least 10 positive training examples and 5 positive test examples. Negative examples for the family are chosen from outside of the positive sequences' fold, and were randomly split into training and test sets in the same ratio as the positive examples. For more details on this data set and the experiments, the reader is referred to Liao and Noble (2002) and Jaakkola et al. (2000).

To measure the quality of the methods, the receiver operating characteristic (ROC) scores, the ROC_{50} scores, and the median rate of false positives (RFP) are used. The ROC score is the normalized area under a curve that plots true positives against false positives for different possible thresholds for classification (Gribskov and Robinson, 1996). The ROC_{50} is the area under the ROC curve up to 50 false positives, and is considered a useful measure of performance for real-world application. The median RFP is the number of false positives scoring as high or better than the median-scoring true positives. These measures were used for evaluation in Jaakkola et al. (2000) and Liao and Noble (2002).

6.7.1 SVM

All methods based on SVM were tested with a common procedure, which we now describe. We used the Gist publicly available SVM software[2] which implements the soft margin optimization algorithm described in Jaakkola et al. (2000). For each kernel k to be tested, the following systematic preprocessing was performed. First, all points were normalized to unit norm in the feature space and separated from the origin by adding a constant to the kernel, resulting in the following kernel:

$$k_{norm}(\mathbf{x}, \mathbf{y}) = \frac{k(\mathbf{x}, \mathbf{y})}{\sqrt{k(\mathbf{x}, \mathbf{x})k(\mathbf{y}, \mathbf{y})}} + 1. \tag{6.13}$$

The necessity to separate the points from the origin stems from the fact that in the implementation we use, all separating hyperplanes are required to pass through the origin. Second, because most classification problems below are very unbalanced in the sense that the numbers of positive and negative examples are very different,

1. Available from `www.cs.columbia.edu/compbio/svm-pairwise` .
2. Available from `http://microarray.cpmc.columbia.edu/gist` .

we used a class-dependent regularization parameter. This amounts to adding to the diagonal a constant $0.02\alpha_+/0.02\alpha_-$ to all positive/negative examples, where α_+/α_- is the fraction of positive/negative examples in the training set (see Liao and Noble, 2002; Jaakkola et al., 2000, for details and justifications).

6.7.2 SVM-Pairwise Kernel

We reproduced the implementation of the best method tested in Liao and Noble (2002). For a given training set of size n, each sequence $\mathbf{x} \in \mathcal{X}$ is represented by a feature vector $U^{\text{pw}}(\mathbf{x})$ of dimension n, where each coordinate is the E-value of the SW scores $SW(\mathbf{x}, \mathbf{y}_i)$ $(1 \leq i \leq n)$ between $\mathbf{x}$ and a sequence $\mathbf{y}_i$ in the training set, as computed by the SSearch software (Lipman and Pearson, 1988). The Smith-Waterman algorithm here uses the BLOSUM 62 substitution matrix, gap opening penalty of 11, and gap extension penalty of 1. Following Liao and Noble (2002) the kernel is defined as a radial basis function kernel on the vectors $U^{pw}(\mathbf{x})$ normalized to unit length, that is:

$$k^{\text{pw}}(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{1}{2\sigma^2}\left(\frac{U^{pw}(\mathbf{x})}{||U^{pw}(\mathbf{x})||} - \frac{U^{pw}(\mathbf{y})}{||U^{pw}(\mathbf{y})||}\right)^2\right),$$

where the width σ is the median Euclidean distance (in the feature space) from any positive training example to the nearest negative example.

6.7.3 SVM-Fisher Kernel

The SVM-Fisher method (Jaakkola et al., 2000) represents any sequence $\mathbf{x} \in \mathcal{X}$ by a fixed-length vector $U(\mathbf{x})$, namely the gradient of the log-likelihood of $\mathbf{x}$ under a profile HMM probabilistic model, with respect to the emission parameters of the model. We followed exactly the procedure described in Jaakkola et al. (2000).

6.7.4 Mismatch Kernel

The mismatch kernel (Leslie et al., 2003b) consists in representing a sequence by the set of fixed-length blocks it contains (typically three to six amino acids long), and augmenting this set by the blocks obtained by mutating a small number of amino acids (typically one or two) in the blocks observed. The mismatch kernel between two sequences is the inner products between these bag-of-blocks representations. We tested the kernel based on blocks of length 5, with up to 1 mutation, as it was reported to have the best performance in Leslie et al. (2003b).

6.7.5 Local Alignment Kernels

As explained in section 6.5 we compute the LA kernels using dynamic programming. These kernels have several parameters: the gap penalty parameters e and d, the

Table 6.1 ROC, ROC_{50} and median RFP averaged over 54 families for different kernels. The LA-eig and LA-ekm kernels with $\beta = +\infty$ correspond to the SW score (modified to become positive definite on the set of proteins used to train the SVM).

Kernel	Mean ROC	Mean ROC_{50}	Mean mRFP
LA-eig ($\beta = +\infty$)	0.908	0.591	0.0654
LA-eig ($\beta = 1$)	0.912	0.612	0.0626
LA-eig ($\beta = 0.8$)	0.908	0.597	0.0679
LA-eig ($\beta = 0.5$)	**0.925**	**0.649**	0.0541
LA-eig ($\beta = 0.2$)	0.923	**0.661**	0.0637
LA-eig ($\beta = 0.1$)	0.868	0.429	0.111
LA-ekm ($\beta = +\infty$)	0.916	0.585	0.0580
LA-ekm ($\beta = 1$)	0.920	0.587	**0.0539**
LA-ekm ($\beta = 0.8$)	0.916	0.585	0.0592
LA-ekm ($\beta = 0.5$)	**0.929**	0.600	**0.0515**
LA-ekm ($\beta = 0.2$)	0.877	0.453	0.125
LA-ekm ($\beta = 0.1$)	0.596	0.052	0.500
Pairwise	0.896	0.464	0.0837
Mismatch	0.872	0.400	0.0837
Fisher	0.773	0.250	0.204

amino acid mutation matrix s, and the factor β which controls the influence of suboptimal alignments in the kernel value. A precise analysis of the effects of these parameters would be beyond the scope of this chapter so we limit ourselves to the analysis of the effect of the β parameter. In order to be consistent with the SVM-pairwise method the substitution matrix is always the BLOSUM 62 matrix and the gap parameters are always ($e = 11$,$d = 1$) below.

6.8 Results

Table 6.1 summarizes the performance of the different methods in terms of ROC, ROC_{50}, and RFP scores averaged over all 54 families tested. We tested the local alignment kernels LA-eig and LA-ekm for several values of β ranging from $+\infty$, in which case they are derived from the SW score (6.12), to $\beta = 0.1$.

These results show that both LA-eig and LA-ekm perform best for β in the range of 0.2 to 0.5, and have almost similar performances in that case. This suggests that the normalization of $\tilde{k}_{LA}^{(\beta)}$ into a positive definite kernel through the empirical kernel map of Schölkopf et al. (2002) or by subtracting the smallest negative eigenvalue from the diagonal has little influence on the final performance.

Second, the fact that the performance of the LA-eig and LA-ekm kernels is better for β in the range of 0.2 to 0.5 than for $\beta = \infty$ shows that the SW score as a

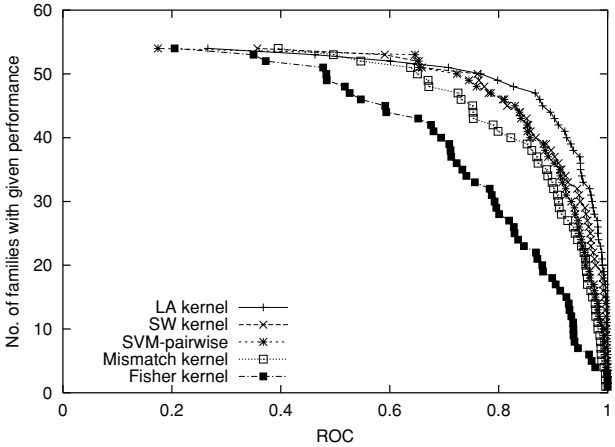


Figure 6.2 ROC score distribution for different kernels. The curve denoted LA kernel corresponds to the LA-eig kernel with $\beta = 0.5$. The curve denoted SW kernel corresponds to the LA-eig kernel with $\beta = \infty$, which is equal to the SW score up to a constant on the diagonal.

kernel is outperformed by variants which take into account suboptimal alignments to quantify the similarity between protein sequences.

The results obtained with the Fisher, pairwise, and mismatch kernels are consistent with Jaakkola et al. (2000), Liao and Noble (2002), and Leslie et al. (2003b). The pairwise and mismatch kernels perform almost at the same level, with some advantage for the pairwise kernel in terms of ROC and ROC_{50} scores. They both outperform the Fisher kernel on this benchmark, but this is likely due to the facts that only a single HMM is used to build the Fisher score vector for each family, on the one hand, and that no protein outside the training data set was used to train the HMM, on the other hand. In more realistic conditions, the Fisher kernel performs roughly at the same level as the mismatch kernel (C. Leslie, private communication).

More important, most of the LA kernels tested slightly outperform all three other methods in this benchmark. As an illustration, the distribution of ROC, ROC_{50}, and median RFP scores for all three methods and the LA-eig kernel with $\beta = 0.5$ and $\beta = \infty$ are shown in figures 6.2, 6.3 and 6.4. In each case a higher curve corresponds to a more accurate remote homology detection method. The LA-eig kernel with $\beta = 0.5$ retrieves more than twice as many families as the best other method tested (the pairwise method) at a ROC_{50} score of 0.8 or higher. This remains true for a wide range of values for β, including $\beta = +\infty$. This means that the SW score as a kernel also outperforms the Fisher, pairwise, and mismatch kernels.

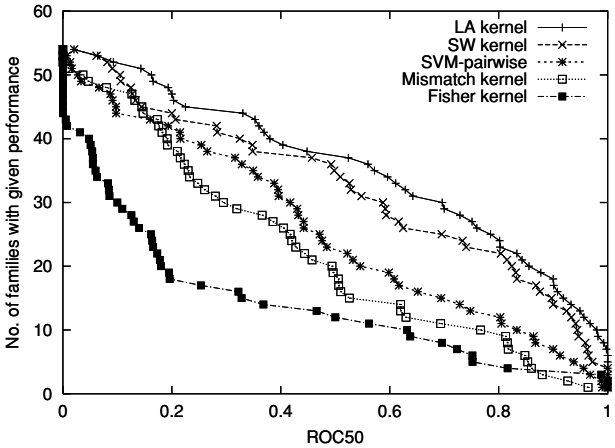


Figure 6.3 ROC_{50} score distribution for different kernels.

6.8.1 Complexity

Another important factor for practical use of these kernels is their computation cost and speed. The mismatch kernel has a linear complexity with respect to the sum of the sequence lengths, and is by far the fastest to operate. The complexity of the LA kernel is proportional to the product of the sequence lengths. Moreover, the LA kernel is faster to compute for $\beta = +\infty$ (SW score) than for other values of β, because in that case all computations can be performed with sum and max operations on integer instead of logarithms on floating point real numbers. In our experiments the computation of the SW kernel was 4 times slower than the computation of the mismatch kernel (using the SSEARCH software for the SW score, and an implementation of the mismatch kernel described in Leslie et al. (2003b), likely to be optimized in the future). The computation of the LA kernel for other values of β was 2 orders of magnitude slower than the case $\beta = \infty$, using a very naive implementation of the dynamic programming algorithm.

The computation of the kernel Gram matrix on the training set for SVM-pairwise and the LA-ekm kernels requires $\mathrm{O}(n^3)$ further operations to multiply the empirical kernel map matrix by its transpose. Only $\mathrm{O}(n^2)$ are approximately required by the LA-eig kernels to compute the smallest eigenvalue using the power method (Golub and Van Loan, 1996) and subtract it from the diagonal. However, in both cases this operation is very fast compared to the time required to compute the LA kernel values or E-values.

Finally, classification of a new sequence with SVM-pairwise or the LA-ekm kernels requires computing the explicit empirical kernel map representation of the sequence, that is, computing n E-value or LA kernels. In the case of the spectrum and LA-eig

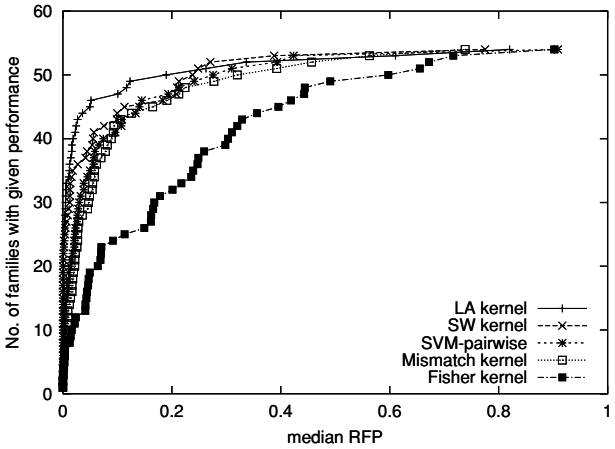


Figure 6.4 Median RFP distribution for different kernels.

kernels, the kernels are only computed between the new sequence and the support vector sequences, which usually form only a subset of the training set. Because the performances of LA-eig and LA-ekm are very similar, this suggests a preference for the former.

6.9 Discussion and Conclusion

In this chapter we introduced a family of kernels specifically adapted to protein sequences, based on the detection of high-scoring local alignments. These kernels are biologically motivated, and extend classic work on local alignment scoring to the framework of kernel functions.

The theoretically valid local alignment kernels we come up with suffer in practice from diagonal dominance. Hence we employed tricks to turn them into useful kernels, by taking a logarithm and adding a constant on the diagonal, or by using the empirical kernel map. The resulting kernels significantly outperform all other state-of-the-art methods on a benchmark experiment of SCOP superfamily recognition, which was designed to simulate the problem of remote homology detection.

The remarkable accuracy of our method certainly comes from the combination of two widely used algorithms. On the one hand, the SVM algorithm is based on a sound mathematical framework and has been shown to perform very well on many real-world applications. One of its particularities is that it can perform classification of any kind of data, such as strings in our case, as soon as a kernel function is provided. On the other hand, local alignment scores, in particular the SW score, have been developed to quantify the similarity of biological sequences.

Their parameters have been optimized over the years to provide relevant measures of similarity for homologous sequences, and they now represent core tools in computational biology.

However, direct pairwise comparisons of sequences through local alignment scores such as the SW scores are often considered naive and weak methods to detect remote homology. They are usually outperformed by methods such as PSI-BLAST which extend pairwise comparisons to pools of sequences extracted iteratively. The main contribution of this chapter is to show that pairwise sequence comparison can be extremely powerful when used as a kernel function combined with an SVM, and that the SW score itself provides a state-of-the-art method for remote homology detection when used as a kernel. An interesting conclusion of our experiments is that the SW score, however, is outperformed by local alignment scores which sum up the contributions of all possible local alignments. Summing up over local alignments has an important cost in terms of computation time due to the operations required with floating point numbers, but can be worth the cost when one is interested in precision more than speed. On the other hand the SW score itself is computed by dynamic programming and is therefore slower to compute than the mismatch kernel which it outperforms. Here again a tradeoff must be found between speed and accuracy, depending on the application. However, due to its wide use in computational biology the SW score has been precomputed and stored in databases such as KEGG's SSDB (Kanehisa et al., 2002) for virtually all known or predicted proteins of sequenced genomes, which suggests that practical applications for the Smith-Waterman kernel could be implemented in relation to such databases.

Suboptimal alignments

The only parameter whose influence was tested is the parameter β of the LA kernel, which controls the importance of the contribution of nonoptimal local alignments in the final score. It should be pointed out here that the optimal values for β we observed (in the range of 0.2 to 0.5) are only optimal for an average performance on the 54 families tested, and that the optimal value for each family might fluctuate. Moreover, a number of other parameters could be modified, in particular the gap penalty parameters and the similarity matrix between amino acids, and the optimal values for β might also depend on these parameters. Further theoretical and practical studies, which are beyond the scope of this chapter, should be performed to evaluate the influence of these parameters. In particular, it would be interesting to know for which values of these parameters the SW score itself is a valid kernel, and which parameter tuning results in the most accurate remote protein homology detection.

Parameter setting

An important open problem with the LA kernels as well as with most other string kernels is the following: how to make the kernel independent of the lengths of the sequences compared. Indeed long sequences typically result in small kernel values when the kernel is normalized with (6.13). While much work has been done to estimate the significance of alignment scores for varying sequence length, these approaches remain difficult to adapt to the kernel framework. The importance of this issue might be underestimated in the benchmark experiment presented in this chapter, because protein sequences in a SCOP family tend to have similar

Length normalization

sequence lengths. However applying kernel-based homology detection in a more realistic setting might reveal important effects on this issue.

While we focused in this paper on the application of the new kernels to the problem of remote homology detection, it should finally be pointed out that possible use of these kernels, and more generally of all other string kernels, go beyond this single goal. In combination with SVM, string kernels can be applied to various classification and regression tasks such as gene function, localization, or structure prediction. Moreover recent studies suggest that kernels provide a useful framework for integrating and performing inference from heterogeneous data (Vert and Kanehisa, 2003b; Yamanishi et al., 2003), which we plan to investigate in the future.

6.10 Acknowledgments

We thank Li Liao and William Stafford Noble for making their data sets and software available, Mark Diekhans for providing information about the Fisher kernel, and Christina Leslie, Eleazar Eskin, and Jason Weston for information and software concerning the mismatch kernel. The computational resource was provided by the Bioinformatics Center, Institute for Chemical Research, Kyoto University and the Supercomputer Laboratory, Kyoto University. Part of this work was supported by a Grant-in-Aid for Scientific Research on Priority Areas (C) "Genome Information Science" from MEXT of Japan.

Appendix A: Proof of Theorem 6.7

In this proof we assume the similarity matrix S and gap penalty function g are fixed, and remove them as subscripts in $s_{S,g}$ and $SW_{S,g}$ for convenience. For any two sequences $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2$, let

$$\Pi_n^0(\mathbf{x}, \mathbf{y}) = \{\pi \in \Pi_n(\mathbf{x}, \mathbf{y}) : \pi_1(n) = |\mathbf{x}|, \pi_2(n) = |\mathbf{y}|\}\,.$$

We can prove a first useful result:

Lemma 6.9 *With the notations of theorem 6.7, the following holds for any two sequences* $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2$ *and* $n \geq 1$*:*

$$\sum_{\pi \in \Pi_n^0(\mathbf{x},\mathbf{y})} \exp\left(\beta s(\mathbf{x}, \mathbf{y}, \pi)\right) = k_0 \star k_a \star (k_g \star k_a)^{n-1}(\mathbf{x}, \mathbf{y}), \tag{6.14}$$

and for $n = 0$*:*

$$\sum_{\pi \in \Pi_0^0(\mathbf{x},\mathbf{y})} \exp\left(\beta s(\mathbf{x}, \mathbf{y}, \pi)\right) = k_0(\mathbf{x}, \mathbf{y}) \tag{6.15}$$

Proof Let us proceed by induction on n. For $n = 0$, $\Pi_n^0(\mathbf{x}, \mathbf{y})$ is reduced to the singleton $\{\emptyset\}$ (the alignment where no position is aligned), which has a null score. As a result:

$$\sum_{\pi \in \Pi_0^0(\mathbf{x},\mathbf{y})} \exp\left(\beta s(\mathbf{x}, \mathbf{y}, \pi)\right) = \exp\left(\beta s(\mathbf{x}, \mathbf{y}, \emptyset)\right) = 1 = k_0(\mathbf{x}, \mathbf{y}),$$

which proves (6.15) for any $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2$.

For $n = 1$, $\Pi_n^0(\mathbf{x}, \mathbf{y})$ is reduced to the singleton $\{(|\mathbf{x}|, |\mathbf{y}|)\}$ (only the last letters are aligned). The score of this alignment is $S(x_{|\mathbf{x}|}, y_{|\mathbf{y}|})$. On the other hand, by (6.4), (6.5), and (6.6), the following holds:

$$k_0 \star k_a(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{x}_1\mathbf{x}_2=\mathbf{x}, \mathbf{y}_1\mathbf{y}_2=\mathbf{y}} k_0(\mathbf{x}_1, \mathbf{y}_1) k_a(\mathbf{x}_2, \mathbf{y}_2) = \exp\left[\beta S(x_{|\mathbf{x}|}, y_{|\mathbf{y}|})\right].$$

This proves (6.14) in the case $n = 1$, for any $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2$.

Let us now suppose that (6.14) is true at the level $n - 1$, and prove it is then true at the level n. Let $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2$ be two sequences. Then any alignment $\pi \in \Pi_n^0(\mathbf{x}, \mathbf{y})$ induces a decomposition $\mathbf{x} = \mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$, with $\mathbf{x}_1 = x_1 \cdots x_{\pi_1(n-1)}$, $\mathbf{x}_2 = x_{\pi_1(n-1)+1} \cdots x_{|\mathbf{x}|-1}$, and $\mathbf{x}_3 = x_{|\mathbf{x}|}$ (and similarly for $\mathbf{y} = \mathbf{y}_1\mathbf{y}_2\mathbf{y}_3$). By construction, the alignment $f(\pi)$ obtained from π by removing the last aligned positions satisfies $f(\pi) \in \Pi_n^0(\mathbf{x}_1, \mathbf{y}_1)$. Conversely, it is easy to see that for any decompositions $\mathbf{x} = \mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$ and $\mathbf{y} = \mathbf{y}_1\mathbf{y}_2\mathbf{y}_3$ with $|\mathbf{x}_3| = |\mathbf{y}_3| = 1$, and for any $\pi \in \Pi_{n-1}^0(\mathbf{x}_1, \mathbf{y}_1)$, we can find a unique $\pi' \in \Pi_n^0(\mathbf{x}, \mathbf{y})$ such that $f(\pi') = \pi$, namely the alignment obtained by adding to the alignment π' the pair $(|\mathbf{x}|, |\mathbf{y}|)$. This bijection enables us to write

$$\sum_{\pi \in \Pi_n^0(\mathbf{x},\mathbf{y})} \exp\left(\beta s(\mathbf{x}, \mathbf{y}, \pi)\right) = \sum_{\mathbf{x}=\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3, \mathbf{y}=\mathbf{y}_1\mathbf{y}_2\mathbf{y}_3} \sum_{\pi' \in \Pi_{n-1}^0(\mathbf{x}_1,\mathbf{y}_1)} \exp\left(\beta s(\mathbf{x}_1, \mathbf{y}_1, f^{-1}(\pi'))\right), \quad (6.16)$$

where the first sum in the right-hand side is over all decompositions of $\mathbf{x}$ and $\mathbf{y}$ with $|\mathbf{x}_3| = |\mathbf{y}_3| = 1$.

By definition of the local alignment score (6.1), we have for any $\pi \in \Pi_n^0(\mathbf{x}, \mathbf{y})$:

$$s(\mathbf{x}, \mathbf{y}, \pi) = s(\mathbf{x}_1, \mathbf{y}_1, f(\pi)) + g(|\mathbf{x}_2|) + g(|\mathbf{y}_2|) + S(x_{|\mathbf{x}|}, y_{|\mathbf{y}|}),$$

and therefore, by (6.7) and (6.6):

$$\exp\left(\beta s(\mathbf{x}, \mathbf{y}, \pi)\right) = \exp\left(\beta s(\mathbf{x}_1, \mathbf{y}_1, f(\pi))\right) . k_g(\mathbf{x}_2, \mathbf{y}_2) . k_a(\mathbf{x}_3, \mathbf{y}_3). \quad (6.17)$$

Observing that $k_a(\mathbf{x}_3, \mathbf{y}_3)$ is null when $\mathbf{x}_3$ or $\mathbf{y}_3$ is not reduced to a single letter, we can plug (6.17) into (6.16) to get

$$\begin{aligned}&\sum_{\pi\in\Pi_n^0(\mathbf{x},\mathbf{y})} \exp\left(\beta s(\mathbf{x},\mathbf{y},\pi)\right)\\ &= \sum_{\mathbf{x}=\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3,\mathbf{y}=\mathbf{y}_1\mathbf{y}_2\mathbf{y}_3} \sum_{\pi\in\Pi_{n-1}^0(\mathbf{x}_1,\mathbf{y}_1)} \exp\left(\beta s(\mathbf{x}_1,\mathbf{y}_1,f(\pi))\right).k_g(\mathbf{x}_2,\mathbf{y}_2).k_a(\mathbf{x}_3,\mathbf{y}_3),\end{aligned} \tag{6.18}$$

where the first sum in the right-hand side is now over all possible decompositions of $\mathbf{x}$ and $\mathbf{y}$ with $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3) \in \mathcal{X}^6$.

Using the induction hypothesis and the definition of convolution we can now conclude as follows:

$$\begin{aligned}&\sum_{\pi\in\Pi_n^0(\mathbf{x},\mathbf{y})} \exp\left(\beta s(\mathbf{x},\mathbf{y},\pi)\right)\\ &\quad= \sum_{\mathbf{x}=\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3,\mathbf{y}=\mathbf{y}_1\mathbf{y}_2\mathbf{y}_3} \sum_{\pi\in\Pi_{n-1}^0(\mathbf{x}_1,\mathbf{y}_1)} \exp\left(\beta s(\mathbf{x}_1,\mathbf{y}_1,\pi)\right).k_g(\mathbf{x}_2,\mathbf{y}_2).k_a(\mathbf{x}_3,\mathbf{y}_3)\\ &\quad= \sum_{\mathbf{x}=\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3,\mathbf{y}=\mathbf{y}_1\mathbf{y}_2\mathbf{y}_3} k_0 \star k_a \star (k_g \star k_a)^{n-2}(\mathbf{x}_1,\mathbf{y}_1).k_g(\mathbf{x}_2,\mathbf{y}_2).k_a(\mathbf{x}_3,\mathbf{y}_3)\\ &\quad= k_0 \star k_a \star (k_g \star k_a)^{n-1}(\mathbf{x},\mathbf{y}),\end{aligned} \tag{6.19}$$

■

Going back to the proof of theorem 6.7, let $n > 0$ and $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}$. Observing that any alignment $\pi \in \Pi_n(\mathbf{x}, \mathbf{y})$ is in fact an element of $\Pi_n^0(\mathbf{x}_1, \mathbf{y}_1)$ where $\mathbf{x}_1 = x_1 \cdots x_{\pi_1(n)}$ and $\mathbf{y}_1 = y_1 \cdots y_{\pi_2(n)}$, we can write

$$\begin{aligned}\sum_{\pi\in\Pi_n(\mathbf{x},\mathbf{y})} \exp\left(\beta s(\mathbf{x},\mathbf{y},\pi)\right) &= \sum_{\mathbf{x}=\mathbf{x}_1\mathbf{x}_2,\mathbf{y}=\mathbf{y}_1\mathbf{y}_2} \sum_{\pi\in\Pi_n^0(\mathbf{x}_1,\mathbf{y}_1)} \exp\left(\beta s(\mathbf{x}_1,\mathbf{y}_1,\pi)\right)\\ &= \sum_{\mathbf{x}=\mathbf{x}_1\mathbf{x}_2,\mathbf{y}=\mathbf{y}_1\mathbf{y}_2} k_0 \star k_a \star (k_g \star k_a)^{n-1}(\mathbf{x}_1,\mathbf{y}_1)\\ &= k_0 \star k_a \star (k_g \star k_a)^{n-1} \star k_0(\mathbf{x},\mathbf{y}),\end{aligned} \tag{6.20}$$

where the first equality is obtained by organizing the alignments in terms of $\pi_1(n)$ and $\pi_2(n)$, the second is the consequence of lemma 6.9, and the last is the definition of convolution. (6.9) now follows by summing this equality over n.

In order to prove (6.10) we derive from (6.9) the following:

$$\frac{1}{\beta}\log k_{LA}^{(\beta)}(\mathbf{x},\mathbf{y}) = \frac{1}{\beta}\log\left(\sum_{\pi\in\Pi(\mathbf{x},\mathbf{y})}\exp\left(\beta s(\mathbf{x},\mathbf{y},\pi)\right)\right)$$
$$= SW(\mathbf{x},\mathbf{y}) + \frac{1}{\beta}\log\left(\sum_{\pi\in\Pi(\mathbf{x},\mathbf{y})}\exp\left[\beta\left(s(\mathbf{x},\mathbf{y},\pi) - SW(\mathbf{x},\mathbf{y})\right)\right]\right). \tag{6.21}$$

By definition of the SW score (6.2) it follows that:

$$\lim_{\beta\to+\infty}\log\left(\sum_{\pi\in\Pi(\mathbf{x},\mathbf{y})}\exp\left[\beta\left(s(\mathbf{x},\mathbf{y},\pi) - SW(\mathbf{x},\mathbf{y})\right)\right]\right) = \log l, \tag{6.22}$$

where l is the number of alignments in $\Pi(\mathbf{x},\mathbf{y})$ with maximal alignment. (6.10) is now a consequence of (6.21) and (6.22), which concludes the proof of theorem 6.7. ■

Appendix B: Proof of Theorem 6.8

For any sequences $(\mathbf{x},\mathbf{y}) \in \mathcal{X}^2$ let us first introduce some notations. For $0 \le i \le |\mathbf{x}|$ and $0 \le j \le |\mathbf{y}|$, we define the following sets of alignments:

$$\begin{aligned}
\Pi_{i,j} &:= \left\{\pi \in \Pi(\mathbf{x},\mathbf{y})\backslash\{\emptyset\} : \pi_1(|\pi|) \le i, \pi_2(|\pi|) \le j\right\},\\
\Pi^0_{i,j} &:= \left\{\pi \in \Pi_{i,j} : \pi_1(|\pi|) = i, \pi_2(|\pi|) = j\right\},\\
\Pi^1_{i,j} &:= \left\{\pi \in \Pi_{i,j} : \pi_1(|\pi|) < i, \pi_2(|\pi|) = j\right\},\\
\Pi^2_{i,j} &:= \left\{\pi \in \Pi_{i,j} : \pi_1(|\pi|) < i, \pi_2(|\pi|) < j\right\}.
\end{aligned} \tag{6.23}$$

Hence $\Pi_{i,j}$ is the set of alignments that only involve up to the first i and j letters of $\mathbf{x}$ and $\mathbf{y}$, and the three subsets $\Pi^0_{i,j}$, $\Pi^1_{i,j}$, and $\Pi^2_{i,j}$ form a partition of $\Pi_{i,j}$. Let us also introduce a score $s_{i,j}$ for the alignments of $\Pi_{i,j}$ derived from the local alignment score (6.1) by

$$\forall \pi \in \Pi_{i,j}, \quad s_{i,j}(\pi) := s(\pi) - g\left(i - \pi_1(|\pi|)\right) - g\left(j - \pi_2(|\pi|)\right). \tag{6.24}$$

In other words, $s_{i,j}$ is an alignment score that penalizes the unaligned part after the last aligned letters, contrary to s. With these definitions we can state a key lemma that explains the meaning of the functions computed in theorem 6.8

Lemma 6.10 *With the notations of theorem 6.8, the following equalities hold for* $0 \leq i \leq |\mathbf{x}|$ *and* $0 \leq j \leq |\mathbf{y}|$*:*

$$\begin{aligned}
M(i,j) &= \sum_{\pi \in \Pi^0_{i,j}} \exp\left(\beta s_{i,j}(\pi)\right) = \sum_{\pi \in \Pi^0_{i,j}} \exp\left(\beta s(\pi)\right), \\
X(i,j) &= \sum_{\pi \in \Pi^1_{i,j}} \exp\left(\beta s_{i,j}(\pi)\right), \\
Y(i,j) &= \sum_{\pi \in \Pi^2_{i,j}} \exp\left(\beta s_{i,j}(\pi)\right), \\
X_2(i,j) &= \sum_{\pi \in \Pi^1_{i,j}} \exp\left(\beta s(\pi)\right), \\
Y_2(i,j) &= \sum_{\pi \in \Pi^2_{i,j}} \exp\left(\beta s(\pi)\right).
\end{aligned}$$

Proof This lemma is proved by induction in (i,j). If $i = 0$ or $j = 0$, then $\Pi_{i,j} = \emptyset$ and the statements are true. To prove the statements for $i > 0$ and $j > 0$, consider first the statement about $M(i,j)$. Let $f : \Pi(\mathbf{x},\mathbf{y}) \to \Pi(\mathbf{x},\mathbf{y})$ be the application that removes the last pair of aligned position in an alignment. Then f is clearly a bijection between $\Pi^{(0)}_{i,j}$ and $\Pi_{i-1,j-1} \cup \{\emptyset\}$, and $s(\pi) = s_{i,j}(\pi) = s_{i-1,j-1}(f(\pi)) + S(x_i, y_j)$ for any $\pi \in \Pi^0_{i,j}$. We therefore can write

$$\begin{aligned}
&\sum_{\pi \in \Pi^0_{i,j}} \exp\left(\beta s(\pi)\right) \\
&\quad = \sum_{\pi \in \Pi^0_{i,j}} \exp\left(\beta s_{i,j}(\pi)\right) \\
&\quad = \sum_{\pi \in \Pi_{i-1,j-1} \cup \{\emptyset\}} \exp\left(\beta s_{i-1,j-1}(\pi) + \beta S(x_i, y_j)\right) \\
&\quad = e^{\beta S(x_i,y_j)} \left(M(i-1,j-1) + X(i-1,j-1) + Y(i-1,j-1) + 1\right) \\
&\quad = M(i,j),
\end{aligned}$$

where the third equality uses the induction hypothesis and the fourth one the definition of $M(i,j)$. The same approach can be followed to prove the other statements of lemma 6.10, so we don't explicitly write them down for lack of space. ■

Let $m = |\mathbf{x}|$ and $n = |\mathbf{y}|$. theorem 6.8 is now a direct consequence of lemma 6.10 by using (6.9) and observing that $\Pi(\mathbf{x},\mathbf{y}) = \Pi_{m,n}$ is the disjoint union of $\{\emptyset\}$, $\Pi^0_{m,n}$, $\Pi^1_{m,n}$ and $\Pi^2_{m,n}$. ■

7 Kernels for Graphs

Hisashi Kashima
Koji Tsuda
Akihiro Inokuchi

This chapter discusses the construction of kernel functions between labeled graphs. We provide a unified account of a family of kernels called *label sequence kernels* that are defined via label sequences generated by graph traversal. For cyclic graphs, dynamic programming techniques cannot simply be applied, because the kernel is based on an infinite dimensional feature space. We show that the kernel computation boils down to obtaining the stationary state of a discrete-time linear system, which is efficiently performed by solving simultaneous linear equations. Promising empirical results are presented in classification of chemical compounds.

7.1 Introduction

Many real-world data are represented not as vectors but as *graphs*, including sequences and trees as special cases. Examples of such data include biological sequences, phylogenetic trees, RNA structures (Durbin et al., 1998), natural language texts (Manning and Schütze, 1999), semistructured data such as HTML and XML (Abiteboul et al., 2000), and so on. In computational biology, graph data are attracting considerable attention in drug design. Chemical compounds can be represented as labeled graphs and their automatic classification to predict the effectiveness or toxicity of drugs is of crucial importance to the rationalization of drug discovery processes (Kramer and De Raedt, 2001; Inokuchi et al., 2000). In protein engineering, three-dimensional structures of proteins are often represented as distance graphs (Holm and Sander, 1993).

Kernel methods such as support vector machines (SVMs) are becoming increasingly popular for their high performance (Schölkopf and Smola, 2002). In kernel methods, all computations are done via a *kernel function*, which is defined as the inner product of two vectors in a feature space. A kernel function needs to be de-

signed to capture the characteristics of the objects appropriately, and at the same time, to be computed efficiently. Furthermore it must satisfy the mathematical property called *positive definiteness*. A kernel function should deliberately be designed to satisfy this property, because ad hoc similarity functions are not always positive definite; see, for example, Shimodaira et al. (2002) and Bahlmann et al. (2002).

Haussler (1999) introduced "convolution kernels", a general framework for handling discrete data structures by kernel methods. In convolution kernels, objects are decomposed into parts, and kernels are defined in terms of the (sub)kernels between parts. After Haussler's seminal paper, a number of kernels for structured data were proposed, for example, Watkins (2000), Jaakkola et al. (2000), Leslie et al. (2003b), Lodhi et al. (2002), and Tsuda et al. (2002b) for sequences, and Vishwanathan and Smola (2003), Collins and Duffy (2002), and Kashima and Koyanagi (2002) for trees. Most of them are basically based on the same idea. An object such as a sequence or a tree is decomposed into substructures, for example, substrings, subsequences, and subtrees, and a feature vector is composed of the counts of the substructures. Since the dimensionality of feature vectors is typically very high, the explicit computations of feature values should be avoided. So most of the kernels adopt efficient procedures such as dynamic programming or suffix trees.

In this chapter, we discuss the construction of kernel functions between labeled graphs.[1] We try to give a unified overview on the recent researches for graph kernels (Kashima and Inokuchi, 2002; Kashima et al., 2003; Gärtner, 2002; Gärtner et al., 2003). A common point of these works is that features are composed of the counts of *label sequences* produced by graph traversal. For the labeled graph shown in figure 7.1, a label sequence is produced by traversing the vertices, and looks like

$$(A, c, C, b, A, a, B), \tag{7.1}$$

where the vertex labels A, B, C, D and the edge labels a, b, c, d appear alternately. The essential difference among the kernels lies in how graphs are traversed and how weights are involved in computing a kernel. We call this family of kernel "label sequence kernels". This family of kernels can be computed efficiently and capture essential features of labeled graphs. As we will see below, label sequence kernels are closely related to the kernels between probability distributions (Jebara and Kondor, 2003), especially the kernels between hidden Markov models (HMMs). Mathematically it is possible to consider a kernel based on more complicated substructures such as subgraphs. However, the practical computation becomes considerably more difficult, because counting the number of all possible subgraphs turns out to be NP-hard (Gärtner et al., 2003).

1. Note that they should be distinguished from kernels in graph-structured input spaces such as kernels between two vertices in a graph, for example, diffusion kernels (Srinivasan et al., 1996; Kondor and Lafferty, 2002; Lafferty and Lebanon, 2003) or kernels between two paths in a graph, for example, path kernels (Takimoto and Warmuth, 2002).

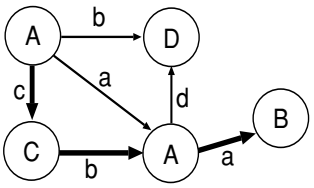


Figure 7.1 An example of labeled graphs. Vertices and edges are labeled by uppercase and lowercase letters, respectively. By traversing along the bold edges, the label sequence (7.1) is produced.

When the graphs are acyclic, the label sequence kernels are computed simply by dynamic programming, as shown in subsection 7.2.3. However, when the graphs are cyclic, label sequences of infinite length can be produced because the traversal may never end. In that case, the number of features becomes infinite. For computing a kernel based on infinite-dimensional vectors, we exploit recursive structures in features, and it will be shown that the kernel computation is reduced to finding the stationary state of a discrete-time linear system (Rugh, 1995), which can be done efficiently by solving simultaneous linear equations with a sparse coefficient matrix.

In the following, we describe the kernel function proposed by Kashima et al. (2003).[2] This kernel is defined as the expectation of a string kernel over all possible label sequences, which is regarded as a special case of *marginalized kernels* (Tsuda et al., 2002b). The relations to other label sequence kernels are discussed, and several extensions are proposed as well. Finally, in order to investigate how well our kernel performs on the real data, we show promising results on classifying chemical compounds.

7.2 Label Sequence Kernel between Labeled Graphs

Labeled graph

In this section, we introduce a kernel between labeled graphs. First of all, let us define a labeled graph. Let Σ_V, Σ_E be the sets of vertex labels and edge labels, respectively. Let $\mathcal{X}$ be a finite nonempty set of vertices, v be a function $v : \mathcal{X} \to \Sigma_V$, $\mathcal{L}$ be a set of ordered pairs of vertices called edges, and e be a function $e : \mathcal{L} \to \Sigma_E$. Then $G = (\mathcal{X}, v, \mathcal{L}, e)$ is a labeled graph with directed edges. figure 7.1 shows such a graph. For the time being, we assume that there are no multiple edges from one vertex to another. Our task is to construct a kernel function $k(G, G')$ between two labeled graphs.

2. Notice that the notations here are in part changed from those in Kashima et al. (2003) for better presentation.

7.2.1 Random Walks on Graphs

For extracting features from graph G, a set of label sequences is produced by random walking. At the first step, $x_1 \in \mathcal{X}$ is sampled from an initial probability distribution $p_s(x_1)$. Subsequently, at the ith step, the next vertex $x_i \in \mathcal{X}$ is sampled subject to a transition probability $p_t(x_i|x_{i-1})$, or the random walk ends with probability $p_q(x_{i-1})$:

$$\sum_{x_i=1}^{|\mathcal{X}|} p_t(x_i|x_{i-1}) + p_q(x_{i-1}) = 1. \tag{7.2}$$

When we do not have any prior knowledge, we can set p_s to be the uniform distribution, the transition probability p_t to be a uniform distribution over the vertices adjacent to the current vertex, and the termination probability p_q to be a small constant probability.

From the random walk, we obtain a sequence of vertices called *path*:

$$\mathbf{x} = (x_1, x_2, \ldots, x_\ell), \tag{7.3}$$

where ℓ is the length of $\mathbf{x}$ (possibly infinite). The probability for the path $\mathbf{x}$ is described as

$$p(\mathbf{x}|G) = p_s(x_1) \prod_{i=2}^{\ell} p_t(x_i|x_{i-1}) p_q(x_\ell).$$

Label sequence

Let us define a *label sequence* as an alternating sequence of vertex labels and edge labels:

$$\mathbf{h} = (h_1, h_2, \ldots, h_{2\ell-1}) \in (\Sigma_V \Sigma_E)^{\ell-1} \Sigma_V.$$

Associated with a path $\mathbf{x}$, we obtain a label sequence

$$\mathbf{h}_\mathbf{x} = (v_{x_1}, e_{x_1,x_2}, v_{x_2}, \ldots, v_{x_\ell}).$$

The probability for the label sequence $\mathbf{h}$ is equal to the sum of the probabilities of the paths emitting $\mathbf{h}$,

$$p(\mathbf{h}|G) = \sum_{\mathbf{x}} \delta(\mathbf{h} = \mathbf{h}_\mathbf{x}) \cdot \left(p_s(x_1) \prod_{i=2}^{\ell} p_t(x_i|x_{i-1}) p_q(x_\ell) \right),$$

where δ is a function that returns 1 if its argument holds, 0 otherwise.

7.2.2 Label Sequence Kernel

Next we define a kernel k_z between two label sequences $\mathbf{h}$ and $\mathbf{h}'$. We assume that two kernel functions, $k_v(v, v')$ and $k_e(e, e')$, are readily defined between vertex labels

Vertex and edge kernels

and edge labels, respectively. We constrain both kernels $k_v(v, v'), k_e(e, e') \geq 0$ to be non-negative.[3] An example of a vertex label kernel is the identity kernel,

$$k_v(v, v') = \delta(v = v'). \tag{7.4}$$

If the labels are defined in $\mathbb{R}$, the Gaussian kernel,

$$k_v(v, v') = \exp(- \parallel v - v' \parallel^2 / 2\sigma^2), \tag{7.5}$$

could be a natural choice (Schölkopf and Smola, 2002). Edge kernels are defined similarly. The kernel for label sequences is defined as the product of label kernels when the lengths of two sequences are equal ($\ell = \ell'$):

$$k_z(\mathbf{h}, \mathbf{h}') = k_v(h_1, h'_1) \prod_{i=2}^{\ell} k_e(h_{2i-2}, h'_{2i-2}) k_v(h_{2i-1}, h'_{2i-1}). \tag{7.6}$$

If the lengths are different ($\ell \neq \ell'$), then k_z is simply zero ($k_z(\mathbf{h}, \mathbf{h}') = 0$).

The function k_z is proved to be a valid kernel function as follows: The set of all possible label sequences $\mathcal{H}$ can be divided into subsets according to their lengths as $\mathcal{H}_1, \mathcal{H}_2, \ldots$. Let us define $k_z^{(j)}$ as k_z whose domain is limited to the subset $\mathcal{H}_j \times \mathcal{H}_j$, then $k_z^{(j)}$ is a valid kernel as it is described as the tensor product of kernels (Schölkopf and Smola, 2002). Now let us expand the domain of $k_z^{(j)}$ to the whole set $\mathcal{H} \times \mathcal{H}$ by assigning zero when one of the inputs is not included in $\mathcal{H}_j$, and call it $\bar{k}_z^{(j)}$. This operation is called zero extension (Haussler, 1999), which preserves positive definiteness. Since k_z is the sum of all $\bar{k}_z^{(j)}$'s, it turns out to be a valid kernel.

Label sequence kernel

Finally, our label sequence kernel is defined as the expectation of k_z over all possible $\mathbf{h}$ and $\mathbf{h}'$.

$$k(G, G') = \sum_{\mathbf{h}} \sum_{\mathbf{h}'} k_z(\mathbf{h}, \mathbf{h}') p(\mathbf{h}|G) p(\mathbf{h}'|G'). \tag{7.7}$$

This kernel is valid, because it is described as an inner product of two vectors $p(\mathbf{h}|G)$ and $p(\mathbf{h}'|G')$.

3. This constraint will play an important role in proving the convergence of our kernel in section 7.2.5.

7.2.3 Efficient Computation of Label Sequence Kernels

The label sequence kernel (7.7) can be expanded as

$$k(G,G') = \sum_{\ell=1}^{\infty} \sum_{\mathbf{h}} \sum_{\mathbf{h}'} \left(k_v(h_1,h'_1) \prod_{i=2}^{\ell} k_e(h_{2i-2},h'_{2i-2}) k_v(h_{2i-1},h'_{2i-1}) \right) \times \left(\sum_{\mathbf{x}} \delta(\mathbf{h}=\mathbf{h}_{\mathbf{x}}) \cdot \left(p_s(x_1) \prod_{i=2}^{\ell} p_t(x_i|x_{i-1}) p_q(x_\ell) \right) \right) \times \left(\sum_{\mathbf{x}'} \delta(\mathbf{h}=\mathbf{h}_{\mathbf{x}'}) \cdot \left(p_s(x'_1) \prod_{i=2}^{\ell} p_t(x'_i|x'_{i-1}) p_q(x'_\ell) \right) \right),$$

where $\sum_{\mathbf{h}} := \sum_{h_1 \in \Sigma_V} \sum_{h_2 \in \Sigma_E} \cdots \sum_{h_{2\ell-1} \in \Sigma_V}$ and $\sum_{\mathbf{x}} := \sum_{x_1=1}^{|\mathcal{X}|} \cdots \sum_{x_\ell=1}^{|\mathcal{X}|}$. The straightforward enumeration of all terms to compute the sum takes a prohibitive computational cost. For cyclic graphs, it is simply impossible because ℓ spans from 1 to infinity. Nevertheless, there is an efficient method to compute this kernel as shown below. The method is based on the observation that the kernel has the following nested structure.

Nested structure

$$k(G,G') = \lim_{L\to\infty} \sum_{\ell=1}^{L} \sum_{x_1,x'_1} s(x_1,x'_1) \left(\sum_{x_2,x'_2} t(x_2,x'_2,x_1,x'_1) \left(\sum_{x_3,x'_3} t(x_3,x'_3,x_2,x'_2) \times \left(\cdots \left(\sum_{x_\ell,x'_\ell} t(x_\ell,x'_\ell,x_{\ell-1},x'_{\ell-1}) q(x_\ell,x'_\ell) \right) \right) \cdots \right) \right), \tag{7.8}$$

where

$$\begin{aligned} s(x_1,x'_1) &:= p_s(x_1) p'_s(x'_1) k_v(v_{x_1}, v'_{x'_1}) \\ t(x_i,x'_i,x_{i-1},x'_{i-1}) &:= p_t(x_i|x_{i-1}) p'_t(x'_i|x'_{i-1}) k_v(v_{x_i}, v'_{x'_i}) k_e(e_{x_{i-1}x_i}, e_{x'_{i-1}x'_i}) \\ q(x_\ell,x'_\ell) &:= p_q(x_\ell) p'_q(x'_\ell). \end{aligned} \tag{7.9}$$

Acyclic Graphs Let us first consider acyclic graphs, that is, directed graphs without cycles. Precisely, it means that if there is a directed path from vertex x_1 to x_2, then there is no directed paths from vertex x_2 to x_1. When a directed graph is acyclic, the vertices can be numbered in a topological order[4] such that every edge from a vertex numbered i to a vertex numbered j satisfies $i < j$ (see figure 7.2).

Since there are no directed paths from vertex j to vertex i if $i < j$, we can employ dynamic programming. When G and G' are directed acyclic graphs, (7.8) can be

Dynamic programming

4. Topological sorting of graph G can be done in $O(|\mathcal{X}| + |\mathcal{L}|)$ (Cormen et al., 1990).

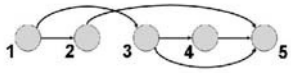


Figure 7.2 A topologically sorted directed acyclic graph. The label sequence kernel can be efficiently computed by dynamic programming running from right to left.

written as

$$k(G,G') = \sum_{x_1.x'_1} s(x_1,x'_1)q(x_1,x'_1) + \lim_{L\to\infty} \sum_{\ell=2}^{L} \sum_{x_1,x'_1} s(x_1,x'_1) \times \qquad (7.10)$$
$$\left(\sum_{x_2>x_1,x'_2>x'_1} t(x_2,x'_2,x_1,x'_1) \left(\sum_{x_3>x_2,x'_3>x'_2} t(x_3,x'_3,x_2,x'_2)\times \right.\right.$$
$$\left.\left. \left(\cdots \left(\sum_{x_\ell>x_{\ell-1},x'_\ell>x'_{\ell-1}} t(x_\ell,x'_\ell,x_{\ell-1},x'_{\ell-1})q(x_\ell,x'_\ell) \right) \right) \cdots \right).$$

The first and second terms correspond to the label sequences of length 1 and those longer than 1, respectively. By defining

$$r(x_1,x'_1) := q(x_1,x'_1) + \lim_{L\to\infty} \sum_{\ell=2}^{L} \left(\sum_{x_2>x_1,x'_2>x'_1} t(x_2,x'_2,x_1,x'_1)\times \right.$$
$$\left. \left(\cdots \left(\sum_{x_\ell>x_{\ell-1},x'_\ell>x'_{\ell-1}} t(x_\ell,x'_\ell,x_{\ell-1},x'_{\ell-1})q(x_\ell,x'_\ell) \right) \right) \cdots \right), \qquad (7.11)$$

we can rewrite (7.10) as the following:

$$k(G,G') = \sum_{x_1,x'_1} s(x_1,x'_1)r(x_1,x'_1).$$

The merit of defining (7.11) is that we can exploit the following recursive equation.

$$r(x_1,x'_1) = q(x_1,x'_1) + \sum_{j>x_1,j'>x'_1} t(j,j',x_1,x'_1)r(j,j'). \qquad (7.12)$$

Since all vertices are topologically ordered, $r(x_1,x'_1)$ for all x_1 and x'_1 can be efficiently computed by dynamic programming (figure 7.3). The worst-case time complexity of computing $k(G,G')$ is $O(c\cdot c'\cdot|\mathcal{X}|\cdot|\mathcal{X}'|)$ where c and c' are the maximum out degree of G and G', respectively.

General Directed Graphs In the case of cyclic graphs, we do not have topologically sorted graphs anymore. This means that we cannot employ the one-pass dynamic programming algorithm for acyclic graphs. However, we can obtain a re-

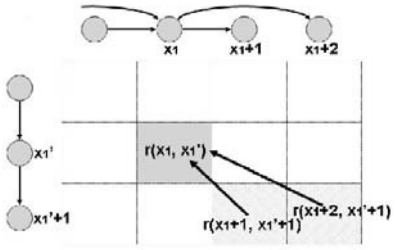


Figure 7.3 Recursion for computing $r(x_1, x_1')$ using recursive equation (7.12). $r(x_1, x_1')$ can be computed based on the precomputed values of $r(x_2, x_2'),\ x_2 > x_1,\ x_2' > x_1'$.

cursive form of the kernel like (7.12), and reduce the problem to solving a system of simultaneous linear equations. Let us rewrite (7.8) as

$$k(G, G') = \lim_{L\to\infty} \sum_{\ell=1}^{L} \sum_{x_1, x_1'} s(x_1, x_1') r_\ell(x_1, x_1'),$$

where for $\ell \geq 2$,

$$r_\ell(x_1, x_1') := \left(\sum_{x_2, x_2'} t(x_2, x_2', x_1, x_1') \left(\sum_{x_3, x_3'} t(x_3, x_3', x_2, x_2') \times \left(\cdots \left(\sum_{x_\ell, x_\ell'} t(x_\ell, x_\ell', x_{\ell-1}, x_{\ell-1}') q(x_\ell, x_\ell') \right) \right) \cdots \right) \right),$$

and $r_1(x_1, x_1') := q(x_1, x_1')$. Replacing the order of summation, we have the following:

$$\begin{aligned} k(G, G') &= \sum_{x_1, x_1'} s(x_1, x_1') \lim_{L\to\infty} \sum_{\ell=1}^{L} r_\ell(x_1, x_1') \\ &= \sum_{x_1, x_1'} s(x_1, x_1') \lim_{L\to\infty} R_L(x_1, x_1'), \end{aligned} \tag{7.13}$$

where

$$R_L(x_1, x_1') := \sum_{\ell=1}^{L} r_\ell(x_1, x_1').$$

Thus we need to compute $R_\infty(x_1, x_1')$ to obtain $k(G, G')$.

Now let us restate this problem in terms of linear system theory (Rugh, 1995).

Linear system

The following recursive relationship holds between r_k and r_{k-1} $(k \geq 2)$:

$$r_k(x_1, x_1') = \sum_{i,j} t(i, j, x_1, x_1') r_{k-1}(i, j). \tag{7.14}$$

Using (7.14), the recursive relationship for R_L also holds as follows:

$$\begin{aligned} R_L(x_1, x_1') &= r_1(x_1, x_1') + \sum_{k=2}^{L} r_k(x_1, x_1') \\ &= r_1(x_1, x_1') + \sum_{k=2}^{L} \sum_{i,j} t(i, j, x_1, x_1') r_{k-1}(i, j) \\ &= r_1(x_1, x_1') + \sum_{i,j} t(i, j, x_1, x_1') R_{L-1}(i, j). \end{aligned} \tag{7.15}$$

Thus R_L can be perceived as a discrete-time linear system (Rugh, 1995) evolving as the time L increases. Assuming that R_L converges (see section 7.2.5 for the convergence condition), we have the following equilibrium equation:

$$R_\infty(x_1, x_1') = r_1(x_1, x_1') + \sum_{i,j} t(i, j, x_1, x_1') R_\infty(i, j). \tag{7.16}$$

Therefore, the computation of our kernel finally boils down to solving simultaneous linear equations (7.16) and substituting the solutions into (7.13).

Matrix computation

Now let us restate the above discussion in the language of matrices. Let $\mathbf{s}$, $\mathbf{r}_1$, and $\mathbf{r}_\infty$ be $|\mathcal{X}| \cdot |\mathcal{X}'|$ dimensional vectors such that

$$\mathbf{s} = (\cdots, s(i, j), \cdots)^\top, \quad \mathbf{r}_1 = (\cdots, r_1(i, j), \cdots)^\top, \quad \mathbf{r}_\infty = (\cdots, R_\infty(i, j), \cdots)^\top, \tag{7.17}$$

respectively. Let the transition probability matrix T be a $|\mathcal{X}||\mathcal{X}'| \times |\mathcal{X}||\mathcal{X}'|$ matrix,

$$[T]_{(i,j),(k,l)} = t(i, j, k, l).$$

Equation (7.13) can be rewritten as

$$k(G, G') = \mathbf{r}_\infty^T \mathbf{s} \tag{7.18}$$

Similarly, the recursive equation (7.16) is rewritten as

$$\mathbf{r}_\infty = \mathbf{r}_1 + T\mathbf{r}_\infty.$$

The solution of this equation is

$$\mathbf{r}_\infty = (I - T)^{-1} \mathbf{r}_1.$$

Finally, the matrix form of the kernel is

$$k(G, G') = (I - T)^{-1} \mathbf{r}_1 \mathbf{s}. \tag{7.19}$$

Computing the kernel requires solving a linear equation or inverting a matrix with $|\mathcal{X}||\mathcal{X}'| \times |\mathcal{X}||\mathcal{X}'|$ coefficients. However, the matrix $I - T$ is actually sparse because the number of non-zero elements of T is less than $c \cdot c' \cdot |\mathcal{X}| \cdot |\mathcal{X}'|$ where

c and c' are the maximum out degree of G and G', respectively [see (7.9) for the definition of T]. Therefore, we can employ efficient numerical algorithms that exploit sparsity (Barrett et al., 1994). In our implementation, we employed a simple iterative method that updates R_L by using (7.15) until convergence starting from $R_1(x_1, x'_1) = r_1(x_1, x'_1)$.

7.2.4 Allowing Multiple Edges between Vertices

Up to this point, we assumed that there are no multiple edges from one vertex to another. However, a slight modification allows incorporation of multiple edges. Suppose that there are $M_{x_{i-1}x_i}$ directed edges from vertex x'_{i-1} to vertex x'_i with labels $e^m_{x_{i-1}x_i}$, and transition probabilities $p^m_t(x_i|x_{i-1})$ $(m = 1, 2, \ldots, M_{x_{i-1}x_i})$. Instead of (7.9), by considering all pair of $e^m_{x_{i-1}x_i}$ and $e^m_{x'_{i-1}x'_i}$, we have only to redefine $t(x_i, x'_i, x_{i-1}, x'_{i-1})$ as the following.

$$t(x_i, x'_i, x_{i-1}, x'_{i-1}) := k(v_{x_i}, v'_{x'_i}) \times \sum_{m=1}^{M_{x_{i-1}x_i}} \sum_{m'=1}^{M'_{x_{i-1}x_i}} p^m_t(x_i|x_{i-1}) p'^{m'}_t(x'_i|x'_{i-1}) k(e^m_{x_{i-1}x_i}, e^{m'}_{x'_{i-1}x'_i})$$

7.2.5 Convergence Condition

Since loops are allowed in general directed graphs, an infinite number of paths can be generated. Therefore some convergence condition is needed to justify (7.16). The following theorem holds:

Theorem 7.1 *The infinite sequence* $\lim_{L\to\infty} R_L(x_1, x'_1)$ *converges for any* $x_1 \in \{1, \cdots, |\mathcal{X}|\}$ *and* $x'_1 \in \{1, \cdots, |\mathcal{X}'|\}$*, if the following inequality holds for* $i_0 \in \{1, \cdots, |\mathcal{X}|\}$ *and* $j_0 \in \{1, \cdots, |\mathcal{X}'|\}$*,*

$$\sum_{i=1}^{|\mathcal{X}|} \sum_{j=1}^{|\mathcal{X}'|} t(i, j, i_0, j_0) q(i, j) < q(i_0, j_0). \tag{7.20}$$

For the proof, see Kashima et al. (2003). The condition (7.20) seems rather complicated, but we can have a simpler condition, if the termination probabilities are constant over all vertices.

Corollary 7.2 *If* $p_q(i) = p'_q(j) = \gamma$ *for any* i *and* j*, the infinite sequence* $\lim_{L\to\infty} R_L(x_1, x'_1)$ *converges if*

$$k_v(v, v') k_e(e, e') < \frac{1}{(1-\gamma)^2}. \tag{7.21}$$

Apparently, the above lemma holds if $0 \le k_v, k_e \le 1$. Standard label kernels such as (7.4) and (7.5) satisfy this condition.

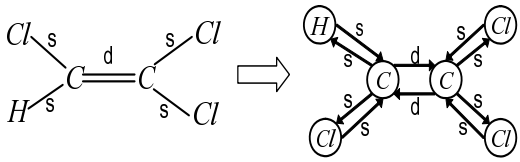


Figure 7.4 A chemical compound is conventionally represented as an undirected graph (*left*). Atom types and bond types correspond to vertex labels and edge labels, respectively. The edge labels s and d denote single and double bonds, respectively. As our kernel assumes a directed graph, undirected edges are replaced by directed edges (*right*).

7.3 Experiments

Chemical compound

We applied our kernel to the prediction of properties of chemical compounds. A chemical compound can naturally be represented as an undirected graph by considering the atom types as the vertex labels, for example, C, Cl, and H, and the bond types as the edge labels, for example, s (single bond) and d (double bond). For our graph kernel, we replaced an undirected edge by two directed edges (figure 7.4) since the kernel assumes directed graphs.

7.3.1 Pattern Discovery Algorithm

We compare our graph kernel with the pattern-discovery (PD) method of Kramer and De Raedt (2001) which is one of the best state-of-the-art methods in predictive toxicology. Like our graph kernel, the PD method counts the number of label sequences appearing in the graph.[5] There are other methods which count more complicated substructures such as subgraphs (Inokuchi et al., 2000), but we focus on Kramer and De Raedt (2001) whose features are similar to ours.

Assume that we have n graphs $G_1, \ldots, G_n$. Let us define $\#(\mathbf{h}, G)$ as the number of appearances of a label sequence $\mathbf{h}$ in G. The PD method identifies the set of all label sequences $\mathcal{H}$ which appear in more than m graphs:

$$\mathcal{H} = \{\mathbf{h} \mid \sum_{i=1}^{n} \delta\left(\#\left(\mathbf{h}, G_i\right) > 0\right) \geq m\},$$

where the parameter m is called the minimum support parameter. Furthermore, it is possible to add extra conditions, for example, selecting only the paths frequent

5. Notice that the definition of label sequences is different from ours in several points, for example, a vertex will not be visited twice in a path. See Kramer and De Raedt (2001) for details.

in a certain class and scarce in the other classes. Each graph G is represented by a vector as

$$G \to (\#(\mathbf{x}_1, G), \ldots, \#(\mathbf{x}_{|\mathcal{H}|}, G)), \tag{7.22}$$

whose dimensionality is the cardinality of $\mathcal{H}$. The PD method is useful for extracting comprehensive features. However, as the minimum support parameter gets smaller, the dimensionality of the feature vectors becomes so large that a prohibitive amount of computation is required. Therefore, the user has to control the minimum support parameter m, such that the feature space does not lose necessary information and, at the same time, computation stays feasible.

The PD method contrasts markedly with our method. Our kernel method puts emphasis on dealing with infinite, but less interpretable features, while the PD method tries to extract a relatively small number of meaningful features. Looking at the algorithms, our method is described by just one equation (7.16), while the PD method's algorithm is rather complicated (De Raedt and Kramer, 2001).

7.3.2 Data sets

We used two data sets, the PTC (predictive toxicology challenge) data set (Helma et al., 2001) and the Mutag data set (Srinivasan et al., 1996). The PTC data set is the result of the following pharmaceutical experiments. Four types of test animals—male mouse (MM), female mouse (FM), male rat (MR), and female rat (FR)—were given 417 compounds. According to its carcinogenicity, each compound was assigned one of the following labels: $\{\mathrm{EE}, \mathrm{IS}, \mathrm{E}, \mathrm{CE}, \mathrm{SE}, \mathrm{P}, \mathrm{NE}, \mathrm{N}\}$, where CE, SE, and P indicate "relatively active," NE and N indicate "relatively inactive," and EE, IS, and E indicate "cannot be decided." To simplify the problem, we relabeled CE, SE, and P as "positive," and NE and N as "negative." The task is to predict whether a given compound is positive or negative for each type of test animal. Thus we eventually had four two-class problems. In the Mutag data set, the task is a two-class classification problem to predict whether each of the 188 compounds has mutagenicity or not. Several statistics of the data sets are summarized in table 7.1.

7.3.3 Experimental Settings and Results

Assuming no prior knowledge, we defined the probability distributions for random walks as follows. The initial probabilities were simply uniform, that is, $p_s(x) = 1/|\mathcal{X}|$. The termination probabilities were determined as a constant γ over all vertices. The transition probabilities $p_t(x|x_0)$ were set as uniform over adjacent vertices. We used (7.4) as the label kernels. In solving the simultanous equations, we employed a simple iterative method (7.15). In our observation, 20 to 30 iterations were enough for convergence in all cases. For the classification algorithm, we used the voted kernel perceptron (Freund and Shapire, 1999), whose performance is known to be comparable to SVMs. In the pattern discovery method, the minimum

Table 7.1 Several statistics of the data sets such as numbers of positive examples (#positive) and negative examples (#negative), maximum degree (max. degree), maximum size of graphs (max. $|\mathcal{X}|$), average size of graphs (avg. $|\mathcal{X}|$), and numbers of vertex labels ($|\Sigma_V|$) and edge labels ($|\Sigma_E|$).

	MM	FM	MR	FR	MUTAG
#POSITIVE	129	143	152	121	125
#NEGATIVE	207	206	192	230	63
MAX. $\lvert\mathcal{X}\rvert$	109	109	109	109	40
AVG. $\lvert\mathcal{X}\rvert$	25.0	25.2	25.6	26.1	31.4
MAX. DEGREE	4	4	4	4	4
$\lvert\Sigma_V\rvert$	21	19	19	20	8
$\lvert\Sigma_E\rvert$	4	4	4	4	4

Table 7.2 Classification accuracies (%) of the pattern discovery method. MinSup shows the ratio of the minimum support parameter to the number of compounds m/n.

MINSUP	MM	FM	MR	FR	MUTAG
0.5%	60.1	57.6	61.3	**66.7**	88.3
1.0%	**61.0**	**61.0**	**62.8**	63.2	87.8
3.0%	58.3	55.9	60.2	63.2	**89.9**
5.0%	60.7	55.6	57.3	63.0	86.2
10%	58.9	58.7	57.8	60.1	84.6
20%	**61.0**	55.3	56.1	61.3	83.5

support parameter was determined as $0.5\%, 1\%, 3\%, 5\%, 10\%$, and 20% of the number of compounds, and the simple dot product in the feature space (7.22) was used as a kernel. In our graph kernel, the termination probability γ was changed from 0.1 to 0.9.

Tables 7.2 and 7.3 show the classification accuracies in the five two-class problems measured by leave-one-out cross-validation. No general tendencies were found to conclude which method is better (the PD was better in MR, FR, and Mutag, but our method was better in MM and FM). Thus it would be fair to say that the performances were comparable in this small set of experiments. Even though we could not show that our method is constantly better, this result is still appealing, because the advantage of our method lies in its simplicity both in concepts and in computational procedures.

7.4 Related Works

We have presented one kernel for graphs based on label sequences, but variants can be obtained by changing the following two points:

Table 7.3 Classification accuracies (%) of our graph kernel. The parameter γ is the termination probability of random walks, which controls the effect of the length of label sequences.

γ	MM	FM	MR	FR	MUTAG
0.1	62.2	59.3	57.0	62.1	84.3
0.2	62.2	61.0	57.0	62.4	83.5
0.3	64.0	61.3	56.7	62.1	**85.1**
0.4	**64.3**	61.9	56.1	63.0	**85.1**
0.5	64.0	61.3	56.1	64.4	83.5
0.6	62.8	61.9	54.4	65.8	83.0
0.7	63.1	62.5	54.1	63.2	81.9
0.8	63.4	**63.4**	54.9	64.1	79.8
0.9	62.8	61.6	**58.4**	**66.1**	78.7

- Removing probabilistic constraints: In our setting, the random walk parameters are determined such that the probabilities of all label sequences sum to 1. One can remove these constraints and simply consider transition "weights," not probabilities.
- Changing the rate of weight decay: The probabilities (or weights) associated with a label sequence could decay as the length of the sequence increases. Variants can be obtained by introducing an extra decaying factor depending on the sequence length.

Geometric and exponential kernels

Recently, Gärtner et al. (2003) proposed two graph kernels called *geometric* and *exponential kernels*. Let $w_t(x_i|x_{i-1})$ denote a weight of transition from x_{i-1} to x_i. Their kernels can be recovered in our framework by setting $p_s(\cdot) = 1$, $p_q(\cdot) = 1$ and replacing the transition probability $p_t(x_i|x_{i-1})$ with

$$\sqrt{\lambda_k} w_t(x_i|x_{i-1})$$

where λ_k is the decaying factor depending on the current sequence length k. In our setting, when the random walk passes through an edge, the probability is multiplied by the same factor regardless of the current sequence length. However, in their setting, the decay rate may get larger when the edge is visited *later*, that is, after traversing many vertices.

In the geometric kernel, λ_k does not depend on k, that is, $\lambda_k = \lambda$. This kernel is quite similar to our kernel and is computed by means of matrix inversion, as in (7.19). An interesting kernel, called the exponential kernel, is derived when

$$\lambda_k = \frac{\beta}{k}.$$

It turns out that this kernel is computed efficiently by the matrix exponential:

$$k(G, G') = \sum_i \sum_j \left[\lim_{L \to \infty} \sum_{\ell=1}^{L} \frac{(\beta T)^\ell}{\ell!} \right]_{ij} = \sum_i \sum_j \left[e^{\beta T} \right]_{ij}.$$

Obviously, possible variants are not limited to these two cases, so there remains a lot to explore.

Label sequence kernels have an intrinsic relationship to the kernels between probability distributions called *probability product kernels* (Jebara and Kondor, 2003). Here the kernel between two probability distributions $\mathbf{p}$ and $\mathbf{p}'$ is defined as

$$k(p, p') = \int_{\Omega} p(\mathbf{x})^{\rho} p'(\mathbf{x})^{\rho} d\mathbf{x} \tag{7.23}$$

Expected likelihood kernel

When $\rho = 1$, the kernel is called *expected likelihood kernel*. Also, when $\rho = 1/2$, the kernel is called Bhattacharrya kernel, which is related to the Hellinger distance. In fact, when edges are not labeled and the vertex kernel is determined as the identity kernel, our kernel can be regarded as the expected likelihood kernel between two Markov models. In such cases. the graph G is perceived as a transition graph of a Markov model and random walking amounts to the emission of symbols. The same idea can be extended to define a kernel for HMMs (Lyngsø et al., 1999). An HMM can be regarded as a labeled graph where edges are not labeled and vertices are *probabilistically* labeled, that is, a vertex randomly emits one of the symbols according to some probability distribution.

Kernels for HMMs

If we regard the kernel $k_z(\mathbf{h}, \mathbf{h}')$ as a joint distribution $p_z(\mathbf{h}, \mathbf{h}')$ that emits a pair of sequences $\mathbf{h}$ and $\mathbf{h}'$, it can be an instance of *rational kernels* (Cortes et al., 2003),

Rational kernels

$$k(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{h}} \sum_{\mathbf{h}'} p_z(\mathbf{h}, \mathbf{h}') p(\mathbf{h}|\mathbf{x}) p'(\mathbf{h}'|\mathbf{x}'),$$

that define a kernel between two probabilistic automata $p(\mathbf{h}|\mathbf{x})$ and $p'(\mathbf{h}'|\mathbf{x}')$ via probabilistic transducer $p_z(\mathbf{h}, \mathbf{h}')$. The rational kernels are not limited to the probabilistic setting, and provide a unified framework for designing kernels via weighted transducers. Cortes et al. (2003) provided no algorithms for acyclic cases. The techniques we introduced in this chapter can be easily applied to the rational kernels for cyclic cases.

7.5 Conclusion

This chapter discussed the design of kernel functions between directed graphs with vertex labels and edge labels. We defined the label sequence kernel by using random walks on graphs, and reduced the computation of the kernel to solving a system of simultaneous linear equations. In contrast to the PD method, our kernel takes into account all possible label sequences without computing feature values explicitly. The structure we dealt with in this chapter is fairly general, and promising in a wide variety of problems in bioinformatics. Potential targets would be DNA and RNA sequences with remote correlations, HTML and XML documents in MEDLINE,

topology graphs, and distance graphs of 3D protein structures, just to mention some.

8 Diffusion Kernels

Risi Kondor
Jean-Philippe Vert

Graphs are one of the simplest types of objects in mathematics. In chapter 7 we saw how to construct kernels between graphs, that is, when the individual examples $\mathbf{x} \in \mathcal{X}$ are graphs. In this chapter we consider the case when the input space $\mathcal{X}$ is itself a graph and the examples are vertices in this graph.

Such a case may arise naturally in diverse contexts. We may be faced with a network, trying to extrapolate known values of some quantity at specific nodes to other nodes. An illustrative example might be the graph of interactions between the proteins of an organism, which can be built from recent high-throughput technologies. Let's say we are trying to learn the localization of proteins in the cell, or their functions. In the absence of other knowledge, a reasonable starting point is to assume that proteins that can interact are likely to have similar localization or functions. Other examples of naturally occurring networks include metabolic and signaling pathways, and also social networks, the World Wide Web, and citation networks.

In other cases we might be faced with something much more intricate that is not itself a network, but can conveniently be modeled as one. Assume we are interested in predicting the physical properties of organic molecules. Clearly, the set of all known organic molecules is very large and it is next to impossible to impose a global metric on it or sensibly fit it into a vector space. On the other hand, it is not so difficult to come up with rules for which molecules are expected to be similar. The saturation of a single bond or the addition of an extra link to a long carbon chain is unlikely to dramatically change the global properties of a molecule. We can ask a human expert to supply us with a set of empirical rules from which we can build a similarity graph and treat our domain as if it were a network.

The challenge is to build learning algorithms that can exploit such graphical structures. The modularity of kernel-based learning suggests that information about the graph should be incorporated in the kernel. Once we have fabricated a good kernel for the graph, we can plug it into our favorite algorithm (support vector

machine [SVM], kernel regression, kernel principal component analysis [KPCA], etc.) and expect the algorithm to perform similarly to how it performs in more conventional settings.

The function of the kernel is to provide a *global* similarity metric, whereas graphs incorporate information on *local* similarity. The kernel must be able to express the degree of similarity between any two examples $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ with fine distinctions in the degree to which $\mathbf{x}$ and $\mathbf{x}'$ are distant from each other in the graph. In contrast, the graph itself only expresses whether $\mathbf{x}$ and $\mathbf{x}'$ are neighbors or not. In section 8.1 we show how the physical process of diffusion suggests a natural way of constructing a kernel from such *local* information, and section 8.2 discusses how to compute the diffusion kernel in specific cases. In section 8.3 we highlight the interpretation of the diffusion kernel in the context of regularization operators.

In section 8.4 we then apply these ideas to the network of chemical pathways in the cell and show how this can boost microarray analysis. Finally, section 8.5 recasts the central ideas of this chapter in a slightly more abstract form and provides an outlook on their role in generalizing kernel-based learning to not just graphs but a wide variety of mathematical objects, from finite groups to Riemannian manifolds.

8.1 Random Walks and Diffusion

The role of the kernel k is to provide a similarity metric on the input space $\mathcal{X}$. Let $\mathcal{X}$ be the vertices, labeled from 1 to n, of an unirected graph G. For now, we assume that G is unweighted, that is, any two vertices i and j in G are either neighbors, denoted $i \sim j$, or they are not neighbors, denoted $i \not\sim j$.

Positive definiteness

A kernel must also satisfy the mathematical requirements of being symmetric and positive definite. Recall that positive definiteness means that for any choice of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \in \mathcal{X}$ and any choice of coefficients $c_1, c_2, \ldots, c_m \in \mathbb{R}$,

$$\sum_{i=1}^{m} \sum_{j=1}^{m} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0,$$

this being the crucial condition for the existence of the feature mapping $\Phi : \mathcal{X} \mapsto \mathcal{F}$ satisfying $k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$. For finite graphs, the kernel can equivalently be specified by an $n \times n$ matrix K, with $K_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. Since k and K are essentially the same object, we shall refer to both as "the kernel" and alternate between the two notations depending on whether we want to emphasize the functional or the matrix aspect.

Shortest-path distance

The simplest measure of similarity on G is the shortest-path distance $d(i, j)$, but it is not clear how to construct a positive definite function from d. Furthermore, d is extremely sensitive to the insertion/deletion of individual edges. In many potential applications, the connectivity information is itself derived from data, and as such is subject to noise. A more robust similarlity measure, perhaps involving averaging over many paths, is more desirable.

Random walks

This leads to the idea of random walks. Recall that a random walk is a process generating paths $z_0 z_1 z_2 \dots z_T$. Starting from a given vertex z_0, at each timestep $t = 1, 2, \dots, T$ the new vertex z_t is chosen at random from among the neighbors of z_{t-1}, each neighbor having the same probability of being selected.

A compact representation of this process is provided by the *adjacency matrix* indexgraph!adjacency matrix

$$A_{ij} = \begin{cases} 1 & \text{if } i \sim j \\ 0 & \text{otherwise} \end{cases}$$

or rather the *normalized adjacency matrix*

$$Q_{ij} = \begin{cases} 1/\gamma_j & \text{if } i \sim j \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

where γ_j is the degree of vertex j, that is, the number of edges incident at j. It is easy to see that if $p_t = (p_1^{(t)}, p_2^{(t)}, \dots, p_n^{(t)})^\top$ describes the probability of finding the random walker at each vertex at time t, then $p_{t+1} = Q p_t$. We say that Q is the transition matrix of the random walk. Applying this relation recursively shows that raising the normalized adjacency matrix to the power T,

$$P_T = Q^T, \tag{8.2}$$

gives the matrix whose i, j element describes the probability of a random walker starting from j being found at i at time T.

Generally, random walks have a tendency to meander about close to their origin. This is because when i and j are close and T is reasonably large, in most graphs there is a very large number of possible length T paths from i to j. When i and j are far apart the choice of paths is much more restricted and $[P_T]_{i,j}$ will be correspondingly small. Unfortunately, P_T is not suitable as a kernel, for a number of reasons:

1. There is no clear prescription for how to choose T. Any choice less than the diameter of G will lead to pairs of vertices not reachable from one another, and a corresponding absolute cutoff in the kernel. Kernels with such limited horizon do not have a sense of the global structure of G. On the other hand, choosing too large a T might make the peak of $[P_T]_{i,j}$ around i very flat, resulting in a kernel unable to differentiate betweeen nearby vertices.

2. If the graph has symmetries, particular choices of T might make certain vertices unreachable. For example, if G is just a cycle, an even number of steps could never take us from vertex i to either of its neighbors. Similarly, if T is odd, we could not get back to the vertex we started from.

3. Finally, and most seriously, P_T is generally not positive definite; in fact, it is not even guaranteed to be symmetric.

To see how to overcome these difficulties we first need to take a brief detour to continuous spaces. Physical ideas borrowed from the continuous case will help us construct a modified random walk leading to a natural kernel on G.

The Gaussian kernel and diffusion

One of the most popular kernels on $\mathcal{X}=\mathbb{R}^N$ is the Gaussian radial basis function kernel (Gaussian RBF or just Gaussian kernel)

$$k(\mathbf{x},\mathbf{x}') = \frac{1}{(2\pi\sigma^2)^{N/2}}\, e^{-\|\,\mathbf{x}-\mathbf{x}'\,\|^2/(2\sigma^2)} \tag{8.3}$$

with length scale parameter σ. It may not be immediately obvious from its functional form, but the Gaussian kernel is positive definite on $\mathbb{R}^N$.

The Gaussian has many famous properties, but for now we are going to concentrate on the fact that fixing $\mathbf{x}'=\mathbf{x}_0$ and letting $t=\sigma^2/2$,

$$k_{\mathbf{x}_0}(\mathbf{x},t) = \frac{1}{(4\pi t)^{N/2}}\, e^{-\|\,\mathbf{x}-\mathbf{x}_0\,\|^2/(4t)}$$

is the solution of the *diffusion equation*

$$\frac{\partial}{\partial t}\, k_{\mathbf{x}_0}(\mathbf{x},t) = \left[\frac{\partial^2}{\partial \mathbf{x}_{(1)}^2} + \frac{\partial^2}{\partial \mathbf{x}_{(2)}^2} + \ldots + \frac{\partial^2}{\partial \mathbf{x}_{(N)}^2}\right] k_{\mathbf{x}_0}(\mathbf{x},t), \tag{8.4}$$

with Dirac delta initial conditions $k_{\mathbf{x}_0}(\mathbf{x},0)=\delta(\mathbf{x}-\mathbf{x}_0)$. Here we use parenthesized indices to make it clear that we are differentiating with respect to the ith coordinate and not the ith training example. The second-order differential operator in (8.4) is called the *Laplacian* and is often denoted simply by Δ, reducing the diffusion equation to

$$\frac{\partial}{\partial t}\, k_{\mathbf{x}_0}(\mathbf{x},t) = \Delta\, k_{\mathbf{x}_0}(\mathbf{x},t)\,. \tag{8.5}$$

The physical meaning of these equations is clear: (8.5) describes how heat, gases, and so on, introduced at $\mathbf{x}_0$, diffuse with time in a homogeneous, isotropic medium. In learning theory, using the Gaussian kernel amounts to evoking a similar picture of the diffusion of labels $y_1, y_2, \ldots, y_m$ in $\mathcal{X}$. Every learning algorithm must make some sort of assumption about how similarity between inputs $\mathbf{x},\mathbf{x}'\in\mathcal{X}$ will lead to similarity between the corresponding outputs (labels) y, y'. The assumption behind the Gaussian kernel is essentially that $y(x)$ is to be approximated by diffusing the training labels to the rest of $\mathcal{X}$. Using a sophisticated algorithm such as a SVM complicates the picture somewhat, but the underlying idea remains the same.

Now we ask how to transplant these ideas to the discrete setting, in particular, to when $\mathcal{X}$ is a graph. Going back to the idea of random walks, the key modification we make to (8.2) is to make time continuous by taking the limit

$$K_\beta = \lim_{s\to\infty} \left(I + \frac{\beta L}{s}\right)^s \qquad s\in\mathbb{N}, \tag{8.6}$$

Diffusion on graphs

which corresponds to a random walk with an infinite number of infinitesimally small steps (I is the identity matrix and β is a real parameter). The time evolution of this "continuous time random walk" is now governed by the matrix L, which we again define as a close relative of the adjacency matrix,

Graph Laplacian

$$L_{ij} = \begin{cases} 1 & \text{if } i \sim j \\ -\gamma_i & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{8.7}$$

The negative elements on the diagonal serve the same function as dividing by γ_j in (8.1): their presence guarantees that each column of K_β, regarded as a probability distribution over vertices, remains normalized. In spectral graph theory L is known as the *graph Laplacian*, already suggesting that we are on the right track in developing the analogy with the continuous case. The kernel (8.6) we call the *diffusion kernel* or the *heat kernel* on G.

Properties of diffusion kernels

By analogy with the exponentiation of real numbers, the limit in (8.6) is called the *matrix exponential* of L:

$$e^{\beta L} = \lim_{s\to\infty} \left(I + \frac{\beta L}{s}\right)^s \qquad s\in\mathbb{N}\,. \tag{8.8}$$

Note that $e^{\beta L}$ yields a matrix, but is not equivalent to componentwise exponentiation $J_{ij} = e^{\beta L_{ij}}$. Matrix exponentiation shares many properties with the ordinary exponential function, including the power series expansion

$$e^{\beta L} = I + \beta L + \frac{\beta^2}{2} L^2 + \frac{\beta^3}{6} L^3 + \ldots$$

and the fact that $e^{\beta L}$ satisfies the differential equation

$$\frac{\partial}{\partial \beta}\, e^{\beta L} = L\, e^{\beta L}. \tag{8.9}$$

One important difference is that in general the identity

$$e^{\beta(A+B)} = e^{\beta A}\, e^{\beta B}$$

does *not* hold in the matrix case.

An important effect of defining kernels in such an exponential form is that we get positive definiteness "for free," as can be seen by writing

$$e^{\beta L} = \lim_{s\to\infty}\left(I + \frac{\beta L}{s}\right)^s = \lim_{2s\to\infty}\left(I + \frac{\beta L}{2s}\right)^{2s}$$

and appealing to the well-known fact that any even power of a symmetric matrix is always positive definite. In fact, it is possible to prove that any continuous family of positive definite matrices K_β indexed by a real parameter β in such a way that $K_0 = I$ is of the form $K_\beta = e^{\beta H}$ for some symmetric matrix H.

Analogies

The diffusion kernel will generally increase with increasing shortest-path distance between vertices, but there is no simple one-to-one relationship between d and K_β. Rather, it is helpful to think of K in terms of actual physical processes of diffusion. The function of the parameter β is to control the extent of the diffusion, or to specify the length scale, similarly to σ in the Gaussian kernel.

The correspondence between diffusion kernels and the Gaussian kernel is spelled out even more clearly by considering an infinite regular square grid on $\mathbb{R}^N$. Restricting ourselves to two dimensions for notational simplicity and labeling the vertices with their integer coordinates, the Laplacian becomes

$$L_{(i_1,i_2),(j_1,j_2)} = \begin{cases} 1 & \text{if } i_1 = j_1 \text{ and } i_2 = j_2 \pm 1 \\ 1 & \text{if } i_2 = j_2 \text{ and } i_1 = j_1 \pm 1 \\ -4 & \text{if } i_1 = j_1 \text{ and } i_2 = j_2 \\ 0 & \text{otherwise}. \end{cases}$$

Applied to a function $f : \mathcal{X} \mapsto \mathbb{R}$ [regarded as a vector indexed by (i_1, i_2) pairs], this gives

$$(Lf)_{i_1,i_2} = f_{i_1,i_2+1} + f_{i_1,i_2-1} + f_{i_1+1,i_2} + f_{i_1-1,i_2} - 4f_{i_1,i_2},$$

which is a well-known formula for the finite differences discretization of the continous Laplace operator Δ, commonly used in the numerical solution of partial differential equations. Hence, L can be regarded as just the discretized counterpart of Δ, and, correspondingly, K can be regarded as the discretized Gaussian RBF. The correspondence can be made exact by proving that in the limit of the grid spacing going to zero, K will indeed approach the Gaussian kernel (8.3).

On more general graphs the key to understanding diffusion kernels is the differential equation (8.9). This expresses that starting from the identity matrix $K_0 = I$, the kernel K_β is generated by successive infinitesimal applications of L. Note that the Laplacian encodes strictly local information about the graph. However, through this continuous process expressed by the differential equation, L is lifted to a smooth global similarity measure on G, which is exactly what a kernel is supposed to be.

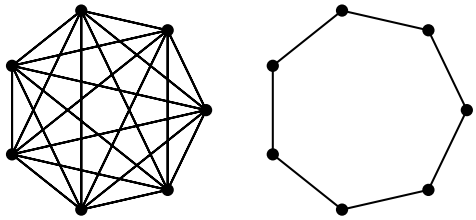


Figure 8.1 Two examples of elementary graphs for which the diffusion kernel can be found in closed form: the complete graph and the closed chain over seven vertices.

8.2 Computing the Diffusion Kernel

At this point, the reader might be wondering how he or she might compute the limit (8.8) on a computer in a finite amount of time. In general, the way to proceed is to compute the normalized eigenvectors $v_1, v_2, \ldots, v_n$ and corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ of L and exploit orthogonality to write

$$L^s = \left(\sum_{i=1}^{n} v_i \, \lambda_i \, v_i^\top \right)^s = \sum_{i=1}^{n} v_i \, \lambda_i^s \, v_i^\top ,$$

from which

$$e^{\beta L} = I + \left(\sum_{i=1}^{n} v_i \, \beta \lambda_i \, v_i^\top \right) + \left(\sum_{i=1}^{n} v_i \, \frac{(\beta \lambda_i)^2}{2} \, v_i^\top \right) + \ldots = \sum_{i=1}^{n} v_i \, e^{\beta \lambda_i} \, v_i^\top , \quad (8.10)$$

reducing matrix exponentiation to real exponentiation. Unfortunately, the complexity of diagonalizing the Laplacian is of order n^3, which threatens to be computationally more expensive than the learning itself for most learning algorithms. However, there are a few special graphs for which the diffusion kernel can be computed in closed form.

Complete graphs

In the complete graph of order n every vertex is linked to every other vertex, so the Laplacian is $L_{ij} = 1 - n\,\delta_{ij}$. In this case (8.9) can be solved explicitly giving

$$k(i,j) = K_{i,j} = \begin{cases} \dfrac{1 + (n-1)\, e^{-n\beta}}{n} & \text{for } \; i = j \\ \dfrac{1 - e^{-n\beta}}{n} & \text{for } \; i \neq j, \end{cases}$$

showing that with increasing β, the kernel relaxes exponentially to $k(i,j) = 1/n$. The asymptotically exponential character of this solution converging to the uniform kernel is a direct consequence of the form of the diffusion equation. We shall see this type of behavior recur in other examples.

Closed chains

When G is a single closed chain, $k(i,j)$ can clearly only depend on the distance $d(i,j)$ along the chain between i and j. Labeling the vertices consecutively from 0 to $n-1$ and fixing i (without loss of generality, taking $i=0$), $k_0(j)=k(0,j)$ can be expressed in terms of its discrete Fourier transform

$$k(0,j) = k_0(j) = \frac{1}{\sqrt{n}} \sum_{\nu=0}^{n-1} \widehat{k}_0(j) \cos \frac{2\pi\nu j}{n} \ .$$

The heat equation implies

$$\frac{d}{d\beta} k_0(j) \ = \ k_0((j+1) \bmod n) - 2\,k_0(j) + k_0((j-1) \bmod n),$$

which after some trigonometry translates into

$$\frac{d}{d\beta} \widehat{k}_0(\nu) = -\,2 \left(1 - \cos \frac{2\pi\nu}{n}\right) \widehat{k}_0(\nu),$$

showing that each Fourier coefficient decays independently. Now applying the inverse Fourier transform, the solution corresponding to the initial condition $k_0(i) = \delta_{i,0}$ at $\beta=0$ can be expressed as

$$k_0(j) = \frac{1}{n} \sum_{\nu=0}^{n-1} e^{-\omega_\nu \beta} \cos \frac{2\pi\nu j}{n},$$

where $\omega_\nu = 2\left(1 - \cos \frac{2\pi\nu}{n}\right)$. Hence the full kernel is

$$k(i,j) = \frac{1}{n} \sum_{\nu=0}^{n-1} e^{-\omega_\nu \beta} \cos \frac{2\pi\nu (i-j)}{n} \ .$$

p-regular trees

There also exist special solutions for tree-shaped graphs, albeit infinite trees with no root. A p-regular tree is an infinite graph with no cycles in which every vertex has exactly p neighbors (figure 8.2). Clearly, such a tree looks the same from every vertex, so $k(i,j)$ can only depend on the shortest-path distance $d(i,j)$. Even for such simple, highly symmetric graphs, the formula for the diffusion kernel is not trivial and has only recently been derived (Chung and Yau, 1999):

$$k(i,j) = \frac{2}{\pi(k-1)} \int_0^\pi \frac{e^{-\beta\left(1 - \frac{2\sqrt{k-1}}{k} \cos x\right)} \sin x \,[\,(k-1)\sin(d+1)\,x - \sin(d-1)\,x\,]}{k^2 - 4\,(k-1)\,\cos^2 x}\, dx$$

for $d = d(i,j) \geq 1$, and

$$k(i,i) \ = \ \frac{2k(k-1)}{\pi} \int_0^\pi \frac{\exp\left(-\beta\,(1 - \frac{2\sqrt{k-1}}{k} \cos x)\right) \sin^2 x}{k^2 - 4\,(k-1)\,\cos^2 x}\, dx$$

for the diagonal elements. The case of infinite rooted trees can be reduced to p-regular trees using a trick from physics called the "method of images," as described in Kondor and Lafferty (2002).

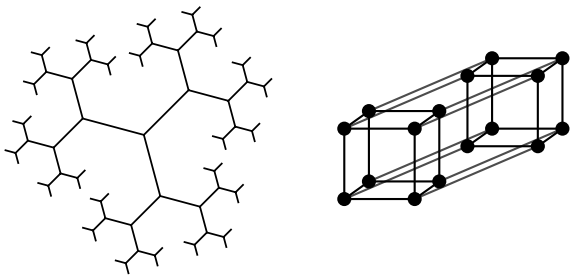


Figure 8.2 *Left,* A few edges of the three-regular tree. The tree extends to infinity in all directions. *Right,* The four-dimensional hypercube can be regarded as the complete graph of order two (two vertices joined by a single edge) "cubed."

Product graphs

Another way to compute diffusion kernels is to reduce complex graphs to elementary ones. Specifically, let G_1 and G_2 be two undirected, unweighted graphs with n_1 and n_2 vertices, respectively. The direct product graph $G = G_1 \times G_2$ will then have $n_1 \cdot n_2$ vertices labeled by pairs of integers (i_1, i_2), with $1 \leq i_1 \leq n_1$ and $1 \leq i_2 \leq n_2$. Two vertices (i_1, i_2) and (j_1, j_2) will then be neighbors either if $i_1 = j_1$ and $i_2 \sim j_2$ or if $i_2 = j_2$ and $i_1 \sim j_1$. The infinite square grid that we encountered at the end of section 8.1 is an example of such a structure. More generally, also encompasssing weighted graphs, the adjacency matrix of the product graph will be related to the adjacency matrices of the factor graphs by

$$A = A_1 \otimes I_2 + I_1 \otimes A_2,$$

where I_1 and I_2 are the $n_1 \times n_1$ and $n_2 \times n_2$ unit matrices, respectively. The Laplacians are similarly related:

$$L = L_1 \otimes I_2 + I_1 \otimes L_2$$

and the corresponding diffusion kernel will be the direct product of the diffusion kernels on the factor graphs:

$$K_\beta = K_\beta^{(1)} \otimes K_\beta^{(2)},$$

as can easily be seen by plugging this into the diffusion equation

$$\frac{\partial}{\partial \beta} K_\beta = L K_\beta$$

and invoking the product rule for differentiation.

The product graph construction makes it possible to compute diffusion kernels on the D-dimensional hypercube, regarded as the D-fold power of a complete graph of order 2:

Hypercubes

$$k(\mathbf{x}, \mathbf{x}') \propto \left(\frac{1 - e^{-2\beta}}{1 + e^{-2\beta}}\right)^{d(\mathbf{x},\mathbf{x}')} = (\tanh \beta)^{d(\mathbf{x},\mathbf{x}')},$$

where $d(\mathbf{x}, \mathbf{x}')$ is the Hamming distance betweeen vertices $\mathbf{x}$ and $\mathbf{x}'$. More generally, we may consider products of complete graphs of orders $g_1, g_2, \ldots g_D$, and compute the kernel

$$k(\mathbf{x}, \mathbf{x}') \propto \prod_{i=1}^{D} \left[\frac{1 - e^{-\beta g_i}}{1 + (g_i - 1)e^{-\beta g_i}}\right]^{d_i(\mathbf{x},\mathbf{x}')},$$

where $d_i(\mathbf{x}, \mathbf{x}') = 0$ if $\mathbf{x}$ and $\mathbf{x}'$ match in the ith index and 1 otherwise. This construction can be used to apply diffusion kernels to learning problems involving categorical data (Kondor and Lafferty, 2002), assuming D distinct attributes with $g_1, g_2, \ldots g_D$ possible values.

8.3 Regularization Theory

The correspondence between kernel-based algorithms and regularization theory is now widely recognized in the machine learning community (Smola et al., 1998; Girosi, 1998; Girosi et al., 1995; Tikhonov and Arsenin, 1977). SVMs, Gaussian processes, and so on, can all be cast in the form of searching some linear space of functions $\mathcal{H}$ to find $f : \mathcal{X} \mapsto \mathcal{Y}$ minimizing the *regularized risk*

$$R_{\text{reg}}[f] = \sum_{i=1}^{m} \mathcal{L}(f(\mathbf{x}_i), y_i) + \Omega[f], \tag{8.11}$$

where $\Omega[f]$ is expressed in terms of a *regularization operator* $P : \mathcal{H} \mapsto \mathcal{H}$ as

$$\Omega[f] = \int_{\mathcal{X}} [(Pf)(\mathbf{x})]^2 \, d\mathbf{x}.$$

Without going into detail, we note that (8.11) expresses a tradeoff between fitting the training data and generalizing well to future examples. Given a loss function $\mathcal{L}$, the first term tries to minimize the error of f on the training set, while the second term stands for the competing objective of restricting f to "desirable" functions according to some criterion embodied in P. The choice of algorithm corresponds to choosing $\mathcal{L}$ and the choice of kernel to choosing the regularization operator.

When $\mathcal{X}$ is a finite graph, f is just a vector, P is a matrix, and the regularization term becomes

$$\Omega[f] = \| Pf \|^2 = f^\top (P^\top P) f. \tag{8.12}$$

Since $P^\top P$ is symmetric, its normalized eigenvectors $v_1, v_2, \ldots, v_n$ form an orthonormal basis, and expressing f as $f = \sum_{i=1}^{n} c_i v_i$, (8.12) becomes

$$\Omega[f] = \sum_{i=1}^{n} \lambda_i c_i^2,$$

where $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues corresponding to $v_1, v_2, \ldots, v_n$. The effect of the regularization term in (8.11) will be to force most of the energy in f to the low eigenvalue modes of $P^\top P$. From the learning theory point of view, our goal is to make f as smooth as possible, in the sense of making few sudden jumps between neighboring vertices. Jagged functions are likely to overfit the training data and not generalize well to future examples. Hence, we are interested in regularization operators whose eigenfunctions form a progression from the smoothest to the roughest functions on our graph.

Connection between K and P

To see how diffusion kernels fare from the regularization theory point of view, it remains to find the connection between the kernel and the regularization operator already alluded to. For concreteness, consider support vector regression, which aims to fit to the data a function f of the form $f(\mathbf{x}) = \langle \boldsymbol{w}, \Phi(\mathbf{x}) \rangle$ for some feature space vector $\boldsymbol{w}$ by minimizing

$$\frac{1}{2} \| \boldsymbol{w} \| + C \sum_{i=1}^{m} | f(\mathbf{x}_i) - y_i |_\epsilon ,$$

where $| f(\mathbf{x}) - y |_\epsilon$ is the ϵ-insensitive loss function $\max \{ 0, | f(\mathbf{x}) - y | - \epsilon \}$ (Vapnik, 1995). As in other kernel methods, the solution of this minimization problem will be of the form

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x}), \tag{8.13}$$

reducing it to finding the coefficients $\alpha_1, \alpha_2, \ldots, \alpha_m$ minimizing

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{m} \sum_{j=1}^{m} | \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) - y_j |_\epsilon . \tag{8.14}$$

When $\mathcal{X}$ is discrete, if we collect the coefficients in a vector, the first term becomes just $\alpha^\top K \alpha$. Similarly, the regularization term (8.12) can be written as $\Omega[f] = \alpha^\top K (P^\top P) K \alpha$. Then comparing (8.14) with (8.11) we see that the two can be made equivalent by setting

$$P^\top P = K^{-1} \quad \text{and} \quad \mathcal{L}(f(\mathbf{x}), y) = 2C | f(\mathbf{x}) - y |_\epsilon .$$

Since K is positive definite, we may just take $P = K^{-1/2}$. The relationship between kernel and regularization is rather direct.

Specifically, referring back to (8.10), L, K, and $P^\top P$ share the same orthonormal system of eigenvectors $v_1, v_2, \ldots, v_n$ and their eigenvalues are related by

$$\lambda_i^{(K)} = \exp(\,\beta\,\lambda_i^{(L)}\,) \qquad \lambda_i^{(P^\top P)} = \exp(\,-\beta\,\lambda_i^{(L)}\,). \tag{8.15}$$

Furthermore, from the definition of the graph Laplacian (8.7),

$$-\,\lambda_i^{(L)} = -\,v_i^\top L\, v_i = \sum_{\mathbf{x}\sim\mathbf{x}'} \left(v_i(\mathbf{x}) - v_i(\mathbf{x}')\right)^2,$$

which can be interpreted as how many edges v_i "violates." Ordering $v_1, v_2, \ldots, v_n$ so that $-\lambda_1 \leq -\lambda_2 \leq \ldots \leq -\lambda_n$, the first eigenvector will always be the constant function on G with eigenvalue 0. The second eigenvector will tend to be positive on roughly one half of G and negative on the other half with a smooth transition in between. The third, fourth, and so on, eigenvectors correspond to successive subdivisions, all the way to the last eigenvectors, which oscillate rapidly, changing sign between most pairs of neigboring vertices. Together with (8.15) this shows that the regularization operator corresponding to the diffusion kernel does indeed establish a basis of functions on G sorted by smoothness.

The natural analogy is the Fourier basis of sine and cosine functions on $\mathbb{R}^N$. In general, K acts as a smoothing operator, since it dampens the "high frequency" modes of the Laplacian, while P is a coarsening operator, because it exaggerates them.

This type of analysis involving operators and their eigensystems (albeit in a somewhat more rigorous form) is at the center of spectral graph theory (Chung, 1997), some of the cornerstones of which are the Laplacian and the heat kernel. In fact, we could have motivated this whole chapter purely by regularization ideas, and derived the diffusion kernel that way, instead of talking about the actual physical process of diffusion, or appealing to the analogy with the Gaussian kernel.

8.4 Applications

In this section we present an application of the diffusion kernel idea to the analysis of gene expression data and metabolic pathways. We first present in subsection 8.4.1 a graph of gene, called the metabolic gene network, which encodes our current knowledge of metabolic pathways. We then show how the regularization operator associated with the diffusion kernel on this graph can be useful for extracting pathway activity from gene expression data, and illustrate this approach by a short analysis of gene expression during two cell cycles. More details on this approach can be found in Vert and Kanehisa (2003b), and more data analysis is presented in Vert and Kanehisa (2003a).

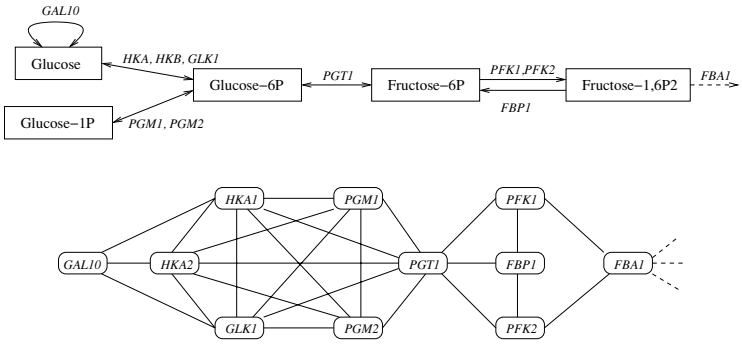


Figure 8.3 *Top*, The first three reactions of the glycolysis pathway, together with the catalyzing enzymes in the yeast *S. cerevisiae*. *Bottom*, The metabolic gene network derived from these reactions by linking two genes whenever they can catalyze two successive reactions.

8.4.1 The Metabolic Gene Network

At the molecular level life can be described as an continual flow of chemical reactions that degrade, synthesize, or transform various molecules for various purposes. In particular, metabolism encompasses the processes by which energy is provided for vital processes and activities, and by which new material is assimilated. Metabolic processes are usually organized into series of chemical reactions, called *metabolic pathways*, that take place sequentially in the cell. As an example, glycolysis is the metabolic pathway in charge of degrading glucose into pyruvate with the concomitant production of adenosine triphosphate (ATP) molecules. Figure 8.3 (top) shows the first three reactions of glycolysis: addition of a phosphate group to the glucose molecule to obtain glucose-6P, followed by an izomerization of glucose-6P into fructose-6P, and by the addition of a second phosphate group to obtain fructose-1,6P2.

Metabolic pathways

Each reaction in a pathway usually requires the presence of a particular molecule, called an enzyme, to occur. Enzymes catalyze reactions, that is, they facilitate the reaction usually by placing the various substrates in a precise spatial configuration. Most enzymes in biochemistry are proteins synthesized by the cell itself, which are encoded in the genome of the organism. In Figure 8.3 (top) the enzymes are characterized by the name of the genes that encode them. For example, the izomerization from glucose-6P to fructose-6P is catalyzed by the protein encoded by the gene PGT1 in yeast. When several genes are associated with a single reaction, it is either because the proteins they encode form a complex, or because several different proteins can catalyze the same reaction.

Metabolic gene network

The *metabolic gene network* is derived from the set of metabolic pathways as follows. It is an undirected graph whose vertices are the genes of a given organism,

and with an edge between two genes whenever the proteins they encode can participate in the catalysis of two successive reactions, that is, two reactions such that the main product of the first one is the main substrate of the second one. As an example, the local metabolic graph network for the yeast derived from the first three reactions of glycolysis is shown in Figure 8.3 (bottom).

When all known metabolic pathways are considered, the resulting metabolic gene network is complex for at least two reasons. First, the same chemical compound (such as glucose) can be present in different metabolic pathways, and therefore edges can link genes which catalyze reactions in different pathways. Second, a given gene can usually catalyze more than one reaction.

In the following we use the metabolic gene network for the yeast *Sacchromyces cerevisiae* derived from the metabolic pathways available in the LIGAND database of chemical compounds of reactions in biological pathways (Goto et al., 2002). The resulting graph contains 774 nodes linked through 16,650 edges.

8.4.2 Gene Expression

Reactions in a metabolic pathway occur when both the necessary subtrates and the necessary enzymes are present. As a result, a cell can control its metabolism by controlling the quantity of each enzyme. Because enzymes are proteins, the first level of control of their concentrations is the control of gene expression. For example, in the bacterium *Escherichia coli*, the presence of tryptophan inhibits the expression of the genes that encode the enzymes which catalyze the reactions of the tryptophan synthesis pathway.

DNA microarrays

DNA microarray technology enables the monitoring of gene expression for all genes of an organism simultaneously. It is therefore tempting to try to make sense of gene expression data in terms of pathways, at least for the genes that encode enzymes. More precisely, by following the expression of enzyme genes through various experiments, one can hope to detect activated or inhibited pathways, suggest new pathways, or detect mistakes in the current pathway databases. As a first step toward these ambitious goals we now present a method for automatically detecting activity levels of known pathways by confronting gene expression data with the metabolic gene network.

8.4.3 Mathematical Formulation

Let us represent the set of genes by the finite set $\mathcal{X}$. The metabolic gene network is a simple graph $\Gamma = (\mathcal{X}, \mathcal{E})$ with the genes as vertices. The set of expression profiles measured by DNA microarray for a gene $\mathbf{x} \in \mathcal{X}$ is a vector $e(\mathbf{x}) \in \mathbb{R}^p$, where p is the number of microarray measurements. By subtracting the mean profile from all genes, we suppose below that the set of profiles is centered, that is, $\sum_{\mathbf{x}\in\mathcal{X}} e(\mathbf{x}) = 0$.

We formulate our problem as the problem of automatically finding profiles which exhibit some coherence with respect to the graph topology. Formally speaking, a profile is a vector $v \in \mathbb{R}^p$. We don't require v to be any actual gene expression

profile, but rather use it to represent some more abstract or hidden information, such as the quantity of some substance in the cell, or the activity of a pathway. Intuitively, if v represents the evolution of such a biological quantity, then expression profiles of genes participating in or affected by this event should exhibit some form of correlation with v.

For a zero-mean candidate profile $v \in \mathbb{R}^p$ (i.e., $\sum_{i=1}^p v_i = 0$), let us therefore call $f_1(\mathbf{x}) := v^T e(\mathbf{x})$ the correlation between the profile v and the gene expression profile $e(\mathbf{x})$. Typically, if v represents the activity level of a pathway where gene $\mathbf{x}$ plays a regulatory role, then $f_1(\mathbf{x})$ is likely to be either strongly positive or strongly negative.

These remarks lead to a key observation. If v represents the activity of a pathway, then $f_1(\mathbf{x})$ is likely to be particularly positive or negative for several of the enzymes $\mathbf{x}$ that catalyze the corresponding reactions. Observed on the metabolic gene graph, this suggests that f_1 should have less variations on average between adjacent nodes if v corresponds to a true biological process than otherwise. Indeed, we can at least expect some local regularities in the regions of the graph that correspond to the pathways related to the process.

Smoothness on the network

This is where the diffusion kernel becomes useful. Recall from section 8.3 that the diffusion kernel K_1 on the metabolic gene network defines a regularization operator $\Omega_1[f]$ on functions $f : \mathcal{X} \to \mathbb{R}$ that is small when f has little high-frequency energy. This suggests looking for candidate profiles v such that $\Omega_1[f_1]$ be as small as possible. Writing f_1 in a dual form $f_1 = K_1\alpha$, this means requiring $\alpha^T K_1 \alpha / \alpha^T \alpha$ to be as small as possible.

On the other hand, minimizing $\Omega_1[f_1]$ over v is likely to be an ill-posed or at least an ill-conditioned problem. First observe that any component of v orthogonal to the linear span of $\{e(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}$ does not modify f_1. This suggests restricting v to this subspace, that is, writing v as $v = \sum_{\mathbf{x}\in\mathcal{X}} \beta(\mathbf{x}) e(\mathbf{x})$, and therefore $f_1 = K_2\beta$ where K_2 is the Gram matrix of the linear kernel $K_2(\mathbf{x}, \mathbf{y}) = e(\mathbf{x})^T e(\mathbf{y})$ and $\beta : \mathcal{X} \to \mathbb{R}$ is a dual form of f_1. Second, when the linear span of $\{e(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}$ is large, eventually the whole space of centered profiles, then a perfectly smooth function might be found whether or not a "true" biological correlation exists between the profiles and the graph. This suggests imposing some form of regularity on v, such as being close to the directions of natural variations between profiles. This is achieved by requiring $\Omega_2[f_1] = \beta' K_2 \beta / \beta' \beta$ to be as small as possible. In order to end up with a computationally feasible formulation, Vert and Kanehisa (2003b) proposed decoupling the problem as follows: find two different functions $f_1 = K_1\alpha$ and $f_2 = K_2\beta$ such that $\Omega_1[f_1]$ and $\Omega_2[f_2]$ are both small, while f_1 and f_2 are as similar as possible. A possible measure of similarity between f_1 and f_2 being their correlations $f_1^\top, f_2$, the different goals can be fulfilled simultaneously by maximizing the following functional:

Natural variations

$$\gamma(\alpha, \beta) := \frac{\alpha^T K_1 K_2 \beta}{\left(\alpha^T \left(K_1^2 + \delta K_1\right) \alpha\right)^{\frac{1}{2}} \left(\beta^T \left(K_2^2 + \delta K_2\right) \beta\right)^{\frac{1}{2}}}, \tag{8.16}$$

where δ is a parameter that controls the tradeoff between regularities of f_1 and f_2 on the one hand, and similarity of f_1 and f_2 on the other hand. It turns out that (8.16) can be seen as a regularized form of canonical component analysis (Bach and Jordan, 2002), equivalent to the following generalized eigenvalue problem:

Kernel CCA

$$\begin{pmatrix} 0 & K_1K_2 \\ K_2K_1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \rho \begin{pmatrix} K_1^2 + \delta K_1 & 0 \\ 0 & K_2^2 + \delta K_2 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{8.17}$$

As pointed out in Bach and Jordan (2002) and Vert and Kanehisa (2003b) this problem can be solved efficiently and results in a series of pairs of features $\{(\alpha_i, \beta_i), i = 1, \ldots, n\}$ with decreasing values of $\gamma(\alpha_i, \beta_i)$. The corresponding profiles can be recovered by $v_i = \sum_{\mathbf{x} \in \mathcal{X}} \beta_i(\mathbf{x}) e(\mathbf{x})$.

8.4.4 Analysis of α Factor Release

In order to illustrate this method on a real-world example we compared the metabolic gene network with a collection of 18 expression measurements for 6198 yeast genes, collected every 7 minutes after cells were synchronized in G_1 by addition of α factor (Spellman et al., 1998). The original goal of Spellman et al. (1998) was to detect genes whose expression exhibits periodicity related to the cell cycle.

The analysis that follows is restricted to the 756 genes of the metabolic gene netwok with an expression profile in this set. The profiles contain 18 points, hence 17 pairs or features with dual coordinates $(\alpha_i, \beta_i)_{i=1,\ldots,17}$ were extracted. We solved (8.17) using the free and publicly available program Octave.[1] Following experiments detailed in Vert and Kanehisa (2003b) the regularization parameter δ of (8.16) was set to 0.01.

Figure 8.4 shows the first two profiles extracted, and table 8.1 contains a list representative of the genes with highest or lowest correlation with each profile, as well as the pathways they participate in in the KEGG database.

The first extracted profile is essentially a strong signal immediately following the beginning of the experiment. Several pathways positively correlated with this pattern are involved in energy metabolism (oxidative phosphorylation, tricarboxylic acid cycle, glycerolipid metabolism), while pathways negatively correlated are mainly involved in protein synthesis (aminoacyl-tRNA biosynthesis, RNA polymerase, pyrimidine metabolism). Hence this profile clearly detects the sudden change of environment, and the priority fueling the start of the cell cycle with fresh energetic molecules rather than synthesizing proteins. This result highlights the experimental bias in the data: while the goal of the experiment was to study the evolution of gene expression during the cell cycle, the strongest signal we detected is related to the need to synchronize the cells by addition of α factor.

Detection of experimental bias

The second extracted profile exhibits a strong sinusoidal shape corresponding to the progression in the cell cycle experiment, but the first one is more visible than

1. Available at `http://www.octave.org`.

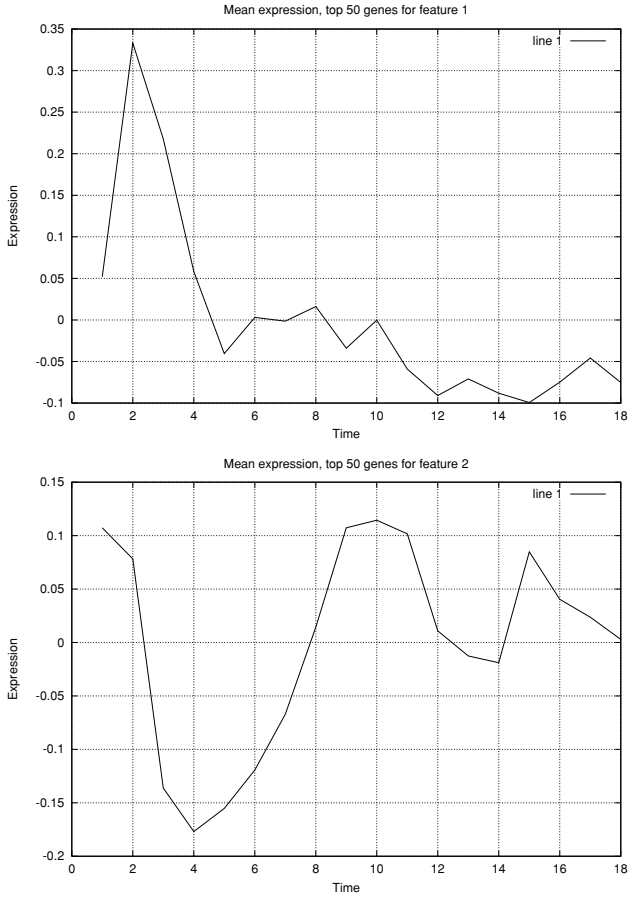


Figure 8.4 First 2 profiles extracted (α factor data set).

Table 8.1 Pathways and genes with highest and lowest scores on the first 2 features extracted.

Feature	Correlation	Main pathways and genes
1	+	Glycolysis/gluconeogenesis (*PGK1, GPM2, ALD4,6*), TCA cycle (*CIT2, MDH1,2, SDH1, LSC1*), pentose phosphate pathway (*RBK1, SOL4, ZWF1, YGR043C*), glycerolipid metabolism (*GPD1,2,3, ALD4,6*), glyoxylate and dicarboxylate metabolism (*MDH1,2, CIT2, ICL2*), sulfur metabolism (*MET2,14,16,17*).
1	-	Pyrimidine metabolism (*RPA12,34,49,190, RPB2,5, RPC53, DUT1, TRR1, POL5, URK1, MIP1, PUS1*), purine metabolism (*RPA12,34,49,190, RPB2,5, RPC53, CDC19, APT2, POL5, MIP1*), aminoacyl-tRNA biosynthesis (*ILS1, FRS2, MES1, YHR020W, GLN4, ALA1, CDC60*), starch and sucrose metabolism (*MPS1, HPR5, SWE1, HSL1, EXG1*).
2	+	Pyrimidine metabolism (*DEG1, PUS1,3,4, URA1,2, CPA1,2,FCY1*), folate biosynthesis (*ENA1,5, BRR2, HPR5, FOL1*), starch and sucrose metabolism (*ENA1,5, BRR2, HPR5, PGU1*), phenylalanine, tyrosine, and tryptophan biosynthesis (*TRP2,3,4, ARO2,7*), sterol biosynthesis (*ERG7,12, HGM1,2*).
2	-	Starch and sucrose metabolism (*CDC7, ENA1, GIN4, HXK2, HPR5, SWE1, UGP1, HSL1, FKS1, MEK1*), purine and pyrimidine metabolism (*POL12, ADK2, DUT1, RNR2, HYS2, YNK1, CDC21*), fructose and mannose metabolism (*MNN1, PMI40, SEC53, HXK2*), cell cycle (*CDC7, GIN4, SWE1, HSL1*).

Detection of the cell cycle

the second one because the synchronization in the yeast colony decreased while the experiment progressed. Several genes directly involved in DNA synthesis . Two cell cycles took place during the (*YNK1*, *RNR2*, *POL12*) can be recognized in the list of genes anticorrelated with the second feature (corresponding to maximum expression in the S phase). Some pathways such as the starch metabolism have genes which exhibit either strong correlation or strong anticorrelation with the second profile, corresponding to the various regimens in the normal cell cycle (e.g., periods of energy storage alternate with periods of energy consumption).

8.5 Extensions

Weighted graphs

In weighted graphs each edge has an associated weight $w_{ij} = w_{ji} > 0$ or $w_{ij} = 0$ if i and j are not connected. Weighted graphs can naturally be incorporated in the diffusion kernel framework by defining the Laplacian as

$$L_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \\ -\sum_{l=1}^{n} w_{il} & \text{if } i = j. \end{cases}$$

The rest of the development follows exactly as in the unweighted case. Unfortunately, no similarly straightforward solution suggests itself for directed graphs ($w_{ij} \neq w_{ji}$), since the symmetry of L is essential for the positive definiteness of k.

A different line of generalizing diffusion kernels, expounded in Smola and Kondor (2003), focuses on replacing $e^{\beta L}$ with a general expansion in terms of the eigenvalues and eigenvectors of L,

Other kernels

$$k = \sum_{i=1}^{n} v_i \, \frac{1}{r(\lambda_i)} \, v_i^\top$$

for some function $r : \mathbb{R} \mapsto \mathbb{R}^+$. Choosing $r(\lambda) = \exp(\sigma^2\lambda/2)$ gives the diffusion kernel, but other choices lead to interesting new kernels such as

$$\begin{aligned} r(\lambda) &= 1 + \sigma^2\lambda && \text{(regularized Laplacian kernel)} \\ r(\lambda) &= (aI - \lambda)^{-p} && (p\text{-step random walk kernel}) \\ r(\lambda) &= \cos(\pi\lambda/4) && \text{(inverse cosine kernel)} \ , \end{aligned}$$

although these generally cannot boast a similarly rich collection of interpretations.

Metric spaces

The most farreaching avenue of generalizing the ideas in this chapter involves applying the same framework to other mathematical objects, not just graphs. The ideas of diffusion and corresponding regularization are natural and well-studied concepts on a whole spectrum of different metric spaces. Indeed, the Laplacian and the spectral theory induced by it are two of the great unifying concepts of mathematics. For want of space here we can only sketch the wealth of possibilities this leads to.

For $\mathcal{X}=\mathbb{R}^N$ we have already encountered the Laplacian

$$\Delta = \frac{\partial^2}{\partial \mathbf{x}_{(1)}^2} + \frac{\partial^2}{\partial \mathbf{x}_{(2)}^2} + \ldots + \frac{\partial^2}{\partial \mathbf{x}_{(N)}^2}$$

and we have seen that the explicit solution of the diffusion equation is the Gaussian kernel

$$k(\mathbf{x}, \mathbf{x}') = \frac{1}{(4\pi\beta)^{N/2}} \, e^{-\| \mathbf{x}-\mathbf{x}' \|/(4\beta)}.$$

It should not come as a surprise that the eigenfunctions of K in this case are the harmonic eigenfunctions

$$\sin(2\pi k \cdot \mathbf{x}) \quad \text{and} \quad \cos(2\pi k \cdot \mathbf{x}), \quad k \in \mathbb{R}^N$$

with corresponding eigenvalues $e^{-\| k \|^2/(4\beta)}$.

Riemannian manifolds

The generalization of the Laplacian to curved spaces (Riemannian manifolds) is

$$\Delta = \frac{1}{\sqrt{\det g}} \sum_{ij} \partial_i \left(\sqrt{\det g} \, g^{ij} \partial_j \right),$$

where g is the metric tensor and ∂_i denotes differentiation with respect to the ith coordinate. Plugging this operator into the diffusion equation (8.5) we can solve for k. Locally the diffusion kernel will be very similar to the Gaussian kernel, but not so further away, due to the curvature and possibly nontrivial topology. Unfortunately, there are very few manifolds on which k can be computed in closed form and one has to resort to using asymptotic expansions.

In various domains, most notably image analysis, data often fall on curved manifolds embedded in a higher-dimensional space. Constraining the kernel, and hence the whole learning problem, to this manifold can improve the performance of learning algorithms. Unfortunately, in most cases the manifold is not known and may have very complicated geometry. Belkin and Niyogi (2002) and Belkin and Niyogi (2003) have proposed approximating the manifold by a mesh obtained by connecting data points, for example, according to a k-nearest neighbor rule. The diffusion kernel on the manifold can then be approximated by the diffusion kernel on this mesh, treated as a graph. What is especially attractive in this procedure is that it provides a natural use for unlabeled data. Unlabeled data points will not, of course, feature in a support vector expansion such as (8.13), but they can still play an important role in the learning process as vertices of the mesh, helping to form a good kernel. This can make learning possible in scenarios where only a very small fraction of data points are labeled.

The statistical manifold

Another context in which manifolds appear in learning theory is information geometry (Amari and Nagaoka, 2000). Consider a family of probability distributions $\{p_\theta(\mathbf{x})\}$ parameterized by $\theta \in \mathbb{R}^d$. The natural metric on this space of distributions, for a variety of reasons that we do not have space to go into here, is the Fisher metric

$$g_{ij} = \mathrm{E}_\theta \left[(\partial_i \ell_\theta)(\partial_j \ell_\theta)\right] = \int \left(\partial_i \log p(\mathbf{x}|\theta)\right) \left(\partial_j \log p(\mathbf{x}|\theta)\right) p(\mathbf{x}|\theta)\, d\mathbf{x},$$

where $\ell_\theta(\mathbf{x}) = \log p(\mathbf{x}|\theta)$. The Riemannian manifold this metric gives rise to is called the statistical manifold. The geometry of such manifolds can get rather involved and explicit calculations on them are almost never possible. However, a few celebrated special cases do exist. Lafferty and Lebanon (2003) have shown how the diffusion kernel can be computed in closed form on the statistical manifold of the spherical normal family and how it can be approximated for the multinomial family, in which case the geometry happens to be identical to that of a quadrant of a hypersphere. Assuming one of these two models generate the data, the kernel between two data points can be defined as the information diffusion kernel between the model fit to one and the other. Combined with SVMs, the authors successfully employ this technique to text classification with impressive results.

Acknowledgments

At about the same time that diffusion kernels were introduced by R.K. and John Lafferty (Kondor and Lafferty, 2002), Mikhail Belkin and Partha Niyogi were independently applying similar ideas from spectral graph theory to learning problems in a slightly different context (Belkin and Niyogi, 2002). The problem of partially labeled data is treated in Belkin and Niyogi (2003). Alex Smola has contributed by extending Laplacian-based kernels on graphs beyond diffusion kernels (Smola and Kondor, 2003) and working out the corresponding regularization theory. The application of diffusion kernels to bioinformatics was pioneered by J.-P.V. and Minoru Kanehisa (Vert and Kanehisa, 2003b) and information diffusion kernels were introduced by Lafferty and Lebanon (2003). In mathematics, the ideas of spectral geometry go back a little bit further, at least to Marquis Pierre Simon de Laplace (1749-1827).

The authors thank all of the above for many exchanges of ideas and for their role in bringing the concepts sketched in this chapter into the mainstream of machine learning. This work was partially supported by NSF grant CCR-0312690.

9 A Kernel for Protein Secondary Structure Prediction

Yann Guermeur
Alain Lifchitz
Régis Vert

Multiclass support vector machines (SVMs) have already proved efficient in protein secondary structure prediction as ensemble methods, to combine the outputs of sets of classifiers based on different principles. In this chapter, their implementation as basic prediction methods, processing the primary structure or the profile of multiple alignments, is investigated. A kernel devoted to the task is introduced, which incorporates high-level pieces of knowledge. Initial experimental results illustrate the potential of this approach.

9.1 Introduction

Protein structure prediction

Knowing the structure of a protein is prerequisite to gaining a thorough understanding of its function. The large-scale sequencing projects which have multiplied in recent years have produced an exploding number of protein sequences. Unfortunately, the number of known protein structures has not increased in the same proportion. Indeed, the experimental methods available to determine the three-dimensional structure, x-ray crystallography and nuclear magnetic resonance (NMR), are highly labor-intensive and do not ensure the production of the desired result (e.g., some proteins simply do not crystallize). As a consequence, predicting the tertiary structure of proteins *ab initio*, that is, starting from their sequences, has become one of the most challenging problems in structural biology. In the sixties, Anfinsen proposed his "thermodynamic hypothesis" (Anfinsen et al., 1963), which implies that there is sufficient information contained in the protein sequence to guarantee correct folding from any of a large number of unfolded states. In other words, the problem of interest can theoretically be solved. However, due to its practical difficulty, high-

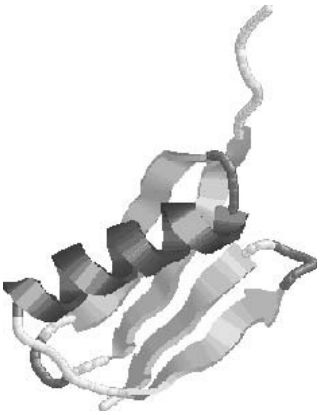


Figure 9.1 Schematic representation of the structural elements of protein G.

lighted, for instance, in Karplus and Petsko (1990), it is seldom tackled directly, but rather through a divide-and-conquer approach. In that context, a useful intermediate step consists in predicting first the secondary structure, which is a way to simplify the prediction problem by projecting the very complicated 3D structure onto one dimension, that is, onto a string of secondary structural assignments for each residue (amino acid). Protein secondary structure refers to regular, repeated patterns of folding of the protein backbone. The two most common folding patterns are the α *helix* and the β *strand*. Figure 9.1 is a schematic representation of the secondary structure of protein G (Derrick and Wigley, 1994), which was obtained with the RASMOL software (Sayle and Milner-White, 1995). This structure has two main parts: an α helix and a β sheet made up of four strands.

Secondary structure

From the point of view of pattern recognition, protein secondary structure prediction can be seen as a 3-class discrimination task, which consists in assigning to each residue of a sequence its conformational state, either α helix, β strand or aperiodic (coil). People started working on this problem as early as the late sixties. Since then, almost all the main families of machine learning methods have been assessed on it. Currently, the best prediction methods are connectionist architectures (see Rost and O'Donoghue, 1997; Baldi and Brunak, 2001; Rost, 2001 for surveys).

Although kernel methods have already found many applications in bioinformatics, as can be seen in other chapters of this book, to the best of our knowledge, they have only been applied to protein secondary structure prediction twice. Hua and Sun (2001a) implemented different combinations of biclass SVMs to perform the prediction from alignment profiles generated by BLAST. Guermeur (2002) and Guermeur et al. (2004) used different multiclass SVMs (M-SVMs) to combine several prediction methods, as well as the modules (BRNNs) of the current best

prediction method, SSpro (Baldi et al., 1999; Pollastri et al., 2002). In both cases, the experimental results appeared promising, which was all the more satisfactory that only standard kernels were used. In this chapter, we build on these initial works, introducing an M-SVM devoted to the prediction of the secondary structure from the primary structure, or profiles of multiple alignments. Its originality rests in the nature of its kernel, designed to exploit expert knowledge on the task. The parameterization of this kernel makes use of an original extension of the principle of kernel alignment (Cristianini et al., 2001, 2002b) to the multiclass case. Once more, experimental results appear promising, as they highlight the relevance of the specification performed. In short, our M-SVM, adequately dedicated, proves superior to a multilayer perceptron in all the contexts where the latter is incorporated in the prediction methods.

The organization of the chapter is as follows. Section 9.2 provides a short introduction to M-SVMs, as well as a description of the standard local approach implemented to predict the secondary structure, an approach based on the use of a sliding window. The main part of our contribution, the specification and parameterization of a kernel exploiting this input, is discussed in section 9.3. This section details our extension of the kernel alignment. Last, the resulting machine is assessed in section 9.4, where it is compared with a multilayer perceptron.

9.2 Multiclass SVMs for Protein Secondary Structure Prediction

This section introduces the M-SVMs, and the general principle of their implementation for protein secondary structure prediction.

9.2.1 M-SVMs

In the early days of the development of the SVM method, multiclass discrimination was implemented with biclass machines, through decomposition schemes. The first of them was the so-called *one-against-the-rest* or *one-per-class* approach (Schölkopf et al., 1995; Vapnik, 1995). Later on came the *pairwise-coupling* decomposition scheme (Mayoraz and Alpaydin, 1998; Weston and Watkins, 1998). The first multiclass SVM, more precisely the first SVM algorithm devoted to a multivariate affine architecture, was the k-class SVM proposed independently by Vapnik and Blanz (Vapnik, 1998), Weston and Watkins (1998), and Bredensteiner and Bennett (1999), among others. Alternative possibilities were then investigated in Crammer and Singer (2001), Lee et al. (2001), Hsu and Lin (2002), and Guermeur (2002). In (Guermeur et al., 2003), all these machines were endowed with a unifying theoretical framework.

Multiclass SVM

All the M-SVMs share the same architecture, which corresponds, in the feature space, to a multivariate affine model. This is expressed formally below. Let $\mathcal{X}$ be the

space of description (input space), $Q \geq 3$ the number of categories, k the Mercer kernel used, and Φ a map into the feature space $\mathcal{F}$ induced by k. Let F be the set of vector-valued functions $f = [f_j]$, $(1 \leq j \leq Q)$, from $\mathcal{X}$ into $\mathbb{R}^Q$, computed by the M-SVMs. We have then precisely :

$$\forall \mathbf{x} \in \mathcal{X}, \ \ \forall j \in \{1, \ldots, Q\}, \ \ f_j(\mathbf{x}) = \langle \mathbf{w}_j, \Phi(\mathbf{x}) \rangle + b_j$$

Thus, each category is associated with one hyperplane, and the discriminant function computed is obtained by application of the standard max rule: a pattern $\mathbf{x}$ is assigned to the category C_{j^*} satisfying $j^* = \text{argmax}_j \{f_j(\mathbf{x})\}$. In its primal formulation, training thus amounts to finding the optimal values of the couples $(\mathbf{w}_j, b_j)$, $(1 \leq j \leq Q)$, for a given choice of the kernel k and the *soft margin* constant C. In the biclass case, this choice is performed so as to maximize the (soft) margin. Strangely enough, in the first papers dealing with M-SVMs, the algorithms proposed were not related, at least explicitly, to the maximization of some notion of margin. This could be due to the fact that the standard pathways used to express the fat-shattering dimension of an SVM in terms of the constraints on the norm of $\mathbf{w}$, such as the use of a Rademacher's sequence (see, e.g., Bartlett and Shawe-Taylor, 1999; Cristianini and Shawe-Taylor, 2000; Gurvits, 2001), do not extend nicely to the multiclass case. Indeed, our efforts to endow the M-SVMs with the standard results derived in the framework of the theory of large margin classifiers (Guermeur et al., 2003) call for additional work. To the best of our knowledge, the only other study on the generalization capabilities of M-SVMs involving an extended notion of margin is reported in Crammer and Singer (2001). As usual, the corresponding quadratic programming (QP) problem is solved in its Wolfe dual form (Fletcher, 1989). Several algorithms can be applied to perform the optimization. Our software of the k-class SVM, used in the experiments described below, and available through the website of kernel machines,[1] implements a variant of the Frank-Wolfe algorithm (Frank and Wolfe, 1956) which includes a decomposition method (see also Elisseeff, 2000).

9.2.2 Selection of the Predictors

Window encoding

The standard way to perform protein secondary structure prediction with statistical discriminant methods consists in applying a local approach. Precisely, the predictors used to predict the conformational state of a given residue are the amino acids contained in a window of fixed size centered on this residue. To code the content of each position in the window, a vector of 22 components is used. Each of the 20 first components corresponds to a specific amino acid (there are 20 of them), whereas the 2 remaining ones are used to take into account unknown amino acids, usually designed by an "X" in the databases, as well as empty positions in the window. Empty positions occur when the window extends over the N-terminus

1. Available from `http://www.kernel-machines.org`.

or the C-terminus of the chain of interest. In short, the coding used to represent the window content is the standard orthonormal one, which induces no correlation between the symbols of the alphabet. What appears a priori as an advantage is an inconvenience here, as is pointed out below. Given a window size $|W| = 2n + 1$ (typically, n will range from 5 to 10), the number of predictors is thus equal to $(2n + 1).22$, only $2n + 1$ of them being equal to 1, the rest being equal to 0. We thus end up with large but very sparse vectors. Things are different when profiles of multiple alignments are used in place of the primary structure. Special attention must be paid to the inclusion of evolutionary information in this form, since it is known to improve significantly the performance of the prediction methods (see, e.g., Rost and Sander, 1993; Geourjon and Deléage, 1995). For the sake of simplicity, details on this alternative possibility are postponed to section 9.3.3.

9.3 Specification of the Kernel

Protein secondary structure prediction is a field that has emerged more than 30 years ago, and since then has been the subject of intensive researches. Nowadays, improvements over the state-of-the-art cannot be expected unless one uses a discriminant method specifically designed for the task. In the case of a kernel method, this means, of course, designing a new kernel. The one that is introduced in this section rests on very simple biological considerations.

9.3.1 Shortcomings of the Standard Kernels

Consider the vector $\mathbf{x}$ used to predict the conformational state of a given residue. Then, according to the choice of predictors described above, $\mathbf{x} = [x_{-n}, \ldots, x_i, \ldots, x_n]^T \in \{0,1\}^{(2n+1).22}$, where x_i is the canonical coding of the amino acid which occupies the ith position in the window. Consequently, the function computed by a Gaussian kernel applied on two window contents $\mathbf{x}$ and $\mathbf{x}'$ can be rewritten as

Standard kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right) = \exp\left(-\frac{(2n+1) - \sum_{i=-n}^{n} \delta_{x_i, x_i'}}{\sigma^2}\right), \qquad (9.1)$$

where δ is the Kronecker symbol. The right-hand side of (9.1) highlights the fact that the kernel only depends on the Hamming distance between the two strings. This is a poor summary of the information contained in the data. Indeed, two segments corresponding to 3D-superposable parts of homolog proteins, and thus sharing the same secondary structure, can differ significantly due to two evolution phenomena, insertion/deletion and substitution. The Hamming distance is very sensitive to the first one, whereas it does not take into account the nature of the substitutions, just their number. As a consequence, one cannot expect such a combination of kernel and coding (things would be similar with the other standard kernels) to

Hamming distance

give satisfactory results for the problem of interest. This simple observation is at the origin of the work on kernel design described in the following subsections, work which makes central use of an original extension to the multiclass case of the notion of kernel alignment.

9.3.2 Multiclass kernel alignment

Framework Kernel alignment was introduced by Cristianini et al. (2001), as a means to assess the degree of fitness of a kernel for a given learning task, and adapt in consequence the Gram matrix to increase this fitness. It is thus basically a method conceived to perform transduction, since the resulting kernel is not available in analytical form. However, we use it here for another purpose, namely to estimate some kernel parameters. The reason for this choice is the following. Consider a family of kernels where each element is characterized by the value of a formal parameter θ belonging to a set Θ. This family, $(k_\theta)_{\theta\in\Theta}$, is supposed to be built upon some knowledge of the task of interest. In order to select a kernel function k_{θ^*} that will give good performance, a natural and practical approach consists in endowing the whole family with a measure of adequacy, and then optimizing this measure with respect to the parameter. A typical example of measure of adequacy, often used in practice, is the score given by a cross-validation procedure. However, choosing it raises two difficulties. First, this can only be done when the set Θ is finite (or was discretized). Second, it is computation-consuming, since the cross-validation procedure is to be run for each value θ_i in Θ. The solution advocated in Chapelle et al. (2002) has the same drawback, since it requires training a SVM at each step of a gradient descent. *Kernel target alignment* is a score which does not exhibit these shortcomings. In what follows, we first define it, and then describe its use to tune a kernel with respect to some parameter.

Kernel alignment Let k and k' be two measurable kernel functions defined on $\mathcal{X}\times\mathcal{X}$, where the space $\mathcal{X}$ is endowed with a probability measure P. The alignment between k and k' is defined as follows:

Kernel alignment

$$A(k,k') = \frac{\langle k,k'\rangle_2}{\|k\|_2\|k'\|_2} = \frac{\int k(\mathbf{x},\mathbf{x}')k'(\mathbf{x},\mathbf{x}')dP(\mathbf{x})dP(\mathbf{x}')}{\sqrt{\int k(\mathbf{x},\mathbf{x}')^2 dP(\mathbf{x})dP(\mathbf{x}')}\sqrt{\int k'(\mathbf{x},\mathbf{x}')^2 dP(\mathbf{x})dP(\mathbf{x}')}}. \tag{9.2}$$

Empirical kernel alignment Let k and k' be two kernel functions defined on $\mathcal{X}\times\mathcal{X}$ and consider a data set $X = (\mathbf{x}_1,\ldots,\mathbf{x}_m)\in\mathcal{X}^m$. The empirical alignment of k with k' with respect to X is the quantity

$$\hat{A}_X(K,K') = \frac{\langle K,K'\rangle_F}{\|K\|_F\|K'\|_F}, \tag{9.3}$$

where K and K' respectively denote the kernel Gram matrices associated with k and k', computed on the sample X, $\langle .,.\rangle_F$ denotes the Frobenius inner product

between matrices, so that $\langle K, K' \rangle_F = \sum_{i=1}^m \sum_{j=1}^m k(\mathbf{x}_i, \mathbf{x}_j) k'(\mathbf{x}_i, \mathbf{x}_j)$, and $\|.\|_F$ is the corresponding norm.

The alignment between two kernels k and k' should be thought of as a measure of their similarity. Roughly speaking, if k' is a well-suited kernel to the problem at hand and k is well aligned with k', then k should also be a good kernel for the same problem. In practice, as the alignment is not computable (since the underlying distribution P is unknown), it is estimated empirically, with (9.3). Some concentration properties of $\hat{A}_X(K, K')$ around its expected value $A(k, k')$ were studied by Cristianini et al. (2001).

Tuning Parameter θ Using Kernel Target Alignment Now, the strategy to tune parameters based on this measure can be summarized as follows:

1. Select a theoretically ideal kernel k_t, hereinafter called the *target kernel*, ideal in the sense that it leads to perfect classification. Practically, the Gram matrix of k_t should be computable.

2. Given a training set of labeled examples $Z = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$, choose θ^* satisfying

$$\theta^* = \operatorname*{argmax}_{\theta \in \Theta} \hat{A}_Z(k_\theta, k_t).$$

In doing so, the conjecture is that the kernel k_{θ^*} will behave well, provided the family $(k_\theta)_{\theta \in \Theta}$ is relevant to the problem at hand. Cristianini et al. (2001) only considered the case of dichotomies. Their ideal kernel is the obvious one, namely $k_t(\mathbf{x}, \mathbf{x}') = yy'$. Our extension to the multiclass case, based on geometric considerations, developed in Vert (2002c), is the following:

$$k_t(\mathbf{x}, \mathbf{x}') = \begin{cases} 1 & if\ y = y' \\ -1/(Q-1) & otherwise \end{cases}$$

This target kernel corresponds to a mapping Φ_t associating each description $\mathbf{x}$ to one of the Q vertices of a $(Q-1)$-dimensional centered simplex, according to the category to which it belongs (see figure 9.2). This clusterization in the feature space is obviously the one that makes the subsequent (multi-) linear separation performed by the M-SVM easiest. Note that under some regularity assumptions on k_θ, $\hat{A}_Z(k_\theta, k_t)$ is differentiable with respect to θ, and can thus be optimized using classic techniques, such as gradient descents.

9.3.3 Incorporating Biological Knowledge in a Convolution Kernel

In what follows, we adopt the terminology of Williamson et al. (2001), where a convolution kernel is a kernel satisfying $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}', 0)$. Our goal is to take into account in such kernels two of the factors which have proved important to predict the secondary structure: the nature of the substitutions between two

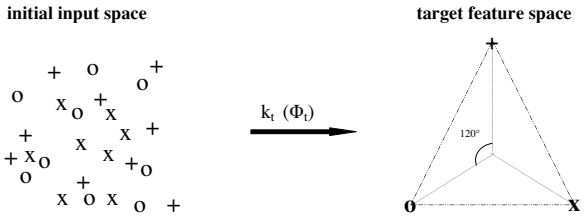


Figure 9.2 The 3-class problem in 2D : the target kernel performs an optimal clustering of the data in the feature space.

segments, and the relative influence of the amino acids involved as a function of their position in the window.

Dot Products Between Amino Acids In subsection 9.2.2, we pointed out the fact that the standard processing of protein sequences for secondary structure prediction involves a canonical orthonormal coding of the amino acids. However, it is common knowledge that this coding is unsatisfactory. Indeed, the biologists have derived many similarity matrices for the amino acids which all differ significantly from the identity. It is the case of the PAM (percent accepted mutations) (Dayhoff et al., 1978) and BLOSUM (blocks substitution matrix) (Henikoff and Henikoff, 1992) matrices, sometimes called substitution matrices, which are especially well suited in their log-odds form. The problem raised by their use in a kernel springs from the fact that they are not symmetric positive definite, and thus are not associated with an underlying dot product. To overcome this difficulty, one can think of several off-the-shelf solutions. Since the matrices are symmetric, one simple way to approximate them with a Gram matrix consists in diagonalizing them and replacing all the negative eigenvalues with 0. Another possibility consists in looking for their projection on the space of symmetric positive definite matrices, the operator being associated with a matrix norm, for instance, the Frobenius one. Although the projection operator will usually not be available in analytical form (the problem to be solved is nonconvex), satisfactory estimates can result from a simple gradient descent. This descent is performed with respect to the components of the vectors representing the amino acids.

Substitution matrix

This change in the coding of the amino acids extends nicely to the case where multiple alignments are used. In that case, the profile presented as input of a connectionist classifier (see, e.g., Rost and Sander, 1993; Jones, 1999; Pollastri et al., 2002) is simply obtained by computing, for each position in the window, a weighted average of the vectors coding the amino acids present in the corresponding position of the alignment. The weight associated with a particular amino acid is its frequency of appearance in this position. In practice, let a_j, $(1 \leq j \leq 22)$, be the coding of the jth amino acid (or an unknown residue, or the empty position), and θ_{ij} its frequency of appearance in the position of the alignment corresponding to the ith position of the sliding window. Then the window can be represented by $\tilde{\mathbf{x}} = [\tilde{x}_{-n}, \ldots, \tilde{x}_i, \ldots, \tilde{x}_n]^T$, with $\tilde{x}_i = \sum_{j=1}^{22} \theta_{ij} a_j$. Thus, in the computation of the kernel, the dot product $\langle x_i, x_i' \rangle$ is simply replaced with

$$\langle \tilde{x}_i, \tilde{x}_i' \rangle = \langle \sum_{j=1}^{22} \theta_{ij} a_j, \sum_{k=1}^{22} \theta'_{ik} a_k \rangle = \sum_{j=1}^{22} \sum_{k=1}^{22} \theta_{ij} \theta'_{ik} \langle a_j, a_k \rangle. \tag{9.4}$$

Influence of the Position in the Window As stated in subsection 9.2.2, the use of the sliding window is standard in protein secondary structure prediction. Many studies have dealt with the choice of its size, or the exploitation of its content. Good illustrations are given in Qian and Sejnowski (1988), Zhang et al. (1992), and Rost and Sander (1993). In short, a too small window will not include enough information on the local conformation, whereas a too large window will incorporate data that risk behavior like noise. A way to overcome this difficulty consists in choosing a priori a large value for the size of the window, and associating each position with a weight (irrespective of the nature of the amino acid), so as to modulate its influence on the subsequent computations. This has already been performed with success by different teams (Gascuel and Golmard, 1988; Guermeur, 1997). An interesting point is that these studies, although they involved very different approaches, produced similar distributions of the weights as a function of the position. This suggests that they were capable of highlighting some intrinsic property of the problem of interest. We thus decided to incorporate such a weighting in our kernel, with the values of the weights being derived through the multiclass kernel alignment.

Position-dependent weight

Both parameterizations, the change in the "dot products between amino acids" and the weighting of the positions in the window, can be applied to any kind of convolution kernel. For the sake of simplicity, their incorporation is illustrated below in the case of a Gaussian kernel processing multiple alignments:

$$k_{\theta,D}(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}') = \exp\left(- \frac{\sum_{i=-n}^{n} \theta_i^2 (\|\tilde{x}_i\|^2 + \|\tilde{x}_i'\|^2 - 2\langle \tilde{x}_i, \tilde{x}_i' \rangle)}{2\sigma^2} \right), \tag{9.5}$$

where θ is the vector of weights and D the matrix of dot products.

9.4 Experimental Results

We have already pointed out the difficulty in developing a state-of-the-art secondary structure prediction method. Nowadays, all the most accurate statistical methods are based on huge hierarchical and modular architectures. The best illustration of this phenomenon is given by the methods PSIPRED (Jones, 1999), SSpro2 (Pollastri et al., 2002), as well as those described by Riis and Krogh (1996) and Petersen et al. (2000). In these architectures, which can incorporate up to several hundred components, the contribution of a single module (ordinarily a neural network) can hardly be assessed. From a general point of view, our goal hereafter is not to obtain recognition rates comparable with those of the above methods, but rather to highlight the fact that in each of the various contexts where an MLP can be used to perform protein secondary structure prediction (as part of a hierarchical classifier, as an ensemble method, etc.), significant benefits result from replacing it with our M-SVM. In that respect, this study can be seen as the natural continuation of those reported in Guermeur (2002) and Guermeur et al. (2004).

9.4.1 Experimental Protocol

To assess our classifier, we used the set of 1096 protein sequences introduced in Guermeur et al. (2004) under the reference P1096. This set was designed so as to meet the toughest requirements in terms of percentage of identity (see Sander and Schneider, 1991 for details). Secondary structure assignment was performed with the DSSP program (Kabsch and Sander, 1983). This assignment is essentially based on hydrogen-bonding patterns. The reduction from 8 to 3 conformational states was derived according to the CASP method, given by: H+G $\rightarrow$ H (α helix), E+B $\rightarrow$ E (β strand), and all the other states in C (aperiodic or coil). This assignment is known to be somewhat harder to predict than the ones used in the literature (see, e.g., Cuff and Barton, 1999). The PSI-BLAST alignments were compiled according to the protocol described in Pollastri et al. (2002).

As stated in the introduction, the prediction of the secondary structure is seldom a goal in its own right. It is primarily a step toward the prediction of the tertiary structure. As a consequence, the main concern of the biologist is the recognition of all the structural elements in their order of appearance in the sequence. A small shift in the relative locations of the true and predicted structures can be tolerated, but the prediction must remain biologically plausible (no helix can be shorter than 4 residues, two periodic structures cannot be consecutive, etc.). As a consequence, the sole per residue recognition rate, hereinafter denoted by Q_3, is not sufficient to characterize the quality of the prediction. To overcome this difficulty, many alternative measures of quality have been proposed. The interested reader will find in Baldi et al. (2000) a review of the subject. In what follows, we use the three most common quality measures: the Q_3, the Pearson's/Matthews' correlation coefficients

C (Matthews, 1975), and the segment overlap measure Sov (Rost et al., 1994; Zemla et al., 1999), which give complementary indications. Whereas each of the Matthews' coefficients characterizes the quality of the prediction for one particular conformational state (α/β/coil), which makes it possible, for instance, to emphasize a poor identification of the sheets, the values of the Sov coefficients give an idea of the prediction accuracy at the segment level, meeting by way of consequence one of the central requirements listed above.

9.4.2 Estimation of the Parameters

The matrix of dot products between amino acids was derived from the similarity matrix introduced in Levin et al. (1986). This choice resulted from the fact that this matrix had been specifically devised to perform secondary structure prediction based on the similarity of small peptides, that is, on local sequence homology (see also Levin and Garnier, 1988; Geourjon and Deléage, 1995). In this context, it has reportedly proved superior to the Dayhoff substitution matrix. Among the two possibilities considered in subsection 9.3.3 to generate the Gram matrix, we chose the one based on diagonalization. However, this choice was primarily made for the sake of reproducibility, since a simple gradient descent gave very similar results (Didiot, 2003). With this set of dot products at hand, the vector of weights θ could be obtained thanks to the implementation of the multiclass target alignment principle, through a stochastic gradient descent procedure (see Vert (2002c) for details). The training set was the set of 1180 sequences used to train SSpro1 and SSpro2. This set, described in Baldi et al. (1999) and Pollastri et al. (2002), is referred to below as P1180. This choice could be made since no sequence in this base is the homolog of a sequence of the P1096 base (see also Guermeur et al., 2003). Figure 9.3 illustrates the resulting values of the coefficients θ_i. This curve is

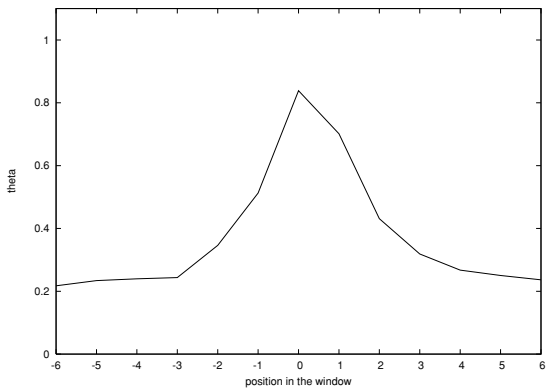


Figure 9.3 Vector θ of (9.5) maximizing the kernel target alignment.

Table 9.1 Relative prediction accuracy of an MLP and the M-SVM on the P1096 data set.

	sequences		alignments	
	MLP	M-SVM	MLP	M-SVM
Q_3	61.6	62.0	72.0	72.3
C_α	0.46	0.47	0.63	0.64
C_β	0.33	0.35	0.53	0.54
C_c	0.38	0.38	0.53	0.54
Sov	53.9	54.2	65.1	65.3
Sov_α	57.8	57.9	66.5	66.7
Sov_β	44.7	46.1	61.5	62.3
Sov_c	57.3	57.3	66.7	66.8

very similar to those mentioned in subsection 9.3.3. One of their common features is a significant asymmetry in favor of the right-hand side context. This intriguing phenomenon, the observation of which utterly rests on a statistical basis, and by no means on the incorporation of a priori knowledge of the task, has found no biological justification so far.

9.4.3 Prediction from the Primary Structure and Multiple Alignments

This subsection describes the implementation and results of two experiments inspired by the pioneering works reported by Qian and Sejnowski (1988), and Rost and Sander (1993). In the former, the M-SVM is compared with an MLP in the case where the input (vector $\mathbf{x}$) is simply given by the content of a window of size 13 sliding on the sequence. In the latter, single sequences are replaced with PSI-BLAST multiple alignments. To perform these experiments, we used the same procedure twice, a standard fivefold cross-validation (the P1096 base was divided into four sets of 219 sequences, and one set of 220 sequences). In both cases, the MLP had one hidden layer of eight units with sigmoid activation functions, and softmax output units. The parameterization of the M-SVM also remained unchanged, with the soft margin constant C being equal to 10.0, and the width of the Gaussian kernel being set to $\sigma^2 = 10.0$. Table 9.1 summarizes the results obtained.

In both configurations, the gain in recognition rate resulting from using the M-SVM in place of the MLP is statistically significant, with confidence exceeding 0.95. The size of the base (255551) compensates for the small value of the increase. However, what is more promising, all measures of accuracy benefit from the change. This is particularly noticeable for the β strands, usually the hardest conformational segments to predict. Hua and Sun (2001a) noticed that the superiority of combinations of (biclass) SVMs over MLPs was highlighted with the Sov coefficients. Although we were unable to duplicate their experiments and get similar results (we observed far lower Sov coefficients than they did), the same conclusion can be

inferred here.

The number of dual variables of an M-SVM is equal to the number of categories minus one times the size of the training set $[(Q-1)m]$. An idea of the complexity of the discriminant function it computes is given by the number of training examples for which at least one of the dual variables is different both from 0 and from C (examples which should lie on one of the margins). In all our experiments (ten trainings of the M-SVM), the ratio of such points ranged between 25% and 30%.

9.5 Discussion

The results of the experiments reported above support the thesis of the superiority of our M-SVM over the standard MLP for the task at hand. However, the real touchstone to judge its usefulness is obviously the incorporation in a state-of-the-art prediction method, as we already did with SSpro2. To pave the way for this new step, one can think of implementing several straightforward improvements. A simple one is the choice of a better matrix of dot products between amino acids (matrix D). Instead of choosing a priori or by any model selection method one specific similarity matrix, one can benefit from the fact that a convex combination of symmetric positive definite matrices is still a symmetric positive definite matrix. This makes it possible to select a whole set of similarity matrices, estimate them with Gram matrices, and compute the optimal combination thanks to the procedure described in subsection 9.3.2. Another useful development is the postprocessing of the conformational scores generated, to produce class posterior probability estimates. These estimates could then be used to compute the observation probability density functions of a hidden Markov model (HMM), as was done, for instance, in Guermeur (2002). Both possibilities are the subject of an ongoing work.

9.6 Conclusion and Future Work

We have described a first attempt to implement M-SVMs to perform protein secondary structure prediction from the primary structure, or profiles of multiple alignments. This study focused on the design of a kernel incorporating high-level knowledge of the task. The process of evolution, which makes two homolog proteins differ in their sequences, while keeping similar folds, is based in two phenomena: substitution and insertion/deletion. If one can think of simple solutions to take the first phenomenon into account with a kernel, as the one we used, the second one raises more difficulties. Indeed, if substituting a multiple alignment for a single sequence provides useful additional evolutionary information, this procedure alone does not solve the problem of the comparison of two window contents by means of a dot product. What if they only differ by an insertion? A priori, a natural way

to overcome this difficulty would consist in making use of the results established by Haussler (1999) or Watkins (2000), regarding dynamic alignment kernels and select, for instance, an adequately designed pair HMM (Durbin et al., 1998) to compute the kernel function. However, this solution currently remains infeasible for such large problems as those we are interested in, due to its prohibitive CPU time requirements coming from too high computational complexity. Cheaper alternatives are thus badly needed. Well suited spectrum (Leslie and Kuang, 2003) and rational (Cortes et al., 2003) string kernels, which can extract the similarity of pairs of strings of unequal length, are potentially good candidates with their efficient linear computational complexity. Their incorporation in our machine is currently under investigation.

Acknowledgments

We gratefully acknowledge the support of the CNRS funded "Action Spécifique : Apprentissage et Bioinformatique." We thank Dr. G. Pollastri for providing us with the data sets used in the experiments, and Prof. A. Ourjoumtsev and D. Eveillard for helping us with the production of figure 9.1. Thanks are also due to Dr. A. Zemla and Dr. C. Venclovas for the availability of the code of the Sov measure, and to Dr. F.D. Maire for carefully reading this manuscript.

III DATA FUSION WITH KERNEL METHODS

10 Heterogeneous Data Comparison and Gene Selection with Kernel Canonical Correlation Analysis

Yoshihiro Yamanishi
Jean-Philippe Vert
Minoru Kanehisa

The integration and comparison of heterogeneous data such as biochemical pathways, genomes, gene functions, and gene expression data is a major issue in postgenomics. While integration strategies often rely on heuristic approaches specifically adapted to the nature of the data to be integrated — such as sequences, graphs, and vectors — we present in this chapter a systematic approach to the integration and comparison of virtually any types of data, as long as relevant kernel functions can be defined on the data to be compared. Tools to measure the correlation between different heterogeneous data sets and to extract sets of genes which share similarities with respect to multiple biological attributes are proposed. The originality of this approach is the extension of the concept of correlation for nonvectorial data, which is made possible by the use of generalized kernel canonical correlation analysis, and its application to the extraction of groups of genes responsible for the detected correlations.

As an application, this approach is tested on its ability to recognize operons in the *Escherichia coli* genome, from the comparison of three data sets corresponding to *functional* relationships among genes in metabolic pathways, *positional* relationships along the chromosome, and *coexpression* relationships as observed by gene expression data.

10.1 Introduction

Developments in high-throughput technologies have filled biological databases with many sorts of genomic data. Examples include genome sequences, signaling and

metabolic pathways (Kanehisa et al., 2002), gene expression data (Eisen et al., 1998), protein-protein interaction data (Ito et al., 2001), phylogenetic profiles (Pellegrini et al., 1999), and several more. Investigating the relationships among these data is an important step toward a better understanding of the functions of the genes and the machinery of the cell. In particular, it is often the case that different data provide different and complementary information about the same underlying objects or processes. To fix the ideas, we focus in this chapter on gene analysis, but the principled approach we follow can easily be applied to the analysis of other biological objects or processes, such as the evolution of a disease, as soon as several different measurements about the objects or processes of interest are available.

Heterogeneous data comparison

Our approach is motivated by the classic idea that comparing different data about the same objects is a way to detect hidden or underlying relationships or phenomena. Let us suppose, for example, that an unusually strong correlation is detected between the presence of a motif in the promoter region of some genes, on the one hand, and the gene expression levels under particular conditions, on the other hand. This correlation might stem from a biological phenomenon linking the sequence and the function of the genes, such as the recognition of the motif by a transcription factor. More generally, comparing different data sets involving the same genes, such as their sequences, expression, promoter regions, or the structure of the encoded proteins, is a way to recognize biological phenomena. Moreover, if a correlation is detected among several data sets, genes mainly responsible for the observed correlation can be detected. One can expect these genes to play a special role in or be affected by the underlying biological phenomenon.

A well-known statistical method to investigate the correlation between different real-valued attributes is canonical correlation analysis (CCA) (Hotelling, 1936). However, classic CCA cannot be applied to nonvectorial genomic data, such as pathways, protein-protein interactions, or gene positions in a chromosome. In this chapter we overcome this issue by using a generalization of CCA, known as kernel CCA (KCCA) proposed by Akaho (2001) and Bach and Jordan (2002), which provides a way to perform a generalized form of CCA between any two types of data as long as kernel functions can be defined on these data. KCCA finds directions simultaneously in the two feature spaces defined by the kernel functions with maximum correlation.

Variants of kernel CCA

As a first contribution we derive two variants of KCCA in order to perform CCA on more than two data sets. The first one, which we call multiple KCCA, is a natural generalization of KCCA to more than two kernel functions. Already suggested by Bach and Jordan (2002), it consists in searching for directions simultaneously in all feature spaces by maximizing the sum of all pairwise correlations between data sets. The second one, which we call integrated KCCA, is a normal KCCA carried out between two kernels which are themselves sums of primary kernels. Integrated KCCA can be useful to extract correlations between two sets of data sets, represented by two sets of kernel functions.

Gene selection

As a second contribution, we propose a method to select genes of interest from

the results of CCA. The method consists in ranking the genes in terms of the absolute value of their projection on a canonical direction, typically the first one. Large absolute values correspond to the genes mainly responsible for the detected correlation, hence selecting these genes is likely to provide groups of genes related to the biological phenomenon behind the correlation.

As an application we consider the problem of detecting operons in prokaryotic genomes. Operons are groups of adjacent genes on the genome which are transcribed together on a single messenger RNA (mRNA) molecule. Genes in an operon often code proteins involved in the same biochemical function, such as enzymes catalyzing successive chemical reactions in a pathway. As a result, the presence of operons in prokaryotes is responsible for a form of correlation among several data sets, because genes which form operons tend to be close to each other along chromosomes, to have similar expression profiles, and to catalyze successive reactions in a pathway. Conversely, one can start from three data sets containing the localization of the genes on the genome, their expression profiles, and the chemical reactions they catalyze in known pathways, and look for correlations among these data sets, in order to finally recover groups of genes, which may form operons. We provide experimental results on the unsupervised detection of operons in the *E.coli* genome by detecting correlations among the KEGG/pathways database of metabolic and signaling pathways, the positions of the genes on the genome, and microarray expression data.

The integration of heterogeneous data has been investigated with a variety of approaches so far. Motivated by graph-theoretical arguments, clusters of genes have been extracted from several biological networks using multiple graph comparison by Ogata et al. (2000) and Nakaya et al. (2001). Using classic clustering algorithms with a distance combining information from expression data and biochemical networks, Hanisch et al. (2002) were able to extract coclusters of genes. An approach using direct kernel operations was proposed by Pavlidis et al. (2001b) to improve the performance of gene function prediction algorithms from expression data and phylogenetic profiles. The use of KCCA was pioneered by Vert and Kanehisa (2003b) and Vert and Kanehisa (2002) in the context of gene function prediction from gene expression data using biochemical networks as side information, and further investigated by Yamanishi et al. (2003) and Vert and Kanehisa (2003a) as a data mining tool to extract information from heterogeneous data.

10.2 Methods

In this section we present the methodology of our work. We review canonical correlation analysis, its generalization as a kernel algorithm, and propose two variants to handle more than two data sets. We then present a method to select genes from the result of CCA analysis, and finally recall the definition of the diffusion kernel used in the experiment.

10.2.1 Classic CCA

Classic CCA

Canonical correlation analysis was introduced by Hotelling (1936) as a way to measure linear relationships between random multivariate vectors $\mathbf{x}_1$ and $\mathbf{x}_2$, of respective dimension N_1 and N_2. It finds two linear transforms, one for each variable, such that one component within each transformed variable is maximally correlated with a single component in the other. More precisely, the first *canonical variates* are defined as the projections of $\mathbf{x}_1$ and $\mathbf{x}_2$ onto unit norm vectors $\alpha_1 \in \mathbb{R}^{N_1}$ and $\alpha_2 \in \mathbb{R}^{N_2}$ defined by

$$(\alpha_1, \alpha_2) := \underset{||a_1||=||a_2||=1}{\arg\max} \left| \text{ corr } \left(a_1^\top \mathbf{x}_1, a_2^\top \mathbf{x}_2\right)\right|, \tag{10.1}$$

where $a^\top$ denotes the transpose of a. The first *canonical correlation* is defined as the maximum value attained in (10.1). Higher-order canonical variates and correlations are defined as in (10.1) under the additional restriction that the kth canonical variate, with $1 \leq k \leq \min(N_1, N_2)$, should be uncorrelated with all canonical variates of lower order. The problem (10.1) has a fairly simple solution (Johnson and Wichern, 1998), where α_1 and α_2 are found by eigenvector decomposition. CCA is a popular tool in exploratory data analysis to investigate the relationship between two kinds of attributes, and has found many applications in economics and medical studies, for example.

10.2.2 Kernel CCA

Kernel CCA is a generalization of CCA using the kernel trick. Proposed independently by Akaho (2001) and Bach and Jordan (2002), it consists in performing a regularized form of CCA in the feature spaces implicitly defined by two different kernels on the same objects. As an example, if objects are genes, kernel CCA can be used to investigate the relationships between gene sequences and gene expression by performing classic CCA between the genes in the features spaces defined respectively by a string kernel and a kernel for expression profiles.

In order to transform CCA into a kernel algorithm, at least two important issues must be addressed:

- Technically, the algorithm to solve CCA must be expressed in a form that only involves the data through their inner products, in order to use the kernel trick (see chapter 2, subsection 2.3.1) and be able to replace each such inner product by the evaluation of a kernel function.
- Theoretically, classic CCA is not adapted to large-dimensional variables. In particular, when the dimension of each space exceeds the number of points available, perfect canonical correlation can always be found between any sets of variables. While this issue is well-known by practitioners of classic CCA, it becomes problematic with kernels that correspond to high-dimensional feature spaces, such as the

Gaussian kernel. To address this issue, some form of regularization must be added to the CCA definition.

Kernel CCA

Both issues have been addressed in the KCCA algorithm which we now present. Further details and references can be found in Akaho (2001) and Bach and Jordan (2002). The goal is to detect correlations between two data sets $\mathbf{x}_1 = \left(\mathbf{x}_1^{(1)}, \cdots, \mathbf{x}_1^{(n)}\right)$ and $\mathbf{x}_2 = \left(\mathbf{x}_2^{(1)}, \cdots, \mathbf{x}_2^{(n)}\right)$, where n is the number of objects, and each data set $\mathbf{x}_1^{(i)}$ / $\mathbf{x}_2^{(i)}$ belongs to some set $\mathcal{X}_1$ / $\mathcal{X}_2$, for $i = 1, \cdots, n$. In the example treated in this chapter, the objects correspond to genes, and each data set corresponds to one representation of the genes. For example, if $\mathcal{X}_1$ is the set of finite-length nucleotide sequences, and $\mathcal{X}_2$ is a vector space of gene expression profiles, then $\mathbf{x}_1^{(i)}$ could be the sequence of the i-th gene studied and $\mathbf{x}_2^{(i)}$ its expression profile.

In order to detect correlations between the two data sets, the objects $\mathbf{x}_1^{(i)}$ /$\mathbf{x}_2^{(i)}$ are mapped to a Hilbert space H_1/ H_2 by a mapping $\phi_1 : \mathcal{X}_1 \to H_1$/ $\phi_2 : \mathcal{X}_2 \to H_2$. Classic CCA can then be performed between the images $\phi_1(\mathbf{x}_1)$ and $\phi_2(\mathbf{x}_2)$ as follows. For any two directions $f_1 \in H_1$ and $f_2 \in H_2$, we can define the projections $u_1 = \left(u_1^{(1)}, \cdots, u_1^{(n)}\right)^\top \in \mathbb{R}^n$ and $u_2 = \left(u_2^{(1)}, \cdots, u_2^{(n)}\right)^\top \in \mathbb{R}^n$ of $\mathbf{x}_1$ and $\mathbf{x}_2$ onto f_1 and f_2 by

$$u_1^{(i)} := \langle f_1, \phi_1(\mathbf{x}_1^{(i)})\rangle, \quad u_2^{(i)} := \langle f_2, \phi_2(\mathbf{x}_2^{(i)})\rangle, \tag{10.2}$$

for $i = 1, \cdots, n$, where $\langle ., . \rangle$ denotes the dot products in the Hilbert spaces H_1 and H_2. The sample mean, variance, and covariance of u_1 and u_2 are respectively defined by

$$\begin{aligned} \hat{mean}\,(u_j) &:= \frac{1}{n}\sum_{i=1}^{n} u_j^{(i)}, \\ \hat{var}(u_j) &:= \frac{1}{n}\sum_{i=1}^{n}\left(u_j^{(i)} - \hat{mean}\,(u_j)\right)^2, \\ \hat{cov}(u_1, u_2) &:= \frac{1}{n}\sum_{i=1}^{n}\left(u_1^{(i)} - \hat{mean}\,(u_1)\right)\left(u_2^{(i)} - \hat{mean}\,(u_2)\right), \end{aligned} \tag{10.3}$$

for $j = 1, 2$. The goal of CCA is to find $f_1 \in H_1$ and $f_2 \in H_2$ that maximize the empirical correlation between u_1 and u_2, defined by:

$$\hat{corr}\,(u_1, u_2) := \frac{\hat{cov}(u_1, u_2)}{(\hat{var}(u_1)\hat{var}(u_2))^{\frac{1}{2}}}. \tag{10.4}$$

The solution to this problem, however, is not unique when the dimension of H_1 or H_2 is larger than the number of samples n: indeed, adding to f_1 or f_2 any vector orthogonal to the linear span of the respective points does not change the projections u_1 and u_2. Moreover, the importance of regularization for CCA in high dimension is a well-known fact discussed, for instance, by Hastie et al. (1995) and Leurgans et al. (1993).

Regularization of CCA

A classic way to regularize CCA is to penalize the Hilbert norm of f_1 and f_2 through the maximization of the following functional instead of (10.4):

$$\gamma(f_1, f_2) := \frac{\hat{cov}(u_1, u_2)}{(\hat{var}(u_1) + \lambda_1 ||f_1||^2)^{\frac{1}{2}} (\hat{var}(u_2) + \lambda_2 ||f_2||^2)^{\frac{1}{2}}}, \tag{10.5}$$

where λ_1 and λ_2 are regularization parameters. When $\lambda_1 = \lambda_2 = 0$, $\gamma(f_1, f_2)$ reduces to the sample correlation (10.4), but when $\lambda_1 > 0$ and $\lambda_2 > 0$, the pair (f_1, f_2) that maximizes (10.5) finds a tradeoff between maximizing the empirical correlation (10.4) and having small norms $||f_j||/\hat{var}(u_j)$ (for $j = 1, 2$).

By homogeneity, maximizing (10.5) is equivalent to maximizing $\hat{cov}(u_1, u_2)$ under the constraints

$$\hat{var}(u_1) + \lambda_1 ||f_1||^2 \leq 1, \quad \hat{var}(u_2) + \lambda_2 ||f_2||^2 \leq 1.$$

Dual formulation

The solution to this problem is obtained by solving the Lagrangian:

$$\begin{aligned} L(f_1, f_2, \rho_1, \rho_2) &= \hat{cov}(u_1, u_2) \\ &+ \frac{\rho_1}{2} \left(1 - \hat{var}(u_1) - \lambda_1 ||f_1||^2\right) + \frac{\rho_2}{2} \left(1 - \hat{var}(u_2) - \lambda_2 ||f_2||^2\right), \end{aligned} \tag{10.6}$$

where ρ_1 and ρ_2 are Lagrange multipliers. From the conditions that the derivatives of L with respect to f_1 and f_2 be equal to 0, we get that f_1 and f_2 must respectively be in the linear span of $\mathbf{x}_1$ and $\mathbf{x}_2$, that is:

$$f_1 = \sum_{j=1}^{n} \alpha_1^{(j)} \phi_1(\mathbf{x}_1^{(j)}), \quad f_2 = \sum_{j=1}^{n} \alpha_2^{(j)} \phi_2(\mathbf{x}_2^{(j)}), \tag{10.7}$$

for some $\alpha_1 \in \mathbb{R}^n$ and $\alpha_2 \in \mathbb{R}^n$. Supposing now that the points are centered in the feature space, that is, $\sum_{i=1}^{n} \phi_1(\mathbf{x}_1^{(i)}) = \sum_{i=1}^{n} \phi_2(\mathbf{x}_2^{(i)}) = 0$, the sample means $\hat{mean}(u_1)$ and $\hat{mean}(u_2)$ are always null, by (10.2) and (10.3). Plugging (10.7) into (10.3), we can then rewrite the sample variance and covariance of u_1 and u_2 in terms of α_1 and α_2:

$$\begin{aligned} \hat{var}(u_1) &= \frac{1}{n} \alpha_1^\top K_1^2 \alpha_1, \\ \hat{var}(u_2) &= \frac{1}{n} \alpha_2^\top K_2^2 \alpha_2, \\ \hat{cov}(u_1, u_2) &= \frac{1}{n} \alpha_1^\top K_1 K_2 \alpha_2, \end{aligned} \tag{10.8}$$

where K_1 and K_2 are the $n \times n$ kernel Gram matrix defined by $K_1(i, j) = k_1(\mathbf{x}_1^{(i)}, \mathbf{x}_1^{(j)}) = \langle \phi_1(\mathbf{x}_1^{(i)}), \phi_1(\mathbf{x}_1^{(j)}) \rangle$ and $K_2(i, j) = k_2(\mathbf{x}_2^{(i)}, \mathbf{x}_2^{(j)}) = \langle \phi_2(\mathbf{x}_2^{(i)}), \phi_2(\mathbf{x}_2^{(j)}) \rangle$ for $1 \leq i, j \leq n$. Observing from (10.7) that the square Hilbert norms of f_1 and f_2 can also be expressed in terms of α_1 and α_2 as follows:

$$||f_1||^2 = \alpha_1^\top K_1 \alpha_1, \quad ||f_2||^2 = \alpha_2^\top K_2 \alpha_2,$$

we finally can rewrite the Lagrangian (10.6) as a function of α_1 and α_2 as follows:

$$L(\alpha_1, \alpha_2, \rho_1, \rho_2) = \frac{1}{n}\alpha_1^\top K_1 K_2 \alpha_2 + \frac{\rho_1}{2n}\left(n - \alpha_1^\top K_1^2 \alpha_1 - n\lambda_1 \alpha_1^\top K_1 \alpha_1\right) + \frac{\rho_2}{2n}\left(n - \alpha_2^\top K_2^2 \alpha_2 - n\lambda_2 \alpha_2^\top K_2 \alpha_2\right).$$

Observing that $K^2 + n\lambda K = \left(K + \frac{n\lambda}{2}\mathbf{I}\right)^2$ up to the second order in λ for any square matrix K ($\mathbf{I}$ represents the identity matrix), and imposing that the derivatives with respect to α_1 and α_2 of the first-order approximation of the Lagrangian be equal to 0, we obtain that the values (α_1, α_2) and (ρ_1, ρ_2) that solve the Lagrangian are solution of the following generalized eigenvalue problem;

$$\begin{pmatrix} \mathbf{0} & K_1 K_2 \\ K_2 K_1 & \mathbf{0} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} = \rho \begin{pmatrix} (K_1 + \frac{n\lambda_1}{2}\mathbf{I})^2 & \mathbf{0} \\ \mathbf{0} & (K_2 + \frac{n\lambda_2}{2}\mathbf{I})^2 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix}. \quad (10.9)$$

It can be shown (Bach and Jordan, 2002) that the canonical correlations are indeed the $\min(N_1, N_2)$ largest generalized eigenvalues of this problem.

Let $\left(\alpha_1^\top, \alpha_2^\top\right)^\top$ be a generalized eigenvector with generalized eigenvalue ρ. From (10.2) and (10.7) we can recover the canonical variate u_1 and u_2 associated with the canonical correlation ρ as follows:

$$u_1 = K_1\alpha_1, \qquad u_2 = K_2\alpha_2.$$

The above derivation is only valid if the points are centered in the feature space. This is not a restriction, however, because for any Gram matrix K of noncentered data points, the Gram matrix $\tilde{K}$ of the centered data points can be computed by $\tilde{K} = N_0 K N_0$ where $N_0 = \mathbf{I} - \frac{1}{n}\mathbf{1}$, where $\mathbf{1}$ is the $n \times n$ matrix composed of ones (Schölkopf et al., 1999).

10.2.3 Multiple KCCA

In this subsection we present an extension of KCCA when more than two kernel matrices are available. This method was suggested by Bach and Jordan (2002) for the purpose of independent component analysis. We refer to it as multiple kernel canonical correlation analysis (MKCCA). It is a natural extension of the ordinary KCCA model described in the previous subsection.

Multiple KCCA

Suppose that we have P data sets $\left(\mathbf{x}_p^{(1)}, \cdots, \mathbf{x}_p^{(n)}\right)$ for $p = 1, 2, \cdots, P$, where $\mathbf{x}_p^{(i)}$ belongs to a set $\mathcal{X}_p$ for $1 \le p \le P$ and $1 \le i \le n$. For each $p = 1, \cdots, P$, suppose that there is a mapping $\phi_p : \mathcal{X}_p \to H_p$ to a Hilbert space H_p, and let K_p be the corresponding Gram matrix, that is:

$$K_p(i, j) = k_p(\mathbf{x}_p^{(i)}, \mathbf{x}_p^{(j)}) = \langle \phi_p(\mathbf{x}_p^{(i)}), \phi_p(\mathbf{x}_p^{(j)}) \rangle,$$

for $1 \le i, j \le n$. Each set of points is supposed to be centered in the feature space.

The goal of MKCCA is to detect directions $f_p \in H_p\,(p = 1, 2, \cdots, P)$ such that the sum of all pairwise correlations between features

$$u_p^{(i)} = \langle f_p, \phi_p(\mathbf{x}_p^{(i)})\rangle, \quad p = 1, \cdots, P, \quad i = 1, \cdots, n, \tag{10.10}$$

be the largest possible, that is, to solve the following problem:

$$\max_{(f_1,\cdots,f_P)\in H_1\times\cdots\times H_P} \sum_{1\le p<q\le P} \hat{corr}(u_p, u_q). \tag{10.11}$$

Following the same approach as the one explained in subsection 10.2.2, the problem (10.11) is regularized with regularization parameters $\lambda_p \geq 0$ for $1 \leq p \leq P$ as follows:

$$\max_{(f_1,\cdots,f_P)\in H_1\times\cdots\times H_P} \sum_{1\le p<q\le P} \frac{\hat{cov}(u_p, u_q)}{(\hat{var}(u_p) + \lambda_p||f_p||^2)^{\frac{1}{2}}\,(\hat{var}(u_q) + \lambda_q||f_q||^2)^{\frac{1}{2}}}. \tag{10.12}$$

It is then easy to derive that the vectors $(f_1, \cdots, f_P)$ solving (10.12) can be expressed as

$$f_p = \sum_{j=1}^{n} \alpha_p^{(j)} \phi_p(\mathbf{x}_p^{(j)}), \tag{10.13}$$

for some vector $\alpha_p \in \mathbb{R}^n$, for $1 \leq p \leq P$, and that the α_p solve the Lagrangian:

$$L = \frac{1}{n} \sum_{1\le p<q\le P} \alpha_p^T K_p K_q \alpha_q + \sum_{p=1}^{P} \frac{\rho_p}{2n} \left(n - \alpha_p^\top K_p^2 \alpha_p - n\lambda_p \alpha_p^\top K_p \alpha_p\right). \tag{10.14}$$

The estimation of canonical correlation scores (CC scores) is now reduced to the following generalized eigenvalue problem:

$$\begin{pmatrix} \mathbf{0} & \cdots & K_1K_P \\ \vdots & \ddots & \vdots \\ K_PK_1 & \cdots & \mathbf{0} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_P \end{pmatrix} = \rho \begin{pmatrix} (K_1 + \frac{n\lambda_1}{2}I)^2 & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & (K_P + \frac{n\lambda_P}{2}I)^2 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_P \end{pmatrix}.$$

The correlated variates can then be obtained by $u_p = K_p\alpha_p$ (for $p = 1, 2, \cdots, P$).

10.2.4 Integrated KCCA

While theoretically sound, the MKCCA approach presented in subsection 10.2.3 may suffer in practice from the fact that by maximizing (10.12), it detects correlations among *all pairs of data sets*. As the number of data sets increases, it is often

the case that no strong signal is present simultaneously in all data sets, except in trivial cases.

In this subsection we propose a variant to perform KCCA on more than two data sets to address this issue. We suppose that the data sets available $(\mathbf{x}_k)_{k=1,\cdots,P}$ are split into two groups, $(\mathbf{x}_p)_{p\in\mathcal{P}}$ and $(\mathbf{x}_q)_{q\in\mathcal{Q}}$, where $\mathcal{P}$ and $\mathcal{Q}$ form a partition of $\{1,\cdots,P\}$. Intuitively, this split should be done in such a way that there is not necessarily a big correlation between the data sets in each split, but that the data sets of one split taken together contain a clear correlation with the data sets of the other split taken together.

More formally, suppose first that the variables are real-valued vectors, that is, $\mathbf{x}_k \in \mathbb{R}^{N_k}$ for $k = 1,\cdots,P$. Then we propose to concatenate the vector representations in each split to obtain two vector representations $\mathbf{x}_\mathcal{P}$ and $\mathbf{x}_\mathcal{Q}$ of the data of dimensions $\sum_{p\in\mathcal{P}} N_p$ and $\sum_{q\in\mathcal{Q}} N_q$ respectively, and to search for canonical correlations between the resulting two vectors. This amounts to solving the generalized eigenvalue problem (10.9) with K_1 and K_2 replaced by $K_\mathcal{P}$ and $K_\mathcal{Q}$, the kernel matrices of the concatenated vectors $\mathbf{x}_\mathcal{P}$ and $\mathbf{x}_\mathcal{P}$. Now, because $k_\mathcal{P}(\mathbf{x}_\mathcal{P}^{(i)},\mathbf{x}_\mathcal{P}^{(j)}) = \sum_{p\in\mathcal{P}} k_p(\mathbf{x}_p^{(i)},\mathbf{x}_p^{(j)})$ and $k_\mathcal{Q}(\mathbf{x}_\mathcal{Q}^{(i)},\mathbf{x}_\mathcal{Q}^{(j)}) = \sum_{q\in\mathcal{Q}} k_q(\mathbf{x}_q^{(i)},\mathbf{x}_q^{(j)})$ for $1 \le i,j \le n$, it follows that

$$K_\mathcal{P} = \sum_{p\in\mathcal{P}} K_p, \quad K_\mathcal{Q} = \sum_{q\in\mathcal{Q}} K_q. \tag{10.15}$$

In the more general case where the data sets are not real vector-valued, but rather belong to more general sets endowed with kernel functions, then the same analysis holds for the vector representations in the features spaces associated with the kernels. In particular (10.15) holds for general kernel functions. Observe that summing up kernels is a convenient way to integrate heterogeneous information, which was, for instance, investigated by Pavlidis et al. (2001b) in the context of gene function prediction from gene expression and phylogenetic profiles.

Integrated kernel CCA

Plugging (10.15) into (10.9), we see that integrated KCCA (IKCCA)can be performed by solving the following generalized eigenvalue problem:

$$\begin{pmatrix} \mathbf{0} & \sum_{p\in\mathcal{P}} K_p . \sum_{q\in\mathcal{Q}} K_q \\ \sum_{q\in\mathcal{Q}} K_q . \sum_{p\in\mathcal{P}} K_p & \mathbf{0} \end{pmatrix} \begin{pmatrix} \alpha_\mathcal{P} \\ \alpha_\mathcal{Q} \end{pmatrix} = \rho \begin{pmatrix} (\sum_{p\in\mathcal{P}} K_p + \frac{n\lambda_\mathcal{P}}{2}\mathbf{I})^2 & \mathbf{0} \\ \mathbf{0} & (\sum_{q\in\mathcal{Q}} K_q + \frac{n\lambda_\mathcal{Q}}{2}\mathbf{I})^2 \end{pmatrix} \begin{pmatrix} \alpha_\mathcal{P} \\ \alpha_\mathcal{Q} \end{pmatrix}.$$

As for KCCA and MKCCA, the correlated variates can again be obtained by $u_\mathcal{P} = K_\mathcal{P}\alpha_\mathcal{P}$ and $u_\mathcal{Q} = K_\mathcal{Q}\alpha_\mathcal{Q}$.

10.2.5 From CCA to Object Selection

Each generalization of CCA presented so far produces several canonical variates for each canonical correlation (2 for KCCA and IKCCA, P for MKCCA). Let us define

the *canonical score* $s \in \mathbb{R}^n$ associated with a given canonical correlation to be the absolute value of the average of the corresponding canonical variates. That is, using the notations of the previous subsection, we respectively define the canonical scores for KCCA, MKCCA, and IKCCA by

Canonical scores

$$s_{KCCA}(i) = \left| \frac{u_1^{(i)} + u_2^{(i)}}{2} \right|, \; s_{MKCCA}(i) = \left| \frac{1}{P} \sum_{j=1}^{P} u_j^{(i)} \right|, \; s_{IKCCA}(i) = \left| \frac{u_{\mathrm{P}}^{(i)} + u_{\mathrm{Q}}^{(i)}}{2} \right|,$$

for $i = 1, \cdots, n$.

The canonical score can be thought of as a quantitative measure of how objects contribute to the canonical correlation. To see this, let us observe, for example, that when u_1 and u_2 are scaled to unit variance

$$\sum_{i=1}^{N} s_{KCCA}(i)^2 = \sum_{i=1}^{N} \left| \frac{u_1^{(i)} + u_2^{(i)}}{2} \right|^2 = \frac{N}{2} \left[1 + \hat{corr}(u_1, u_2) \right].$$

Similar results hold for s_{MKCCA} and s_{IKCCA}. This shows that the correlation $\hat{corr}(u_1, u_2)$ between canonical variates is the sum of individual canonical scores.

In the case where the canonical correlation is due to some hidden phenomenon, this suggests that objects with large canonical scores are more likely to be involved in the phenomenon than others. If one is interested in the detection of such objects, it therefore makes sense to select those objects that have a canonical score above a given threshold.

Link with spectral clustering

It is worth observing that this method of selecting objects bears some similarity to recently studied spectral clustering methods (Weiss, 1999; Ng et al., 2002) which perform data clustering after embedding the data in a feature space using the first eigenvectors of the kernel Gram matrix. In our case, we use the canonical directions instead of the principal directions to map the data, and need to average over the canonical directions found in different feature spaces in order to obtain a one-dimensional mapping. Selecting the objects with large scores then corresponds to a simple clustering method that separates numbers with large absolute values from the others. Of course, this does not make sense if the correlation is due to the presence of two or more different classes of points that one wants to separate as different clusters, in which case large positive canonical variates should be separated from large negative variates, as most clustering methods would do. However, it makes sense in the cases where the canonical correlation is due to the presence of a number of small "interesting" clusters separated from a large bulk of "noninteresting" objects, and where the goal is to detect the "interesting" objects. We illustrate such a case below, in the problem of detecting operons in a genome.

The link with spectral clustering methods is particularly clear when a single kernel matrix K is considered. Similarly to KCCA, kernel principal component analysis (KPCA) searches a direction f of the Hilbert space H that defines a variate

$u^{(i)} = \langle f, \phi\left(\mathbf{x}^{(i)}\right)\rangle$ with maximum variance (where $||f||$ is fixed). As explained in chapter 2, subsection 2.3.4, this is equivalent to the following problem:

$$\min_{\hat{var}(u)=1} ||f||. \tag{10.16}$$

Suppose now that we perform KCCA between the kernel K and itself. From (10.5) it is obvious that the two variates found are equal ($f_1 = f_2$), and that the functional to maximize becomes

$$\gamma_{PCA}(f) = \frac{\hat{var}(u)}{\hat{var}(u) + \lambda||f||^2},$$

where we suppose that $\lambda_1 = \lambda_2 = \lambda$ in (10.5). By homogeneity the maximization of γ_{PCA} is equivalent to the following problem:

$$\max_{\hat{var}(u)=1} \frac{1}{1 + ||f||^2},$$

which is equivalent to (10.16). This shows that KCCA (and any of its generalization to more than two kernels) boils down to KPCA when the kernels are equal.

10.2.6 Diffusion Kernel

In the experiments we perform below, some of the data sets consist of graphs whose nodes are the objects of interest. As an example, metabolic pathways or the organization of the genes on a genome can be represented by graphs with genes as nodes. In order to use such data sets in the KCCA framework, the information contained in the graph must be encoded into a kernel function. We perform this transformation of a graph into a kernel using the diffusion kernel, proposed by Kondor and Lafferty (2002) and reviewed in chapter 8, which we now briefly recall.

Suppose that we have an undirected, unweighted graph $\Gamma = (V, E)$. The opposite Laplacian of this graph is the matrix

$$\mathbf{H}_{ij} = \begin{cases} 1 & \text{for } i \sim j, \\ -d_i & \text{for } i = j, \\ 0 & \text{otherwise,} \end{cases} \tag{10.17}$$

where $i \sim j$ means that the ith and jth genes are joined by an edge on the graph, and d_i is the number of edges emanating from the ith vertex. The exponential of the matrix $\beta\mathbf{H}$ is defined as

$$\exp(\beta\mathbf{H}) = \lim_{m\to\infty} \left(\mathbf{I} + \frac{\beta\mathbf{H}}{m}\right)^m, \tag{10.18}$$

where β is a positive constant. This is equivalent to the following expansion:

$$\exp(\beta\mathbf{H}) = \mathbf{I} + \beta\mathbf{H} + \frac{\beta^2}{2}\mathbf{H}^2 + \frac{\beta^3}{3!}\mathbf{H}^3 + \cdots. \tag{10.19}$$

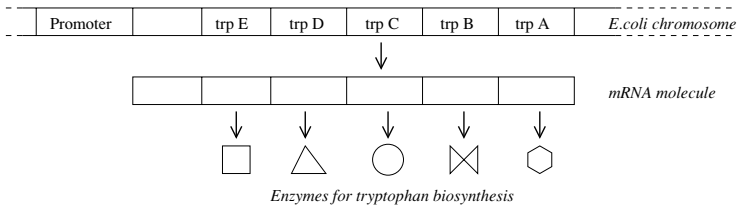


Figure 10.1 The clustered genes in *E.coli* that code for enzymes responsible for the synthesis of the amino acid tryptophan. The five genes are transcribed as a single mRNA molecule, a feature that allows their expression to be controlled coordinately. Such a cluster of genes is called an operon.

The resulting matrix is symmetric and positive definite. It is therefore a valid kernel called the diffusion kernel (Kondor and Lafferty, 2002), which can be thought of as a generalization of the Gaussian radial basis function (RBF) kernel to a discrete setting.

10.3 Experimental Results

In order to test the various generalizations of KCCA (section 10.2.3 and section 10.2.4) and the object selection method (section 10.2.5) presented so far on real-world data, we performed a series of experiments with the goal of detecting operons in the *E.coli* genome.

10.3.1 Operon Detection

Operons

In most bacterial genomes, functionally coupled gene clusters are often adjacent to one another on the genome and regulated under the same upstream promoter, therefore transcribed as one long polycistronic mRNA. Such clusters of genes are called *operons*. As an example, figure 10.1 shows the well-studied tryptophan operon which contains five genes translated into five enzymes responsible for the synthesis of tryptophan. Experimental detection or confirmation of operons is time-consuming (Walters et al., 2001) and relatively difficult to implement in the laboratory as a high-throughput process. Computational prediction of operons has therefore gained increased attention in recent years, either by sequence analysis only (Yada et al., 2001; Salgado et al., 2000; Ermolaeva et al., 2001) or by combining multiple information (Ogata et al., 2000; Zheng et al., 2002).

10.3.2 Data

Because genes that code for enzymes in operons are closely located on the genome, are coregulated, and often catalyze related reactions in metabolic pathways, they should be responsible for a form of correlation between three sorts of data: the

position of genes on the genome, their expression as measured by DNA microarray, and the position of the chemical reactions they catalyze in metabolic pathways. We therefore tried to automatically detect correlations between these three sorts of data using CCA, and to detect genes likely to belong to operons by selecting the genes mostly responsible for the detected correlations.

We therefore collected three sorts of data for the genes of the bacterium *E.coli* and derived three kernel matrices for the 740 genes common to all three data sets as follows.

Pathways

Pathway data were extracted from the KEGG/LIGAND database of chemical compounds and reactions in biological pathways (Goto et al., 2002), which can be freely downloaded from the KEGG database (Kanehisa et al., 2002). This database contains thousands of metabolic reactions known to take place in various organisms, together with the substrates involved and the classification of the catalyzing enzyme as an EC number. From this database we created an undirected graph with genes of *E.coli* as vertices, where two vertices are linked when the genes encode enzymes that can catalyze two successive reactions in a pathway. The resulting graph, called the *gene metabolic network*, is described in more detail in chapter 8, subsection 8.4.1. From this graph of genes we built a diffusion kernel as explained in subsection 10.2.6 with the parameter β set to 1.

Gene positions

The positions of the genes on genomes were obtained from the KEGG/GENES database, which contains various genomic information such as gene names, positions along chromosomes, and their amino acid sequences. From this we computed a matrix of pairwise gene distance,where the distance d_{ij} between gene i and gene j is defined by the number of nucleotides between the end of the ith gene and the start of the jth gene along the chromosomes. We then derived a distance kernel by the formula $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-d_{ij}/h)$, where h is a parameter set to 10^5.

Gene expression

Finally the gene expression data for 48 experiments[1] on the genes *E.coli K-12* were downloaded from the KEGG/EXPRESSION database, a repository of expression data for *Saccharomyces cerevisiae*, *E.coli*, and *Bacilla subtilis*. Given the (R,G) fluorescence intensity pairs for each gene on each array (where R=red for Cy5 and G=green for Cy3), we evaluated the expression level by the log ratio $\log(R_S - R_B)/(G_S - G_B)$, where G_B is control-background, G_S is control-signal, R_B is target-background, and R_S is target-signal, respectively. We then used a Gaussian RBF kernel with unit width to obtain the expression Gram matrix.

This results in three 740×740 kernel Gram matrices, which we denote by $K_{pathway}$, K_{genome} and $K_{expression}$ below.

1. With identification numbers ex0000287 to ex0000334 in the KEGG/EXPRESSION database.

Table 10.1 List of experiments performed to detect operons in the *E.coli* genome. For OKCCA and IKCCA methods, an ordinary KCCA is performed between the two kernels in the columns Kernel 1 and Kernel 2. For the MKCCA method, a multiple KCCA is performed between the three kernels. For the KPCA method, an ordinary KPCA is performed on Kernel 1. In each case, operons are then predicted by the gene selection method described in subsection 10.2.5

Name	Abbr.	Method	Kernel 1	Kernel 2	Kernel 3
OKCCA-a	O-a	KCCA	$K_{pathway}$	K_{genome}	-
OKCCA-b	O-a	KCCA	K_{genome}	$K_{expression}$	-
OKCCA-c	O-a	KCCA	$K_{expression}$	$K_{pathway}$	-
MKCCA	M	MKCCA	$K_{pathway}$	K_{genome}	$K_{expression}$
IKCCA-a	I-a	IKCCA	$K_{genome} + K_{expression}$	$K_{pathway}$	-
IKCCA-b	I-a	IKCCA	$K_{expression} + K_{pathway}$	K_{genome}	-
IKCCA-c	I-a	IKCCA	$K_{pathway} + K_{genome}$	$K_{expression}$	-
KPCA-a	S-a	KPCA	$K_{pathway}$	-	-
KPCA-b	S-a	KPCA	K_{genome}	-	-
KPCA-c	S-a	KPCA	$K_{expression}$	-	-

10.3.3 Experiments

We performed successively ordinary KCCA (OKCCA) between the three possible pairs of kernels, MKCCA between the three kernels, and IKCCA between all splits of the three kernels into two groups. To confirm the improvement due to the comparison and integration of several attributes, we also performed ordinary kernel PCA on each single data set (e.g., pathway alone, genome alone, and expression alone). Table 10.1 summarizes these experiments.

We then applied the gene selection procedure described in subsection 10.2.5 with a varying threshold, and compared the set of genes selected at a given threshold with a database of known operons (Ito et al., 1999). By varying the threshold, we computed the number of selected genes that really belong to a known operon (true positives) as a function of the number of selected genes that do not belong to a known operon (false positive). We therefore obtained a receiver operating characteristic (ROC) curve (Gribskov and Robinson, 1996), that is a plot of true positive as a function of false positives, for each CCA method.

10.3.4 Results

Figure 10.2 shows multiple cross-scatterplots of the first canonical variates obtained with the MKCCA method between pathway, genome, and expression. Figures 10.3 shows scattersplots of the first canonical variates obtained with the IKCCA-a,-b, and -c methods. In these scatterplots each point corresponds to one gene. The diagonal shapes of the clouds of points indicate that correlations have been detected

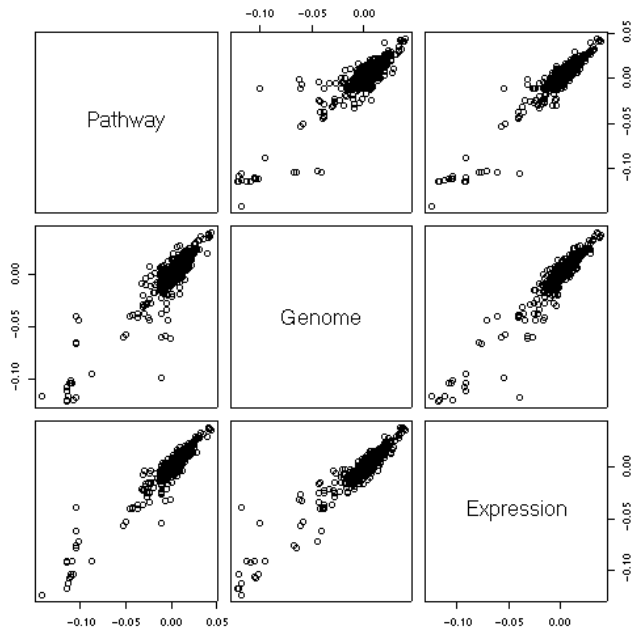


Figure 10.2 Multiple cross-scatterplots of the first canonical variates in MKCCA. In each plot, a circle corresponds to a gene. MKCCA extracts one canonical variate for each of the three data sets: pathway, genome, and expression. These plots highlight the pairwise correlations between these canonical variates.

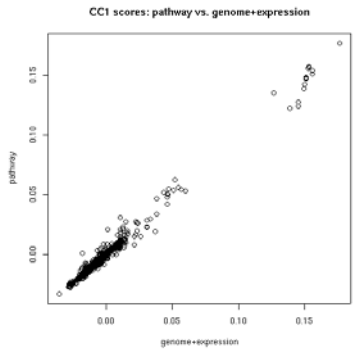
in all cases. The correlations detected are mostly due to the genes with high or low scores, in particular in MKCCA, IKCCA-a, and IKCCA-b.

Operons are likely to form clusters simultaneously in all feature spaces defined by the three kernels considered. As a result, they might be the cause behind the first canonical correlation, in which case genes that form operons are more likely to contribute strongly to the canonical correlation than are others. This motivates the use of the object selection methods described in subsection 10.2.5 with the goal of detecting genes that belong to operons. It should be noted here that we don't try to separate different operons, but rather to separate operon genes from the rest.

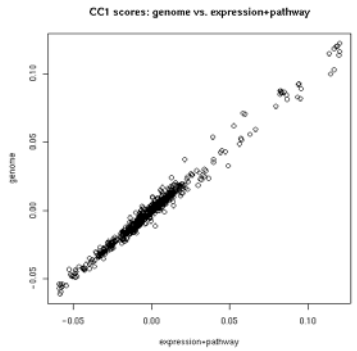
Operon detection

Figure 10.4 shows the ROC curves for the task of detecting operon genes with the object selection method described in subsection 10.2.5 applied to each CCA method in table 10.1. Compared with the performance of the approach applied to a single data set, the detection rates have been improved by the comparison and integration of several data sets. Table 10.2 shows the number of genes correctly

(a)



(b)



(c)

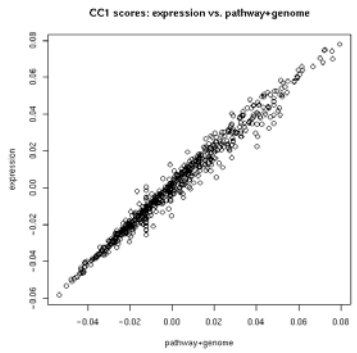


Figure 10.3 Scatterplots of the first canonical variates in IKCCA-*a* (pathway vs. genome + expression), IKCCA-*b* (genome vs. expression + pathway), and IKCCA-*c* (expression vs. pathway + genome). In each case, the plot highlights the canonical correlation between the two variates extracted by IKCCA.

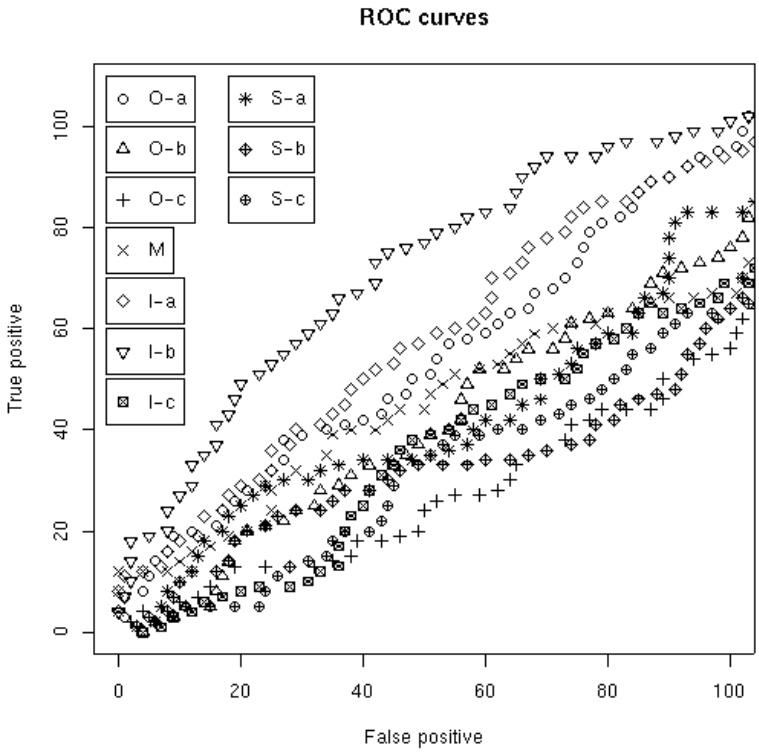


Figure 10.4 ROC curves for the detection of operon genes. O-a, -b, -c indicate OKCCA-a, -b, -c, respectively; M indicates MKCCA; I-a, -b, -c indicate IKCCA-a, -b, -c respectively; S-a, -b, -c indicate KPCA-a, -b, -c respectively. For each method, the gene selection method described in subsection 10.2.5 was performed with a varying threshold in order to vary the number of genes selected. These curves show the number of selected genes that belong to known operons (true positives on the y-axis) as a function of the number of genes selected even though they don't belong to known operons (false positives on the x-axis).

Table 10.2 Number of correctly detected genes based on the first canonical scores in each KCCA. We set the threshold such that 10% of all genes with high scores (74 out of 740 genes) are selected.

Operon (# of genes)	O-a	O-b	O-c	M	I-a	I-b	I-c
Biotin metabolism (3)	3	1	0	3	3	3	0
Fatty acid (short-chain) metabolism (3)	0	3	0	2	0	3	3
Fumarate reductase (4)	4	0	2	4	4	4	0
Galactose metabolism (4)	4	0	0	4	3	4	1
Glycerol-3-phosphate dehydrogenase (3)	0	3	3	3	3	3	3
Menaquinone (vitamin K_2) biosynthesis (5)	0	3	0	0	4	0	0
NADH dehydrogenase (13)	0	0	0	0	0	13	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Total number (280)	39	34	27	37	42	52	28

selected by each method for several known operons when we set the threshold such that 10% of all genes (74 out of 740 genes) are selected, for instance.

The best operon detection performance is obtained by IKCCA-b, which corresponds to correlations between K_{genome} and $K_{pathway} + K_{expression}$. Next are IKCCA-a, corresponding to correlations between $K_{pathway}$ and $K_{genome} + K_{expression}$, and OKCCA-a, corresponding to correlations between K_{genome} and $K_{pathway}$. The worst methods are OKCCA-c, IKCCA-c, and OKCCA-b, which correspond to correlations between $K_{expression}$ and another kernel involving $K_{pathway}$ or K_{genome}, or both.

Kernel hierarchy

These results suggest several remarks. First, a clear hierarchy appears between the three kernels. K_{genome} is the one that contains the most information about operons, as seen from the good performance of the methods that detect correlations between K_{genome} alone and other kernels. It is closely followed by $K_{pathway}$. $K_{expression}$ is clearly less related to operons, as shown by the poor performance of the experiments where canonical correlations were driven by $K_{expression}$. The major contribution of K_{genome} in the detection of operons makes sense, by the definition itself of operons, which are clusters of genes on the genome. The relatively poor performance of OKCCA-b (K_{genome} vs. $K_{expression}$), and more generally of all experiments involving $K_{expression}$ alone, seems to indicate that the quality of the expression data used is poor, since genes in an operon are supposed to be coregulated. In contrast, the good performance of $K_{pathway}$ suggests that the pathway database is of reasonable quality.

Advantage of kernel combination

Second, in spite of the poor quality of the expression data, it appears that the best performance is obtained by using the three kernels in the form of canonical correlations between K_{genome} and $K_{pathway}+K_{expression}$. This means that IKCCA-b is able to somehow denoise the expression data and extract from a combination of pathway information and expression data a meaningful correlation with the genome data that outperforms the correlation detected by each data set alone with the

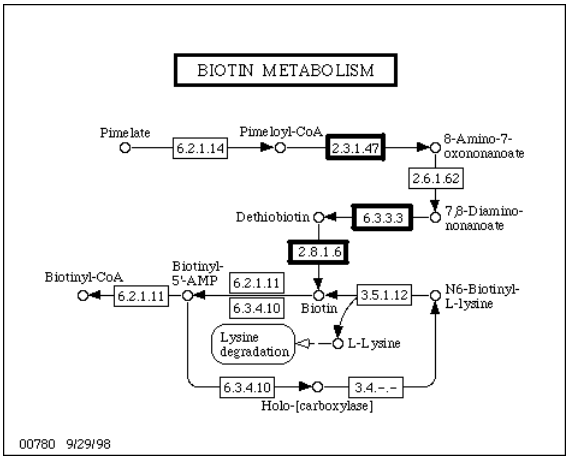


Figure 10.5 An example of known operons in the operon data library. The genes in known operons are represented by the corresponding EC numbers, and are outlined in bold boxes.

genome data. This experiment is a typical example where IKCCA is more relevant than MKCCA and OKCCA, because of the difference in the information about operons contained in each data set.

Visualization

For visualization the genes detected can be mapped to the KEGG/pathway visualization tool. As an example, figure 10.5 shows a very local picture of the large metabolic network, namely biotin metabolism, together with 3 genes known to form an operon (the genes respectively marked 2.3.1.47, 6.3.3.3, and 2.8.1.6). Figure 10.6, on the other hand, shows the genes selected by the IKCCA-a method which belong to the biotin metabolism, when the threshold of the gene selection procedure is set in such a way that 10% (74) of all genes are selected. The three known operon genes are selected, as well as a fourth gene (*JW0757*) annotated 2.6.1.62. Figure 10.7 shows the positions of the four selected genes on the genome, where genes *JW0757*, *JW0758*, *JW0759*, and *JW0761* on the genome correspond to their product enzymes EC 2.6.1.62, EC 2.8.1.6, EC 2.3.1.47, and EC 6.3.3.3 respectively in the pathway. One can observe that the four genes selected catalyze four successive reactions in the biotin pathway, and they are adjacent on the genome. The reason why the gene *JW0757* does not belong to the operon formed by the three other genes is that its translation direction is different from that of the other genes. This difference is an important factor in the mechanism of transcription, because a transcription starts from the promoter at the beginning of genes in the same direction. This suggests that further improvements might result from taking into account the direction of the genes in the genome kernel function, which currently only contains distance information.

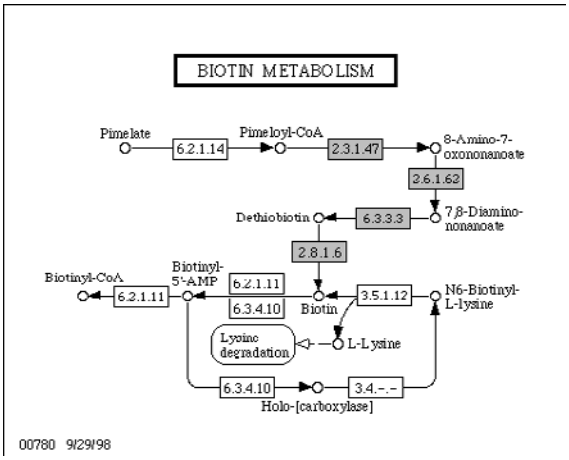


Figure 10.6 An example of operons predicted by IKCCA. The genes selected are represented by the corresponding EC numbers, and colored in gray.

Figure 10.7 The three genes *JW0758*, *JW0759*, and *JW0761* (corresponding to EC 2.8.1.6, EC 2.3.1.47, and EC 6.3.3.3 respectively in the biotin pathway) are part of an operon. Our gene selection method included the gene *JW0757* (corresponding to EC 2.6.1.62 in the pathway) in the operon because it is close to the other genes in the genome and has a similar function. This is a mistake, however, because the orientation of this gene, which corresponds to the direction of translation, is opposite that of the remaining genes.

10.4 Discussion and Conclusion

In this chapter we proposed various approaches to investigate the correlation between heterogeneous genomic data. We proposed several generalized formulations of ordinary KCCA and derived a gene selection procedure based on the newly introduced canonical score. The integration of different types of genomic data (e.g., biochemical pathways, genomes, and expression data) is a key problem in computational biology nowadays. When data types are different (e.g., graphs, strings, and vectors), integration strategies often rely on various heuristic approaches, which depend on the types of data. The originality of our approach is the extension of the concept of correlation for nonvectorial data and integration of genomic data in a rigorous mathematical framework common to all types.

The proposed methods enable us to automatically find correlated directions, along which high/low scoring genes tend to share similarities with respect to multiple biological attributes. These methods give encouraging results on the problem of recognizing the genes that belong to operons in the *E. coli* genome, by comparing three data sets corresponding to *functional* relationships between genes in metabolic pathways, *positional* relationships along the chromosome, and *coexpression* relationships as observed by gene expression data. We observed that generalized KCCAs (MKCCA and IKCCA) outperform ordinary KCCAs in this context. In our preliminary results the number of correct operon candidates selected by MKCCA at a given rate of false predictions tends to be smaller than that selected by the best choice of IKCCA, that is, when the genome data set is compared to the combination of the pathway and the expression data sets. One explanation for this difference in performance might be the fact that MKCCA looks for correlations simultaneously among all pairs of data sets. It would work well if the genes in an operon were systematically similar to each other with respect to all three sources of information we used. To the contrary, in our IKCCA setting, we relax the constraint of having a correlation between gene positions in the pathways and gene expression (which alone gave the worst results), and rather focus on detection of correlations between positions on the genome on the one hand, and positions on the pathways *or* expression profile on the other. Due to noise and errors in the data, this less constrained problem might detect biological phenomena (operons in our case) more easily than the MKCCA approach. We conjecture that as the number of data sets increases, the performance of MKCCA might decrease because it becomes too difficult to impose correlation constraints between any two data sets. In that case it might be more efficient to try to detect correlations between a smaller number of data sets, obtained themselves by combining the initial data sets available, as we did in IKCCA.

From the viewpoint of algorithms, much work remains to be done on testing the influence of kernel parameters on the final performance of the methods. Any real-world application of these methods might require a fine-tuning of each kernel, as well as of the regularization parameters used in the KCCA algorithms.

Finally, it should be pointed out that the canonical variates extracted from the comparison of several data sets can be used as new representations of the genes themselves. This avenue was investigated by Vert and Kanehisa (2003b) with promising results for gene function prediction from heterogeneous data.

Acknowledgments

Y.Y. and M.K. were supported by grants from the Ministry of Education, Culture, Sports, Science and Technology of Japan, the Japan Society for the Promotion of Science, and the Japan Science and Technology Corporation. The computational resource was provided by the Bioinformatics Center, Institute for Chemical Research, Kyoto University.

11 Kernel-Based Integration of Genomic Data Using Semidefinite Programming

Gert R. G. Lanckriet
Nello Cristianini
Michael I. Jordan
William Stafford Noble

An important theme in bioinformatics is the leveraging of different descriptions of a biological phenomenon, each capturing different aspects of the phenomenon. Many sources of information concerning genes and proteins are now available, such as sequence, expression, interaction, and regulation data. More data types are going to be available in the near future, such as array-based fitness profiles and protein-protein interaction data from mass spectrometry. A variety of inferential problems in bioinformatics—such as gene function prediction, prediction of protein structure and localization, and inference of regulatory and metabolic networks—could benefit from a methodology that allows different sources of information to be treated in a unified way, merging them into a single representation rather than using only the description that is judged to be the most relevant at hand.

This chapter describes a computational framework for integrating and drawing inferences from a collection of genome-wide measurements. Each data set is represented via a kernel function, which defines generalized similarity relationships between pairs of entities, such as genes or proteins. The kernel representation is both flexible and efficient, and provides a principled framework in which many types of data can be represented, including vectors, strings, trees, and graphs. Furthermore, kernel functions derived from different types of data can be combined in a straightforward fashion—recent advances in the theory of kernel methods have provided efficient algorithms to perform such combinations in an optimal way. These methods formulate the problem of optimal kernel combination as a convex optimization problem that can be solved with semidefinite programming techniques.

After introducing the semidefinite programming techniques, we illustrate their use in two domains: (1) the problem of identifying membrane proteins in yeast, and (2) the prediction of functional classifications of proteins in yeast. We base these

predictions on a variety of sources of information, including amino acid sequence, hydropathy profiles, gene expression data, known protein-protein interactions, and known protein complexes. We show that a support vector machine (SVM) trained from all of these data, using the combined kernel, performs significantly better than the same algorithm trained on any single type of data, and better than previously described approaches.

11.1 Introduction

Much research in computational biology involves drawing statistically sound inferences from collections of data. For example, the function of an unannotated protein sequence can be predicted based on an observed similarity between that protein sequence and the sequence of a protein of known function. Related methodologies involve inferring related functions of two proteins if they occur in fused form in some other organism, if they co-occur in multiple species, if their corresponding messenger RNAs (mRNAs) share similar expression patterns, or if the proteins interact with one another.

It seems natural that, while all such data sets contain important pieces of information about each gene or protein, the comparison and fusion of these data should produce a much more sophisticated picture of the relations among proteins, and a more detailed representation of each protein. Especially the recent availability of multiple types of genome-wide data that provide biologists with complementary views of a single genome highlights the need for machine learning algorithms that unify these views and exploit this fused representation. Combining information from different sources contributes to forming a complete picture of the relations between the different components of a genome, enhancing the total information about the problem at hand.

In yeast, for example, for a given gene we typically know the protein it encodes, that protein's similarity to other proteins, its hydrophobicity profile, the mRNA expression levels associated with the given gene under hundreds of experimental conditions, the occurrences of known or inferred transcription factor binding sites in the upstream region of that gene, and the identities of many of the proteins that interact with the given gene's protein product or form a complex with it. Each of these distinct data types provides one view of the molecular machinery of the cell. In the near future, research in bioinformatics will focus more and more heavily on methods of data fusion.

One problem with this approach, however, is that genomic data come in a wide variety of data formats: expression data are expressed as vectors or time series; protein sequence data as strings from a 20-symbol alphabet; gene sequences are strings from a different (4-symbol) alphabet; protein-protein interactions are best expressed as graphs, and so on.

This chapter presents a computational and statistical framework for integrating heterogeneous descriptions of the same set of genes, proteins, or other entities. The

approach relies on the use of kernel-based statistical learning methods that have already proved to be very useful tools in bioinformatics (Jaakkola et al., 1999; Brown et al., 2000; Furey et al., 2000; Zien et al., 2000). These methods represent the data by means of a kernel function, which defines similarities between pairs of genes, proteins, and so on. Such similarities can be quite complex relations, implicitly capturing aspects of the underlying biological machinery. One reason for the success of kernel methods is that the kernel function takes relationships that are implicit in the data and makes them explicit, so that it is easier to detect patterns. Each kernel function thus extracts a specific type of information from a given data set, thereby providing a partial description or view of the data. The goal of this chapter is to find a kernel that best represents all of the information available for a given statistical learning task. Given many partial descriptions of the data, we solve the mathematical problem of combining them using a convex optimization method known as semidefinite programming (SDP) (Boyd et al., 1994; Nesterov and Nemirovsky, 1994; Vandenberghe and Boyd, 1996). This SDP-based approach (Lanckriet et al., 2002) yields a general methodology for combining many partial descriptions of data that are statistically sound, as well as computationally efficient and robust.

In order to demonstrate the feasibility of these methods, we describe two problems: identifying membrane proteins in yeast and predicting the function of yeast proteins. Both problems are statistical learning problems in which a single type of feature derived from the protein sequence provides only partial information. We demonstrate that incorporating knowledge derived from the amino acid sequences, protein complex data, hydropathy profiles, gene expression data, and known protein-protein interactions significantly improves classification performance relative to previously described methods and relative to our method trained on any single type of data.

We begin by describing related work. Afterward, the main ideas of the kernel approach to pattern analysis are explained and SDP techniques are introduced as an advanced instance of convex optimization. After presenting the necessary mathematical background, we describe how different kernels defined on different data can be integrated using SDP techniques to provide a unified description. Finally, we describe the two biological applications of membrane protein recognition and protein function prediction in yeast.

11.2 Related Work

Considerable work has been devoted to the problem of automatically integrating genomic data sets, leveraging the interactions and correlations between them to obtain more refined and higher-level information. Previous research in this field can be divided into three classes of methods.

The first class treats each data type independently. Inferences are made separately from each data type, and an inference is deemed correct if the various data agree.

This type of analysis has been used to validate, for example, gene expression and protein-protein interaction data (Ge et al., 2001; Grigoriev, 2001; Mrowka et al., 2003), to validate protein-protein interactions predicted using five different methods (von Mering et al., 2002), and to infer protein function (Marcotte et al., 1999). A slightly more complex approach combines multiple data sets using intersections and unions of the overlapping sets of predictions (Jansen et al., 2002).

The second formalism to represent heterogeneous data is to extract binary relations between genes from each data source, and represent them as graphs. As an example, sequence similarity, protein-protein interaction, gene coexpression, or closeness in a metabolic pathway can be used to define binary relations between genes. Several groups have attempted to compare the resulting gene graphs using graph algorithms (Nakaya et al., 2001; Tanay et al., 2002), in particular to extract clusters of genes that share similarities with respect to different sorts of data.

The third class of techniques uses statistical methods to combine heterogeneous data. For example, Holmes and Bruno use a joint likelihood model to combine gene expression and upstream sequence data for finding significant gene clusters (Holmes and Bruno, 2000). Similarly, Deng et al. (2003b) use a maximum likelihood method to predict protein-protein interactions and protein function from three types of data. Alternatively, protein localization can be predicted by converting each data source into a conditional probabilistic model and integrating via Bayesian calculus (Drawid and Gerstein, 2000). The general formalism of graphical models, which includes Bayesian networks and Markov random fields as special cases, provides a systematic methodology for building such integrated probabilistic models. As an instance of this methodology, Deng et al. (2003a) developed a Markov random field model to predict yeast protein function. They found that the use of different sources of information indeed improved prediction accuracy when compared to using only one type of data.

This chapter describes a fourth type of data fusion technique, also statistical, but of a more nonparametric and discriminative flavor. The method, described in detail below, consists of representing each type of data independently as a matrix of kernel similarity values. These kernel matrices are then combined to make overall predictions. An early example of this approach, based on fixed sums of kernel matrices, showed that combinations of kernels can yield improved gene classification performance in yeast, relative to learning from a single kernel matrix (Pavlidis et al., 2001b). The current work takes this methodology further—we use a *weighted* linear combination of kernels, and demonstrate how to estimate the kernel weights from the data. This yields not only predictions that reflect contributions from multiple data sources but also yields an indication of the relative importance of these sources.

Markov random field

The graphical model formalism, as exemplified by the Markov random field model of Deng et al. (2003a), has several advantages in the biological setting. In particular, prior knowledge can be readily incorporated into such models, with standard Bayesian inference algorithms available to combine such knowledge with data. Moreover, the models are flexible, accommodating a variety of data types and providing a modular approach to combining multiple data sources. Classic

discriminative statistical approaches, on the other hand, can provide superior performance in simple situations, by focusing explicitly on the boundary between classes, but tend to be significantly less flexible and less able to incorporate prior knowledge. As we discuss in this chapter, however, recent developments in kernel methods have yielded a general class of discriminative methods that readily accommodate nonstandard data types (such as strings, trees, and graphs), allow prior knowledge to be brought to bear, and provide general machinery for combining multiple data sources.

11.3 Kernel Methods

Kernel methods work by embedding data items (genes, proteins, etc.) into a vector space $\mathcal{F}$, called a *feature space*, and searching for linear relations in such a space. This embedding is defined implicitly, by specifying an inner product for the feature space via a positive semidefinite *kernel function*: $k(\mathbf{x}_1, \mathbf{x}_2) = \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle$, where $\Phi(\mathbf{x}_1)$ and $\Phi(\mathbf{x}_2)$ are the embeddings of data items $\mathbf{x}_1$ and $\mathbf{x}_2$. Note that if all we require in order to find those linear relations are inner products, then we do not need to have an explicit representation of the mapping Φ, nor do we even need to know the nature of the feature space. It suffices to be able to evaluate the kernel function, which is often much easier than computing the coordinates of the points explicitly. Evaluating the kernel on all pairs of data items yields a symmetric, positive definite matrix K known as the *kernel matrix*, which can be regarded as a matrix of generalized similarity measures among the data points.

The kernel-based binary classification algorithm that we describe in this chapter, the *1-norm soft margin support vector machine* (Boser et al., 1992; Schölkopf and Smola, 2002), forms a linear discriminant boundary in feature space $\mathcal{F}$, $f(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + b$, where $\mathbf{w} \in \mathcal{F}$ and $b \in \mathbb{R}$. Given a labeled sample $S_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $\mathbf{w}$ and b are optimized to maximize the distance ("margin") between the positive and negative class, allowing misclassifications (therefore "soft margin"):

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n \\ & \xi_i \geq 0, \quad i = 1, \ldots, n, \end{aligned} \tag{11.1}$$

where C is a regularization parameter, trading off error against margin. By considering the corresponding dual problem of (11.1), one can prove (see, e.g., Schölkopf and Smola, 2002) that the weight vector can be expressed as $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \Phi(\mathbf{x}_i)$, where the support values α_i are solutions of the following dual *quadratic program* (QP):

$$\max_{\boldsymbol{\alpha}} \quad 2\boldsymbol{\alpha}^T\mathbf{e} - \boldsymbol{\alpha}^T \mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y}) \boldsymbol{\alpha} \;:\; C \geq \boldsymbol{\alpha} \geq 0, \;\; \boldsymbol{\alpha}^T\mathbf{y} = 0. \tag{11.2}$$

The first stage of processing in a kernel method is thus to reduce the data by computing the kernel matrix. Given this matrix, and given the labels y_i, we can throw away the original data; the problem of fitting the SVM to data reduces to an optimization procedure that is based entirely on the kernel matrix and the labels. Different kernels correspond to different embeddings of the data and thus can be viewed as capturing different notions of similarity. For example, in a space derived from amino acid sequences, two genes that are close to one another will have protein products with very similar amino acid sequences. This amino acid space would be quite different from a space derived from microarray gene expression measurements, in which closeness would indicate similarity of the expression profiles of the genes. Finally, an unlabeled data item $\mathbf{x}_{new}$ can be classified by computing the linear function

$$f(\mathbf{x}_{new}) = \mathbf{w}^T \Phi(\mathbf{x}_{new}) + b = \sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}_{new}) + b.$$

If $f(\mathbf{x}_{new})$ is positive, then we classify $\mathbf{x}_{new}$ as belonging to class $+1$; otherwise, we classify $\mathbf{x}_{new}$ as belonging to class -1.

11.4 Semidefinite Programming

In this section we review the basic definition of SDP as well as some important concepts and key results. Details and proofs can be found in Boyd and Vandenberghe (2001).

SDP (Nesterov and Nemirovsky, 1994; Vandenberghe and Boyd, 1996; Boyd and Vandenberghe, 2001) deals with the optimization of convex functions over the convex cone[1] of symmetric, positive definite matrices

$$\mathcal{P} = \left\{ X \in \mathbb{R}^{p \times p} \mid X = X^T, X \succeq 0 \right\},$$

or affine subsets of this cone. As explained earlier, every positive definite and symmetric matrix is a kernel matrix, and conversely, every kernel matrix is symmetric and positive definite. Therefore $\mathcal{P}$ can be viewed as a search space for possible kernel matrices. This consideration leads to the key problem addressed in this chapter—we wish to specify a convex cost function that will enable us to learn the optimal kernel matrix within $\mathcal{P}$ using SDP.

1. $S \subseteq \mathbb{R}^d$ is a convex cone iff $\forall \mathbf{x}, \mathbf{y} \in S, \forall \lambda, \mu \geq 0 \ : \lambda \mathbf{x} + \mu \mathbf{y} \in S$.

11.4.1 Definition

A *linear matrix inequality* (LMI) is a constraint of the form

$$F(\mathbf{u}) := F_0 + u_1 F_1 + \ldots + u_q F_q \preceq 0.$$

Here, $\mathbf{u}$ is the vector of decision variables, and $F_0, \ldots, F_q$ are given symmetric $p \times p$ matrices. The notation $F(\mathbf{u}) \preceq 0$ means that the symmetric matrix F is negative semidefinite. Note that such a constraint is in general a *nonlinear* constraint; the term "linear" in the name LMI merely emphasizes that F is affine in $\mathbf{u}$. Perhaps the most important feature of an LMI constraint is its convexity: the set of $\mathbf{u}$ that satisfy the LMI is a convex set.

An LMI constraint can be seen as an *infinite* set of scalar, affine constraints. Indeed, for a given $\mathbf{u}$, $F(\mathbf{u}) \preceq 0$ iff $\mathbf{z}^T F(\mathbf{u})\mathbf{z} \leq 0$ for every $\mathbf{z}$; every constraint indexed by $\mathbf{z}$ is an affine inequality in the ordinary sense, that is, the left-hand side of the inequality is a scalar, composed of a linear term in $\mathbf{u}$ and a constant term. Alternatively, using a standard result from linear algebra, we may state the constraint $F(\mathbf{u}) \preceq 0$ as

$$\forall Z \in \mathcal{P} \ : \ \mathrm{trace}(F(\mathbf{u})Z) \leq 0. \tag{11.3}$$

This can be seen by writing down the spectral decomposition of Z and using the fact that $\mathbf{z}^T F(\mathbf{u})\mathbf{z} \leq 0$ for every $\mathbf{z}$.

An SDP is an optimization problem with a linear objective, and linear matrix inequality and affine equality constraints.

Definition 11.1 *A semidefinite program is a problem of the form*

$$\begin{aligned} \min_{\mathbf{u}} \quad & \mathbf{c}^T \mathbf{u} \\ \textit{subject to} \quad & F^j(\mathbf{u}) = F_0^j + u_1 F_1^j + \ldots + u_q F_q^j \preceq 0, \quad j = 1, \ldots, L \\ & A\mathbf{u} = \mathbf{b}, \end{aligned} \tag{11.4}$$

where $\mathbf{u} \in \mathbb{R}^q$ is the vector of decision variables, $\mathbf{c} \in \mathbb{R}^q$ is the objective vector, and matrices $F_i^j = (F_i^j)^T \in \mathbb{R}^{p \times p}$ are given.

By convexity of their LMI constraints, SDPs are convex optimization problems. The usefulness of the SDP formalism stems from two important facts. First, despite the seemingly very specialized form of SDPs, they arise in a host of applications; second, there exist "interior-point" algorithms to globally solve SDPs that have extremely good theoretical and practical computational efficiency (Vandenberghe and Boyd, 1996).

One very useful tool to reduce a problem to an SDP is the so-called Schur complement lemma.

Lemma 11.2 (Schur complement lemma) *Consider the partitioned symmetric matrix*

$$X = X^T = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix},$$

where A and C are square and symmetric. If $\det(A) \neq 0$, we define the Schur complement of A in X by the matrix $S = C - B^T A^{-1} B$. The Schur complement lemma states that if $A \succ 0$, then $X \succeq 0$ iff $S \succeq 0$.

To illustrate how this lemma can be used to cast a nonlinear convex optimization problem as an SDP, consider the following result:

Lemma 11.3 *The quadratically constrained quadratic program (QCQP)*

$$\begin{aligned} \min_{\mathbf{u}} \quad & f_0(\mathbf{u}) \\ \textit{subject to} \quad & f_i(\mathbf{u}) \leq 0, \quad i = 1, \ldots, M, \end{aligned} \tag{11.5}$$

with $f_i(\mathbf{u}) \triangleq (A_i\mathbf{u} + \mathbf{b}_i)^T (A_i\mathbf{u} + \mathbf{b}_i) - \mathbf{c}_i^T \mathbf{u} - d_i$, is equivalent to the SDP:

$$\begin{aligned} \min_{\mathbf{u},t} \quad & t \\ \textit{subject to} \quad & \begin{pmatrix} I & A_0\mathbf{u} + \mathbf{b_0} \\ (A_0\mathbf{u} + \mathbf{b_0})^T & \mathbf{c_0}^T\mathbf{u} + d_0 + t \end{pmatrix} \succeq 0, \\ & \begin{pmatrix} I & A_i\mathbf{u} + \mathbf{b}_i \\ (A_i\mathbf{u} + \mathbf{b}_i)^T & \mathbf{c}_i^T\mathbf{u} + d_i \end{pmatrix} \succeq 0, \quad i = 1, \ldots, M. \end{aligned} \tag{11.6}$$

This can be seen by rewriting the QCQP (11.5) as

$$\begin{aligned} \min_{\mathbf{u},t} \quad & t \\ \text{subject to} \quad & t - f_0(\mathbf{u}) \geq 0, \\ & -f_i(\mathbf{u}) \geq 0, \quad i = 1, \ldots, M. \end{aligned}$$

Note that for a fixed and feasible $\mathbf{u}$, $t = f_0(\mathbf{u})$ is the optimal solution. The convex quadratic inequality $t - f_0(\mathbf{u}) = (t + \mathbf{c_0}^T\mathbf{u} + d_0) - (A_0\mathbf{u} + \mathbf{b_0})^T I^{-1} (A_0\mathbf{u} + \mathbf{b_0}) \geq 0$ is now equivalent to the following LMI, using the Schur complement lemma 11.2:

$$\begin{pmatrix} I & A_0\mathbf{u} + \mathbf{b_0} \\ (A_0\mathbf{u} + \mathbf{b_0})^T & \mathbf{c_0}^T\mathbf{u} + d_0 + t \end{pmatrix} \succeq 0.$$

Similar steps for the other quadratic inequality constraints finally yield (11.6), an SDP in standard form (11.4), equivalent to (11.5). This shows that a QCQP can be cast as an SDP. Of course, in practice a QCQP should not be solved using general-purpose SDP solvers, since the particular structure of the problem at hand can be efficiently exploited. The above does show that QCQPs, and in particular, linear programming problems, belong to the SDP family.

11.4.2 Duality

An important principle in optimization—perhaps the most important principle—is that of *duality*. To illustrate duality in the case of an SDP, we first review basic concepts in duality theory and then show how they can be extended to SDP. In particular, duality will give insights into optimality conditions for the SDP.

Consider an optimization problem with n variables and m scalar constraints

$$\begin{aligned} \min_{\mathbf{u}} \quad & f_0(\mathbf{u}) \\ \text{subject to} \quad & f_i(\mathbf{u}) \leq 0, \quad i = 1, \ldots, m, \end{aligned} \tag{11.7}$$

where $\mathbf{u} \in \mathbb{R}^n$. In the context of duality, problem (11.7) is called the *primal problem*; we denote its optimal value p^*. For now, we do not assume convexity.

Definition 11.4 (Lagrangian) *The* Lagrangian $\mathcal{L} : \mathbb{R}^{n+m} \to \mathbb{R}$ *corresponding to the minimization problem (11.7) is defined as*

$$\mathcal{L}(\mathbf{u}, \boldsymbol{\lambda}) = f_0(\mathbf{u}) + \lambda_1 f_1(\mathbf{u}) + \ldots + \lambda_m f_m(\mathbf{u}).$$

The $\lambda_i \in \mathbb{R},\ i = 1, \ldots, m$ *are called* Lagrange multipliers *or* dual variables.

One can now notice that

$$h(\mathbf{u}) = \max_{\boldsymbol{\lambda} \geq 0} \mathcal{L}(\mathbf{u}, \boldsymbol{\lambda}) = \begin{cases} f_0(\mathbf{u}) & \text{if } f_i(\mathbf{u}) \leq 0,\ i = 1, \ldots, m \\ +\infty & \text{otherwise.} \end{cases} \tag{11.8}$$

So, the function $h(\mathbf{u})$ coincides with the objective $f_0(\mathbf{u})$ in regions where the constraints $f_i(\mathbf{u}) \leq 0,\ i = 1, \ldots, m$ are satisfied and $h(\mathbf{u}) = +\infty$ in infeasible regions. In other words, h acts as a "barrier" of the feasible set of the primal problem. Thus we can as well use $h(\mathbf{u})$ as objective function and rewrite the original primal problem (11.7) as an *unconstrained* optimization problem:

$$p^* = \min_{\mathbf{u}} \max_{\boldsymbol{\lambda} \geq 0} \mathcal{L}(\mathbf{u}, \boldsymbol{\lambda}). \tag{11.9}$$

The notion of weak duality amounts to exchanging the "min" and "max" operators in the above formulation, resulting in a lower bound on the optimal value of the primal problem. Strong duality refers to the case where this exchange can be done without altering the value of the result: the lower bound is actually equal to the optimal value p^*. While weak duality always holds, even if the primal problem (11.9) is not convex, strong duality may not hold. However, for a large class of generic convex problems, strong duality holds.

Lemma 11.5 (Weak duality) *For all functions* $f_0, f_1, \ldots, f_m$ *in (11.7), not necessarily convex, we can exchange the* max *and the* min *and get a lower bound on* p^**:*

$$d^* = \max_{\boldsymbol{\lambda} \geq 0} \min_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \boldsymbol{\lambda}) \leq \min_{\mathbf{u}} \max_{\boldsymbol{\lambda} \geq 0} \mathcal{L}(\mathbf{u}, \boldsymbol{\lambda}) = p^*.$$

The objective function of the maximization problem is now called the (Lagrange) dual function.

Definition 11.6 (Lagrange dual function) *The* (Lagrange) dual function $g : \mathbb{R}^m \to \mathbb{R}$ *is defined as*

$$\begin{aligned} g(\boldsymbol{\lambda}) &= \min_{\mathbf{u}} \ \mathcal{L}(\mathbf{u}, \boldsymbol{\lambda}) \\ &= \min_{\mathbf{u}} \ f_0(\mathbf{u}) + \lambda_1 f_1(\mathbf{u}) + \ldots + \lambda_m f_m(\mathbf{u}). \end{aligned} \tag{11.10}$$

Furthermore $g(\boldsymbol{\lambda})$ *is concave, even if the* $f_i(\mathbf{u})$ *are not convex.*

The concavity can easily be seen by considering first that for a given $\mathbf{u}$, $\mathcal{L}(\mathbf{u}, \boldsymbol{\lambda})$ is an affine function of $\boldsymbol{\lambda}$ and hence is a concave function. Since $g(\boldsymbol{\lambda})$ is the pointwise minimum of such concave functions, it is concave.

Definition 11.7 (Lagrange dual problem) *The* Lagrange dual problem *is defined as*

$$d^* = \max_{\boldsymbol{\lambda} \geq 0} g(\boldsymbol{\lambda}).$$

Since $g(\boldsymbol{\lambda})$ is concave, this will always be a convex optimization problem, even if the primal is not. By *weak duality*, we always have $d^* \leq p^*$, even for nonconvex problems. The value $p^* - d^*$ is called the duality gap. For *convex* problems, we usually (although not always) have *strong duality* at the optimum, i.e.,

$$d^* = p^*,$$

which is also referred to as a *zero duality gap*. For convex problems, a sufficient condition for zero duality gap is provided by *Slater's condition*:

Lemma 11.8 (Slater's condition) *If the primal problem (11.7) is convex and is* strictly feasible, *i.e.,* $\exists \ \mathbf{u}_0 \ : \ f_i(\mathbf{u}_0) < 0, \ i = 1, \ldots, m$, *then*

$$p^* = d^*.$$

11.4.3 SDP Duality and Optimality Conditions

Consider for simplicity the case of an SDP with a single LMI constraint, and no affine equalities:

$$p^* = \min_{\mathbf{u}} \ \mathbf{c}^T \mathbf{u} \text{ subject to } F(\mathbf{u}) = F_0 + u_1 F_1 + \ldots u_q F_q \preceq 0. \tag{11.11}$$

The general case of multiple LMI constraints and affine equalities can be handled by elimination of the latter and using block-diagonal matrices to represent the former as a single LMI.

The classic Lagrange duality theory outlined in the previous section does not directly apply here, since we are not dealing with finitely many constraints in

scalar form; as noted earlier, the LMI constraint involves an infinite number of such constraints, of the form (11.3). One way to handle such constraints is to introduce a Lagrangian of the form

$$\mathcal{L}(\mathbf{u}, Z) = \mathbf{c}^T\mathbf{u} + \text{trace}(ZF(\mathbf{u})),$$

where the dual variable Z is now a symmetric matrix, of same size as $F(\mathbf{u})$. We can check that such a Lagrange function fulfills the same role assigned to the function defined in definition 11.4 for the case with scalar constraints. Indeed, if we define $h(\mathbf{u}) = \max_{Z\succeq 0}\mathcal{L}(\mathbf{u}, Z)$, then

$$h(\mathbf{u}) = \max_{Z\succeq 0}\mathcal{L}(\mathbf{u}, Z) = \begin{cases} \mathbf{c}^T\mathbf{u} & \text{if } F(\mathbf{u}) \preceq 0, \\ +\infty & \text{otherwise.} \end{cases} \tag{11.12}$$

Thus, $h(\mathbf{u})$ is a barrier for the primal SDP (11.11), that is, it coincides with the objective of (11.11) on its feasible set, and is infinite otherwise. Notice that to the LMI constraint we now associate a multiplier *matrix*, which will be constrained to the positive semidefinite cone.

In the above, we made use of the fact that, for a given symmetric matrix F,

$$\Theta(F) := \sup_{Z\succeq 0} \text{trace}(ZF)$$

is $+\infty$ if F has a positive eigenvalue, and zero if F is negative semidefinite. This property is obvious for diagonal matrices, since in that case the variable Z can be constrained to be diagonal without loss of generality. The general case follows from the fact that if F has the eigenvalue decomposition $F = U\Lambda U^T$, where Λ is a diagonal matrix containing the eigenvalues of F, and U is orthogonal, then $\text{trace}(ZF) = \text{trace}(Z'\Lambda)$, where $Z' = U^TZU$ spans the positive semidefinite cone whenever Z does.

Using the above Lagrangian, one can cast the original problem (11.11) as an unconstrained optimization problem:

$$p^* = \min_{\mathbf{u}} \max_{Z\succeq 0} \mathcal{L}(\mathbf{u}, Z).$$

By weak duality, we obtain a lower bound on p^* by exchanging the min and max:

$$d^* = \max_{Z\succeq 0} \min_{\mathbf{u}} \mathcal{L}(\mathbf{u}, Z) \leq \min_{\mathbf{u}} \max_{Z\succeq 0} \mathcal{L}(\mathbf{u}, Z) = p^*.$$

The inner minimization problem is easily solved analytically, due to the special structure of the SDP. We obtain a closed form for the (Lagrange) dual function:

$$\begin{aligned} g(Z) = \min_{\mathbf{u}} \mathcal{L}(\mathbf{u}, Z) &= \min_{\mathbf{u}} \mathbf{c}^T\mathbf{u} + \text{trace}(ZF_0) + \sum_{i=1}^{q} u_i \,\text{trace}(ZF_i) \\ &= \begin{cases} \text{trace}(ZF_0) & \text{if } c_i = -\text{trace}(ZF_i),\ i = 1, \ldots, q \\ -\infty & \text{otherwise.} \end{cases} \end{aligned}$$

The dual problem can be explicitly stated as follows:

$$\begin{aligned} d^* &= \max_{Z \succeq 0} \min_{\mathbf{u}} \mathcal{L}(\mathbf{u}, Z) \\ &= \max_{Z} \operatorname{trace}(ZF_0) \;\text{ subject to }\; Z \succeq 0,\; c_i = -\operatorname{trace}(ZF_i),\; i = 1, \ldots, q. \end{aligned} \tag{11.13}$$

We observe that the above problem is an SDP, with a single LMI constraint and q affine equalities in the matrix dual variable Z.

While weak duality always holds, strong duality may not, even for SDPs. Not surprisingly, a Slater-type condition ensures strong duality. Precisely, if the primal SDP (11.11) is strictly feasible, that is, there exist a $\mathbf{u}_0$ such that $F(\mathbf{u}_0) \prec 0$, then $p^* = d^*$. If, in addition, the dual problem is also strictly feasible, meaning that there exist $Z \succ 0$ such that $c_i = \operatorname{trace}(ZF_i)$, $i = 1, \ldots, q$, then both primal and dual optimal values are attained by some optimal pair $(\mathbf{u}^*, Z^*)$. In that case, we can characterize such optimal pairs as follows. In view of the equality constraints of the dual problem, the duality gap can be expressed as

$$\begin{aligned} p^* - d^* &= \mathbf{c}^T \mathbf{u}^* - \operatorname{trace}(Z^* F_0) \\ &= -\operatorname{trace}(Z^* F(\mathbf{u}^*)). \end{aligned}$$

A zero duality gap is equivalent to $\operatorname{trace}(Z^* F(\mathbf{u}^*)) = 0$, which in turn is equivalent to $Z^* F(\mathbf{u}^*) = O$, where O denotes the zero matrix, since the product of a positive semidefinite and a negative semidefinite matrix has zero trace iff it is zero.

To summarize, consider the SDP (11.11) and its Lagrange dual (11.13). If either problem is strictly feasible, then they share the same optimal value. If both problems are strictly feasible, then the optimal values of both problems are attained and coincide. In this case, a primal-dual pair $(\mathbf{u}^*, Z^*)$ is optimal iff

$$\begin{aligned} &F(\mathbf{u}^*) \preceq 0, \\ &Z^* \succeq 0, \\ &c_i = -\operatorname{trace}(Z^* F_i), \quad i = 1, \ldots, q, \\ &Z^* F(\mathbf{u}^*) = O. \end{aligned}$$

The above conditions represent the expression of the general Karush-Kuhn-Tucker (KKT) conditions in the SDP setting. The first three sets of conditions express that $\mathbf{u}^*$ and Z^* are feasible for their respective problems; the last condition expresses a complementarity condition.

For a pair of strictly feasible primal-dual SDPs, solving the primal minimization problem is equivalent to maximizing the dual problem and both can thus be considered simultaneously. Algorithms indeed make use of this relationship and use the duality gap as a stopping criterion. A general-purpose program such as SeDuMi (Sturm, 1999) handles those problems efficiently. This code uses interior-point methods for SDP (Nesterov and Nemirovsky, 1994); these methods have a worst-case complexity of $O(q^2 p^{2.5})$ for the general problem (11.11). In practice, problem structure can be exploited for great computational savings: for example, when $F(\mathbf{u}) \in \mathbb{R}^{p \times p}$ consists of L diagonal blocks of size p_i, $i = 1, \ldots, L$, these

methods have a worst-case complexity of $O(q^2(\sum_{i=1}^{L} p_i^2)p^{0.5})$ (Vandenberghe and Boyd, 1996).

11.5 Kernel Methods for Data Fusion

Given multiple related data sets (e.g., gene expression, protein sequence, and protein-protein interaction data), each kernel function produces, for the yeast genome, a square matrix in which each entry encodes a particular notion of similarity of one yeast protein to another. Implicitly, each matrix also defines an embedding of the proteins in a feature space. Thus, the kernel representation casts heterogeneous data—variable-length amino acid strings, real-valued gene expression data, a graph of protein-protein interactions—into the common format of kernel matrices.

The kernel formalism also allows these various matrices to be combined. Basic algebraic operations such as addition, multiplication, and exponentiation preserve the key property of positive definiteness, and thus allow a simple but powerful algebra of kernels (Berg et al., 1984). For example, given two kernels K_1 and K_2, inducing the embeddings $\Phi_1(\mathbf{x})$ and $\Phi_2(\mathbf{x})$, respectively, it is possible to define the kernel $K = K_1 + K_2$, inducing the embedding $\Phi(\mathbf{x}) = [\Phi_1(\mathbf{x}), \Phi_2(\mathbf{x})]$. Of even greater interest, we can consider parameterized combinations of kernels. In this chapter, given a set of kernels $\mathcal{K} = \{K_1, K_2, \ldots, K_m\}$, we will form the linear combination

$$K = \sum_{i=1}^{m} \mu_i K_i. \tag{11.14}$$

As we have discussed, fitting an SVM to a single data source involves solving the quadratic program (11.2) based on the kernel matrix and the labels. It is possible to extend this optimization problem not only to find optimal discriminant boundaries but also to find optimal values of the coefficients μ_i in (11.14) for problems involving multiple kernels (Lanckriet et al., 2002). In the case of the 1-norm soft margin SVM, we want to minimize the same cost function (11.1), now with respect to both the discriminant boundary and the μ_i. Since the primal problem (11.1) is convex and strictly feasible, strong duality holds for (11.1) and (11.2) according to lemma 11.8:

$$\begin{aligned} \omega_{S1}(K) &= \mathbf{w}_*^T \mathbf{w}_* + C \sum_{i=1}^{n} \xi_{i,*} \\ &= \max_{\boldsymbol{\alpha}} \; 2\boldsymbol{\alpha}^T \mathbf{e} - \boldsymbol{\alpha}^T \mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y}) \boldsymbol{\alpha} \;:\; C \geq \boldsymbol{\alpha} \geq 0, \;\; \boldsymbol{\alpha}^T \mathbf{y} = 0. \end{aligned} \tag{11.15}$$

where $\mathbf{e}$ is a vector containing ones and $\mathbf{w}_*$ and $\xi_{i,*}$ the optimal values of the primal variables $\mathbf{w}$ and ξ_i. Training an SVM for a given kernel $K \succeq 0$ yields the minimal value (11.15) of (11.1) which is obviously a function of the particular choice of K, as is expressed explicitly in (11.15) as a dual problem. Let us now optimize this quantity with respect to the kernel matrix $K = \sum_{i=1}^{m} \mu_i K_i$, that is, let us try to

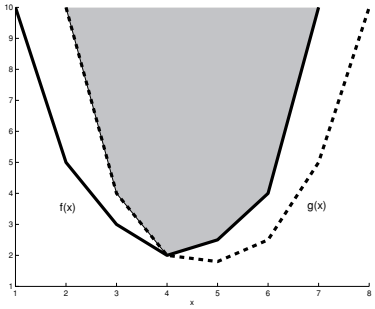


Figure 11.1 Given two convex functions $f(x)$ and $g(x)$, their pointwise maximum $\max\{f(x), g(x)\}$ will also be convex, as can easily be seen from the convexity of the shaded area (called the epigraph).

find the weights $\boldsymbol{\mu} \in \mathbb{R}^m$ for which the corresponding embedding shows minimal $\omega_{S1}(K)$, keeping the trace of K constant:

$$\min_{\boldsymbol{\mu}\in\mathbb{R}^m, K\succeq 0} \omega_{S1}(K) \qquad \text{s.t. trace}(K) = c, \quad K = \sum_{i=1}^{m} \mu_i K_i, \tag{11.16}$$

where c is a constant. Note that the constraint $K \succeq 0$ emerges very naturally because the optimal kernel matrix must indeed come from the cone of positive definite matrices. We first notice a fundamental property of the quantity $\omega_{S1}(K)$, a property that is crucial for the remainder of this discussion.

Proposition 11.9 *The quantity*

$$\omega_{S1}(K) = \max_{\boldsymbol{\alpha}} \; 2\boldsymbol{\alpha}^T\mathbf{e} - \boldsymbol{\alpha}^T \mathit{diag}(\mathbf{y}) K \mathit{diag}(\mathbf{y})\boldsymbol{\alpha} \;:\; C \geq \boldsymbol{\alpha} \geq 0, \;\; \boldsymbol{\alpha}^T\mathbf{y} = 0,$$

is convex in K.

This is easily seen by considering first that $2\boldsymbol{\alpha}^T\mathbf{e} - \boldsymbol{\alpha}^T\text{diag}(\mathbf{y})K\text{diag}(\mathbf{y}))\boldsymbol{\alpha}$ is an affine function of (the entries of) K, and hence is a convex function as well. Secondly, we notice that $\omega_{S1}(K)$ is the pointwise maximum of such convex functions and is thus convex. This last statement is illustrated in a discrete case in figure 11.1. It shows how the pointwise maximum of two functions is convex. This can be extended for an infinite set of functions, for example, indexed by $\boldsymbol{\alpha}$ in this case.

Problem (11.16) is now a convex optimization problem. The following theorem shows that, for $K = \sum_{i=1}^{m} \mu_i K_i$, this problem can be cast as an SDP:

Theorem 11.10 *Given a labeled sample $S_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ with corresponding set of labels $\mathbf{y} \in \mathbb{R}^n$ and a set of kernel matrices $\{K_i\}_{i=1}^m$, the kernel*

matrix $K = \sum_{i=1}^m \mu_i K_i$ *that optimizes (11.16) can be found by solving the following convex optimization problem which is an SDP:*

$$\begin{aligned} \min_{\boldsymbol{\mu},t,\lambda,\boldsymbol{\nu},\boldsymbol{\delta}} \quad & t \\ \textit{subject to} \quad & \textit{trace}\left(\sum_{i=1}^m \mu_i K_i\right) = c, \\ & \sum_{i=1}^m \mu_i K_i \succeq 0, \\ & \begin{pmatrix} \textit{diag}(\mathbf{y})(\sum_{i=1}^m \mu_i K_i)\textit{diag}(\mathbf{y}) & \mathbf{e}+\boldsymbol{\nu}-\boldsymbol{\delta}+\lambda\mathbf{y} \\ (\mathbf{e}+\boldsymbol{\nu}-\boldsymbol{\delta}+\lambda\mathbf{y})^T & t - 2C\boldsymbol{\delta}^T\mathbf{e} \end{pmatrix} \succeq 0, \\ & \boldsymbol{\nu},\boldsymbol{\delta} \geq 0. \end{aligned} \tag{11.17}$$

Proof After substitution of $\omega_{S1}(K)$ as defined in (11.15), (11.16) becomes

$$\begin{aligned} \min_{\boldsymbol{\mu}\in\mathbb{R}^m, K\succeq 0} \max_{\boldsymbol{\alpha}} \quad & 2\boldsymbol{\alpha}^T\mathbf{e} - \boldsymbol{\alpha}^T \mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y})\boldsymbol{\alpha} \\ \text{subject to} \quad & C \geq \boldsymbol{\alpha} \geq 0, \;\; \boldsymbol{\alpha}^T\mathbf{y} = 0, \;\; \mathrm{trace}(K) = c, \;\; K = \sum_{i=1}^m \mu_i K_i, \end{aligned} \tag{11.18}$$

with c a constant. Assume that $K \succ 0$, hence $\mathrm{diag}(\mathbf{y})K\mathrm{diag}(\mathbf{y}) \succ 0$ (the following can be extended to the general semidefinite case). We note that $\omega_{S1}(K)$ is convex in K (because of proposition 11.9) and thus in $\boldsymbol{\mu}$, since K is a linear function of $\boldsymbol{\mu}$. Given the convex constraints in (11.18), the optimization problem is thus certainly convex in $\boldsymbol{\mu}$. We write this as

$$\begin{aligned} \min_{\boldsymbol{\mu}\in\mathbb{R}^m, K\succeq 0, t} t \;:\; & t \geq \max_{\boldsymbol{\alpha}} \; 2\boldsymbol{\alpha}^T\mathbf{e} - \boldsymbol{\alpha}^T \mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y})\boldsymbol{\alpha}, \\ & C \geq \boldsymbol{\alpha} \geq 0, \;\; \boldsymbol{\alpha}^T\mathbf{y} = 0, \;\; \mathrm{trace}(K) = c, \;\; K = \sum_{i=1}^m \mu_i K_i. \end{aligned} \tag{11.19}$$

We will now express $t \geq \max_{\boldsymbol{\alpha}} \; 2\boldsymbol{\alpha}^T\mathbf{e} - \boldsymbol{\alpha}^T \mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y})\boldsymbol{\alpha}$ as an LMI using duality. In particular, we express the constraint using the dual minimization problem. This will allow us to drop the minimization and use the Schur complement lemma to obtain an LMI. We explain this now in more detail.

Define the Lagrangian of the maximization problem (11.2) by

$$\mathcal{L}(\boldsymbol{\alpha},\boldsymbol{\nu},\lambda,\boldsymbol{\delta}) = 2\boldsymbol{\alpha}^T\mathbf{e} - \boldsymbol{\alpha}^T \mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y})\boldsymbol{\alpha} + 2\boldsymbol{\nu}^T\boldsymbol{\alpha} + 2\lambda\mathbf{y}^T\boldsymbol{\alpha} + 2\boldsymbol{\delta}^T(C\mathbf{e} - \boldsymbol{\alpha}),$$

where $\lambda \in \mathbb{R}$ and $\boldsymbol{\nu},\boldsymbol{\delta} \in \mathbb{R}^n$. By duality, we have

$$\omega_{S1}(K) = \max_{\boldsymbol{\alpha}} \min_{\boldsymbol{\nu}\geq 0, \boldsymbol{\delta}\geq 0, \lambda} \mathcal{L}(\boldsymbol{\alpha},\boldsymbol{\nu},\lambda,\boldsymbol{\delta}) = \min_{\boldsymbol{\nu}\geq 0, \boldsymbol{\delta}\geq 0, \lambda} \max_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha},\boldsymbol{\nu},\lambda,\boldsymbol{\delta}),$$

where $\boldsymbol{\nu} \geq 0 \Leftrightarrow \nu_i \geq 0$ for $i = 1, \ldots, n$, similarly for $\boldsymbol{\delta}$. Since $\mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y}) \succ 0$, at the optimum, we have

$$\boldsymbol{\alpha} = \left(\mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y})\right)^{-1} (\mathbf{e} + \boldsymbol{\nu} - \boldsymbol{\delta} + \lambda \mathbf{y}),$$

and can form the dual problem

$$\omega_{S1}(K) = \min_{\boldsymbol{\nu} \geq 0,\, \boldsymbol{\delta} \geq 0,\, \lambda} (\mathbf{e} + \boldsymbol{\nu} - \boldsymbol{\delta} + \lambda \mathbf{y})^T \left(\mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y})\right)^{-1} (\mathbf{e} + \boldsymbol{\nu} - \boldsymbol{\delta} + \lambda \mathbf{y}) + 2C\boldsymbol{\delta}^T \mathbf{e}$$

We obtain that for any $t > 0$, the constraint $\omega_{S1}(K) \leq t$ is true iff there exist $\boldsymbol{\nu} \geq 0$, $\boldsymbol{\delta} > 0$ and λ such that

$$(\mathbf{e} + \boldsymbol{\nu} - \boldsymbol{\delta} + \lambda \mathbf{y})^T \left(\mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y})\right)^{-1} (\mathbf{e} + \boldsymbol{\nu} - \boldsymbol{\delta} + \lambda \mathbf{y}) + 2C\boldsymbol{\delta}^T \mathbf{e} \leq t,$$

or, equivalently (using the Schur complement lemma), such that

$$\begin{pmatrix} \mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y}) & \mathbf{e} + \boldsymbol{\nu} - \boldsymbol{\delta} + \lambda \mathbf{y} \\ (\mathbf{e} + \boldsymbol{\nu} - \boldsymbol{\delta} + \lambda \mathbf{y})^T & t - 2C\boldsymbol{\delta}^T \mathbf{e} \end{pmatrix} \succeq 0$$

holds. Taking this into account, (11.19) can be expressed as

$$\begin{aligned} \min_{\boldsymbol{\mu} \in \mathbb{R}^m, K, t, \lambda, \boldsymbol{\nu}, \boldsymbol{\delta}} \quad & t \\ \text{subject to} \quad & \mathrm{trace}(K) = c, \\ & K = \sum_{i=1}^{m} \mu_i K_i \succeq 0, \\ & \begin{pmatrix} \mathrm{diag}(\mathbf{y}) K \mathrm{diag}(\mathbf{y}) & \mathbf{e} + \boldsymbol{\nu} - \boldsymbol{\delta} + \lambda \mathbf{y} \\ (\mathbf{e} + \boldsymbol{\nu} - \boldsymbol{\delta} + \lambda \mathbf{y})^T & t - 2C\boldsymbol{\delta}^T \mathbf{e} \end{pmatrix} \succeq 0, \\ & \boldsymbol{\nu} \geq 0, \\ & \boldsymbol{\delta} \geq 0, \end{aligned} \tag{11.20}$$

which yields (11.17) after substituting $K = \sum_{i=1}^{m} \mu_i K_i$ to eliminate K. Notice that $\boldsymbol{\nu} \geq 0 \Leftrightarrow \mathrm{diag}(\boldsymbol{\nu}) \succeq 0$, and thus an LMI; similarly for $\boldsymbol{\delta} \geq 0$. ■

Notice that the optimization problem (11.17) is an SDP in the standard form (11.4). This leads to a general method for learning the optimal combination of kernel matrices as an SDP problem, which can be solved via efficient interior-point algorithms (Vandenberghe and Boyd, 1996). Although efficient, these algorithms will still have a worst-case complexity $O(n^{4.5})$ in this particular case, according to the complexity results mentioned in subsection 11.4.3.

In this discussion, the K_i are positive definite by construction; thus $K \succeq 0$ is automatically satisfied if the weights μ_i are constrained to be non-negative. We will now point out an additional advantage of the restriction $\boldsymbol{\mu} \geq 0$: it will allow us to cast the SDP (11.17) as a *quadratically constrained quadratic program*, which has beneficial computational effects by lowering the efficiency of the computation to $O(n^3)$ in terms of the number of data points. Also, the constraint can result in

improved numerical stability—it prevents the algorithm from using large weights with opposite sign that cancel. Finally, Lanckriet et al. (2002) show that the constraint also yields better estimates of the generalization performance of these algorithms.

Solving the original learning problem (11.16) subject to the extra constraint $\boldsymbol{\mu} \geq 0$ yields

$$\begin{aligned} \min_{\boldsymbol{\mu} \in \mathbb{R}^m, K} \max_{\boldsymbol{\alpha} \,:\, C \geq \boldsymbol{\alpha} \geq 0, \boldsymbol{\alpha}^T \mathbf{y} = 0} \quad & 2\boldsymbol{\alpha}^T \mathbf{e} - \boldsymbol{\alpha}^T \operatorname{diag}(\mathbf{y}) K \operatorname{diag}(\mathbf{y}) \boldsymbol{\alpha} \\ \text{subject to} \quad & \operatorname{trace}(K) = c, \\ & K \succeq 0, \\ & K = \sum_{i=1}^{m} \mu_i K_i, \\ & \boldsymbol{\mu} \geq 0, \end{aligned}$$

when $\omega_{S1}(K)$ is expressed using (11.15). We can omit the second constraint, because this is implied by the last two constraints, since $K_i \succeq 0$. If we let $\operatorname{trace}(K_i) = r_i$, where $\mathbf{r} \in \mathbb{R}^m$, the problem reduces to

$$\begin{aligned} \min_{\boldsymbol{\mu}} \max_{\boldsymbol{\alpha} \,:\, C \geq \boldsymbol{\alpha} \geq 0, \boldsymbol{\alpha}^T \mathbf{y} = 0} \quad & 2\boldsymbol{\alpha}^T \mathbf{e} - \boldsymbol{\alpha}^T \operatorname{diag}(\mathbf{y}) \left(\sum_{i=1}^{m} \mu_i K_i \right) \operatorname{diag}(\mathbf{y}) \boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{\mu}^T \mathbf{r} = c, \\ & \boldsymbol{\mu} \geq 0. \end{aligned}$$

We can write this as

$$\begin{aligned} & \min_{\boldsymbol{\mu} \,:\, \boldsymbol{\mu} \geq 0, \boldsymbol{\mu}^T \mathbf{r} = c} \quad \max_{\boldsymbol{\alpha} \,:\, C \geq \boldsymbol{\alpha} \geq 0, \boldsymbol{\alpha}^T \mathbf{y} = 0} 2\boldsymbol{\alpha}^T \mathbf{e} - \boldsymbol{\alpha}^T \operatorname{diag}(\mathbf{y}) \left(\sum_{i=1}^{m} \mu_i K_i \right) \operatorname{diag}(\mathbf{y}) \boldsymbol{\alpha} \\ = & \min_{\boldsymbol{\mu} \,:\, \boldsymbol{\mu} \geq 0, \boldsymbol{\mu}^T \mathbf{r} = c} \quad \max_{\boldsymbol{\alpha} \,:\, C \geq \boldsymbol{\alpha} \geq 0, \boldsymbol{\alpha}^T \mathbf{y} = 0} 2\boldsymbol{\alpha}^T \mathbf{e} - \sum_{i=1}^{m} \mu_i \boldsymbol{\alpha}^T \operatorname{diag}(\mathbf{y}) K_i \operatorname{diag}(\mathbf{y}) \boldsymbol{\alpha} \\ = & \max_{\boldsymbol{\alpha} \,:\, C \geq \boldsymbol{\alpha} \geq 0, \boldsymbol{\alpha}^T \mathbf{y} = 0} \quad \min_{\boldsymbol{\mu} \,:\, \boldsymbol{\mu} \geq 0, \boldsymbol{\mu}^T \mathbf{r} = c} 2\boldsymbol{\alpha}^T \mathbf{e} - \sum_{i=1}^{m} \mu_i \boldsymbol{\alpha}^T \operatorname{diag}(\mathbf{y}) K_i \operatorname{diag}(\mathbf{y}) \boldsymbol{\alpha}, \end{aligned}$$

The interchange of the order of the minimization and the maximization is justified by standard results in convex optimization (see, e.g., Boyd and Vandenberghe, 2001) since the objective is convex in $\boldsymbol{\mu}$ (it is linear in $\boldsymbol{\mu}$) and concave in $\boldsymbol{\alpha}$, and because the minimization problem is strictly feasible in $\boldsymbol{\mu}$ and the maximization problem strictly feasible in $\boldsymbol{\alpha}$. We thus obtain

$$\begin{aligned} & \max_{\boldsymbol{\alpha} \,:\, C \geq \boldsymbol{\alpha} \geq 0, \boldsymbol{\alpha}^T \mathbf{y} = 0} \quad \min_{\boldsymbol{\mu} \,:\, \boldsymbol{\mu} \geq 0, \boldsymbol{\mu}^T \mathbf{r} = c} 2\boldsymbol{\alpha}^T \mathbf{e} - \sum_{i=1}^{m} \mu_i \boldsymbol{\alpha}^T \operatorname{diag}(\mathbf{y}) K_i \operatorname{diag}(\mathbf{y}) \boldsymbol{\alpha} \\ = & \max_{\boldsymbol{\alpha} \,:\, C \geq \boldsymbol{\alpha} \geq 0, \boldsymbol{\alpha}^T \mathbf{y} = 0} \left[2\boldsymbol{\alpha}^T \mathbf{e} - \max_{\boldsymbol{\mu} \,:\, \boldsymbol{\mu} \geq 0, \boldsymbol{\mu}^T \mathbf{r} = c} \left(\sum_{i=1}^{m} \mu_i \boldsymbol{\alpha}^T \operatorname{diag}(\mathbf{y}) K_i \operatorname{diag}(\mathbf{y}) \boldsymbol{\alpha} \right) \right] \\ = & \max_{\boldsymbol{\alpha} \,:\, C \geq \boldsymbol{\alpha} \geq 0, \boldsymbol{\alpha}^T \mathbf{y} = 0} \left[2\boldsymbol{\alpha}^T \mathbf{e} - \max_i \left(\frac{c}{r_i} \boldsymbol{\alpha}^T \operatorname{diag}(\mathbf{y}) K_i \operatorname{diag}(\mathbf{y}) \boldsymbol{\alpha} \right) \right]. \end{aligned}$$

Finally, this can be reformulated as follows:

$$\begin{aligned} \max_{\boldsymbol{\alpha},t} \quad & 2\boldsymbol{\alpha}^T\mathbf{e} - ct \\ \text{subject to} \quad & t \geq \frac{1}{r_i}\boldsymbol{\alpha}^T \text{diag}(\mathbf{y}) K_i \text{diag}(\mathbf{y})\boldsymbol{\alpha}, \quad i = 1, \ldots, m \\ & \boldsymbol{\alpha}^T\mathbf{y} = 0, \\ & C \geq \boldsymbol{\alpha} \geq 0, \end{aligned} \tag{11.21}$$

or, when the K_i are normalized ($[K_i]_{jj} = 1$, $j = 1, \ldots, n$, such that $r_i = n$):

$$\begin{aligned} \max_{\boldsymbol{\alpha},t} \quad & 2\boldsymbol{\alpha}^T\mathbf{e} - ct \\ \text{subject to} \quad & t \geq \frac{1}{n}\boldsymbol{\alpha}^T \text{diag}(\mathbf{y}) K_i \text{diag}(\mathbf{y})\boldsymbol{\alpha}, \quad i = 1, \ldots, m \\ & \boldsymbol{\alpha}^T\mathbf{y} = 0, \\ & C \geq \boldsymbol{\alpha} \geq 0. \end{aligned} \tag{11.22}$$

This problem is a convex optimization problem, more precisely a *quadratically constrained quadratic program* (Boyd and Vandenberghe, 2001). Thus, the SDP (11.17) can be cast as a QCQP, which improves the efficiency of the computation to $O(n^3)$ in terms of the number of data points. The optimal weights μ_i, $i = 1, \ldots, m$, can be recovered from the primal-dual solution found by standard software such as SeDuMi (Sturm, 1999).

Thus, by solving a QCQP, we are able to find an adaptive combination of kernel matrices—and thus an adaptive combination of heterogeneous information sources—that solves our classification problem. The output of our procedure is a set of weights μ_i and a discriminant function based on these weights. We obtain a classification decision that merges information encoded in the various kernel matrices, and we obtain weights μ_i that reflect the relative importance of these information sources.

11.6 Two Biological Applications

In this section, we illustrate the kernel-based approach for fusing heterogeneous genomic data using SDP for two biologically important problems: membrane protein prediction and protein function prediction in yeast. More details can be found in Lanckriet et al. (2003) for membrane protein recognition, and in Lanckriet et al. (2004) for the protein function classification.

11.6.1 Membrane Protein Classification

Membrane protein

Membrane proteins are proteins that anchor in one of various membranes in the cell. Many membrane proteins serve important communicative functions. Generally, each membrane protein passes through the membrane several times. The transmembrane

Table 11.1 Kernel functions. The table lists the seven kernels used to compare proteins, the data on which they are defined, and the method for computing similarities. The final kernel, K_{RND}, is included as a control. All kernels matrices, along with the data from which they were generated, are available from `noble.gs.washington.edu/sdp-svm`.

Kernel	Data	Similarity measure
K_{SW}	protein sequences	Smith-Waterman
K_B	protein sequences	BLAST
K_{HMM}	protein sequences	Pfam HMM
K_{FFT}	hydropathy profile	FFT
K_{LI}	protein interactions	linear kernel
K_D	protein interactions	diffusion kernel
K_E	gene expression	radial basis kernel
K_{RND}	random numbers	radial basis kernel

regions of the amino acid sequence are typically hydrophobic, whereas the nonmembrane portions are hydrophilic. This specific hydrophobicity profile of the protein allows it to anchor itself in the cell membrane.

Hydrophathy profile

Because the hydrophobicity profile of a membrane protein is critical to its function, this profile is better conserved in evolution than the specific amino acid sequence. Therefore, classic methods for determining whether a protein spans a membrane (Chen and Rost, 2002) depend upon a *hydropathy profile*, which plots the hydrophobicity of the amino acids along the protein (Engleman et al., 1986; Black and Mould, 1991; Hopp and Woods, 1981). In this subsection, we build on these classic methods by developing a kernel function that is based on the low-frequency alternation of hydrophobic and hydrophilic regions in membrane proteins. However, we also demonstrate that the hydropathy profile provides only partial evidence for transmembrane regions. Additional information is gleaned from sequence homology and from protein-protein interactions.

Note that, in general, membrane protein prediction consists in predicting the locations of multiple transmembrane regions within a single protein. In this example, however, for the purposes of demonstrating the SDP method, we focus on the binary prediction task of differentiating between membrane and nonmembrane proteins.

Kernels for Membrane Protein Prediction For the task of membrane protein classification we experiment with seven kernel matrices derived from three different types of data: four from the primary protein sequence, two from protein-protein interaction data, and one from mRNA expression data. These are summarized in table 11.1.

Pairwise comparison kernel

Protein Sequence: Smith-Waterman, BLAST, and Pfam HMM Kernels A homolog of a membrane protein is likely also to be located in the membrane. Therefore, we define three kernel matrices based upon standard homology detection methods. The first two sequence-based kernel matrices (K_{SW} and K_B) are gener-

ated using the BLAST (Altschul et al., 1990) and Smith-Waterman (SW) (Smith and Waterman, 1981) pairwise sequence comparison algorithms, as described previously (Liao and Noble, 2002). Because matrices of BLAST or SW scores are not necessarily positive definite, we represent each protein as a vector of scores (BLAST and SW log E-values, respectively) against all other proteins. Defining the similarity between proteins as the inner product between the score vectors (the so-called empirical kernel map, Tsuda, 1999) leads to a valid kernel matrix, one for the BLAST score and one for the SW score. Note that including in the comparison set proteins with unknown subcellular locations allows the kernel to exploit these unlabeled data. The third kernel matrix (K_{HMM}) is a generalization of the previous pairwise comparison-based matrices in which the pairwise comparison scores are replaced by expectation values derived from hidden Markov models (HMMs) in the Pfam database (Sonnhammer et al., 1997). These similarity measures are not specific to the membrane protein classification task.

Pfam kernel

FFT kernel

Protein Sequence: FFT Kernel In contrast, the fourth sequence-based kernel matrix (K_{FFT}) directly incorporates information about hydrophobicity patterns, which are known to be useful in identifying membrane proteins. The kernel uses hydropathy profiles generated from the Kyte-Doolittle index (Kyte and Doolittle, 1982). This kernel compares the frequency content of the hydropathy profiles of the two proteins. After prefiltering the hydropathy profiles, their Fourier transforms (describing the frequency content) are computed using a fast Fourier transform (FFT) algorithm. The frequency contents of different profiles are compared by applying a Gaussian kernel function, $k(\mathbf{x}_1, \mathbf{x}_2) = \exp(-||\mathbf{x}_1 - \mathbf{x}_2||^2/2\sigma)$ with width $\sigma = 10$, to the corresponding vectors of FFT values. This kernel detects periodicities in the hydropathy profile, a feature that is relevant to the identification of membrane proteins and complementary to the previous, homology-based kernels.

Protein Interactions: Linear and Diffusion Kernels We expect information about protein-protein interactions to be informative in this context for two reasons. First, hydrophobic molecules or regions of molecules tend to interact with each other. Second, transmembrane proteins are often involved in signaling pathways, and therefore different membrane proteins are likely to interact with a similar class of molecules upstream and downstream in these pathways (e.g., hormones upstream or kinases downstream). The two protein interaction kernels are generated using medium- and high-confidence interactions from a database of known interactions (von Mering et al., 2002). These interactions can be represented as an interaction matrix, in which rows and columns correspond to proteins, and binary entries indicate whether the two proteins interact.

The first interaction kernel matrix (K_{LI}) is comprised of linear interactions, that is, inner products of rows and columns from the centered, binary interaction matrix. The more similar the interaction pattern (corresponding to a row or column from the interaction matrix) is for a pair of proteins, the larger the inner product will be.

Diffusion kernel

An alternative way to represent the same interaction data is to consider the proteins as nodes in a large graph. In this graph, two proteins are linked when they interact and otherwise not. Kondor and Lafferty (2002) propose a general method for establishing similarities between the nodes of a graph, based on a random walk on the graph. This method efficiently accounts for all possible paths connecting two nodes, and for the lengths of those paths. Nodes that are connected by shorter paths or by many paths are considered more similar. The resulting *diffusion kernel* generates the second interaction kernel matrix (K_D).

An appealing characteristic of the diffusion kernel is its ability, like the empirical kernel map, to exploit unlabeled data. In order to compute the diffusion kernel, a graph is constructed using all known protein-protein interactions, including interactions involving proteins whose subcellular locations are unknown. Therefore, the diffusion process includes interactions involving unlabeled proteins, even though the kernel matrix only contains entries for labeled proteins. This allows two labeled proteins to be considered close to one another if they both interact with an unlabeled protein.

Gene Expression: Radial Basis Kernel Finally, we also include a kernel constructed entirely from microarray gene expression measurements. A collection of 441 distinct experiments was downloaded from the Stanford Microarray Database (`genome-www.stanford.edu/microarray`). These data provide us with a 441-element expression vector characterizing each gene. A Gaussian kernel matrix (K_E) is computed from these vectors by applying a Gaussian kernel function with width $\sigma = 100$ to each pair of 441-element vectors, characterizing a pair of genes. Note that we do not expect that gene expression will be particularly useful for the membrane classification task. We do not need to make this decision a priori, however; as explained in the following section, our method is able to provide an a posteriori measure of how useful a data source is relative to the other sources of data. We thus include the expression kernel in our experiments to test this aspect of the method.

Experimental Design In order to test our kernel-based approach in the setting of membrane protein classification, we use as a gold standard the annotations provided by the Munich Information Center for Protein Sequences Comprehensive Yeast Genome Database (CYGD) (Mewes et al., 2000). The CYGD assigns subcellular locations to 2318 yeast proteins, of which 497 belong to various membrane protein classes. The remaining approximately 4000 yeast proteins have uncertain location and are therefore not used in these experiments.

The primary input to the classification algorithm is the collection of kernel matrices from table 11.1. Using the SDP techniques described above, we find an optimal combination of the seven kernel matrices, and the resulting matrix is used to train an SVM classifier.

For comparison with the SDP/SVM learning algorithm, we consider several classic biological methods that are commonly used to determine whether a Kyte-

Doolittle plot corresponds to a membrane protein, as well as a state-of-the-art technique using HMMs to predict transmembrane helices in proteins (Krogh et al., 2001; Chen and Rost, 2002). The first method relies on the observation that the average hydrophobicity of membrane proteins tends to be higher than that of nonmembrane proteins, because the transmembrane regions are more hydrophobic. We therefore define f_1 as the average hydrophobicity, normalized by the length of the protein. We will compare the classification performance of our statistical learning algorithm with this metric.

Clearly, however, f_1 is too simplistic. For example, protein regions that are not transmembrane only induce noise in f_1. Therefore, an alternative metric filters the hydrophobicity plot with a low-pass filter and then computes the number, the height, and the width of those peaks above a certain threshold (Chen and Rost, 2002). The filter is intended to smooth out periodic effects. We implement two such filters, choosing values for the filter order and the threshold based on Chen and Rost (2002). In particular, we define f_2 as the area under the 7th-order low-pass filtered Kyte-Doolittle plot and above a threshold value 2, normalized by the length of the protein. Similarly, f_3 is the corresponding area using a 20th-order filter and a threshold of 1.6.

Finally, the transmembrane HMM (TMHMM) web server (`www.cbs.dtu.dk/services/TMHMM`) is used to make predictions for each protein. In Krogh et al. (2001), transmembrane proteins are identified by TMHMM using three different metrics: the expected number of amino acids in transmembrane helices, the number of transmembrane helices predicted by the N-best algorithm, and the expected number of transmembrane helices. Only the first two of these metrics are provided in the TMHMM output. Accordingly, we produce two lists of proteins, ranked by the number of predicted transmembrane helices (T_{PH}) and by the expected number of residues in transmembrane helices (T_{ENR}).

Each algorithm's performance is measured by splitting the data into a training and test set in a ratio of 80:20. We report the receiver operating characteristic (ROC) score, which is the area under a curve that plots the true positive rate as a function of the false positive rate for differing classification thresholds (Hanley and McNeil, 1982; Gribskov and Robinson, 1996). The ROC score measures the overall quality of the ranking induced by the classifier, rather than the quality of a single point in that ranking. An ROC score of 0.5 corresponds to random guessing, and an ROC score of 1.0 implies that the algorithm succeeded in putting all of the positive examples before all of the negatives. Each experiment is repeated 30 times with different random splits in order to estimate the variance of the performance values.

Results We performed computational experiments which study the performance of the SDP/SVM approach as a function of the number of data sources, compare the performance of the method to classic biological methods and state-of-the-art techniques for membrane protein classification, and study the robustness of the method to the presence of noise.

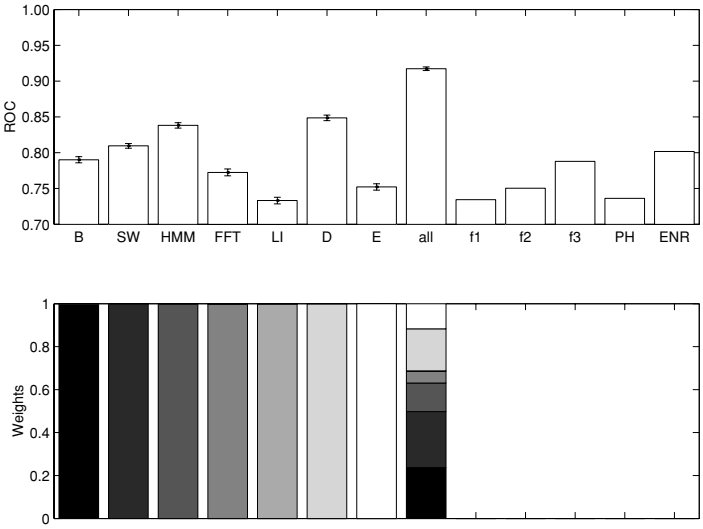


Figure 11.2 Combining data sets yields better classification performance. The height of each bar is proportional to the ROC score of the given membrane protein classification method. The bars labeled B to E and *all* correspond to SDP/SVM methods, the bars labeled f are hydropathy profile metrics, and the bars labeled PH and ENR refer to the TMHMM methods as defined in the text. Error bars indicate standard error across 30 random train/test splits. The heights of the gray level bars below each plot indicate the relative weight of the different kernel matrices in the optimal linear combination.

The results from the first three experiments are summarized in figure 11.2. The plot illustrates that SDP/SVM learns significantly better from the heterogeneous data than from any single data type. The mean ROC score using all seven kernel matrices (0.9174 ± 0.0025) is significantly higher than the best ROC score using only one matrix (0.8487 ± 0.0039 using the diffusion kernel). This improvement corresponds to a change in test set accuracy of 7.3%, from 81.3% to 88.6%.

As expected, the sequence-based kernels yield good individual performance. This is evident from the ROC scores. Furthermore, when all seven matrices are used at once, the SDP assigns relatively large weights to the sequence-based kernels. These weights are as follows: $\mu_B = 1.66$, $\mu_{SW} = 1.83$, $\mu_{HMM} = 0.93$, $\mu_{FFT} = 0.39$, $\mu_{LI} = 0.01$, $\mu_D = 1.37$ and $\mu_E = 0.82$ (note that for ease of interpretation, we scale the weights such that their sum is equal to the number m of kernel matrices). Thus, two of the three kernel matrices that receive weights larger than 1 are derived from the amino acid sequence. The Smith-Waterman kernel yields better results than the BLAST kernel, reflecting the fact that BLAST is a heuristic search

procedure, whereas the Smith-Waterman algorithm guarantees finding the optimal local alignment of two sequences.

The results also show that the interaction-based diffusion kernel is more informative than the expression kernel. Not only has the diffusion kernel an individual ROC score which is significantly higher than the expression kernel, the SDP also assigns a weight of 1.37 to the diffusion kernel, whereas the expression kernel receives a weight of 0.82. Accordingly, removing the diffusion kernel reduces the ROC score from 0.9174 to 0.8984, whereas removing the expression kernel has a smaller effect, leading to a ROC score of 0.9033. Further description of the results obtained when various subsets of kernels are used is provided in Lanckriet et al. (2003).

Figure 11.2 also compares the membrane protein classification performance of the SDP/SVM method with that of previously described techniques. The results confirm that using learning in this context dramatically improves the results relative to the simple hydropathy profile approach. Also, the SDP/SVM improves upon the performance of the TMHMM approach, even when the SVM algorithm uses only the sequence data K_{SW} or K_{HMM} (ROC of 0.8096 ± 0.0033 or 0.8382 ± 0.0038 vs. 0.8018, respectively).

While the SDP/SVM algorithm is a discriminative method that attempts to find a decision boundary that separates positive and negative instances of membrane proteins, the TMHMM is a generative method that simply attempts to model the membrane proteins. As an illustration of the difference, it is known that the TMHMM tends to yield false positives for sequences containing signal peptides—hydrophobic sequences in the N-terminal regions of proteins (Chen and Rost, 2002). The SDP/SVM approach tends to avoid these false positives, because signal peptides appear among the negative instances in the training set. Indeed, as shown in Lanckriet et al. (2003), signal peptides tend to be highly ranked by the TMHMM, and are more uniformly spread within the SDP/SVM rankings.

Finally, in order to test the robustness of our approach, a second experiment was performed in which a randomly generated kernel matrix K_{RND} was included among the kernel matrices used as input to our algorithm. This kernel matrix was generated by sampling 100-element vectors for each protein, where each component of each vector was sampled independently from a standard normal distribution, and then computing inner products of the 100-element vectors to form K_{RND}. A control classifier trained using only the random kernel yields an ROC score of 0.5, indicating that K_{RND} is indeed uninformative for the classification problem at hand. More important, when a classifier is trained using all seven real kernels plus K_{RND}, SDP assigns the random kernel a weight that is close to zero. Thus, the ROC score derived from seven matrices does not change when the random matrix is added, indicating that the method is robust to the presence of noisy, irrelevant data.

Table 11.2 Functional categories. The table lists the 13 CYGD functional classifications used in these experiments. The class listed as "others" is a combination of four smaller classes: (1) cellular communication/signal transduction mechanism, (2) protein activity regulation, (3) protein with binding function or cofactor requirement (structural or catalytic), and (4) transposable elements, viral and plasmid proteins.

	Category	Size
1	metabolism	1048
2	energy	242
3	cell cycle & DNA processing	600
4	transcription	753
5	protein synthesis	335
6	protein fate	578
7	cellular transport mechanisms & transport	479
8	cell rescue, defense, & virulence	264
9	interaction with cellular environment	193
10	cell fate	411
11	control of cellular organization	192
12	transport facilitation	306
13	others	81

11.6.2 Yeast Function Prediction

As a second test for our kernel-based approach, we follow the experimental paradigm of Deng et al. (2003a). The task is predicting functional classifications associated with yeast proteins, and we use as a gold standard the functional catalog provided by the MIPS Comprehensive Yeast Genome Database (CYGD—`mips.gsf.de/proj/yeast`). The top-level categories in the functional hierarchy produce 13 classes (see table 11.2). These 13 classes contain 3588 proteins; the remaining yeast proteins have uncertain function and are therefore not used in evaluating the classifier. Because a given protein can belong to several functional classes, we cast the prediction problem as 13 binary classification tasks, one for each functional class.

The primary input to the classification algorithm is a collection of kernel matrices representing different types of data. In order to compare the SDP/SVM approach to the Markov random field (MRF) method of Deng et al., we perform two variants of the experiment: one in which the five kernels are restricted to contain precisely the same binary information as used by the MRF method, and a second experiment in which two of the kernels use richer representations and a sixth kernel is added.

Kernels for Protein Function Prediction For the first kernel, the domain structure of each protein is summarized using the mapping provided by SwissProt v7.5 (`us.expasy.org/sprot`) from protein sequences to Pfam domains (`pfam.wustl.edu`). Each protein is characterized by a 4950-bit vector, in which each bit represents the presence or absence of one Pfam domain. The kernel function K_{Pfam}

is simply the inner product applied to these vectors. This bit vector representation was used by the MRF method. In the second experiment, the domain representation is enriched by adding additional domains (Pfam 9.0 contains 5724 domains) and by replacing the binary scoring with log E-values derived by comparing the HMMs with a given protein using the HMMER software toolkit (`hmmer.wustl.edu`).

Diffusion kernel

Three kernels are derived from CYGD information regarding three different types of protein interactions: protein-protein interactions, genetic interactions, and coparticipation in a protein complex, as determined by tandem affinity purification (TAP). All three data sets can be represented as graphs, with proteins as nodes and interactions as edges. As explained before, each interaction graph allows establishing similarities among proteins through the construction of a corresponding diffusion kernel. This generates three interaction kernel matrices, K_{Gen}, K_{Phys} and K_{TAP}. Because direct physical interaction is not necessarily guaranteed when two proteins participate in a complex, a smaller diffusion constant—this parameter is required to construct a diffusion kernel (see Kondor and Lafferty, 2002)—is used to construct K_{TAP}, that is, $\tau = 1$ instead of $\tau = 5$ for the others.

The fifth kernel is generated using 77 cell cycle gene expression measurements per gene (Spellman et al., 1998). Two genes with similar expression profiles are likely to have similar functions; accordingly, Deng et al. convert the expression matrix to a square binary matrix in which a 1 indicates that the corresponding pair of expression profiles exhibits a Pearson's correlation greater than 0.8. We use this matrix to form a diffusion kernel K_{Exp}. In the second experiment, a Gaussian kernel is defined directly on the expression profiles: for expression profiles $\mathbf{x}_1$ and $\mathbf{x}_2$, the kernel is $k(\mathbf{x}_1, \mathbf{x}_2) = \exp(-||\mathbf{x}_1 - \mathbf{x}_2||^2/2\sigma)$ with width $\sigma = 0.5$.

Pairwise comparison kernel

In the second experiment, we construct one additional kernel matrix by applying the Smith-Waterman pairwise sequence comparison algorithm (Smith and Waterman, 1981) to the yeast protein sequences. Each protein is represented as a vector of Smith-Waterman log E-values, computed with respect to all 6355 yeast genes. The kernel matrix K_{SW} is computed using an inner product applied to pairs of these vectors. This matrix is complementary to the Pfam domain matrix, capturing sequence similarities among yeast genes, rather than similarities with respect to the Pfam database.

Results Each algorithm's performance is measured by performing fivefold cross-validation three times. For a given split, we again evaluate each classifier by reporting the ROC score on the test set. For each classification, we measure 15 ROC scores (three fivefold splits), which allows us to estimate the variance of the score.

The experimental results are summarized in figure 11.3. The figure shows that, for each of the 13 classifications, the ROC score of the SDP/SVM method is better than that of the MRF method. Overall, the mean ROC improves from 0.715 to 0.854. The improvement is consistent and statistically significant across all 13 classes. An additional improvement, though not as large, is gained by replacing the expression and Pfam kernels with their enriched versions. The most improvement is offered

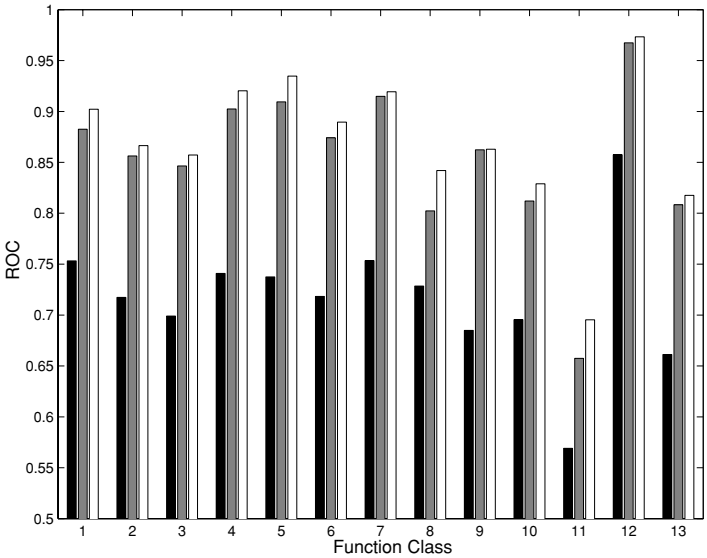


Figure 11.3 Classification performance for the 13 functional classes. The height of each bar is proportional to the ROC score. The standard deviation across the 15 experiments is usually 0.01 or smaller, so most of the depicted differences are significant. Black bars correspond to the MRF method of Deng et al.; gray bars correspond to the SDP/SVM method using five kernels computed on binary data, and white bars correspond to the SDP/SVM using the enriched Pfam kernel and replacing the expression kernel with the SW kernel.

by using the enriched Pfam kernel and replacing the expression kernel with the Smith-Waterman kernel. The resulting mean ROC is 0.870. Again, the improvement occurs in every class, although some class-specific differences are not statistically significant.

Table 11.3 provides detailed results for a single functional classification, the transport facilitation class. The weight assigned to each kernel indicates the importance that the SDP/SVM procedure assigns to that kernel. The Pfam and Smith-Waterman kernels yield the largest weights, as well as the largest individual ROC scores. Note that the combination of kernels performs significantly better than any single kernel. Results for the other 12 classifications are similar.

Table 11.3 Kernel weights and ROC scores for the transport facilitation class. The table shows, for both experiments, the mean weight associated with each kernel, as well as the ROC score resulting from learning the classification using only that kernel. The final row lists the ROC score using all kernels.

Kernel	Binary data		Enriched kernels	
	Weight	ROC	Weight	ROC
K_{Pfam}	2.21	.9331	1.58	.9461
K_{Gen}	0.18	.6093	0.21	.6093
K_{Phys}	0.94	.6655	1.01	.6655
K_{TAP}	0.74	.6499	0.49	.6499
K_{Exp}	0.93	.5457	—	.7126
K_{SW}	—	—	1.72	.9180
all	—	.9674	—	.9733

11.7 Discussion

We have described a general method for combining heterogeneous genome-wide data sets in the setting of kernel-based statistical learning algorithms, and we have demonstrated an application of this method to the problems of classifying yeast membrane proteins and protein function prediction in yeast. The resulting SDP/SVM algorithm yields significant improvement relative to an SVM trained from any single data type, relative to both state-of-the-art and classical biological methods for membrane protein prediction, as well as relative to a previously proposed graphical model approach for fusing heterogeneous genomic data. Moreover, the performance of the algorithm improves consistently in our experiments as additional genome-wide data sets are added to the kernel representation, if the additional data contain complementary information.

Kernel-based statistical learning methods have a number of general virtues as tools for biological data analysis. First, the kernel framework accommodates not only the vectorial and matrix data that are familiar in classic statistical analysis but also more exotic data types such as strings, trees, and graphs. The ability to handle such data is clearly essential to the biological domain. Second, kernels provide significant opportunities for the incorporation of more specific biological knowledge, as we have seen with the FFT kernel and the Pfam kernel, and unlabeled data, as in the diffusion and Smith-Waterman kernels. Third, the growing suite of kernel-based data analysis algorithms require only that data be reduced to a kernel matrix; this creates opportunities for standardization. Finally, as we have shown here, the reduction of heterogeneous data types to the common format of kernel matrices allows the development of general tools for combining multiple data types. Kernel matrices are required only to respect the constraint of positive definiteness, and thus the powerful technique of SDP can be exploited to derive general procedures for combining data of heterogeneous format and origin.

We thus envision the development of general libraries of kernel matrices for biological data, such as those that we have provided at `noble.gs.washington.edu/sdp-svm`, that summarize the statistically relevant features of primary data, encapsulate biological knowledge, and serve as inputs to a wide variety of subsequent data analyses. Indeed, given the appropriate kernel matrices, the methods that we have described here are applicable to problems such as the prediction of protein metabolic, regulatory, and other functional classes; the prediction of protein subcellular locations; and the prediction of protein-protein interactions.

12 Protein Classification via Kernel Matrix Completion

Taishin Kin
Tsuyoshi Kato
Koji Tsuda

The three-dimensional structure of a protein provides crucial information for predicting its function. However, as it is still a far more difficult and costly task to measure 3D coordinates of atoms in a protein than to sequence its amino acid composition, often we do not know the 3D structures of all the proteins at hand. Let us consider a kernel matrix that consists of kernel values representing protein similarities in terms of their 3D structures where some of the entries are *missing* because structural information about some proteins are not available whereas their amino acid sequences are readily available. This chapter proposes to estimate the missing entries by means of another kernel matrix derived from amino acid sequences. Basically, a parametric model is created from the sequence kernel matrix, and the missing entries of the structure's kernel matrix are estimated by fitting this model to existing entries. For model fitting, we adopt two algorithms: *single e-projection* and *em algorithm* based on the information geometry of positive definite matrices. For evaluating and demonstrating the performance of our method, we performed protein classification experiments by using support vector machines (SVMs). Our results show that these algorithms can effectively estimate the missing entries.

12.1 Introduction

Protein structure and sequence

One of the major issues in bioinformatics is the functional annotation of proteins. Proteins are molecules which play a variety of important roles (functions) in every living organism. The function of a protein is determined by its shape, which is usually called a 3D or tertiary structure, or more simply, structure. Therefore, protein structure is one of the major factors for investigating the mechanisms of

proteins. The constant growth of a protein structure database such as the protein data bank (PDB) (Berman et al., 2000) might be proof of its importance. However, the protein structures are not always available because measuring 3D coordinates of every atom of a nano-scale molecule requires very expensive and intensive experiments. On the other hand, protein amino acid sequences are abundantly available, as shown by the rapid growth of databases such as Swiss-Prot (O'Donovan et al., 2002). We can find this fact with a very simple comparison: there are 129,463 sequences in Swiss-Prot and 19,375 structures in PDB as of this writing. Thus, ongoing research is endeavoring to achieve practical prediction of protein structures from their amino acid sequences (see figure 12.1 for a brief description of the relation between sequence and structure of a protein). Although varieties of prediction methods have been proposed, the prediction of an exact tertiary structure remains one of the most difficult problems because mechanisms behind the relation between structure and sequence are not fully clarified yet. Nevertheless it is highly probable that a sequence contains certain information to infer its structure.

In this chapter, we estimate the *relation* between two structures instead of estimating the structure itself, following the ideas in Tsuda et al. (2003). As in other chapters of this book, we represent n proteins by an $n \times n$ kernel matrix, which is a positive definite similarity matrix where the (i, j)th entry is the similarity between the ith protein and the jth protein. When we do not have structures for all proteins, we have to leave the entries of the kernel matrix for unavailable structures as missing. Obviously, due to missing entries, kernel learning algorithms such as support vector machines cannot be applied. Our aim is to complete the missing entries so that the learning algorithms can work on the completed kernel matrix. Basically we create a parametric model from another kernel matrix derived from sequences, and fit the model to the existing entries to complete the missing ones. Our algorithm is derived from a mathematical theory known as *information geometry* (Amari, 1995). Finally, we show promising results in protein classification experiments.

This chapter is organized as follows: Section 12.2 describes some definitions used in our algorithm. Section 12.3 introduces the information geometry to the space of positive definite matrices. Based on the geometric concepts, two algorithms for matrix approximation are presented in section 12.4. Then the protein structure classification experiment is described in section 12.5. We present our conclusions in section 12.6.

12.2 Kernel Matrix Completion

Kernel matrices for structure and sequence

Let us consider a kernel function as the similarity measure between two proteins. There are two types of such functions: k_{st} for structure similarity and k_{sq} for sequence similarity. We define the two matrices as follows:

- Structure kernel matrix D: $[D]_{ij} = k_{st}(x_i, x_j), \qquad i, j = 1, \cdots, l$

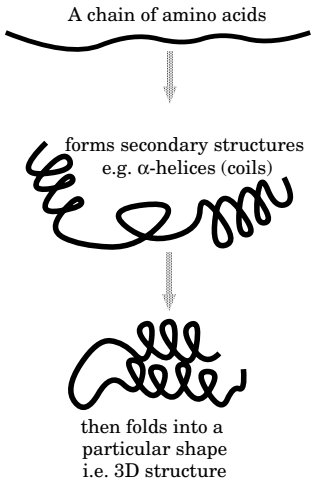


Figure 12.1 An overview of protein folding, which illustrates how an amino acid sequence folds into a protein molecule. A chain of amino acids is a product of translation of messenger RNA. The sequence forms secondary structures such as α helices (coils) or β sheets during its folding process. The secondary structure is the fundamental element of a protein structure. Finally, the sequence folds further into a specific shape which is the key property in determining the function of the protein. The shape of a protein is called its 3D structure.

- Sequence kernel matrix M: $[M]_{ij} = k_{sq}(x_i, x_j), \qquad i, j = 1, \cdots, l$

where $[M]_{ij}$ is the (i, j)th element of a matrix M and x_i denotes the ith protein. The two matrices are assumed to be positive definite such that they are compatible with kernel methods.

Now the task is to estimate the missing entries of the structure matrix using the sequence matrix. We create a *parametric model* of admissible matrices from the sequence matrix, and estimate missing entries by fitting the model to existing entries. According to our previous paper (Tsuda et al., 2003), we define the parametric model as all *spectral variants* of the sequence matrix which have the same eigenvectors but different eigenvalues (Cristianini et al., 2002b).

In order to fit a parametric model, the distance between two matrices has to be determined. A common way is to define the Euclidean distance between matrices (e.g., the Frobenius norm) and make use of the Euclidean geometry. Recently Vert and Kanehisa (2003b) tackled the incomplete matrix approximation problem by means of kernel canonical correlation analysis (CCA). Also Cristianini et al. (2002b) proposed a similarity measure called "alignment," which is basically the cosine between two matrices. In contrast to these methods, which are based on Euclidean geometry, this chapter follows an alternative way: we define the Kullback-

Kullback-Leibler divergence

Leibler (KL) divergence between two kernel matrices and make use of Riemannian information geometry (Ohara et al., 1996). The KL divergence is derived by relating a kernel matrix to a covariance matrix of Gaussian distribution. The primal advantage is that the KL divergence allows us to use the *em* algorithm (Amari, 1995) to approximate an incomplete kernel matrix. The e and m steps are formulated as convex programming problems; moreover, they can be solved analytically when spectral variants are used as a parametric model.

12.3 Information Geometry of Positive Definite Matrices

In this section, we introduce the information geometry of the space of positive definite matrices. Only necessary parts of the theory are presented here, so the reader is referred to Ohara et al. (1996) and Amari and Nagaoka (2000) for details.

Gaussian distribution

Let us define the set of all $d \times d$ positive definite matrices as $\mathcal{P}$. The first step is to relate a $d \times d$ positive definite matrix $P \in \mathcal{P}$ to the Gaussian distribution with mean 0 and covariance matrix P:

$$p(\mathbf{x}|P) = \frac{1}{(2\pi)^{d/2} \det P^{1/2}} \exp\left(-\frac{1}{2}\mathbf{x}^\top P^{-1}\mathbf{x}\right). \tag{12.1}$$

Exponential family

It is well known that the Gaussian distributions form an exponential family. An exponential family is a set of distributions that can be written in the following canonical form:

$$p(\mathbf{x}|\theta) = \exp(\theta^\top \mathbf{r}(\mathbf{x}) - \psi(\theta)),$$

where $\mathbf{r}(\mathbf{x})$ is the vector of sufficient statistics, $\theta \in \Re^\rho$ is called the natural parameter, and $\psi(\theta)$ is the normalization factor. When (12.1) is rewritten in the canonical form, we have the sufficient statistics as

$$\mathbf{r}(\mathbf{x}) = -\left(\frac{1}{2}x_1^2, \frac{1}{2}x_2^2, \ldots, \frac{1}{2}x_d^2, x_1x_2, x_2x_3, \ldots, x_{d-1}x_d\right)^\top,$$

and the natural parameter as

$$\theta = \left([P^{-1}]_{11}, [P^{-1}]_{22}, \ldots, [P^{-1}]_{dd}, [P^{-1}]_{12}, [P^{-1}]_{23}, \ldots, [P^{-1}]_{d-1,d}\right)^\top.$$

θ-coordinate system

From the viewpoint of information geometry, the natural parameter θ provides a coordinate system (Amari and Nagaoka, 2000) to specify a positive definite matrix P, which is called the θ-coordinate system (or the e-coordinate system). On the other hand, there is an alternative representation for the exponential family. Let us define the mean of $r_i(x)$ as η_i: For example, when $r_i(x) = x_s x_t$,

$$\eta_i = \int x_s x_t p(\mathbf{x}|\theta) d\mathbf{x} = P_{st}.$$

η-coordinate system

This new set of parameters η_i provides another coordinate system, called the η-coordinate system (or the m-coordinate system):

$$\eta = (P_{11}, P_{22}, \ldots, P_{dd}, P_{12}, P_{23}, \ldots, P_{d-1,d})^\top .$$

Let us consider the following curve $\theta(t)$ connecting two points θ_1 and θ_2 linearly in θ coordinates:

$$\theta(t) = t(\theta_2 - \theta_1) + \theta_1.$$

When written is the matrix form, this reads

$$P^{-1}(t) = t(P_2^{-1} - P_1^{-1}) + P_1^{-1}.$$

e-flat

This curve is regarded as a straight line from the exponential viewpoint and is called an exponential geodesic or e-geodesic. In particular, each coordinate curve $\theta_i = t$, $\theta_j = c_j \ \ (j \neq i)$ is an e-geodesic. When the e-geodesic between any two points in a manifold $\mathcal{S} \subseteq \mathcal{P}$ is included in $\mathcal{S}$, the manifold $\mathcal{S}$ is said to be e-flat. On the other hand, the mixture geodesic or m-geodesic is defined as

$$\eta(t) = t(\eta_2 - \eta_1) + \eta_1.$$

In the matrix form, this reads

$$P(t) = t(P_2 - P_1) + P_1.$$

m-flat

When the m-geodesic between any two points in $\mathcal{S}$ is included in $\mathcal{S}$, the manifold $\mathcal{S}$ is said to be m-flat.

In information geometry, the distance between probability distributions is defined as the KL divergence (Amari and Nagaoka, 2000):

$$KL(p, q) = \int p(x) \log \frac{p(x)}{q(x)} dx.$$

By relating a positive definite matrix to the covariance matrix of Gaussian (12.1), we have KL divergence for two matrices P, Q:

$$KL(P, Q) = tr(Q^{-1}P) + \log\det Q - \log\det P - d. \tag{12.2}$$

With respect to a manifold $\mathcal{S} \subseteq \mathcal{P}$ and a point $P \in \mathcal{P}$, the projection from P to $\mathcal{S}$ is defined as the point in $\mathcal{S}$ closest to P. Since the KL divergence is asymmetric, there are two kinds of projection:

- e-projection: $Q^* = \operatorname{argmin}_{Q \in \mathcal{S}} KL(Q, P)$.
- m-projection: $Q^* = \operatorname{argmin}_{Q \in \mathcal{S}} KL(P, Q)$.

It is known that the m-projection to an e-flat submanifold is unique, and the e-projection to an m-flat manifold is unique (Amari and Nagaoka, 2000).

12.4 Approximating an Incomplete Kernel Matrix

In this section, we describe the *em* algorithm to approximate an incomplete kernel matrix (Tsuda et al., 2003). Let $x_1, \ldots, x_\ell \in \mathcal{X}$ be the set of samples of interest. In supervised learning cases, this set includes both training and test sets; thus we are considering the transductive setting (Vapnik, 1998). Let us assume that the data are available for the first n samples, and unavailable for the remaining $m := \ell - n$ samples. Denote by K_I an $n \times n$ kernel matrix, which is derived from the data for the first n samples. In our experiments, K_I is derived from the protein 3D structures.

Incomplete kernel matrix

The incomplete kernel matrix is described as

$$D = \begin{pmatrix} K_I & D_{vh} \\ D_{vh}^\top & D_{hh} \end{pmatrix}, \tag{12.3}$$

where D_{vh} is an $n \times m$ matrix and D_{hh} is an $m \times m$ symmetric matrix. Since D has missing entries, it cannot be presented as a point in $\mathcal{P}$. Instead, all the possible kernel matrices form a manifold

Data manifold

$$\mathcal{D} = \{D \mid D_{vh} \in \Re^{n \times m},\ \ D_{hh} \in \Re^{m \times m},\ \ D_{hh} = D_{hh}^\top,\ \ D \succ 0\},$$

where $D \succ 0$ means that D is strictly positive definite. We call it the *data manifold* as in the conventional EM algorithm (Ikeda et al., 1999). It is easy to verify that $\mathcal{D}$ is an m-flat manifold; hence, the e-projection to $\mathcal{D}$ is unique.

Spectral variants

Next let us define the parametric model to approximate D. Here the model is derived as the spectral variants of K_B, which is an $\ell \times \ell$ auxiliary kernel matrix derived from another information source. Let us decompose K_B as

$$K_B = \sum_{i=1}^{\ell} \lambda_i \mathbf{v}_i \mathbf{v}_i^\top,$$

where λ_i and $\mathbf{v}_i$ are the ith eigenvalue and eigenvector, respectively. Define

$$M_i = \mathbf{v}_i \mathbf{v}_i^\top, \tag{12.4}$$

then all the spectral variants are represented as

$$\mathcal{M} = \{M \mid M = \sum_{j=1}^{\ell} \beta_j M_j,\ \ \beta \in \Re^\ell,\ \ M \succ 0\}$$

Model manifold

We call it the model manifold (Ikeda et al., 1999). For notational simplicity, we

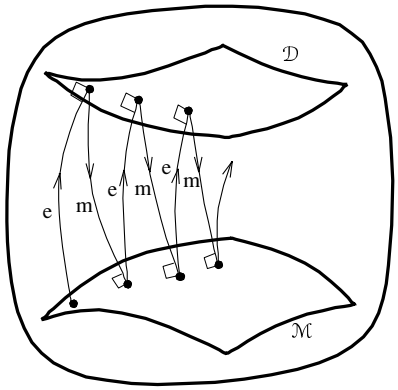


Figure 12.2 Information geometric picture of *em* algorithm. The data manifold $\mathcal{D}$ corresponds to the set of all completed matrices, whereas the model manifold $\mathcal{M}$ corresponds to the set of all spectral variants of a auxiliary matrix. The nearest points are found by gradually minimizing the KL divergence by repeating e- and m- projections.

choose a different parametrization of $\mathcal{M}$ [1]:

$$\mathcal{M} = \{M \mid M = (\sum_{j=1}^{\ell} b_j M_j)^{-1}, \ \ \mathbf{b} \in \Re^{\ell}, \ \ M \succ 0\}, \tag{12.5}$$

where $b_j = 1/\beta_j$. It is easily seen that the manifold $\mathcal{M}$ is e-flat and m-flat at the same time. Such a manifold is called dually flat.

Our approximation problem is formulated as finding the nearest points in two manifolds: Find $D \in \mathcal{D}$ and $M \in \mathcal{M}$ that minimize KL divergence $KL(D, M)$. In geometric terms, this problem is to find the nearest points between e-flat and m-flat manifolds. It is well known that such a problem can be solved by an alternating procedure called the *em* algorithm (Amari, 1995). The *em* algorithm gradually minimizes the KL divergence by repeating the e-step and m-step alternately (figure 12.2).

In the e-step, the following optimization problem is solved with a fixed M: Find $D \in \mathcal{D}$ that minimizes $KL(D, M)$ using (12.2). This is rewritten as follows: Find D_{vh} and D_{hh} that minimize

$$L_e = tr(DM^{-1}) - \log\det D, \tag{12.6}$$

1. $M^{\top}M^{-1} = (\sum \beta_j M_j)^{\top}(\sum 1/\beta_j M_j) = \sum \beta_j/\beta_j M_j^{\top} M_j = \sum M_j^{\top} M_j = I$.

subject to the constraint that $D \succ 0$. Notice that this constraint is not needed, because

$$\log\det D = \sum_{i=1}^{\ell} \log \mu_i,$$

where μ_i is the ith eigenvalue of D. Here $\log\det D$ is defined when all eigenvalues are positive. So, at the optimal solution, D is necessarily positive definite, because the KL divergence is infinite otherwise. As indicated by information geometry, this is a convex problem, which can readily be solved by any reasonable optimizer. Moreover the solution is obtained in a closed form: let us partition M^{-1} as

$$M^{-1} = \begin{pmatrix} S_{vv} & S_{vh} \\ S_{vh}^{\top} & S_{hh} \end{pmatrix}. \tag{12.7}$$

The solution to (12.6) is directly obtained by filling the missing entries in the matrix D with following forms:

$$D_{vh} = -K_I S_{vh} S_{hh}^{-1}, \tag{12.8}$$

$$D_{hh} = S_{hh}^{-1} + S_{hh}^{-1} S_{vh}^{\top} K_I S_{vh} S_{hh}^{-1}. \tag{12.9}$$

In the m-step, the following optimization problem is solved with D being fixed: Find $M \in \mathcal{M}$ that minimizes $KL(D, M)$. This is rewritten as follows: Find $\mathbf{b} \in \Re^{\ell}$ that minimizes

$$L_m = \sum_{j=1}^{\ell} b_j tr(M_j D) - \log\det(\sum_{j=1}^{\ell} b_j M_j) \tag{12.10}$$

subject to the constraint that $\sum_{j=1}^{\ell} b_j M_j \succ 0$. Notice that this constraint can be ignored as well. When $\{M_j\}_{j=1}^{\ell}$ are defined as (12.4), the closed-form solution of (12.10) is obtained as

$$b_i = 1/tr(M_i D), \qquad i = 1, \ldots, \ell. \tag{12.11}$$

For detailed derivation of (12.8), (12.9), and (12.11), the reader is referred to Tsuda et al. (2003).

12.5 Protein Structure Classification Experiment

We perform protein classification experiments by using our kernel completion algorithms. Here we use a fully curated database of protein structures called SCOP (Murzin et al., 1995), where proteins are classified into categories such that both structural and evolutionary relatedness are reflected. The categories are organized hierarchically as the following levels: class, fold, superfamily, and family. At the finest level, a family contains proteins with clear evolutionary relationship.

A set of proteins in a superfamily is considered to have the same evolutionary origin but it is not detectable at the level of sequences. Those in the same fold have major structural similarity, but evolutionary origins may be different. In this experiment, we used the following three superfamilies: NAD(P)-binding Rossmann-fold domains (6 families, 105 proteins) and (trans)glycosidases (4 families, 62 proteins). We classified the proteins in a superfamily into families by using our algorithms. In the case of (trans)glycosidases, 62 proteins are classified into 6 classes. Additionally, we used TIM beta/alpha-barrel protein fold (4 superfamilies, 90 proteins), where each protein is classified into one of superfamilies.

12.5.1 Kernels

We now describe how to obtain the $n \times n$ incomplete matrix K_I and the $l \times l$ auxiliary matrix K_B in this experiment.

The Incomplete Matrix K_I The incomplete matrix is obtained by structural similarities of proteins. The structural similarities are computed using results of MATRAS (Kawabata and Nishikawa, 2000) which is a software to measure a structural similarity of proteins. MATRAS yields several values to measure the similarity. We use values of *Rdis* which is the normalization (ranges from 0 to 100) of *ScDIS*, a similarity score for inter residual distances. We compute *Rdis* for every pair of proteins, which gives an $n \times n$ similarity matrix S_I. Since S_I is not positive definite, we modified S_I to be positive definite by cutting off non-positive eigenvectors (Roth et al., 2003). Let λ_i, ν_i denote the ith eigenvalue and eigenvector of S_I, respectively. Then, K_I is obtained as

$$K_I = \sum_{i:\lambda_i > 0} \lambda_i \nu_i \nu_i^\top .$$

The Base Matrix K_B As the base matrix K_B, we use the second-order marginalized count kernel (MCK) (Tsuda et al., 2002b) in which the entry represents sequence similarities between two proteins. MCK is a general framework that includes Fisher kernel (Jaakkola and Haussler, 1999). Second-order MCK is shown to be more efficient than Fisher kernel for the protein structure classification problem (Tsuda et al., 2002b). MCK exploits parameters of a latent variable model for its kernel computation. The latent variable model is used to represent implicit features of proteins such as secondary structures (i.e. α helices and β sheets). Here we use a hidden Markov model (HMM) which has a tri state full-connection network. Although such a network is not supported by any biological knowledge, it is supposed to capture implicit features of proteins. In order to build a kernel matrix derived from a homogeneous probability distribution, we train the HMM with all the protein sequences. Therefore, no class-specific information is explicitly given to the HMM.

12.5.2 Kernel Completion Methods

Other than the *em* algorithm presented in the previous section, we applied two simpler methods, namely *single e-projection* and *k-linear interpolation*. Let us describe details of them.

Single *e*-Projection The method *single e-projection* performs only one *e*-projection from K_B to the data manifold D; it does not do *m*-projection. So D can be obtained very fast since the method needs no iterations. However, we found that this method does not work well when the diagonal elements of D and M differ significantly. So we normalize K_I as follows:

$$K_I' := AK_IA \tag{12.12}$$

where A is the $n \times n$ diagonal matrix with entries

$$[A]_{ii} = \sqrt{\frac{[K_B]_{ii}}{[K_I]_{ii}}}. \tag{12.13}$$

The transformation in (12.12) adjusts the norms of feature vectors while the angles between feature vectors are kept the same. Notice that the *em* algorithm does not need normalization, because additional variables b_i can automatically absorb the difference of norms.

k-Linear Interpolation As an alternative method, we make use of the nearest sequences for completing missing entries. Suppose the structure is missing for the r-th protein ($n+1 \leq r \leq \ell$), and let us estimate the entries of the structure kernel d_{ri} for $i = 1, \ldots, n$. First, we identify the k-neighbors of protein r in terms of the sequence kernel M, that is, sort the entries m_{rj} ($j = 1, \cdots, n$) and take the k largest ones. Let us denote the identified indices as $j_1, \cdots, j_k$. Then, the structure kernel d_{ri} is determined as

$$d_{ri} = \begin{cases} \frac{1}{k} \sum_{a=1}^{k} d_{j_a,i} & i = 1 \cdots n \\ \frac{1}{k^2} \sum_{a=1}^{k} \sum_{b=1}^{k} d_{j_a,i} d_{j_b,i} & i = n+1 \cdots \ell \end{cases}$$

This amounts to estimating the feature vector of protein r as the center of gravity of other feature vectors corresponding to $j_1, \cdots, j_k$. We call this method *k-linear interpolation*. In the following, we chose $k = 3$ as a result of preliminary cross-validation experiments.

12.5.3 Experimental Design

For evaluating the performance of kernel completion methods, we observe the accuracies of the SVM in the following experiments. Given a complete kernel matrix of structures, we remove randomly chosen rows/columns. The fraction of removed rows/columns is changed from 10% to 90% by a 10-point step. After

completing the missing entries with one of the completion methods, the whole set of samples is randomly shuffled and divided into 50% training and 50% test sets. The accuracies of SVM are computed on these training and test sets. The regularization parameter C is determined through five fold cross-validations on the training set. When there are more than two classes, the classification problem is interpreted as several two-class problems by means of the one-against-all scheme, then several different classifiers are joined based on the largest-score-wins criteria.

Each experiment is iterated 100 times to average random effects. Classification accuracies solely using the sequence kernel K_B are computed to serve as references for comparing the algorithms discussed here.

12.5.4 Results

Several kernel matrices are illustrated in figure 12.3 so that the reader can grasp a visual intuition of how the kernel matrix is completed by the different algorithms. The SVM accuracies of the three completion algorithms on each task are shown in figure 12.4. The *em* algorithm performs the best when the fraction of missing entries is relatively small ($< 50\%$). In two data sets, namely the NAD(P) and TIM barrels, it significantly outperforms single e-projection. However, as the fraction of missing entries increases, the accuracy suddenly falls down. The reason is deemed as *overfitting*, because the increase in missing entries also increases the number of parameters. Single e-projection performs constantly well on all three data sets. It shows its best performances when the missing fraction is very large. However, when the missing fraction is small, it performs poorer than *em* algorithm. This is considered as the effect of *early stopping*. As often observed in neural network training, stopping the optimization before convergence sometimes avoids overfitting (Haykin, 1998). Finally, the k-linear interpolation performed quite well in two data sets [NAD(P) and glycosidases], although it is a simple heuristic. However, its accuracy was always worse than one of the two other principled methods.

Dotted flat lines in the figures show the accuracies of the sequence kernels. Thus the completion does not make sense if the accuracy is lower than this level. In all three experiments, the accuracies of completed matrices are better than that level until the 80% to 90% missing fraction. Obviously it is promising result because it implies that only partial information of structure can enhance the whole classification performance significantly.

12.6 Conclusion

In this chapter, we presented an algorithm for compensating an incomplete kernel matrix by utilizing a base kernel matrix of another information source. The algorithm is based on information geometry which provides metrics for the space of kernel matrices. Our algorithm can be a powerful tool in many situations where one information source is precise but expensive and the other source is noisy but

cheap. Thanks to our algorithm, we can get a better kernel matrix by combining a cheaper complete matrix with a more precise incomplete matrix. One such situation is proteins, from which we can extract the sequences easily but the structures are costly. The experimental results on the protein data set reveal a remarkable performance of our method. Nevertheless, it is worth noting other parametric models for the model manifold $\mathcal{M}$. For example, since all eigenvalues are fixed in *single e-projection* while all eigenvalues are adaptive in *em algorithm*, the other model might be one that permits a part of eigenvalues to be adaptive.

Although we only discussed the case in which two information sources were available, it is interesting to consider a method which utilizes more information sources than two. For example, we may be able to make use of other information sources like class labels. In future works, we look forward to developing information geometric methods to combine multiple kernel matrices.

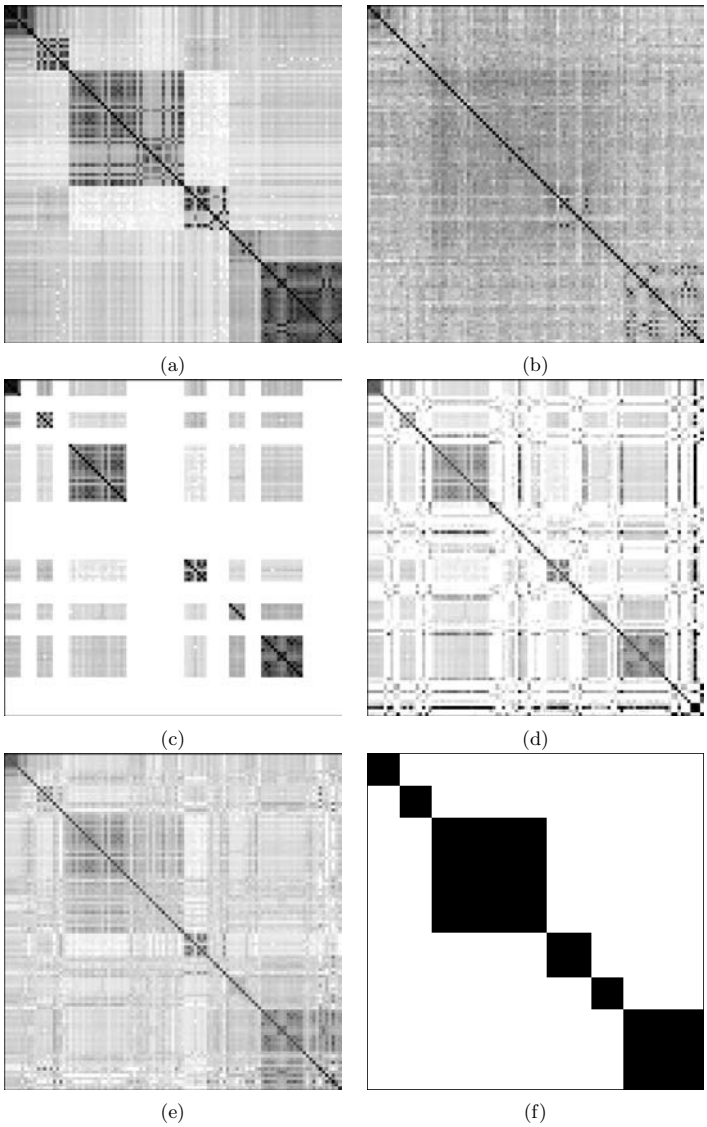


Figure 12.3 Raster of 62×62 kernel matrices for (trans)glycosidases. (a) Complete matrix of structure similarity (D_c). (b) Complete matrix of sequence similarity (M). (c) D with 50% missing elements. (d) D where missing elements are filled by using *single e-projection*. (e) Another D filled by using *em algorithm*. (f) The ideal kernel matrix showing three classes of this protein superfamily.

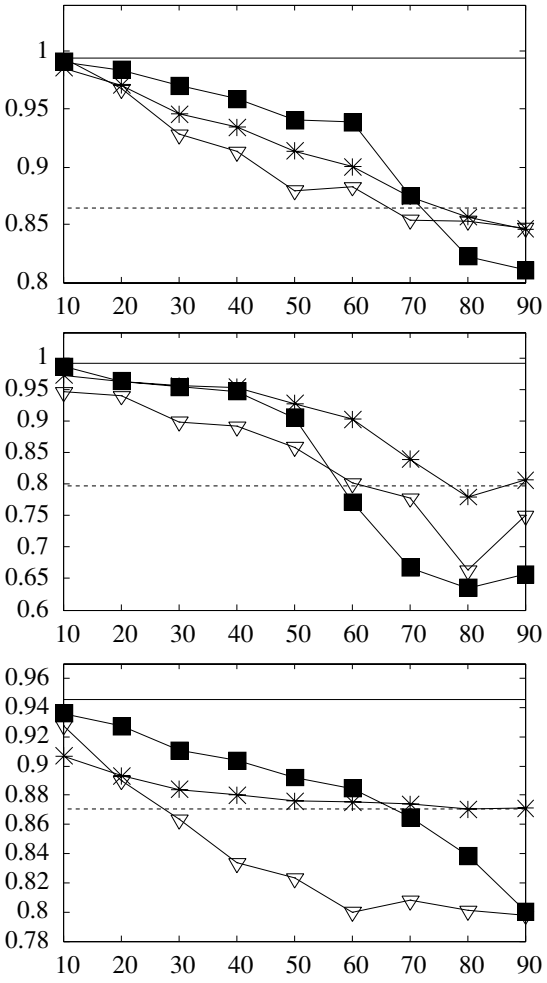


Figure 12.4 Classification accuracies for NAD(P)-binding Rossmann fold domains (top), (trans) glycosidases (middle), and TIM beta/alpha-barrels (bottom). The horizontal axis represents fraction of missing values. Three curves are shown in each figure: *square*, *em* algorithm; *star*, single *e*-projection; and *triangle*, 3-linear interpolation. The solid horizontal line indicates the accuracy without missing values. The dashed line indicates the accuracy using sequences only.

IV ADVANCED APPLICATION OF SUPPORT VECTOR MACHINES

13 Accurate Splice Site Detection for *Caenorhabditis elegans*

Gunnar Rätsch
Sören Sonnenburg

We propose a new system for predicting the splice form of *Caenorhabditis elegans* genes. As a first step we generate a clean set of genes from available exressed sequence tags (EST) and complete complementary (cDNA) sequences. From all such genes we then generate potential acceptor and donor sites as they would be required by any gene finder. This leads to a clean set of true and decoy splice sites. In a second step we use support vector machines (SVMs) with appropriately designed kernels to learn to distinguish between true and decoy sites. Using the newly generated data and the novel kernels we could considerably improve our previous results on the same task.

In the last step we design and test a new *splice finder system* that combines the SVM predictions with additional statistical information about splicing. Using this system we are able to predict the exon-intron structure of a given gene with known translation initiation and stop codon site. The system has been tested successfully on a newly generated set of genes and compared with GenScan. We found that our system predicts the correct splice form for more than 92% of these genes, whereas GenScan only achieves 77.5% accuracy.

13.1 Introduction

Splice sites

Splice sites are locations on DNA at the boundaries of exons (which code for protein parts) and introns (which do not). The more accurately a splice site can be located, the easier and more reliable it becomes to locate the genes on DNA. For this reason, accurate splice site detectors are valuable components of state-of-the-art gene finders (Burge and Karlin, 1997; Reese et al., 1997; Salzberg et al., 1998; Delcher et al., 1999; Pertea et al., 2001). Furthermore, since ever-larger chunks

of DNA are to be analyzed by gene finders, the problem of accurate splice site recognition has never been more important.

Support vector machines

SVMs (see, e.g., Vapnik, 1995; Müller et al., 2001; Schölkopf and Smola, 2002), with their strong theoretical roots, are known to be excellent algorithms for solving classification problems. They have been successfully applied to several bioinformatics problems (see, e.g., Jaakkola and Haussler, 1999; Zien et al., 2000; Brown et al., 2000; Tsuda et al., 2002a, and other chapters in this book). In this chapter we apply SVMs to two binary classification problems: the discrimination of donor sites (those at the exon-intron boundary) from decoys for these sites, and the discrimination of acceptor sites (those at the intron-exon boundary) from decoys for these sites. For this study we consider different kernels, in particular the so-called *locality improved* kernel that we proposed in Zien et al. (2000) for translation initiation site (TIS) recognition, the standard polynomial kernel, the SVM-pairwise kernel using alignment scores (Liao and Noble, 2002), the TOP kernel (related to the Fisher kernel; cf. Jaakkola and Haussler, 1999; Tsuda et al., 2002a), and, in addition, a polynomial-like kernel – the *weighted degree kernel*.

Decoys

Although present-day splice site detectors, e.g., based on neural networks or hidden Markov models (HMMs) are reported to perform at a fairly good level (Reese et al., 1997; Rampone, 1998; Cai et al., 2000), several of the reported performance numbers should be interpreted with caution, for a number of reasons. First of all, these results are based on *small* data sets of a limited number (one or two) of organisms. Now that many genomes have been fully sequenced, these results will need to be re-evaluated. Second, often only the single site prediction of acceptor and donor sites is considered, whereas the higher goal is to use it within a gene finder and it is uncertain how good the predictors perform in this setting. Third, issues in generating negative examples (decoys) were, if recognized, not adequately documented. In some cases (e.g., IP-data in Rampone, 1998), the decoy examples were chosen to be so weak that almost any reasonable method would achieve a high accuracy. In some other cases it was neither described how the decoys were chosen nor were the data sets made publicly available. Their reported performance is therefore incomparable with external studies, since the choice of data sets, in particular the decoys, can make a tremendous difference in the measured performance. Moreover, note that the choice of decoys is actually connected with the intended purpose of the system (i.e., use in a gene finding system).

In this study we put particular care into designing an appropriate splice data set for *C. elegans*. First, we derive a clean set of true splice sites from matching complete cDNA and ESTs to the genomic sequence. From them we generate a set of genes. Our decoy examples are chosen to be the sites that a splice finder asks predictions for but are actually not true sites (see details in the appendix). The generation of the data in this way ensures that the resulting classifier performs well in a setting in which a splice finder would actually use it. Previously mentioned approaches do not take this issue into account, since the training set decoys and test set decoys (when used in a splice finder) are generated from completely different

probability densities. Moreover, it poses theoretical problems and often leads to a great performance loss in practice.

Splice forms

We train and evaluate the SVM on the newly generated data set. We find great performance improvements in predicting single splice sites compared to our previous study on a related *C. elegans* splice data set (Sonnenburg et al., 2002). However, the single site predictions alone do not suffice to determine the splice form (i.e., the exon-intron structure) of a gene. We therefore *design a splice finder* that uses our donor and acceptor predictions and finds a *consistent segmentation* of a given pre-messenger RNA (pre-mRNA) sequence into exons and introns. We assume here that the start (translation initiation) and end (stop codon) positions on the RNA sequence are known.[1] Our splice finder system uses additional knowledge such as intron and exon length statistics to produce biologically plausible splice forms.

A complete understanding of splice sites not only helps to *correctly* predict the spliced mRNA and thus proteins from DNA but can also be of great help in localizing genes. Actually several other sites, like translation initiation start sites and stop codons, branch sites, promoters and terminators of transcription, and various transcription factor binding sites belonging to the class of *local sites* can help to detect genes (Haussler, 1998). Compared to a gene finder that finds genes and locates their exons, our system only solves the subtask of predicting the coding parts (i.e., locating the exons within a gene) of the sequence when the start and end positions are given. Therefore we do not consider the detection of other local sites. However, the proposed system can quite easily be combined with commonly used techniques for gene finding leading to a full gene finder system.

In this chapter the main focus is on improving the signal sensor for the detection of splice sites. As a second step we use the improved sensor to accurately predict the splice form of a gene, when the start and end position of the coding region is given. In a carefully designed and fair experiment we compare the results of our splice finder with a state-of-the-art gene finder – GenScan (Burge and Karlin, 1997) – and find dramatic performance improvements.

The chapter is organized as follows: In section 13.2 we review some biological background on splicing. In section 13.3 we describe the different kernels and methods we used in this study. In section 13.4 we present our experimental results: (a) We compare the methods on the basic classification task, showing the power of kernel methods. (b) We show interesting relations of the predicted activity of a splice site to the position in the gene. (c) Finally, we compare our best methods with the state-of-the-art method GenScan and find greatly superior performance of our methods in the task of predicting the splice form of a gene. In the appendix to this chapter we describe in detail the data generation process.

1. We do not consider splicing in the $5'$ and $3'$ untranslated regions (UTRs).

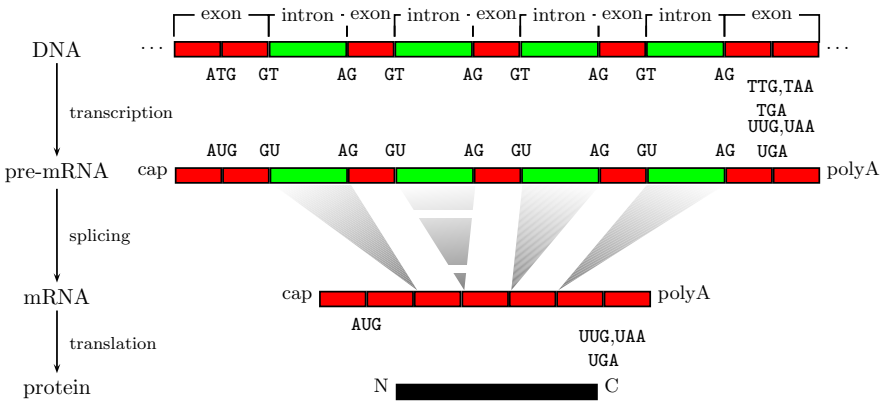


Figure 13.1 The major steps in protein synthesis. See text for details. The idea is from Chuang and Roth (2001) and Lewin (2000)

13.2 Biological Background

Each chromosome consists of coding and intergenic regions. The latter makes up most of the chromosome. But where on the chromosome are the genes located and what is a gene? A *gene* can be defined as a region of DNA that controls a certain characteristic. It corresponds to a sequence used in the production of a specific protein. While the question about the location of genes has not been sufficiently answered yet, a number of stages that are involved in the expression of genes (i.e., the process of synthesizing proteins from genes) have been identified (Lewin, 2000). These steps are carried out sequentially (see figure 13.1):

1. Activation of gene structure
2. Initiation of transcription
3. Elongation of the transcript
4. Post-processing
5. Transport of mRNA to cytoplasm
6. Translation of mRNA

It was discovered that genes are "activated" in a celltype-specific fashion. A gene may be active in cells composing one kind of tissue, but "inactive" in other kinds of tissue. When the precondition – that the gene be active – is fulfilled, it can be transcribed, that is, the process by which a copy of the gene is synthesized and which is encoded on only one strand of the double-stranded DNA (the coding strand for that gene). The copy is not DNA, but single-stranded precursor messenger

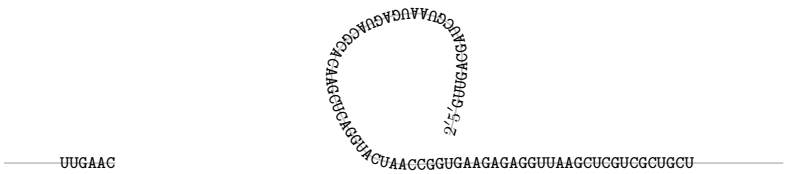


Figure 13.2 Illustration of the splicing process. First a cut is made at the 5′ site right before the `GT`. Then splicing proceeds through a lariat, in which the 5′ terminus generated at the end of the intron becomes linked by a 5′-2′ bond to a nucleotide within the intron. The target nucleotide is an `A` in a sequence that is called the branch site (Lewin, 2000). In the second stage, a cut is made at the 3′ site. The free intron is released and both exons are joined at their ends.

ribonucleic acid (pre-mRNA). The chemistry of RNA requires that the role of thymine is taken over by the nucleotide uracil (U). For convenience, we will use uracil and thymine synonymously. The transcription starts when the enzyme RNA polymerase binds to the *promoter*, which is a special region located upstream[2] of the first nucleotide that is transcribed into pre-mRNA. From this *starting point*, RNA polymerase moves *downstream* in the $5' \to 3'$ direction, continuously synthesizing pre-mRNA until a terminator sequence is reached. In the post-processing step, the pre-mRNA is transformed into mRNA. One necessary step in the process of obtaining mature mRNA is called *splicing*. The coding sequence of a eukaryotic gene is "interrupted" by noncoding regions called *introns*. A gene starts with an exon and may then be interrupted by an *intron*, followed by another exon, intron and so on until it ends in an exon. In the splicing process, introns are removed (cf. figure 13.2.)

As a result, there are two different splice sites: the exon-intron boundary, referred to as the donor site or 5′ site (of the intron) and the intron-exon boundary, that is the acceptor or 3′ site. Splice sites have quite strong consensus sequences, i.e. almost each position in a small window around the splice site is representative of the most frequently occurring nucleotide when many existing sequences are compared in an alignment. For example, the 5′ site's consensus is $\mathtt{A}_{64}\mathtt{G}_{73}\mathtt{G}_{100}\mathtt{T}_{100}\mathtt{A}_{62}\mathtt{A}_{68}\mathtt{G}_{84}\mathtt{T}_{63}$, while the 3′ site's consensus is $\mathtt{C}_{65}\mathtt{A}_{100}\mathtt{G}_{100}$, where the subscripts denote the frequency of the symbol in percent (Lewin, 2000). The dimers `GT` and `AG` can therefore be used to identify potential donor and acceptor sites. Unfortunately the pairs `GT...AG` occur very frequently; for example, in human DNA (which is $\approx 3 \cdot 10^9$ nucleotides in size), `GT` occurs about 1 billion times. For some crude estimate of, say, 10^5 genes with ten exons each, only 0.1% of the possible splice sites are *real* splice sites. One can analogously estimate the number of occurrences of `AG` which leads to a similar result. Therefore it is not enough to look at pairs of `GT...AG`, evidently there is some intrinsic property around the splice site telling the *spliceosome*, i.e. the large biological splicing apparatus consisting of an array

2. Upstream means closer to the 5′ end, while downstream means closer to the 3′ end.

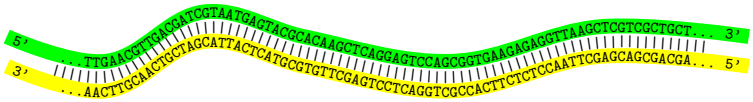


Figure 13.3 The two strands of DNA in an ASCII-character chain. As a result of a sequencing project, only one of these sequences is given, since adenine is always connected to thymine, and guanine to cytosine.

of proteins and ribonucleoproteins, to start the splicing mechanism. Neither is it known what exactly this property is, nor what the details of the reaction involving RNA and the spliceosome are. While the *canonical splice sites* `GT...AG` make up the vast majority of splice sites, other possible combinations, as, for example, `GT...TG`, have been discovered (Burset et al., 2000). We will not take these *noncanonical* sites into account, which would make splice site detection even more difficult.

What is known about splicing?

- The splicing process takes place in the *nucleus*.
- Exons are relatively short, 100 to 200 nucleotides (*nt*) while introns are often longer than 1 knt.
- An average mammalian gene has 7 to 8 exons spread over $\approx$ 16 knt.
- There are no reading frames in introns.
- Splice sites are generic: They do not have a specificity for individual RNA precursors, and individual precursors do not convey specific information (such as secondary structure) that is needed for splicing.
- The apparatus for splicing is not tissue-specific: RNA can usually be spliced properly by any cell, whether or not it is usually synthesized in that cell.
- Experiments show that any $5'$ splice site can in principle be connected to any $3'$ splice site, that is, only *local* information is relevant in the splicing process.
- In *higher* eukaryotes, 18 to 40 nt upstream of the $3'$ site, lies the branch site. To this site the `GU` from the $5'$ site connects to an `A` of the branch site.

Thus, the sequences needed for splicing are the short consensus sequences at the $5'$ and $3'$ splice sites and at the branch site. In higher eukaryotes mutations or deletions of the branch site result in a *proximate* $3'$ *site* to be taken. The branch site therefore identifies the $3'$ site to be used as the target for connection to the $5'$ site (Lewin, 2000), but its removal does not prevent splicing.

After pre-mRNA has been spliced to form mRNA, the splicing product is transported from the nucleus to the cytoplasm. There, in the step of translation, mRNA is read as *codons*, i.e. as triplets of nucleotides. Hence, there are three different *reading frames*, that is, ways of reading triplets of RNA (one for each of the possible start positions: $0, 1,$ or 2). Sixty-one of the $4^3 = 64$ possible codons code for 20 amino acids, while the remaining three (`UAG,UAA,UGA`) are termination codons,

Figure 13.4 Illustration of how acceptor site examples are constructed. Windows of fixed length are taken around the splice site, while `AG` is aligned to be at the same position in all examples. The left part, including the `AG`, is intronic, while the rest is exonic.

which mark the end of a gene. The translation begins almost always at the start codon `AUG`, called the *translation initiation site* (TIS). However, only the minority of the codon `AUG`, which represents the amino acid methionine, really signals the translation initiation. SVMs have been successfully used to model this site (Zien et al., 2000). When a stop codon is reached, the translation is terminated and the sequence of amino acids – the result of the translation process – forms a long polypeptide, the *protein*.

13.2.1 Data

To gain some understanding about what the data look like, we give some examples:

A fully sequenced genome consists of all the chromosomes found in a cell of the respective organism. Each sequenced chromosome is a sequence of the characters `A,C,G,T`, like that in figure 13.3.

When dealing with splice sites, the examples are aligned such that `AG` appears at the same position in all examples, while each example consists of a fixed number of nucleotides around the site (figure 13.4).

In this chapter we consider the problem of distinguishing true splice sites from decoys. Most learning algorithms need positive as well as negative examples for learning. While it is relatively simple to extract positive examples from, for example, cDNA matches (cf. subsection 1.1.1), it is less obvious how to determine negative examples. We do this as follows: from matching cDNA to the genomic sequence we generate a set of "virtual genes" and simulate a run of a splice finder for the whole gene. Our negative examples are chosen to be the sites that any splice finder would ask predictions for, but are not true sites. A detailed explanation of how the splice data set was generated is given in the appendix. All data sets used in this study together with more information are publicly available from `http://ida.first.fhg.de/splice`.

13.3 Methods

In the following we describe three methods for different subtasks of splice site recognition. The *first task* is to classify a given sequence whether it be a donor or not and whether it be an acceptor site or not (two two-class problems). This is done by training Markov models (MMs) or SVMs on the training data and tuning their hyperparameters on the validation data. The *second task* is to predict the splice form for a given sequence. In this step we use the scores provided by the single site detectors (first task) for every appearing `AG` and `GT` dimer. The challenge is to find a splice form that consistently combines all predictions. It turns out that it is beneficial to combine the single site scores with available statistical information about the sequences and certain rules about the structure of the resulting splice product (e.g., open reading frames). In the second step it is assumed that the number of exons to be found is given in advance. The *third task* is to determine the numbers of exons – using the results of the first two stages for different number of exons. For a complete prediction of a splice form from a given sequence, one first computes the single site scores for each potential splice site, then builds the probabilistic model and computes the optimal splice form for various numbers of exons, and finally selects one of the splice forms (i.e., exon number) which is the result of the prediction.

13.3.1 Identifying a Single Splice Site

Machine learning classification methods aim at estimating a classification function $g : \mathcal{X} \to \{\pm 1\}$ using labeled training data from $\mathcal{X} \times \{\pm 1\}$ such that g will correctly classify unseen examples (test data). In our case, input space $\mathcal{X}$ will contain simple representations of sequences $\{A, C, G, T\}^N$, while ± 1 corresponds to true splice and decoy sites, respectively. We will use the posterior log-odds of a simple probabilistic model and SVMs using different kernels as classifiers.

13.3.1.1 Posterior Log-Odds

The posterior log-odds of a probabilistic model with parameters $\boldsymbol{\theta}$ are defined by

$$f(\mathbf{x}) := \log\left(P(y = +1|\mathbf{x}, \boldsymbol{\theta})\right) - \log\left(P(y = -1|\mathbf{x}, \boldsymbol{\theta})\right) \tag{13.1}$$

$$= \log\left(P(\mathbf{x}|\boldsymbol{\theta}^+)\right) - \log\left(P(\mathbf{x}|\boldsymbol{\theta}^-)\right) + b, \tag{13.2}$$

Markov model

where b is the bias. We use Markov chains

$$P(\mathbf{x}|\boldsymbol{\theta}^\pm) = P(x_1, \ldots, x_N|\boldsymbol{\theta}^\pm) = P(x_1, \ldots, x_\omega|\boldsymbol{\theta}^\pm) \prod_{i=\omega+1}^{N} P(x_i|x_{i-1}, \ldots, x_{i-\omega}, \boldsymbol{\theta}^\pm) \tag{13.3}$$

as, for instance, described in Durbin et al. (1998). Each factor in this product has to be estimated in model training, that is, one counts how often each symbol appears

at each position in the training data conditioned on every possible $x_{i-1}, \ldots, x_{i-\omega}$. Then, for given model parameters $\boldsymbol{\theta}$ we have

$$P(\mathbf{x}|\boldsymbol{\theta}^{\pm}) = \theta_0^{\pm}(x_1, \ldots, x_\omega) \prod_{i=\omega+1}^{N} \theta_i^{\pm}(x_i, \ldots, x_{i-\omega}) \tag{13.4}$$

where $\theta_0^{\pm}$ is an estimate for $P(x_1, \ldots, x_\omega)$ and $\theta_i(x_i, \ldots, x_{i-\omega})$ an estimate for $P(x_i|x_{i-1}, \ldots, x_{i-\omega})$. As the alphabet has four letters, each model has $(N - \omega + 1) \cdot 4^{\omega+1}$ parameters and the maximum likelihood estimate is given by

$$\theta_0(s_1, \ldots, s_\omega) = \frac{1}{m+\tau}\left(\sum_{k=1}^{m} \mathbf{I}(s_1 = x_1^k \wedge \cdots \wedge s_\omega = x_\omega^k) + \tau\right)$$
$$\theta_i(s_i, \ldots, s_{i-\omega}) = \frac{\sum_{k=1}^{m} \mathbf{I}(s_i = x_i^k \wedge \cdots \wedge s_{i-\omega} = x_{i-\omega}^k) + \tau}{\sum_{k=1}^{m} \mathbf{I}(s_i = x_{i-1}^k \wedge \cdots \wedge s_{i-\omega} = x_{i-\omega}^k) + 4\tau},$$

where $\mathbf{I}(\cdot)$ is the indicator function, k enumerates over the number of observed sequences m, and τ the commonly used pseudo-count (a model parameter; cf. Durbin et al., 1998). The bias b is tuned by minimizing the number of misclassifications on the validation set. Finally, $g(\mathbf{x}) = \text{sign}(f(\mathbf{x}))$ defines the classifier we obtain from the posterior log-odds.

13.3.1.2 SVM and Kernels for Splice Site Detection

As the second method we use SVMs as described in chapter 2. The generated classification function can be written as

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{m} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b\right), \tag{13.5}$$

where $y_i \in \{-1, +1\}$ $(i = 1, \ldots, m)$ is the label of example $\mathbf{x}_i$. The α_i's are Lagrange multipliers and b is the usual bias which are the results of SVM training. The kernel k is the *key ingredient* for learning with SVMs. It implicitly defines the feature space and the mapping Φ via

$$k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle. \tag{13.6}$$

In the following paragraphs we describe the kernels which are used in this study.

Polynomial kernel

As the well-known (homogeneous) *Polynomial kernel* of degree d

$$k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^d.$$

is originally defined on real-valued inputs, it cannot directly be applied to discrete data, like DNA. However, a commonly used trick is to map the alphabet A,C,G,T into a binary representation: $\mathbf{x} \in \{A, C, G, T\}^N$ is represented as

$$\begin{aligned}\tilde{\mathbf{x}} = (&\mathbf{I}(x_1 = A), \mathbf{I}(x_1 = C), \mathbf{I}(x_1 = G), \mathbf{I}(x_1 = T),\\ &\mathbf{I}(x_2 = A), \mathbf{I}(x_2 = C), \mathbf{I}(x_2 = G), \mathbf{I}(x_2 = T),\\ &\ldots,\\ &\mathbf{I}(x_N = A), \mathbf{I}(x_N = C), \mathbf{I}(x_N = G), \mathbf{I}(x_N = T))^\top .\end{aligned}$$

Now we can apply the standard polynomial kernel $k(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}') = \langle \tilde{\mathbf{x}}, \tilde{\mathbf{x}}' \rangle^d$. This kernel takes all correlations of matches $\mathbf{I}(\tilde{x}_i = \tilde{x}'_i)$ up to order d into account. The features used for learning are position-dependent. They carry *local and global* information about the sequence, as, for instance, any position is combined with any other position to form a feature in the kernel feature space (generated by raising the scalar product to the power of d; cf. Müller et al., 2001). Note that this kernel can be computed in an efficient manner directly on the input space by

$$k(\mathbf{x}, \mathbf{x}') = \left(\sum_{i=1}^{N} \mathbf{I}(x_i = x'_i) \right)^d . \tag{13.7}$$

Locality improved kernel

The so-called locality improved (LI) kernel has been proved useful in the context of TIS recognition (Zien et al., 2000). It essentially works like the polynomial kernel but only considers local correlations within a small window. It is obtained by comparing the two sequences locally within a window of length $2l + 1$ around a sequence position, where one counts matching nucleotides. The resulting fraction of hits is taken to the dth power, where d reflects the order of local correlations (within the window) that we expect to be of importance:

$$\mathrm{win}_p(\mathbf{x}, \mathbf{x}') = \left(\frac{1}{2l+1} \sum_{j=-l}^{+l} \mathbf{I}(x_{p+j} = x'_{p+j}) \right)^d , \tag{13.8}$$

where $p = l+1, \ldots, N-l$. These window scores are then summed up over the length of the sequence using a weighting w_p which linearly decreases to both ends of the sequence, that is, $w_p = \begin{cases} p-l & p \le N/2 \\ N-p-l+1 & p > N/2 \end{cases}$. Then we have the following kernel:

$$k(\mathbf{x}, \mathbf{x}') = \sum_{p=l+1}^{N-l} w_p \mathrm{win}_p(\mathbf{x}, \mathbf{x}'). \tag{13.9}$$

The weighting allows one to emphasize regions of the sequence which are believed to be of higher importance (in our case the center, which is the location of the site). Note that the definition of the LI kernel by Zien et al. (2000) is slightly different from ours. Previously the weighting was inside the window which was not very effective. Moreover, the proposed version of the kernel can be computed $2l+1$ times faster than the original one.

Weighted degree kernel

In a similar approach one counts the matches between two sequences $\mathbf{x}$ and $\mathbf{x}'$ between the words $\boldsymbol{u}_{\omega,i}(\mathbf{x})$ and $\boldsymbol{u}_{\omega,i}(\mathbf{x}')$ where $\boldsymbol{u}_{\omega,i}(\mathbf{x}) = x_i x_{i+1} \dots x_{i+\omega-1}$ for all i and $1 \leq \omega \leq d$. The parameter ω denotes the order (length of the word) to be compared. The weighted degree kernel is defined as

$$k(\mathbf{x}, \mathbf{x}') = \sum_{\omega=1}^{d} w_\omega \sum_{i=1}^{N-d} \mathbf{I}(\boldsymbol{u}_{\omega,i}(\mathbf{x}) = \boldsymbol{u}_{\omega,i}(\mathbf{x}')) \tag{13.10}$$

where we chose the weighting to be $w_k = d - \omega + 1$, that is, higher-order matches get lower weights. This kernel emphasizes position-dependent information and decreases the influence of higher-order matches. It can be computed very efficiently without even extracting and enumerating all words from the sequences.[3] Note that this kernel is similar to the spectrum kernel as proposed by Leslie et al. (2002), with the main difference that the weighted degree kernel uses position-specific information.

TOP kernel from Markov models

Similarly to the well-known Fisher Kernel (Jaakkola and Haussler, 1999), the main idea of the TOP kernel (cf. Tsuda et al., 2002a) is to incorporate prior knowledge via a given probabilistic model. It is derived from the tangent vectors of posterior log-odds (TOP) and especially designed for classification. It was shown to outperform the Fisher kernel on protein family classification tasks (Tsuda et al., 2002a). It is defined as $k(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}), \boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}') \rangle$, where

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}) := (v(\mathbf{x}, \boldsymbol{\theta}), \partial_{\theta_1} v(\mathbf{x}, \boldsymbol{\theta}), \dots, \partial_{\theta_p} v(\mathbf{x}, \boldsymbol{\theta}))^\top \tag{13.11}$$

and

$$v(\mathbf{x}, \boldsymbol{\theta}) = \log(P(y = +1|\mathbf{x}, \boldsymbol{\theta})) - \log(P(y = -1|\mathbf{x}, \boldsymbol{\theta})). \tag{13.12}$$

For the MM described in subsection 13.3 this kernel is particularly simple to compute. Then we have $\partial_{\theta_{i,j}} v(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{I}(x_i = j)/\theta_{i,j}$ and hence the kernel is computed as

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') = v(\mathbf{x}, \boldsymbol{\theta}) v(\mathbf{x}', \boldsymbol{\theta}) \\ + \mathbf{I}(x_1 = x'_1 \wedge \dots, \wedge x_\omega = x'_\omega)/(\theta_0^+(x_1, \dots, x_\omega))^2 \\ + \mathbf{I}(x_1 = x'_1 \wedge \dots, \wedge x_\omega = x'_\omega)/(\theta_0^-(x_1, \dots, x_\omega))^2 \\ + \sum_{i=\omega+1}^{N} \mathbf{I}(x_i = x'_i \wedge \dots, \wedge x_{i-\omega} = x'_{i-\omega})/(\theta_i^+(x_i, \dots, x_{i-\omega}))^2 \\ + \sum_{i=\omega+1}^{N} \mathbf{I}(x_i = x'_i \wedge \dots \wedge x_{i-\omega} = x'_{i-\omega})/(\theta_i^-(x_i, \dots, x_{i-\omega}))^2. \end{aligned} \tag{13.13}$$

SVM-pairwise

Moreover, we use the approach of Liao and Noble (2002) in which Smith-Waterman alignment scores (Smith and Waterman, 1981) make up the SVM's

3. An implementation can be downloaded from the mentioned website.

feature space.[4] Here, we use a randomly chosen subset of 250 positive and 250 negative examples and compute the alignment scores of a sequence to all 500 reference sequences. Computing the alignment to *all* training points, as done in Liao and Noble (2002), was not feasible for our problem. This procedure generates a 500-dimensional vector describing each sequence which we used as the input to a linear kernel, that is, $k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$.

Spectrum kernel

Additionally, we considered the spectrum kernel (Leslie et al., 2002). However, since it does not contain the information where the subsequences are located, it did not seem appropriate to use it for our problem. Moreover, it performed very poorly in some preliminary tests on our data sets (not shown).

Normalization

Finally, a remark regarding the normalization of the kernels is in order. In the case of the TOP kernel and SVM-pairwise, we first normalized each feature to have zero mean and standard deviation 1. Moreover, we normalized all kernels except the LI kernel such that the vectors in feature space $\Phi(\mathbf{x})$ have length 1. This can be done efficiently by redefining the kernel as follows:

$$\tilde{k}(\mathbf{x}, \mathbf{x}') = \frac{k(\mathbf{x}, \mathbf{x}')}{\sqrt{k(\mathbf{x}, \mathbf{x}) k(\mathbf{x}', \mathbf{x}')}}. \tag{13.14}$$

In particular, the latter normalization solved many convergence problems of the SVM optimizer. In the case of the LI kernel the normalization was not necessary, since it never led to any convergence problems without normalization.

13.3.2 Predicting the Splice Form

Using the techniques of the previous section we can identify single splice sites. In some cases, however, the classifier will predict false positives or miss a site. In these cases it is beneficial to incorporate more knowledge about the splice product. In particular, that after every donor has to follow an acceptor site and that an exon has to have an open reading frame. The input to the second stage of the system as described in this section will be a list of single sites together with the SVM or MM scores. From this list we generate a consistent splice form for a particular gene.

At this stage we assume that the number E of exons of a gene is known and we have already generated a list of potential exons (containing an open reading frame) from a virtual gene as described in the appendix. Then the task is to predict the E exons that constitute the coding region of a (virtual) gene.

We first define a probabilistic model (HMM) of a gene and then – by using the Viterbi algorithm – find the most likely path to predict which of the potential exons are used. The model uses the following components:

1. *Probability density estimates over the lengths of introns and exons.*
On the training and validation data we counted how often which length of exons

4. Thanks to A. Zien for providing the code.

and introns appear. We used a log-scaled histogram with 100 bins and with pseudo-count 1 to estimate the probabilities $p_e^l(\mathbf{s})$ and $p_i^l(\mathbf{s})$ that a sequence $\mathbf{s}$ of a certain length is an exon or an intron, respectively.

2. *Probabilities for frame shifts in splicing.*
Here, we counted how often a frame shift of zero, one, or two nucleotides appears on the basis of the length of exons in training and validation data (pseudo-count one). We denote this probability by $p^{fs}(\mathbf{s})$ for a sequence $\mathbf{s}$.

3. *Probability density estimates for the GC content of introns and exons.*
We estimated the average GC content of exons ($\mu_e = 40.31\%$) and introns ($\mu_i = 35.9\%$). We assume a simple Gaussian model for the GC content and use the following probabilities for a sequence $\mathbf{s}$: $p^{gc}(\mathbf{s}) \sim \exp(|gc(\mathbf{s}) - \mu|^2/\sigma^2)$, where $gc(\mathbf{s})$ denotes the GC content of $\mathbf{s}$, μ can either be μ_e or μ_i, and σ is the standard deviation which we used for both models ($\sigma = 5.56\%$, as estimated from the exon sequences).

4. *Scores derived from the SVM predictions.*
SVMs do not output posterior probabilities. The usual way to deal with this problem (Platt, 2001) is to transform the output of the SVM with the sigmoid function $\theta(\cdot)$ to obtain a probability-like score. In our model we used $\hat{p}^{svm}(\mathbf{s}) = \frac{1}{2} + \theta(af(\mathbf{s}) + b))/2$, where $f(\cdot)$ is the SVM prediction and the constants $a = \frac{3}{4}$ and $b \approx -\frac{3}{4}$ were found to be optimal on the validation set (depending on the kernel).

5. *Rule about the length of the translated region.*
We designed the HMM in a way such that only such exons are predicted whose lengths add up to a number that is a multiple of 3.

For a given splicing S of a gene into exons $\mathbf{e}_1, \ldots, \mathbf{e}_E$ and introns $\mathbf{i}_1, \ldots, \mathbf{i}_{E-1}$ we can compute the *score* of S, $\hat{p}(S)$, as follows:

$$p^{gc}(\mathbf{e}_1)p^{fs}(\mathbf{e}_1)p_e^l(\mathbf{e}_1)\prod_{j=2}^{E}\hat{p}_{don}^{svm}(\mathbf{ei}_j)\; p_i^{gc}(\mathbf{i}_j)p_i^l(\mathbf{i}_j)\; \hat{p}_{acc}^{svm}(\mathbf{ie}_j)\; p_e^{gc}(\mathbf{e}_j)p^{fs}(\mathbf{e}_j)p_e^l(\mathbf{e}_j), \tag{13.15}$$

where $\mathbf{ei}_j$ is the donor site sequence at the boundary of the $(j-1)$th exon and $(j-1)$th intron and $\mathbf{ie}_j$ is the sequence on the border of the $(j-1)$th intron and the jth exon. Using standard techniques (Viterbi algorithm; cf. Durbin et al., 1998) we can now find the most likely sequence of exons and introns which forms our prediction. Note that the model above is not normalized in a probabilistic sense, as the sum over all paths do not sum to 1. Moreover, since the score tends to decrease exponentially with the number of exons it cannot directly be used for predicting the number of exons.

All parameters and probabilities have been estimated using the training and validation set, but not the test set.

13.3.3 Predicting the Number of Exons

One of the most important things to know about the gene is the number of exons. If this prediction is wrong, all subsequent steps have to fail. There are several ways to estimate the number of exons. The common approach is to choose the most likely number of exons for a probabilistic model like the above, this may require some additional normalization terms in (13.15). In this chapter we follow a different approach, using the SVM predictions only.

Our prediction works as follows: We start by assuming that the gene has E exons (starting with two) and compute the optimal splicing as described in subsection 13.3.2. Then we sum up the SVM scores for all donor and acceptor sites, that is, $s_E = \sum_{j=2}^{E} \left(f_{don}(\mathbf{ei}_j) + f_{acc}(\mathbf{ie}_j) \right)$. This score is compared with the score s_{E+1} for $E+1$ exons. If $s_E \geq s_{E+1} + \alpha$, then one predicts E exons. Otherwise, one increases E and repeats the loop. Here α is a parameter of the algorithm and controls whether one tends to predict too few or too many exons.

The rule is indeed very simple, but it works surprisingly well – as we will see later. It might have a biochemical interpretation, in that the splicing process stops in the first local minimum of some energy – assuming the SVM outputs are related to the energy.

13.4 Results and Discussion

In this section we discuss the experimental results obtained with our methods on the left-out set of genes (cf. appendix). First we only consider the accuracy of identifying a single donor and acceptor site. For the best methods we then compare our splice predictions with the ones of GenScan (Burge and Karlin, 1997).

13.4.1 Accuracy of Single Donor and Acceptor Predictions

Setup and Model Selection We start with generating true splice and decoy examples for training, validation, and test genes as described in the appendix. Here, we choose a window of ± 30 nt around the site with the consensus `AG` or `GT` dimer centered. A similar window length has been used in previous studies (cf. Sonnenburg, 2002; Sonnenburg et al., 2002). To be able to apply the SVM, we have to find the complexity parameter C, controlling the tradeoff between training error and complexity, and the kernel parameters. For instance, in the case of the LI kernel this is the degree d and window size l. To select these hyper parameters, we train the SVM on the training set and evaluate it on the validation set for different settings of C, d, and l. We tried $C = [0.25, 0.5, 0.75, 1, 2, 5, 10, 20]$, $d = 1, \ldots, 5$ and $l = 1, \ldots, 6$. For acceptor predictions we found $C = 0.75$, $d = 4$, and $l = 3$ as optimal parameters. For donor predictions $C = 1$, $d = 3$, and $l = 2$ are optimal. We omit the details on the model selection for the other methods, but they are

Table 13.1 Classification error and ROC score for Markov models, TOP kernel, SVM-pairwise, polynomial kernel, locality improved kernel, and weighted degree kernel. Note that the data sets are quite unbalanced: the constant classifier $g(\mathbf{x}) = -1$ would achieve an error of about 2.89%. Hence, the classification errors shown should be taken with care. Also, they depend heavily on the choice of the bias.

	Test Error		ROC Score	
	Donor	Acceptor	Donor	Acceptor
Markov	1.85%	1.54%	98.23%	98.88%
TOP	1.82%	1.66%	98.32%	98.70%
Pairwise	2.17%	1.94%	97.60%	98.00%
Polynomial	1.91%	1.53%	98.31%	98.95%
Locality	1.81%	1.44%	98.48%	99.08%
Weighted degree	1.79%	1.45%	98.47%	99.05%

available from the website `http://ida.first.fhg.de/splice` together with the data.

Generalization Performance For the selected hyper parameters we measure the performance on the test set. For acceptor predictions the test set contained 75,905 examples (2132 true sites). For donor predictions we have 73,784 test sequences (2132 true sites). In table 13.1 we summarize the results of our comparison. We show the classification error on the test set and also the receiver operating characteristic (ROC) score (the area under the ROC curve). We have a few observations:

- The TOP kernel slightly improves the simple probabilistic model for donor sites, but is (according to the ROC scores) not as good as, for instance, the SVM with LI or weighted degree kernel.
- The simple polynomial kernel performs surprisingly well, given that the feature space contains the correlation of all positions up to order 4 (in our case). It is only slightly worse than the specialized kernels such as the LI or weighted degree kernel.
- While the test errors for the MM are considerably greater than the ones for the polynomial-like kernels, the ROC score is very similar. For this reason we chose to include the Markov model in the GenScan comparison.
- The SVM-pairwise method did not perform well. This might be due to the small set of reference sequences. We tried to double the number but could not measure significant differences in performance.

We can conclude that the LI and the weighted degree kernel are best suited for the task of identifying single splice sites. The latter and the MM are chosen for the comparison with GenScan in section 13.4.3.

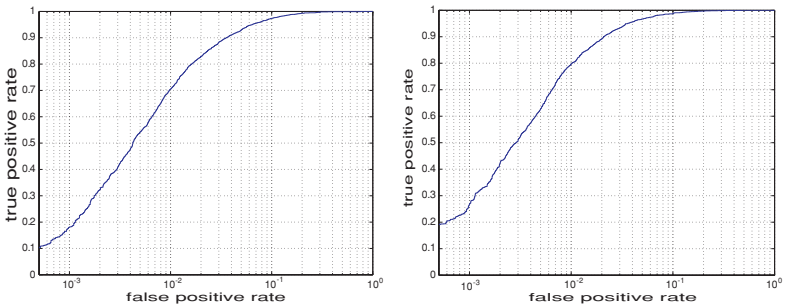


Figure 13.5 ROC curves for donor (*left*) and acceptor (*right*) predictions on the test set. The false-positive rate is plotted in log-scale.

In figure 13.5 we show ROC curves for acceptor and donor predictions of the SVM using the LI kernel. In Sonnenburg et al. (2002) and Sonnenburg (2002), we obtained considerably worse results for prediction on *C. elegans* using SVMs with the same kernel (e.g., only 97.3% accuracy for acceptor sites). In this case, the data were generated differently and the observed improvement suggests that the current data set is much cleaner and thus allows more accurate predictions. Also, in our previous study we have shown that our method is superior to a state-of-the-art method such as NN-BRAIN (Rampone, 1998). In future work we will compare our method with other prediction methods such as NetGene (Hebsgaard et al., 1996) and GeneSplicer (Pertea et al., 2001).

13.4.2 Splice Site Activity Variations for Different Intron Ranks

There seems to be evidence that the removal of introns occurs more or less sequentially in a certain order. In Lewin (2000) we found an example with seven introns where the fifth and sixth introns are removed first, then the fourth and seventh introns, followed by the first and second, and at last, by the third intron. It is conjectured that the third intron is often removed last, while the fifth or sixth intron is usually removed first. This was explained by the fact that after removing an intron the conformation of the mRNA is changed, other sites become available for splicing, and the conformation changes determine the order of intron removal.

Here we show that the order in which the introns are removed might rather be explained by the "activity" of the splice sites, in particular of the donor sites. In order to show this, we use the previously described SVMs for donor and acceptor sites and compute the median of all donor and acceptor scores of introns that appear at a certain position within the gene (figure 13.6). The rank of an intron is determined before assembling the virtual genes and hence is based on the true location within a gene. Since some introns might be missing, the rank might be estimated as too low (in particular for larger ranks). We chose the median to obtain a robust estimate.

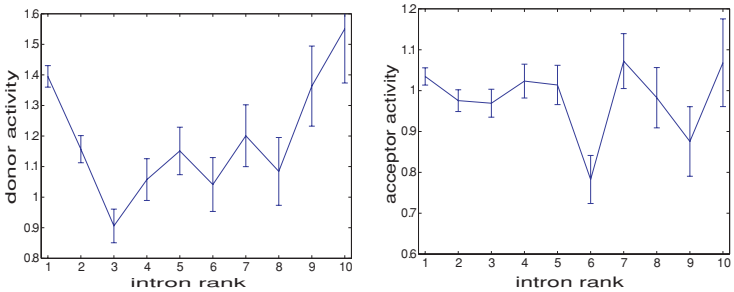


Figure 13.6 Median of the SVM predictions for donor (*left*) and acceptor (right) sites of introns with different ranks within a gene. The error bar indicates a confidence interval for the estimate.

We observe that the third donor site has indeed the lowest "activity" and one may need the most energy[5] to remove the third intron and hence the removal is slowest. Furthermore, we can quite accurately reproduce the previously described order of intron removal, when considering only the first seven intron positions: first, the seventh and second, then the fifth, fourth, and sixth, and at last the third intron. The removal of the second intron seems to be out of order. Additionally, the first intron's donor sites seem to have a particularly high score. However, according to Lewin (2000), it is not removed first. We conjecture that there are interactions with other processes, such as adding the cap structure, that delay the removal of the first intron (cf. Proudfoot et al., 2002). The same argument might apply to a lesser extent to the second intron. The order of the remaining introns matches reasonably well the order described in Lewin (2000). This suggests that also the biochemical activity of a site seems to determine the splicing order and not only the tertiary structure of the mRNA.

The activities of the acceptor sites stay reasonably constant for different ranks, except for two very weak positions: the sixth and the ninth introns' acceptors. The deviation is quite significant, but we have not yet found a biochemical explanation for it. Although one could conclude from these results that the acceptor sites only play a minor role in determining the splicing order, additional results on a new data set (not shown) suggest that the acceptor can also be important in determining the splicing order (cf. Rätsch and Sonnenburg, 2004).

5. Again, we assume that the SVM output is related to the biochemical activity of a site related to the needed energy of the biochemical reaction.

Table 13.2 Accuracies of GenScan and our methods: first using our exon predictions ($\hat{E}$) and second, for completeness, assuming knowledge of the correct number of exons (E). The line "all correct" states the fraction of genes for which all splice sites have been correctly predicted. The line "splice correct" denotes the fraction of correctly spliced genes given that the number of exons was predicted correctly. The line "exon—no. correct" states the fraction of genes for which the number of exons was correctly determined. Note that the top line is the product of the following two lines.

	GenScan	Markov model		LI Kernel		WD Kernel	
		$\hat{E}$	E	$\hat{E}$	E	$\hat{E}$	E
all correct	77.5%	90.0%	n/a	92.4%	n/a	**92.7%**	n/a
splice correct	87.7%	97.4%	96.1%	98.2%	97.9%	**98.7%**	**98.3**%
exon—no. correct	88.3%	92.4%	n/a	**94.0%**	n/a	93.9%	n/a

13.4.3 Comparison with GenScan

In a last experiment we compared our splice form prediction method as described in subsection 13.3.2 with a state-of-the-art method – GenScan. Since GenScan is designed to solve the harder task of finding complete genes, including promoters, initial and terminal exons, and so on, we needed to take special care that the experimental setup allowed a fair comparison. The idea is to run GenScan on all virtual genes in the test set and determine for which genes it found the correct start of the initial exon and the end of the terminal exon. All other genes were not used in this comparison. This procedure is necessary, since the start/end position is assumed to be available to our algorithm.

We used GenScan with the *Arabidopsis* genome setting, which seems a reasonable choice for *C. elegans*.[6] The original test set contains 1686 virtual genes of which GenScan finds the correct start and end position for 889 sequences. We say splice form is correctly predicted, if *all* exons and introns are predicted correctly. The prediction accuracy (i.e. the fraction of correctly predicted splice forms) of GenScan on this restricted set of genes is given in table 13.2.

To use our splice site prediction we first have to determine the number of exons as described in subsection 13.3.3. This method has the hyperparameter α, which we find by maximizing the prediction accuracy on the validation set. We find $\alpha = 1.7$ to be optimal for the LI and weighted degree-kernel. Given the number of exons, we employ the method described in subsection 13.3.2 using the tuned SVMs as in subsection 13.4.1. The results are given in table 13.2 (on the same reduced set of genes).

We found that our simple method for predicting the number of exons is quite accurate (93.9%) – given its simplicity, which corresponds to an error rate of less

6. All other available choices perform much worse and an optimized *C. elegans* setting is not available.

than half of that of GenScan. The achieved accuracy suggests that mainly the biochemical activity of the splice sites (estimated with the SVM score) determine whether an intron is removed. Only to a lesser extend does it depend on the tertiary structure (which would be very hard to predict from the short sequences given to the SVM) or other properties of the sequence.

The overall prediction accuracy of the best method is 92.7%, while GenScan performs much worse (77.5%). The improvement is mainly due to better splice prediction when given the correct number of exons (cf. table 13.2). For this part the improved predictions using SVMs enriched with some statistical information is crucial. Our statistical modeling is still quite basic and covers only a few of many statistical aspects. We conjecture that our predictions will be improved when using a more refined statistical model.

13.5 Conclusion

In this work we designed a new splice finder system that accurately predicts the splice form for *C. elegans* genes. A main achievement is a newly generated set of EST and complete cDNA confirmed virtual genes. Only by using this set of genes were we able to train and tune a support vector classifier which very accurately predicts whether a site is a true splice site or a decoy. We tested several kernels for SVMs and found that the LI kernel and the weighted degree kernel are best suited for the single site prediction. Enriched with statistical information such as intron and exon length statistics and a simple heuristic for predicting the number of exons, our splice finder system was able to translate the high accuracy on the single site prediction into a considerably improved accuracy on the splice form prediction: Our system makes only a third of the number of mistakes of GenScan (92.7% vs. 77.5% accuracy).

Note, however, that in this work we only considered virtual genes generated from ESTs and complete cDNA, i.e., they tend to be too short (some exons and introns at both ends may be missing). Moreover, we considered only the subset of genes for which GenScan detected the correct position of TIS and stop codon. This introduces an additional bias toward genes that are easy to predict.[7]

It is our goal to further improve our results. A better statistical modeling (e.g., normalization of the model) may lead to a better exon number prediction. Additionally, a cleaner data set will likely lead to accuracy improvements in the single site prediction. Moreover, we have not considered the prediction of the TIS and stop codon positions, for which a particularly designed support vector classifier might be beneficial when used within a gene finder. Other directions to consider are alternative splicing and noncanonical splice sites.

7. Recently we performed experiments on a newly generated set of complete genes on which GenScan and our method performes less accurately (our method performed much better thann GenScan though).

Acknowledgments

We particularly thank A. Pannek for great discussions and for proofreading the manuscript. Moreover, we thank S. Heymann, K. Tsuda, A. Zien, A. Zahler, C. Sugnet, K.-R. Müller, A. Smola, M. Warmuth, C. Leslie, and E. Eskin for stimulating discussions. This work was partially funded by DFG under contract JA 379/9-2, JA 379/7-2, MU 987/1-1, and NSF grant CCR-9821087 and supported by an award under the Merit Allocation Scheme of the National Facility of the Australian Partnership for Advanced Computing. Part of this work was done while G. R. was at the Australian National University in Canberra.

Appendix: Data Generation

Data generation proceeds in two steps: First, we generate a list of virtual genes from complete cDNA matches to the genomic sequence, and second, we derive true and decoy donor and acceptor splice examples from the virtual genes with known splice sites.

Generating Virtual Genes from EST and Complete cDNA Sequences

To generate a clean splice data set we started with the genomic sequence (*C. elegans* Sequencing Consortium, 1998) and the set of known EST and complete cDNA sequences (Benson et al., 1999) of *C. elegans*, which we downloaded from the Intronerator website (Kent and Zahler, 2000). We employed NCBI Blast (Altschul et al., 1990) to match all cDNA sequences to the genomic DNA (with default parameters) and obtained a list of matches, of which we only used the matches without gaps on the cDNA and with at least 80% identity. Using these matches we generated lists of introns, internal exons, and potential initial and terminal exons for each cDNA sequence as follows:

- **Introns** An intron is defined by a large gap on the DNA in the match of cDNA to DNA. We only considered gaps of length at most 500 knt. If cDNA matches to different positions on DNA, then we use the one leading to the largest fraction of used cDNA and the smallest gap on the DNA (preferring smaller introns). The nonmatching DNA sequence is assumed to be an intron and added to the intron list for the current cDNA sequence, if the consensus sequence `GT...AG` was found. (We only consider so-called canonical splice sites, i.e., sites with the consensus `GT` or `AG` at the splice site.)
- **Internal Exons** A match between cDNA and DNA is considered to be an internal exon if there are two adjacent introns in the intron list of the cDNA sequence.
- **Initial/terminal exons** If only one intron at the boundary of a cDNA match was found, then it is assumed to be an initial/terminal exon. The length of the exon is given by the cDNA match, but not all of it is part of the coding region. Hence,

we check for open reading frames and fix the length of the exon by cutting it at the position of the appearing stop codon (`TAA`, `TAG`, or `TGA`). However, the exact start/end position cannot be derived from the cDNA matches. We included this exon in the list of initial/terminal exons if it did not overlap with any internal exon on the same strand.

In the next step we removed redundancies by removing exact duplicates within the intron and internal exon lists. Furthermore we removed initial exons with identical ends and terminal exons with equal start positions. This left 24,191 introns and 41,245 exons (35% initial, 30% internal, 35% terminal). The number of introns derived by (Kent and Zahler, 2000) using different techniques is slightly larger.

For this study we were interested in generating a clean splice data set and therefore wanted to exclude the effects of alternative splicing. Thus, we removed introns and exons of all genes for which we could find an intron overlapping with an exon or an exon within an intron. We found evidence for alternative splicing for 2560 introns on 1228 genes (out of 9247 total). Finally, we ended up with 19,544 introns and 35,464 exons in 8019 remaining genes.

By considering all exons and introns of one gene, we can determine the positions of the introns relative to each other and compute ranks for each intron within the gene. Furthermore, we can check whether introns or exons are missing. Each contiguous sequence of maximal length and at least two exons (with matching introns) we call a *virtual gene,* of which we found 4413.

Unfortunately, it happens rather often that different cDNA sequences of one gene are labeled with different gene identifiers, matched to the same location on the DNA. Hence one will generate introns labeled with different gene identifiers in the first step, which are then omitted. Therefore, we may end up with contiguous introns/exons with different gene identifiers. This poses a problem for the above-described method and we miss a considerable amount of (virtual) genes. It will be possible, in the future, to find all matching sequences, independent of the labeling.

Each virtual gene may or may not start with an initial exon and end with a terminal exon. To properly define the start and end of the virtual gene, which will be needed later for comparison with GenScan, we remove the first half of the first and the last half of the last exon and append fixed sequences, which have been derived from gene *Y48G1C_55.c* on chromosome 1 of *C. elegans* as follows. We generated two sequences: (1) 571 nt upstream of the TIS (including the promoter) to 51 nt downstream and (2) 51 nt of the last exon to 783 nt downstream from the stop codon. This gene and these positions have been chosen such that GenScan finds the correct TIS and stop codon as often as possible for our set of genes.

Finally, we have a set of 4413 virtual genes with defined start and end positions, and with all splice sites known. We randomly chose 2153 for training, 574 for validation (tuning), and 1686 for testing.

Generating Donor and Acceptor Sites

From the sets of virtual genes with known splice sites we can straightforwardly derive positive training sequences for donor and acceptor sites. It is more difficult to come up with an appropriate set of negative sequences (decoys), which is needed for training a supervised learning algorithm. In Pertea et al. (2001) decoy sites were chosen randomly. In Reese et al. (1997) a window of ± 40 nt around true sites was used, from which a list of sequences containing the consensus dimer at the correct position was generated. However, the chosen window size is quite arbitrary and also has a great effect on the performance (cf. Sonnenburg, 2002). There are two additional problems: (1) One may generate decoy examples that can logically not be a boundary of an exon (since there is, e.g., no open reading frame) and (2) for long exons or introns one misses the majority of potential splice sites in the interior of the sequence and will therefore not be very successful in applying the splice detector in a gene finder.

Arguably, the best way to generate the decoy sequences for use in a gene finder would be to run the first pass of a gene finder. The result is a list of potential exons, which all have the property to start after the `AG` dimer (followed by an open reading frame; in our case at least 12 nt in length) and to end before the `GT` dimer. From this list one then generates the list of potential donor and acceptor sites, excluding the true sites and using the remaining sites as decoy examples. This way we obtain about 180,000 donor and 195,000 acceptor decoy examples and 8150 true donor/acceptor examples. Note that the above-described method does not require another gene finder, but only a list of potential exons with acceptor and donor sites that any other gene finder would also consider to find the splice form.

14 Gene Expression Analysis: Joint Feature Selection and Classifier Design

Balaji Krishnapuram
Lawrence Carin
Alexander Hartemink

Recently developed high-throughput technologies—including oligonucleotide arrays (Lockhart et al., 1996), DNA microarrays (Schena et al., 1995), and serial analysis of gene expression (SAGE, Velculescu et al., 1995)—enable us to simultaneously quantify the expression levels of thousands of genes in a population of cells. As one application of these technologies, gene expression *profiles* can be generated from a collection of cancerous and noncancerous tumor tissue samples and then stored in a database. Kernel methods like the *support vector machine* (SVM) and the *relevance vector machine* (RVM) have been shown to accurately predict the disease status of an undiagnosed patient by statistically comparing his or her profile of gene expression levels against a database of profiles from diagnosed patients (Golub et al., 1999; Furey et al., 2000; Alon et al., 1999; Ramaswamy et al., 2001; Li et al., 2002). Despite this early success, the presence of a significant number of irrelevant features—here genes in the profile that are unrelated to the disease status of the tissue—makes such analysis somewhat prone to the *curse of dimensionality*.

Intuitively, overcoming the curse of dimensionality requires that we build classifiers relying on information exclusively from the genes in the profile that are truly relevant to the disease status of the tissue. This problem of identifying the features most relevant to the classification task is known as *feature selection*. In this chapter, we review current methods of feature selection, focusing especially on the many recent results that have been reported in the context of gene expression analysis. Then we present a new Bayesian EM algorithm that jointly accomplishes the classifier design and feature selection tasks. By combining these two problems and solving them together, we identify only those features that are most useful in performing the classification itself. Experimental results are presented on several gene expression data sets. The biological significance of the genes identified by the method is also briefly assessed.

14.1 Common Issues in Classifying Gene Expression Profiles

For simplicity, we consider in this chapter the problem of diagnosing whether or not a patient has a specific disease, which can be viewed as a binary supervised classification or pattern recognition task. In situations where several diseases are possible, standard strategies for extending binary classifiers to multiclass problems may be used to generalize the analysis presented here.

Notation

Let us assume that we are presented with a database of m profiles each measuring the expression level of N genes, $X = \{\mathbf{x}_i \in \mathbb{R}^N\}_{i=1}^m$. Additionally, we are also provided with the corresponding set of class labels indicating the disease status, $Y = \{y_i \in \{-1, +1\}\}_{i=1}^m$. Assuming a parametric form for the functional relationship between $\mathbf{x}$ and the posterior probability of class membership as $P(y = 1|\mathbf{x}) = f_{\boldsymbol{\beta}}(\mathbf{x})$, during the training phase we seek to find the optimal parameters $\boldsymbol{\beta}$ based on the evidence of the training data, $D = \{X, Y\}$.

Data characteristics

Because of the significant cost and effort required to perform these experiments, currently available databases typically contain fewer than one hundred profiles, though each profile quantifies the expression levels of several thousand genes. Due to the high dimensionality and the small sample size of the experimental data, it is often possible to find a large number of classifiers that can separate the training data perfectly, but their diagnostic accuracy on unseen test samples is quite poor. In terms of risk, even when we design a classifier to minimize the empirical risk, $\hat{\boldsymbol{\beta}} = \text{argmin}_{\boldsymbol{\beta}} R_{\text{emp}}(f_{\boldsymbol{\beta}})$, the true risk, $R(f_{\hat{\boldsymbol{\beta}}})$, of the resulting classifier remains large. However, as demonstrated later in this chapter, when presented with the expression levels of only a small subset of diagnostically relevant genes, several methods of classifier design achieve equally good generalization. Thus, we may conclude that the choice of feature selection methods is often more important than the choice of classifier for gene expression analysis.

14.2 A Review of Feature Selection Methods for Kernel Machines

Definition

Let us assume that the genes are indexed by the variable $j \in \{1, 2, \ldots, N\}$. We can denote any subset of the genes by S where $S \subset \{1, 2, \ldots, N\}$. We define $X^{(S)}$ to be the database of expression profiles restricted to the genes contained in the subset S. Thus, $X^{(S)} = \{\mathbf{x}_i^{(S)} \in \mathbb{R}^{|S|}\}_{i=1}^m$, where $|S|$ denotes the cardinality of S. In feature selection, we are interested in identifying the subset of genes $\hat{S}$ whose expression levels are most relevant to classification or diagnosis. There are three principal reasons for our interest in feature selection:

Reasons for feature selection

1. We can improve the generalization performance—or out-of-sample accuracy—of our classifier by identifying only the genes that are relevant to the prediction of the disease diagnosis. This effect is attributable to the overcoming of the *curse of dimensionality* mentioned earlier.

2. If it is possible to identify a small set of genes that is indeed capable of providing complete discriminatory information, inexpensive diagnostic assays for only a few genes might be developed and be widely deployed in clinical settings.

3. Knowledge of a small set of diagnostically relevant genes may provide important insights into the mechanisms responsible for the disease itself.

Caveats

In considering this last point, it is necessary to clearly distinguish between the relevance of a gene to the biological mechanism underlying the disease and its relevance to building good diagnostic prediction algorithms. On the one hand, not all biologically relevant genes may need to be used when building accurate diagnostic classifiers; on the other hand, a high correlation between disease status and gene expression level does not necessarily imply that the expression of that particular gene has somehow *caused* the disease. So diagnostic relevance is neither a necessary nor a sufficient condition for biological relevance. Consequently, genes selected for their diagnostic relevance should be validated for biological relevance by follow-up studies of the literature or experimental analysis. Nevertheless, the fact remains that good feature selection methods can often serve as excellent guides to identifying small subsets of genes for further investigation.

Three approaches to feature selection

In the current literature three basic approaches to feature selection predominate (Blum and Langley, 1997; Kohavi and John, 1997): *filter*, *wrapper*, and *embedded*. Filter methods consider the problem of feature selection in isolation from the problem of classifier design; typically a subset of features is first selected in a preprocessing step, and then only the selected features are subsequently used to design a classifier. Thus, filter methods are independent of the technique for classifier design and they may be used in conjunction with any such algorithm. In contrast, wrapper methods search through the space of possible feature subsets and measure the quality of a particular subset S by estimating the accuracy of a classifier designed using only the features in S, either directly or indirectly (using theoretical bounds as approximations). Hence, wrapper methods search for optimal feature subsets for use with a specific classifier design algorithm. Finally, in embedded methods, feature selection and classifier design are accomplished jointly. We study several examples of all three methods below.

14.2.1 Filter Methods of Feature Selection

FDR

Perhaps the simplest filtering scheme is to evaluate each feature individually based on its ability to distinguish between the disease categories (i.e., ability to predict class labels). We may compute the *Fisher discriminant ratio* (FDR) of each gene j, as

$$FDR(j) = \frac{(\mu_+^{(j)} - \mu_-^{(j)})^2}{(\sigma_+^{(j)})^2 + (\sigma_-^{(j)})^2}, \tag{14.1}$$

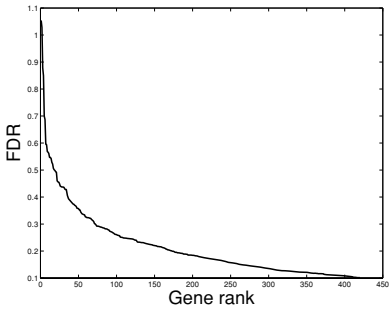


Figure 14.1 Plot of Fisher discriminant ratio of genes with FDR values at least 10% of the value of the gene with the highest FDR value. The genes have been sorted by decreasing FDR value.

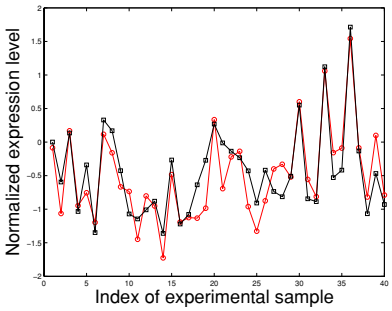


Figure 14.2 Normalized expression levels of genes ranked 3 and 11 according to FDR value. For perceptual clarity, only the expression level on the tumor samples is shown. Again for perceptual clarity, only two genes are depicted here, but the same effect is obtained for large clusters of genes.

where $\mu_+^{(j)}$, $\mu_-^{(j)}$, $\sigma_+^{(j)}$, and $\sigma_-^{(j)}$ represent the class-conditional means and standard deviations, respectively, for the expression level of gene j. Under the assumption that the class-conditional density functions are Gaussian, larger values of FDR(j) suggest that gene j is better able to distinguish between classes when used as the single independent predictor of disease status. Consequently, selecting genes with the highest FDR values is often employed as a simple technique for feature selection.

To study the limitations of this method, let us consider the gene expression data provided by Alon et al. (1999) where the aim is to distinguish colon tumors from normal cells. Figure 14.1 shows the FDR values of genes from this data set, sorted by decreasing FDR value. Note that the expression levels of over 400 genes are well

correlated with the diagnostic class labels, as determined by FDR. This presents far too large a set to reasonably consider if the importance of each of these genes to the disease mechanism must be experimentally validated, or if interactions between the genes are to be explored using automated computational network inference methods.

Redundant features

In looking more closely at these 400 genes, we uncover a great deal of redundancy. For example, consider figure 14.2, which shows the expression profiles of the 3rd and 11th ranked genes. These two genes are highly correlated and thus their expression profiles provide similar information. In fact, large subsets of the genes with high FDR values exist that encode essentially the same information from the point of view of classifier design, implying a heavy redundancy of information among the most prominent features. As a result, considering all the genes with high FDR values provides little more information than that obtained using the single highest gene. Certainly, from the perspective of understanding the disease mechanism, it is important to distinguish between genes whose expression level is not correlated with the diagnosis (i.e., irrelevant features) and sets of genes which are highly correlated with each other (i.e., redundant features). However, from the perspective of classifier design, both of these provide potential problems. Hence, we may conclude that while FDR may identify a large number of relevant genes, the identified set likely has heavy redundancy. This suggests that more insightful feature selection methods may be able to identify much smaller sets of relevant genes that are nevertheless equally effective in diagnosing the disease status of a patient.

PCA and other projection techniques

One commonly suggested mechanism for addressing this redundancy problem is to first identify a low-dimensional space such that the projections of the samples onto that space are linear combinations of the original features. This is accomplished using principal component analysis (PCA) or linear discriminant analysis, and has been used with some success in gene expression analysis, for example, by Khan et al. (2001). Unfortunately, since the new predictors are now linear combinations of all of the genes at once, we no longer have the advantage of requiring few experimental measurements for inexpensive clinical deployment and we lose most of the mechanistic insights as well. Similar problems are also associated with several nonlinear extensions of these projection methods.

14.2.2 Wrapper Methods of Feature Selection

The main limitation of methods that select features independently of one another—like FDR—is that they are typically designed under the assumption of independent and thus nonredundant features. This limitation can be largely overcome by considering feature selection as a search over the space of all possible feature subsets. Exploiting the natural partial ordering properties of the space of subsets, we can either start with an empty set and successively add features, or start with the set of all features and successively remove them. The former approach is referred to as forward selection while the latter is referred to as backward elimination; note that a combination of the two approaches is also possible.

Forward selection

As an example of forward feature selection, using any classifier design algorithm, we might first look for the single most discriminative feature. We could then look for the single additional feature that gives the best class discrimination *when considered along with the first feature* (note the difference between this approach and that of FDR). We could keep augmenting the feature set iteratively in this greedy fashion until cross-validation error estimates are minimized; if adding additional features leads to lower estimates of generalization performance, we stop. As an example, Xiong et al. (2001) used this kind of greedy forward feature selection approach along with a Fisher linear discriminant classifier to analyze gene expression data for tumor classification. Note that this approach requires a large number of classifiers to be built, but the cost of building each such classifier is quite small since the number of features considered is not very large at any point.

Backward elimination

In backward elimination, we start by building a classifier using all the features. Assessing the relative importance of the features in the resulting classifier, we iteratively eliminate the least important features and again build a classifier based on the remaining feature subset. Since the final subset may be only a small fraction of the size of the original feature set, we can accelerate the elimination process by removing large sets of irrelevant features in each iteration. This reduces the number of intermediate classifiers that need to be built during the process. Even with such an optimization, however, each of these iterations may still require a large amount of effort since the feature set remains large for several iterations. Therefore kernel classifiers are a natural choice in this framework since their computational requirements scale well to high feature dimensionality.

By varying the algorithm used for classifier design and the criterion used to measure feature relevance, a family of related backward elimination methods have been developed for gene expression analysis; we use the remainder of this subsection to examine a number of them.

Wrapper methods based on minimization of error bounds

The leave-one-out error L is an unbiased estimator of the generalization performance of classifiers. Hyperplane classifiers like the SVM are often of the form

$$f(\mathbf{x}; \mathbf{w}, b) = (\mathbf{w} \cdot \mathbf{x}) + b. \tag{14.2}$$

For classifiers of this form, the following radius/margin bound for L is well known (Vapnik, 1995):

$$L \leq 4R^2 \|\mathbf{w}\|_2^2. \tag{14.3}$$

where R is the radius of the smallest sphere in the kernel-induced feature space that contains all the data, and $\mathbf{w}$ specifies the hyperplane classifier identified by the SVM in that space. Assuming that R changes little with the choice of features, minimizing the margin $\|\mathbf{w}\|_2$ should result in a tighter bound for L. Guyon et al. (2002) use this intuition as the basis for assessing the importance of the features at each iteration of their wrapper method. The result is an algorithm they term *recursive feature elimination* (RFE) that performs feature selection by iteratively training an SVM classifier with the current set of genes and removing the genes with

the smallest weight in the resulting hyperplane. In related work, Weston et al. (2000) attempt feature selection by minimizing (14.3), using a gradient descent algorithm on the feature scaling factors rather than eliminating them. Retaining the backward elimination approach of Guyon et al. (2002), Rakotomamonjy (2003) discusses two generalizations of these algorithms. First, use of a tighter span bound on L as compared to (14.3) is considered. Second, for both (14.3) and the span bound, the magnitude of the gradient of the error bounds with respect to the feature scale is used as a measure of feature relevance. Intuitively, removing the features with the smallest gradient has the least effect on the generalization error bound. Hence in each iteration an SVM classifier is computed, the features are ranked according to the gradients of the error bounds, and the least significant features are removed.

Comparison of wrapper methods

Among these bound minimization algorithms for feature selection with the SVM, the RFE has empirically been observed to achieve the best results on classification tasks using gene expression data (Rakotomamonjy, 2003). Zhu and Hastie (2003) have shown that RFE with a *penalized kernel logistic regression* (PKLR) classifier in place of an SVM classifier achieves classification accuracy equal to that of RFE with the SVM, but with two additional benefits: RFE with the PKLR tends to find an even more parsimonious feature subset than RFE with the SVM, and this approach also provides posterior probabilities of class membership. In other related work, Ambroise and McLachlan (2002) have found that the RFE achieves a maximum of about 3% improvement in classification error rates as compared to equivalent forward selection methods using the SVM classifier.

14.2.3 Embedded Methods of Feature Selection

Zero-norm minimization

For constructing hyperplane classifiers of the form (14.2), Weston et al. (2003b) provide an algorithm to approximately solve the following problem:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}\in\mathbb{R}^N}{\operatorname{argmin}} \; \lambda \|\mathbf{w}\|_0 + \|\boldsymbol{\xi}\|_0 \,, \tag{14.4}$$

subject to

$$y_i \left((\mathbf{w}\cdot\mathbf{x}_i) + b\right) \geq 1 - \xi_i.$$

Recall that the l_0 norm of a vector—denoted as $\|\cdot\|_0$ above—is equal to the number of non-zero elements in it. Thus, the algorithm of Weston et al. (2003b) tries to jointly minimize a weighted sum of the number of features used in the hyperplane classifier and the number of misclassifications on the training data. As a consequence, feature selection is incorporated fundamentally into the algorithm for classifier design. Though the choice of the l_0 norm constitutes an extreme example of feature selection enforced as part of classifier design, several recently developed algorithms for classifier design have incorporated regularizers based instead on the l_1 norm: $\|\mathbf{w}\|_1 = \sum |\mathrm{w}_i|$, which is also often referred to as the lasso penalty.

The family of l_1 regularized algorithms share interesting theoretical properties and relationships. A theorem from Mangasarian (1999) establishes the equivalence

between l_p margin maximization and l_q distance maximization where $\frac{1}{p} + \frac{1}{q} = 1$. Using this result, Rosset et al. (2003) have shown that one-norm SVM, exponential boosting, and l_1-regularized logistic regression all converge to the *same* nonregularized classifier in the limit as $\lambda \to 0$. This classifier is shown to be a hyperplane that maximizes the l_∞ distance from the closest points on either side. Friedman et al. (2004) consider situations that are not uncommon in gene expression analysis: for example, $m = 100$ and $N = 10,000$. They argue that in a sparse scenario where only a small number of true coefficients w_i are non-zero, the l_1 margin regularizer works better than the normal l_2 margin used in penalized logistic regression, SVM, and so on; in the nonsparse scenario, neither regularizer fits coefficients well due to the curse of dimensionality. Based on these observations, they propose the *bet on sparsity principle* for high-dimensional problems which encourages using the l_1 penalty.

One-norm SVM

One example of an l_1 regularized method is the one-norm SVM of Fung and Mangasarian (2002):

$$\hat{\mathbf{w}} = \underset{\mathbf{w}\in\mathbb{R}^N}{\operatorname{argmin}} \lambda \|\mathbf{w}\|_1 + \|\boldsymbol{\xi}\|_1 , \tag{14.5}$$

subject to

$$y_i \left((\mathbf{w} \cdot \mathbf{x}_i) + b\right) \geq 1 - \xi_i.$$

Sparse logistic regression

Replacing the hinge loss function of the SVM in (14.5) with the logistic loss function leads to an objective function that is the *maximum a posteriori* (MAP) solution of logistic regression with a Laplacian prior (Roth, 2003):

$$\begin{aligned} \hat{\mathbf{w}} &= \underset{\mathbf{w}\in\mathbb{R}^N}{\operatorname{argmin}} \lambda \|\mathbf{w}\|_1 + \sum_{i=1}^{m} \log\left(1 + \exp\left(-y_i f\left(\mathbf{x}_i; \mathbf{w}\right)\right)\right) \\ &= \underset{\mathbf{w}\in\mathbb{R}^N}{\operatorname{argmax}} \frac{1}{\lambda} \exp\left(-\lambda \|\mathbf{w}\|_1\right) \prod_{i=1}^{m} \frac{1}{\left(1 + \exp\left(-y_i f\left(\mathbf{x}_i; \mathbf{w}\right)\right)\right)}. \end{aligned} \tag{14.6}$$

Sparse probit regression, RVM, JCFO

Other closely related algorithms for sparse hyperplane classifier design may be obtained by changing the logistic regression to the probit regression (Figueiredo, 2003), or by changing the Laplacian prior to a Student's t prior (Tipping, 2001). The latter is the basis of the RVM. All these variations provide comparable feature selection and classification accuracy. While all the embedded methods described in this section can be used to perform joint feature selection and classifier design in the context of simple hyperplane classifiers, they are not able to accomplish joint feature selection and classifier design in the context of nonlinear kernel classifiers. In the next section we derive a new algorithm for *joint classifier and feature optimization* (JCFO) that extends these methods to accomplish joint feature selection and classifier design in the context of nonlinear kernel classifiers.

14.3 The Joint Classifier and Feature Optimization Algorithm

Basic intuition behind JCFO

The basic intuition behind our approach to solving the feature selection and classifier design problems jointly is to extend the set of parameters to be learned: we will estimate not only the weight parameters $\boldsymbol{\alpha}$ associated with basis functions but also a vector of (non-negative) scaling factors $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_N]^T$ associated with the N features. Consequently, we consider functions of the form

$$f(\mathbf{x}) = \Phi\left(\boldsymbol{\alpha}^T \boldsymbol{h}_{\boldsymbol{\theta}}(\mathbf{x})\right) = \Phi\left(\alpha_0 + \sum \alpha_i \, \psi_i(\mathbf{x}, \boldsymbol{\theta})\right), \tag{14.7}$$

where $\Phi(z)$ is the probit link function; $\boldsymbol{\alpha}$ is a vector of weights $[\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_k]^T$; and $\boldsymbol{h}_{\boldsymbol{\theta}}(\mathbf{x}) = [1, \psi_1(\mathbf{x}, \boldsymbol{\theta}), \ldots, \psi_k(\mathbf{x}, \boldsymbol{\theta})]^T$; and $\psi_i(\mathbf{x}, \boldsymbol{\theta}) : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ are (possibly nonlinear) basis functions. Please note that $\Phi(z)$ is *not* used in this chapter as the map into feature space induced by some kernel; rather $\Phi(z) = (2\pi)^{-1/2} \int_{-\infty}^{z} \exp\left(-x^2/2\right) dx$.

Kernel basis functions

Although our formulation allows arbitrary basis functions, we shall focus on the important case where $\psi_i(\mathbf{x}, \boldsymbol{\theta}) = k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)$ is some symmetric Mercer kernel function (Cristianini and Shawe-Taylor, 2000), parameterized by $\boldsymbol{\theta}$. Accordingly, the dimension of both $\boldsymbol{\alpha}$ and $\boldsymbol{h}_{\boldsymbol{\theta}}(\mathbf{x})$ is $m + 1$. The only condition we place on the kernel $k_{\boldsymbol{\theta}}$ is that its dependence on $\boldsymbol{\theta}$ is such that smaller values of θ_j correspond to smaller influence of $x^{(j)}$ and $x_i^{(j)}$ in $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)$; in particular, if $\theta_j = 0$, then $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)$ must not depend on $x^{(j)}$ or $x_i^{(j)}$. Examples of such functions are scaled Gaussian kernels, $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i) = \exp\{-(\mathbf{x} - \mathbf{x}_i)^T \operatorname{diag}(\boldsymbol{\theta}) (\mathbf{x} - \mathbf{x}_i)\}$, and nth order polynomial kernels, $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i) = (1 + \mathbf{x}^T \operatorname{diag}(\boldsymbol{\theta}) \mathbf{x}_i)^n$.

Bayesian learning of parameters

Our algorithm may be characterized as a Bayesian approach to learning the weight parameters $\boldsymbol{\alpha}$ and the scaling $\boldsymbol{\theta}$. Accordingly, in the next section we define priors over both $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ that reflect our *a priori* belief that most of the elements of these vectors are identically zero. Subsequently, we jointly estimate the maximum MAP values of $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ using an expectation maximization algorithm.

Related work

It should be noted that Seeger (2000) and Williams and Barber (1998) also attempt similar aims of joint feature scale identification and classifier design in a Bayesian setting; however, those methods have not been applied to problems in gene expression analysis. Despite similar objectives, the three methods differ significantly in their algorithmic details; we only derive the JCFO hereafter. Though we do not have experimental evidence to prove it, we expect all three methods to perform comparably and enjoy similar benefits.

14.3.1 Sparsity-Promoting Priors and Their Hierarchical Decomposition

Laplacian priors promote sparsity

We seek to find classifiers (i.e., to estimate $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$) that not only predict class probabilities accurately but also do so with few non-zero elements in either $\boldsymbol{\alpha}$ or $\boldsymbol{\theta}$. When using a kernel basis function formulation, sparsity in $\boldsymbol{\alpha}$ corresponds to finding a small subset of training samples that are representative of the classes (as in Tipping, 2001; Figueiredo and Jain, 2001), while sparsity in $\boldsymbol{\theta}$ corresponds to

implicit feature selection. As already pointed out in subsection 14.2.3, it is well-known that sparsity can be encouraged by a Laplacian prior (see, e.g., Tibshirani, 1996).

For the prior on $\boldsymbol{\alpha}$, we have $p(\boldsymbol{\alpha}|\eta) \propto \exp(-\eta \|\boldsymbol{\alpha}\|_1) = \prod_j \exp(-\eta |\alpha_j|)$, where η is a hyperparameter whose choice will be addressed below. In the case of $\boldsymbol{\theta}$, since the parameters are all non-negative, we have $p(\boldsymbol{\theta}|\nu) \propto \exp(-\nu \|\boldsymbol{\theta}\|_1) = \prod_l \exp(-\nu\theta_l)$, for all $\theta_l \geq 0$, zero otherwise.

Hierarchical priors

MAP estimates of $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ cannot be found in closed form. To circumvent this difficulty, we consider the following two-level hierarchical model. Each α_i is given a zero-mean Gaussian prior with its own variance τ_i: $p(\alpha_i|\tau_i) = N(\alpha_i|0, \tau_i)$. Further, the variances τ_i have independent exponential hyperpriors, $p(\tau_i|\gamma_1) \propto \exp(-\gamma_1\tau_i/2)$, for $\tau_i \geq 0$. The effective prior can be obtained by integrating out τ_i:

$$p(\alpha_i|\gamma_1) = \int_0^\infty p(\alpha_i|\tau_i)p(\tau_i|\gamma_1)d\tau_i \propto \exp(-\sqrt{\gamma_1}\,|\alpha_i|) \tag{14.8}$$

showing that a Laplacian prior is equivalent to a two-level hierarchical model characterized by zero-mean Gaussian priors with independent exponentially distributed variances. For each parameter θ_i, since it is non-negative, we adopt non-negative Gaussian priors:

$$p(\theta_i|\rho_i) = \begin{cases} 2N(\theta_i|0, \rho_i) & \text{if } \theta_i \geqslant 0 \\ 0 & \text{if } \theta_i < 0 \end{cases} \tag{14.9}$$

and again the ρ_i have independent exponential hyperpriors: $p(\rho_i|\gamma_2) \propto \exp(-\gamma_2\rho_i/2)$, for $\rho_i \geqslant 0$. The effective prior on θ_i is thus exponential, as desired:

$$p(\theta_i|\gamma_2) \propto \begin{cases} \exp(-\sqrt{\gamma_2}\,\theta_i) & \text{if } \theta_i \geqslant 0 \\ 0 & \text{if } \theta_i < 0 \end{cases} \tag{14.10}$$

14.3.2 The JCFO algorithm

Missing data interpretation of probit link

The hierarchical decomposition of the Laplacian priors just described opens the door to the use of an EM algorithm for computing the estimates of $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ by treating the τ_i and ρ_i as missing data. Furthermore, the probit link allows an interpretation in terms of hidden variables that facilitates the derivation of such an EM algorithm (Figueiredo and Jain, 2001; Albert and Chib, 1993). Specifically, let $z(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\theta}) = \boldsymbol{\alpha}^T \boldsymbol{h}_{\boldsymbol{\theta}}(\mathbf{x}) + \epsilon$, where ϵ is a zero-mean unit-variance Gaussian random variable. If the classifier is defined as $y = \text{sign}(z(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\theta}))$, then we recover the probit model, since

$$P(y = 1|\mathbf{x}) = P\left(\boldsymbol{\alpha}^T \boldsymbol{h}_{\boldsymbol{\theta}}(\mathbf{x}) + \epsilon > 0\right) = \Phi\left(\boldsymbol{\alpha}^T \boldsymbol{h}_{\boldsymbol{\theta}}(\mathbf{x})\right). \tag{14.11}$$

Given the data $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, consider the corresponding vector of missing variables $\mathbf{z} = [z_1, \ldots, z_m]^T$, as well as the vectors of missing variables $\boldsymbol{\tau} = [\tau_1, \ldots, \tau_{m+1}]^T$ and $\boldsymbol{\rho} = [\rho_1, \ldots, \rho_N]^T$. If $\mathbf{z}$, $\boldsymbol{\tau}$, and $\boldsymbol{\rho}$ were known, we would

have an easier estimation problem for $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$: We would effectively have the observation model $\mathbf{z} = \boldsymbol{H}_{\boldsymbol{\theta}}\boldsymbol{\alpha} + \boldsymbol{\epsilon}$, with Gaussian priors on $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ (variances given by $\boldsymbol{\tau}$ and $\boldsymbol{\rho}$), where $\boldsymbol{H}_{\boldsymbol{\theta}} = [\boldsymbol{h}_{\boldsymbol{\theta}}(\mathbf{x}_1), \boldsymbol{h}_{\boldsymbol{\theta}}(\mathbf{x}_2), \ldots, \boldsymbol{h}_{\boldsymbol{\theta}}(\mathbf{x}_m)]^T$ is the design matrix, and $\boldsymbol{\epsilon}$ is a vector of independent and identically distributed zero-mean unit-variance Gaussian random variables. This suggests using an EM algorithm to find a local maximum of the posterior $p(\boldsymbol{\alpha}, \boldsymbol{\theta}|D)$.

EM algorithm

The EM algorithm will produce a sequence of estimates $\widehat{\boldsymbol{\alpha}}^{(t)}$ and $\widehat{\boldsymbol{\theta}}^{(t)}$ by alternating between two steps:

- **E-step.** Conditioned on D and on the current estimates $\widehat{\boldsymbol{\alpha}}^{(t)}, \widehat{\boldsymbol{\theta}}^{(t)}$, compute the expected value of the complete log-posterior, $p(\boldsymbol{\alpha}, \boldsymbol{\theta}|D, \mathbf{z}, \boldsymbol{\tau}, \boldsymbol{\rho})$, denoted as $Q\left(\boldsymbol{\alpha}, \boldsymbol{\theta} \;\middle|\; \widehat{\boldsymbol{\alpha}}^{(t)}, \widehat{\boldsymbol{\theta}}^{(t)}\right)$:

$$Q\left(\boldsymbol{\alpha}, \boldsymbol{\theta} \;\middle|\; \widehat{\boldsymbol{\alpha}}^{(t)}, \widehat{\boldsymbol{\theta}}^{(t)}\right) = \int p\left(\mathbf{z}, \boldsymbol{\tau}, \boldsymbol{\rho} \;\middle|\; D, \widehat{\boldsymbol{\alpha}}^{(t)}, \widehat{\boldsymbol{\theta}}^{(t)}\right) \log p(\boldsymbol{\alpha}, \boldsymbol{\theta}|D, \mathbf{z}, \boldsymbol{\tau}, \boldsymbol{\rho})\, d\mathbf{z}\, d\boldsymbol{\tau}\, d\boldsymbol{\rho}$$

- **M-step.** Update the estimates to $\left(\widehat{\boldsymbol{\alpha}}^{(t+1)}, \widehat{\boldsymbol{\theta}}^{(t+1)}\right) = \underset{\boldsymbol{\alpha}, \boldsymbol{\theta}}{\operatorname{argmax}}\, Q\left(\boldsymbol{\alpha}, \boldsymbol{\theta} \;\middle|\; \widehat{\boldsymbol{\alpha}}^{(t)}, \widehat{\boldsymbol{\theta}}^{(t)}\right)$.

E-step

As shown in the appendix, the E-step reduces to the following three analytical expressions:

$$v_i \equiv \mathbf{E}\left[z_i \;\middle|\; D, \widehat{\boldsymbol{\alpha}}^{(t)}, \widehat{\boldsymbol{\theta}}^{(t)}\right] = \boldsymbol{h}_{\boldsymbol{\theta}}^T(\mathbf{x}_i)\widehat{\boldsymbol{\alpha}}^{(t)} + \frac{y_i\, N\left(\boldsymbol{h}_{\boldsymbol{\theta}}^T(\mathbf{x}_i)\widehat{\boldsymbol{\alpha}}^{(t)} \mid 0, 1\right)}{\frac{(1+y_i)}{2} - y_i\, \Phi\left(-\boldsymbol{h}_{\boldsymbol{\theta}}^T(\mathbf{x}_i)\widehat{\boldsymbol{\alpha}}^{(t)}\right)} \tag{14.12}$$

$$\omega_i \equiv \mathbf{E}\left[\tau_i^{-1} \;\middle|\; D, \widehat{\boldsymbol{\alpha}}_i^{(t)}, \gamma_1\right] = \gamma_1 \left|\widehat{\boldsymbol{\alpha}}_i^{(t)}\right|^{-1} \tag{14.13}$$

$$\delta_i \equiv \mathbf{E}\left[\rho_i^{-1} \;\middle|\; D, \widehat{\boldsymbol{\theta}}_i^{(t)}, \gamma_2\right] = \gamma_2 \left(\widehat{\boldsymbol{\theta}}_i^{(t)}\right)^{-1} \tag{14.14}$$

Defining vector $\boldsymbol{v} = [v_1, \ldots, v_m]^T$ and matrices $\boldsymbol{\Omega} = \operatorname{diag}(\omega_1, \ldots, \omega_{m+1})$, and $\boldsymbol{\Delta} = \operatorname{diag}(\delta_1, \ldots, \delta_N)$, the Q function can be written as

$$Q\left(\boldsymbol{\alpha}, \boldsymbol{\theta} \;\middle|\; \widehat{\boldsymbol{\alpha}}^{(t)}, \widehat{\boldsymbol{\theta}}^{(t)}\right) = -\boldsymbol{\alpha}^T \boldsymbol{H}_{\boldsymbol{\theta}}^T \boldsymbol{H}_{\boldsymbol{\theta}} \boldsymbol{\alpha} + 2\, \boldsymbol{\alpha}^T \boldsymbol{H}_{\boldsymbol{\theta}}^T \boldsymbol{v} - \boldsymbol{\alpha}^T \boldsymbol{\Omega}\, \boldsymbol{\alpha} - \boldsymbol{\theta}^T \boldsymbol{\Delta} \boldsymbol{\theta}. \tag{14.15}$$

M-step

As for the M-step, since $\boldsymbol{H}_{\boldsymbol{\theta}}$ is generally neither linear nor quadratic in $\boldsymbol{\theta}$, Q cannot be maximized analytically w.r.t. $\boldsymbol{\theta}$. Moreover, the optimizations w.r.t. $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ cannot be pursued independently. However, we observe that given any $\boldsymbol{\theta}$, the optimal $\boldsymbol{\alpha}$ is simply

$$\widehat{\boldsymbol{\alpha}}_{\boldsymbol{\theta}}^{(t+1)} = (\boldsymbol{\Omega} + \boldsymbol{H}_{\boldsymbol{\theta}}^T \boldsymbol{H}_{\boldsymbol{\theta}})^{-1} \boldsymbol{H}_{\boldsymbol{\theta}}^T \boldsymbol{v} = \boldsymbol{\kappa}(\boldsymbol{I} + \boldsymbol{\kappa} \boldsymbol{H}_{\boldsymbol{\theta}}^T \boldsymbol{H}_{\boldsymbol{\theta}} \boldsymbol{\kappa})^{-1} \boldsymbol{\kappa} \boldsymbol{H}_{\boldsymbol{\theta}}^T \boldsymbol{v}, \tag{14.16}$$

where $\boldsymbol{\kappa} = \gamma_1^{-1/2} \operatorname{diag}\left(\sqrt{\left|\widehat{\alpha}_1^{(t)}\right|}, \ldots, \sqrt{\left|\widehat{\alpha}_{m+1}^{(t)}\right|}\right)$ is a matrix introduced to enable a stable numerical implementation, since the sparsity-promoting properties of the hierarchical priors will drive several of the α_i to zero. Since we can maximize analytically w.r.t. $\boldsymbol{\alpha}$, we are left with the maximization w.r.t. $\boldsymbol{\theta}$ of $Q(\widehat{\boldsymbol{\alpha}}_{\boldsymbol{\theta}}^{(t+1)}, \boldsymbol{\theta}|\widehat{\boldsymbol{\alpha}}^{(t)}, \widehat{\boldsymbol{\theta}}^{(t)})$.

Here, we are forced to employ numerical optimization to obtain $\widehat{\boldsymbol{\theta}}^{(t+1)}$; in the results presented below, we use conjugate gradient. Note that our model assumes that each $\theta_i \geq 0$; this can be accomplished by the reparameterization $\theta_i = \exp(\zeta_i)$, and solving for each ζ_i instead.

Computational complexity

The computational complexity of the algorithm just outlined remains moderate for problems with m on the order of a few hundred and N on the order of a few thousand, due to the use of some simple approximations (Krishnapuram et al., 2003). For all of the gene expression data sets analyzed in the next section, we could complete the training of a classifier in under half an hour on a 800 MHz Pentium III machine with unoptimized MATLAB code. The computational bottleneck is clearly the matrix inversion in (14.16), which becomes impractical for large m; this problem is endemic among kernel methods.

14.4 Experimental Studies Comparing the Methods

Two types of experiments

In order to assess the success of the JCFO and to compare its performance against other popular classifiers we performed two types of experiments. In the first experiment, we synthetically generated data with a large number of irrelevant features, in order to study the effect of overfitting on several algorithms. In the second series of experiments, we used measured gene expression data and compared the methods based on their cross-validation estimate of error rates. The genes identified by the JCFO were also evaluated against the known literature for their biological significance. These experiments are outlined below in more detail.

In keeping with standard practice on kernel classifiers, in all experiments the data sets were mean-centered and normalized to unit variance. The hyperparameters γ_1 and γ_2 were adjusted by using a hold-out test set of 10% of the data. The chosen values were then used with the entire data set to obtain the classifiers.

14.4.1 Effect of Irrelevant Predictor Variables on Kernel Classifiers

To assess the extent to which different state-of-the-art kernel classifiers are affected by the presence of irrelevant predictor variables, we generated N-dimensional Gaussian data from two classes with means $\mu_1 = -\mu_2 = [1/\sqrt{2}, 1/\sqrt{2}, 0, 0, \ldots, 0]^T$ with both covariances identity matrices. Independent of N, the Bayes error rate is $\Phi(-1|0,1) \simeq 0.1587$ and the optimal classifier uses only the first two variables. All algorithms were trained with 100 samples per class, and the accuracy of the learned classifier was tested on an independent test set of 1000 samples. Results were averaged over 20 random draws of the training set, and the procedure repeated for several feature dimensions N. The average error rates plotted in figure 14.3 show that the JCFO algorithm is far more resistant to the presence of irrelevant variables than the other methods considered: SVM, RVM, and sparse probit (Figueiredo and Jain, 2001). Among the kernel methods compared in this

experiment, only the JCFO has the ability to identify the scale of the input features, so it is quite understandable that the JCFO is largely immune to the presence of irrelevant features. Furthermore, for almost all random draws of the training set, the JCFO reported non-zero scaling factors θ_i for only the two relevant features. The performance degradation of the other methods in the presence of irrelevant features shows explicitly that feature selection is crucial, even for large margin techniques like the SVM.

14.4.2 Cancer Diagnosis Using Gene Expression Data

Benchmarks

Next, we considered two publicly available gene expression data sets that have been analyzed by several authors, though each has adopted a slightly different experimental setup. These data sets are characterized by small training sets (a few tens) but very high feature dimensionality (several thousands). The first one, originally studied by Golub et al. (1999), contains expression values of $N = 7129$ genes from $m = 72$ samples of two classes of leukemia: 47 of acute myeloid leukemia (AML) and 25 of acute lymphoblastic leukemia (ALL). The second data set, originally analyzed by Alon et al. (1999), contains expression levels of $N = 2000$ genes from 40 tumor and 22 normal colon tissues. Both data sets contain a large number of redundant and irrelevant features (genes). Strong feature selection is thus both possible and desirable.

Diagnostic accuracies of diagnostic classification schemes are commonly assessed using a cross-validation scheme. In an leave-one-out cross-validation (LOOCV), the accuracy of diagnostic prediction on each sample is assessed based on classifiers built using the remaining $m-1$ samples as a training set. Table 14.1 reports the LOOCV results for the JCFO and several other learning methods, including Adaboosting, the SVM, the RVM, logistic regression, and sparse probit regression.

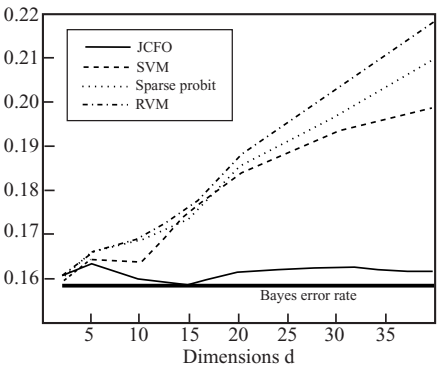


Figure 14.3 Effect of irrelevant features: error rates of most kernel classifiers increase as irrelevant features are introduced, but the JCFO is largely immune.

Table 14.1 LOOCV accuracy of cancer diagnosis based on gene expression data for several classifiers (percent correct; a higher number is better).

Classifier	**AML/ALL**	**Colon tumor**
Adaboost (decision stumps) (Ben-Dor et al., 2000)	95.8	72.6
SVM (quadratic kernel) (Ben-Dor et al., 2000)	95.8	74.2
SVM (linear kernel) (Ben-Dor et al., 2000)	94.4	77.4
RVM (linear kernel)	94.4	80.6
RVM (no kernel: on feature space)	97.2	88.7
Logistic regression (no kernel: on feature space)	97.2	71.0
Sparse probit regression (quadratic kernel)	95.8	84.6
Sparse probit regression (linear kernel)	97.2	91.9
Sparse probit regression (no kernel: on feature space)	97.2	85.5
JCFO (quadratic kernel)	98.6	88.7
JCFO (linear kernel)	100.0	96.8

Table 14.2 Cross-validation estimates of accuracy of cancer diagnosis on colon tumor gene expression data for several classifiers, reproduced from results reported in the literature (percent corect; a higher number is better; see text for details of specific experimental setup in each case).

Classifier	**Colon tumor**
RVM (no kernel: on feature space) (Li et al., 2002)	85.4
SVM (RFE) (Weston et al., 2003b)	87.5
Zero-norm minimization (Weston et al., 2003b)	88.9
SVM (radius/margin gradient) (Rakotomamonjy, 2003)	88.9

Results from the literature

For further perspective, the mean cross-validation error rates reported in a number of papers in the literature on the colon data are reproduced in table 14.2. Slightly different preprocessing has been adopted in certain cases by different authors, leading to small differences in quoted results of those methods; most important, the number of training and testing samples used has a significant impact on the estimate of variance associated with the reported accuracy. Li et al. (2002) split the available 62 samples into 50 used in training their method and 12 test samples used to obtain unbiased estimates of their accuracy. They repeat this process for 100 random draws of the training set. Weston et al. (2003b) use the same split but repeat their experiments for 500 random draws of the training set; hence their mean error rates should be comparable, even though their estimate of variance of error rates may not be directly comparable.

The JCFO typically identified around 25 features (genes) as important for the classification task. In contrast, using the RVM or the sparse probit algorithm with

Table 14.3 Most important genes for distinguishing between AML and ALL, as selected by the JCFO (Krishnapuram et al., 2004)

θ_i	**Gene Name**	**Gene Description**
1.14	MPO	myeloperoxidase
0.83	$HOXA9$	homeo box A9
0.81	$APOC1$	apolipoprotein C-I
0.77	$PPGB$	protective protein for β-galactosidase (galactosialidosis)
0.70	$NME4$	nonmetastatic cells 4, protein expressed in
0.67	$PTGS2$	prostaglandin-endoperoxide synthase 2
0.56	$CD171$	human CD171 protein
0.51	$NEK3$	NIMA (never in mitosis gene a)-related kinase 3
0.46	$CST3$	cystatin C (amyloid angiopathy and cerebral hemorrhage)
0.42	$EPB72$	erythrocyte membrane protein band 7.2(stomatin)
0.42	DF	D component of complement (adipsin)
0.41	$PTMA$	prothymosin α (gene sequence 28)
0.41	$HSPA8$	heat shock 70-kDa protein 8
0.40	$CYP2C18$	cytochrome P-450, subfamily IIC, polypeptide 18
0.35	$CD33$	CD33 antigen ($gp67$)
0.34	$PRG1$	proteoglycan 1, secretory granule
0.33	$SERPING1$	serine (or cysteine) proteinase inhibitor, clade G, member 1
0.32	$FTH1$	ferritin, heavy polypeptide 1
0.30	$ALDH1A1$	aldehyde dehydrogenase 1 family, member A1
0.29	$LTC4S$	leukotriene C4 synthase
0.27	$MYBL1$	$v-myb$ myeloblastosis viral oncogene homolog (avian)-like 1
0.26	$ITGA2B$	α_{2b} integrin, alpha 2b (platelet glycoprotein IIb, antigen CD41B)
0.23	$MACMARCKS$	macrophage myristoylated alanine-rich C kinase substrate

no kernel to perform feature selection (as discussed in subsection 14.2.3) resulted in classifiers based on around 100 genes. More critically, for the RVM, each time the training set was changed in the LOOCV procedure, the genes identified as relevant to the discrimination changed significantly. In other words, this method was not stable to slight changes in the data set, rendering its results less biologically trustworthy.

Biological relevance of identified genes

Table 14.3 lists the genes identified by the JCFO as being most important for making a diagnostic decision in distinguishing between AML and ALL. The reported values of $\boldsymbol{\theta}$ in these tables were obtained by taking the mean of the $\boldsymbol{\theta}$ obtained for each of the classifiers designed in the LOOCV. Almost all genes selected by the JCFO are of known relevance to the AML/ALL distinction. In particular, *CST3*, *CD33*, *DF*, *HOXA9*, *LTC4S*, *PRG1*, *CTSD*, and *EPB72* were all determined both by the JCFO and in Golub et al. (1999) to be predictive of AML. In addition, the JCFO revealed myeloperoxydase (MPO) to be of paramount importance (though it was not uncovered in Golub et al., 1999)). MPO is known to occur in virtually all cells of the myeloid lineage and none of the lymphoid lineage;

Table 14.4 LOOCV accuracy of breast cancer diagnosis based on gene expression data for several classifiers (percent correct; a higher number is better; see text for descriptions of the data sets).

Classifier	Duke ER	Duke LN	Lund
SVM (linear kernel)	97.4	78.9	87.9
RVM (linear kernel)	94.7	92.1	88.5
RVM (no kernel)	89.5	81.6	96.5
Sparse probit regression (linear kernel)	97.4	89.5	86.2
Sparse probit regression (no kernel)	84.2	89.5	96.5
JCFO (linear kernel)	97.4	94.7	98.3

antibodies to MPO are used as clinical determinants of AML. Many others genes selected by the JCFO are known to play a role in myeloid/lymphoid differentiation, and a few novel genes have been identified as well. Similar results hold for the case of the colon data as well.

Breast cancer data

We also examined three different breast cancer data sets. The first was a Duke University study in which $m = 38$ breast tumors were classified based on estrogen receptor (ER) status. The second was a Duke University study in which the same $m = 38$ breast tumors were classified based on lymph node (LN) involvement status. The third was a set of $m = 58$ breast tissues collected by researchers at Lund University in Sweden. To accelerate the time required to complete the full LOOCV for all of the methods, and to give as much performance benefit as possible to the other classification methods that suffer more from the curse of dimensionality, the three breast cancer data sets were pared in advance to include only the 2000 most relevant genes as determined by FDR. Restricting the set of available genes in this way does not improve the accuracy of the JCFO because it is designed to perform feature selection as part of its optimization, but the smaller set of relevant initial features does improve the accuracy of the other methods.

In table 14.4, we present a full leave-one-out cross-validation study for each of the three data sets to compare the accuracy of the diagnostic classification reported by the JCFO against that of the SVM, the RVM, and sparse probit regression. We consider only kernels that seemed to perform reasonably well in the previous tests (as shown in table 14.1).

14.5 Discussion

An analysis of tables 14.1 and 14.4 suggests that for disease diagnosis tasks based on gene expression data, the JCFO provides classification accuracy superior to that of other popular methods like the SVM, RVM, and sparse probit regression. The statistical significance of this difference is unclear, though the fact that the JCFO performs well on a wide range of different tasks may be suggestive. Indeed, while

we have not presented results on other low-dimensional data sets widely used as benchmarks for classification, similar results have been obtained there as well. This may be understandable since the JCFO has two different and complementary kinds of regularization, one based on sparse choice of kernel basis functions and the other on feature selection.

Also, table 14.2 seems to indicate that a variety of modern embedded feature selection methods such as the RVM, RFE, radius/margin bound minimization, and zero-norm minimization methods seem to perform comparably in terms of error rates of the learned classifier, and the differences between all of them are statistically small. However, the latter conclusion is somewhat tenuous due to the limited number of experimental results available for analysis; we are currently attempting to run those methods on more data sets to gather more statistical evidence. The relatively high classification accuracy obtained by all of these methods, including the JCFO, is indicative of the fact that gene expression data provide an excellent basis for distinguishing between disease classes.

Besides improved predictive accuracy in cross-validation experiments, the JCFO also provides the posterior probability of class membership as opposed to a hard classification decision as provided by the SVM, though this property is common to several other methods as well (like Gaussian processes). It tends to be more parsimonious in identifying only a small number of diagnostically relevant genes. We have also found that most of the genes have typically been implicated in earlier results published in the literature.

14.6 Availability of Software

MATLAB implementations of sparse probit regression, RVM, JCFO, and forward feature selection with the SVM can be obtained by emailing the first author. The code is provided freely for noncommercial use.

MATLAB implementations of zero-norm minimization, RFE, and radius/margin bound minimization methods are publicly available in the Spider library from: `http://www.kyb.tuebingen.mpg.de/bs/people/spider/index.html`.

MATLAB implementations of the one-norm SVM software are publicly available from: `http://www.cs.wisc.edu/dmi/svm/`.

C++ implementations of the Generalized-LASSO, an algorithm similar to the RVM, are available from: `http://www.informatik.uni-bonn.de/~roth/GenLASSO/`.

S-Plus implementations of the RFE for penalized kernel logistic regression for academic research use were obtained by contacting Zhu and Hastie (2003) directly.

Acknowledgments

The authors thank the editors for their gracious invitation to contribute a chapter to this volume. They also gratefully acknowledge the assistance of Pallavi Pratapa

in generating some of the JCFO classification results and Mario Figueiredo for helpful comments regarding the text.

Appendix: Derivation for Q-Function and E-Step

As a first step, we see that the complete log-posterior on the learning parameters $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$, including the hidden variables $\boldsymbol{\tau}$, $\boldsymbol{\rho}$, and $\mathbf{z}$, is

$$\begin{aligned}\log P(\boldsymbol{\alpha},\boldsymbol{\theta}|\mathbf{y},\mathbf{z},\boldsymbol{\rho},\boldsymbol{\tau}) &\propto \log P(\mathbf{z}|\boldsymbol{\alpha},\boldsymbol{\theta}) + \log P(\boldsymbol{\alpha}|\boldsymbol{\tau}) + \log P(\boldsymbol{\theta}|\boldsymbol{\rho}) \\ &\propto -\|\boldsymbol{H}\boldsymbol{\alpha}-\mathbf{z}\|^2 - \boldsymbol{\alpha}^T\boldsymbol{T}\boldsymbol{\alpha} - \boldsymbol{\theta}^T\boldsymbol{R}\boldsymbol{\theta} \\ &\propto -\mathbf{z}^T\mathbf{z} - \boldsymbol{\alpha}^T\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{\alpha}-2\mathbf{z}) - \boldsymbol{\alpha}^T\boldsymbol{T}\boldsymbol{\alpha} - \boldsymbol{\theta}^T\boldsymbol{R}\boldsymbol{\theta},\end{aligned} \tag{14.17}$$

where the matrix $\boldsymbol{T} = diag(\tau_1^{-1},\tau_2^{-1},\ldots,\tau_M^{-1})$ and $\boldsymbol{R} = diag(\rho_1^{-1},\rho_2^{-1},\ldots,\rho_d^{-1})$. Thus, the Q function is

$$\begin{aligned}Q\left(\boldsymbol{\alpha},\boldsymbol{\theta}\,\middle|\,\hat{\boldsymbol{\theta}}^{(t)},\hat{\boldsymbol{\alpha}}^{(t)}\right) = E\Big[&-\mathbf{z}^T\mathbf{z} - \boldsymbol{\alpha}^T\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{\alpha}-2\mathbf{z}) \\ &- \boldsymbol{\alpha}^T\boldsymbol{T}\boldsymbol{\alpha} - \boldsymbol{\theta}^T\boldsymbol{R}\boldsymbol{\theta}\,\Big|\,\mathbf{y},\hat{\boldsymbol{\alpha}}^{(t)},\hat{\boldsymbol{\theta}}^{(t)}\Big].\end{aligned} \tag{14.18}$$

Since we seek to maximize the Q-function w.r.t. $\boldsymbol{\alpha}$ in the EM algorithm, terms like $E\left[-\mathbf{z}^T\mathbf{z}\,\middle|\,\mathbf{y},\hat{\boldsymbol{\theta}}^{(t)},\hat{\boldsymbol{\alpha}}^{(t)}\right]$ that do not involve $\boldsymbol{\alpha}$ or $\boldsymbol{\theta}$ can be effectively ignored in the M-step, and thus are irrelevant in the E-step as well. Therefore, the Q-function simplifies to

$$\begin{aligned}Q\left(\boldsymbol{\alpha},\boldsymbol{\theta}\,\middle|\,\hat{\boldsymbol{\theta}}^{(t)},\hat{\boldsymbol{\alpha}}^{(t)}\right) = &-\boldsymbol{\alpha}^T\boldsymbol{H}^T\boldsymbol{H}\boldsymbol{\alpha} + 2\boldsymbol{\alpha}^T\boldsymbol{H}^T E\left[\mathbf{z}\,\middle|\,\mathbf{y},\hat{\boldsymbol{\theta}}^{(t)},\hat{\boldsymbol{\alpha}}^{(t)}\right] \\ &- \boldsymbol{\alpha}^T E\left[\boldsymbol{T}\,\middle|\,\mathbf{y},\hat{\boldsymbol{\theta}}^{(t)},\hat{\boldsymbol{\alpha}}^{(t)}\right]\boldsymbol{\alpha} - \boldsymbol{\theta}^T E\left[\boldsymbol{R}\,\middle|\,\mathbf{y},\hat{\boldsymbol{\theta}}^{(t)},\hat{\boldsymbol{\alpha}}^{(t)}\right]\boldsymbol{\theta}.\end{aligned} \tag{14.19}$$

The E-step thus simplifies to computing the expectations associated with each of these terms. Fortunately, each of these computations can be expressed in closed form, as shown below.

As for the term associated with the expectation of $\mathbf{z}$, we have

$$v_i = E\left[z^{(i)}\,\middle|\,\mathbf{y},\hat{\boldsymbol{\alpha}}^{(t)},\hat{\boldsymbol{\theta}}^{(t)}\right] = \begin{cases}\boldsymbol{h}^T(\mathbf{x}^{(i)})\hat{\boldsymbol{\alpha}}^{(t)} + \frac{N\left(\boldsymbol{h}^T(\mathbf{x}^{(i)})\hat{\boldsymbol{\alpha}}^{(t)}\middle|0,1\right)}{1-\Phi\left(-\boldsymbol{h}^T(\mathbf{x}^{(i)})\hat{\boldsymbol{\alpha}}^{(t)}\right)}, & \text{if } y^{(i)}=1 \\ \boldsymbol{h}^T(\mathbf{x}^{(i)})\hat{\boldsymbol{\alpha}}^{(t)} - \frac{N\left(\boldsymbol{h}^T(\mathbf{x}^{(i)})\hat{\boldsymbol{\alpha}}^{(t)}\middle|0,1\right)}{\Phi\left(-\boldsymbol{h}^T(\mathbf{x}^{(i)})\hat{\boldsymbol{\alpha}}^{(t)}\right)}, & \text{if } y^{(i)}=0,\end{cases} \tag{14.20}$$

which follows from the observation that $z^{(i)}$ is distributed as a Gaussian with mean $\boldsymbol{h}^T(\mathbf{x}^{(i)})\hat{\boldsymbol{\alpha}}^{(t)}$, but left-truncated at zero if $y^{(i)}=1$, and right-truncated at zero if $y^{(i)}=0$. This expression can be simplified further to remove the case statement, as shown earlier in (14.12).

After some further algebraic manipulations, it can be shown that for the expectation of τ_i^{-1} is given by

$$\omega_i = E\left[\tau_i^{-1} \;\middle|\; \mathbf{y}, \hat{\boldsymbol{\alpha}}_i^{(t)}, \gamma_1\right] = \frac{\int_0^\infty \tau_i^{-1} P(\tau_i|\gamma_1) P\left(\hat{\boldsymbol{\alpha}}_i^{(t)} \;\middle|\; \tau_i\right) d\tau_i}{\int_0^\infty P(\tau_i|\gamma_1) P\left(\hat{\boldsymbol{\alpha}}_i^{(t)} \;\middle|\; \tau_i\right) d\tau_i} = \gamma_1 \left|\hat{\boldsymbol{\alpha}}_i^{(t)}\right|^{-1}. \tag{14.21}$$

The last term in the E-step computation is associated with the expectation of $\boldsymbol{R}$, and a manipulation similar to that above yields the following:

$$\delta_i = E\left[\rho_i^{-1} \;\middle|\; \mathbf{y}, \hat{\boldsymbol{\alpha}}^{(t)}, \hat{\boldsymbol{\theta}}_i^{(t)}, \gamma_2\right] = \gamma_2 \left(\hat{\boldsymbol{\theta}}_i^{(t)}\right)^{-1}. \tag{14.22}$$

15 Gene Selection for Microarray Data

Sepp Hochreiter
Klaus Obermayer

In this chapter we discuss methods for gene selection on data obtained from the microarray technique. Gene selection is very important for microarray data (a) as a preprocessing step to improve the performance of classifiers or other predictors for sample attributes; (b) in order to discover relevant genes, that is, genes which show specific expression patterns across the given set of samples; and (c) to save costs, for example, if the microarray technique is used for diagnostic purposes. We introduce a new feature selection method based on the support vector machine (SVM) technique. The new feature selection method extracts a sparse set of genes, whose expression levels are important for predicting the class of a sample (e.g., "positive" vs. "negative" therapy outcome for tumor samples from patients). For this purpose the support vector technique is used in a novel way: instead of constructing a classifier from a minimal set of most informative samples (the so-called support vectors), the classifier is constructed using a minimal set of most informative features. In contrast to previously proposed methods, however, features rather than samples now formally assume the role of support vectors. We introduce a protocol for preprocessing, feature selection, and evaluation of microarray data. Using this protocol we demonstrate the superior performance of our feature selection method on data sets obtained from patients with certain types of cancer (brain tumor, lymphoma, and breast cancer), where the outcome of chemo- or radiation therapy must be predicted based on the gene expression profile. The feature selection method extracts genes (the so-called support genes) which are correlated with therapy outcome. For classifiers based on these genes, generalization performance is improved compared to previously proposed methods.

15.1 Introduction

Gene expression profiles obtained by the microarray technique provide a snapshot of the expression values of up to some 10000 genes in a particular tissue sample. The advantage of the microarray method—namely to monitor a large number of variables of a cell's (or a piece of tissue's) state, however, often turns out to be difficult to exploit. The number of samples is small and the level of noise is high, which makes it difficult to detect the small number of genes relevant to the task at hand. Therefore, specific gene selection methods must be designed to reliably extract relevant genes.

15.1.1 Microarray Technique

The microarray technique (Southern, 1988; Lysov et al., 1988; Drmanac et al., 1989; Bains and Smith, 1988) is a recent technique which allows monitoring of the concentration of many kinds of messenger RNA (mRNA) simultaneously in cells of a tissue sample and provides a snapshot of the pattern of gene expression at the time of preparation (Wang et al., 1998; Gerhold et al., 1999). The so-called DNA microarrays allow for the first time the simultaneous measurement of up to 10,000 expression levels providing valuable information about whole genetic networks. DNA microarrays allow a search for genes related to certain properties of the sample tissue and extraction of related genes via dependencies in their expression pattern.

Figure 15.1 depicts the microarray procedure. mRNA is extracted from the samples (step 1) and reverse-transcribed to complementary DNA (cDNA) (step 2). This "target" cDNA is then coupled to a fluorescent dye (step 3). The target cDNA is then hybridized with a large number of probes of immobilized DNA (steps 4 and 5) which had been synthesized and fixed to different locations of the DNA chip during fabrication. The cDNA from the samples binds to their corresponding probes on the chip (step 5). After cleaning, the chip is scanned with a confocal microscope and the strength of the fluorescent light is recorded (step 6). Genes which are predominantly expressed in the sample give rise to bright spots of strong fluorescent light. No expression is indicated by weak fluorescent light. After segmentation of the stained locations on the chip and a correction for background intensity, intensity values are transformed to real numbers for every location (step 7). After processing, the data from several experiments with different samples are collected and represented in matrix form, where columns correspond to tissue samples, rows correspond to genes, and matrix entries describe the result of a measurement of how strong a particular gene was expressed in a particular sample.

Microarray noise

Expression values as measured by the DNA microarray technique are noisy. First, there exists biological noise, because samples do not show the same "expression state" and exactly the same levels of mRNA even if they belong to the same class or the same experimental condition. Then there is noise introduced by the microarray measurement technique. Sources of noise include tolerances in chip properties

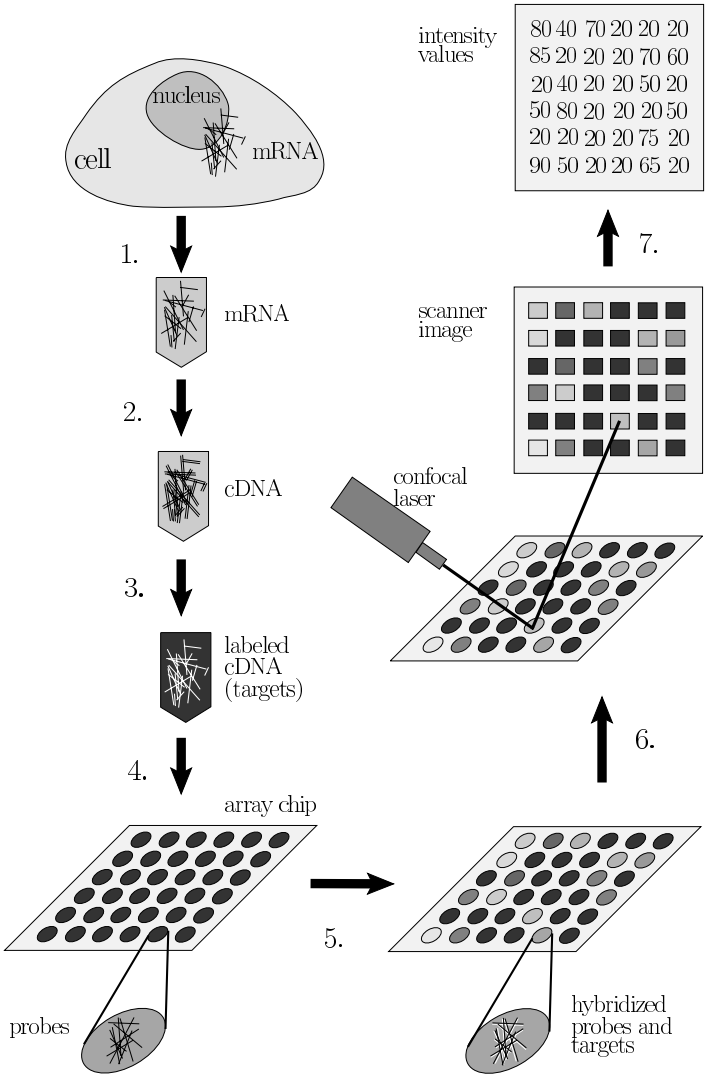


Figure 15.1 The microarray technique (see text for explanation).

which originate from the fabrication process, different efficiencies for the mRNA extraction and reverse transcription process, variations in background intensities, nonuniform labeling of the cDNA targets (the dye may bind multiple times and with different efficiencies), variations in dye concentration during labeling, pipette errors, temperature fluctuations and variations in the efficiency of hybridization, and scanner deviations. The effect of measurement noise can be reduced by averaging over multiple measurements using the same sample but it usually remains large. Measurement noise is not always Gaussian. Hartemink et al. (2001), for example, found that the measurement noise distribution of the logarithmic expression values has heavy tails.

15.1.2 Gene Selection for Microarrays

Gene selection aims at three goals:

Why gene/feature selection?

- Data preprocessing to improve the prediction quality of machine learning approaches
- Identification of indicator genes (this would aid in the interpretation and understanding of the data)
- Reducing costs, if microarray data are used, for example, for diagnostic purposes.

Data preprocessing This is an important issue in machine learning if the input dimension is larger than the number of samples. Kohavi and John (1997), for example, found that decision tree approaches like ID3 (Quinlan, 1986), CART (Breiman et al., 1984), and C4.5 (Quinlan, 1993), as well as instance-based (e.g., K-nearest neighbor) methods, degrade in performance when the number of features is larger than a minimal set of relevant features. The Naive-Bayes method is reported to be more robust to irrelevant features but the prediction accuracy decreases if correlated features are present. Also, Kittler (1986) observed decreasing performance of machine learning methods for large feature sets.

Curse of dimensionality

The reduction in performance for data sets with many attributes is known as the "curse of dimensionality" (Bellman, 1961). According to Stone (1980), the number of training examples has to increase exponentially with the number of dimensions to ensure that an estimator also performs well for higher-dimensional data. Otherwise overfitting (high variance in model selection) occurs, that is, fitting of the selected model to noise in the training data. On the other hand, if the model class is chosen to be smooth so that the variance of model selection is restricted (low overfitting), then underfitting (high bias of model selection) occurs, that is, the training data are not approximated well enough. The latter is shown by Friedman (1997) who demonstrated for K-nearest neighbor classifiers that the curse of dimensionality leads to large bias. Practical applications confirm the theory: many input dimensions lead to poor generalization performance. Fewer features, on the other hand, should improve generalization for equal training error.

For microarray data the situation is especially difficult because the number of features (genes) is often more than 10 times larger than the number of examples (tissue samples). The high level of noise additionally complicates the selection of relevant genes of microarray data. Both facts, the large number of genes and the presence of noise, led Guyon et al. (2002) to state that "the features selected matter more than the classifier used" for DNA microarray data classification, a fact which will be confirmed by our analysis later on.

Microarray data require gene selection

Identification of Genes This refers to the identification of genes whose expression values change with the sample class. Genes which show different expression values in a control condition when compared to the condition under analysis are useful to differentiate between these conditions and should be extracted (see Jäger et al., 2003). The knowledge of the relevant genes can then be exploited in two ways. First, cellular mechanisms can be understood and active pathways may be identified. Second, target genes or target proteins for causing or avoiding conditions can be detected. In medical applications both kinds of information are highly relevant to diagnosis and drug design. Note, however, that the selection of genes for prediction and the selection of genes whose expression levels are correlated lead to different sets. Redundant sets of genes, which are the outcome of the latter task, may lead to a reduced performance of the former task. On the other hand, genes selected for the purpose of prediction may not include genes strongly correlated to each other in order to keep the number of features small.

Identify genes

Reducing Costs This refers to the costs of large scale microarray studies, for example, for diagnostic purposes. Small gene ensembles lead to cheaper chips (fewer probes on a chip), to savings in manpower (fewer experiments), and to easier interpretable experiments (Jäger et al., 2003).

Reducing costs

15.1.3 Feature Selection Methods to Extract Relevant Genes

In the previous subsection we stated three important reasons why it is necessary to reduce the number of genes, that is, to reduce the number of a sample's expression values obtained by the microarray technique. These expression values are considered as features of the sample in the field of machine learning. In order to reduce the number of genes and to select the most important genes, machine learning methods, called "feature selection" methods, must be applied. Because of the special structure of the microarray data, namely the large number of noisy features, not all previously proposed feature selection methods are suited for the analysis of microarray data. Feature selection methods should be able to cope with many features but few samples, to remove redundancies, and to consider dependencies of whole subsets of features.

To address both the suitability for and the performance on microarray data, this chapter consists of two parts. In the first, methodological, part we review previous approaches and then derive a feature selection method, which is particularly suited

to the peculiarities of microarray data. In the second, application, part we assess the algorithms' performance and provide benchmark results. More precisely, this chapter is organized as follows. In section 15.2 we review feature selection methods with respect to their ability to address the particular constraints of microarray data. Then we introduce the new feature selection method in section 15.3. In section 15.4 we describe a "gene selection protocol" which is then evaluated together with the new feature selection method in section 15.5. Benchmark results on microarray data are provided using several previously described approaches.

15.2 Review of Feature Selection Methods

For simplicity let us consider a classification task where the objects to classify are described by vectors with a fixed number of components (the features). The training set consists of vectors which are labeled by whether the according object belongs to a class or not and—again for reasons of simplicity—we assume that there are only two classes. Given the training data, a classifier should be selected which assigns correct class labels to the feature vectors. The goal of machine learning methods is not only to select a classifier which performs well on the training set but which also correctly classifies new examples, that is, which correctly predicts future events.

Feature selection vs. feature construction

There are two classes of preprocessing methods which are commonly used to improve machine learning techniques: *feature selection* and *feature construction* methods. Feature construction methods compute new features as a combination of the original ones and are often used for dimensionality reduction. Many popular methods for feature construction are based on linear combinations of the original features, that is, on projections of data points into low-dimensional spaces, like *projection pursuit* (e.g., see Friedman and Tukey, 1974; Friedman and Stuetzle 1981; Huber, 1985), *principal component analysis* (PCA; e.g., Oja, 1982; Jolliffe, 1986; Jackson, 1991), or *independent component analysis* (ICA; e.g., Cardoso and Souloumiac, 1993; Jutten and Herault, 1991; Bell and Sejnowski, 1995; Hochreiter and Schmidhuber, 1999; Hyvärinen et al., 2001). More recently, nonlinear feature construction algorithms based on kernel methods (Cristianini et al., 2002a) and the information bottleneck idea (Tishby et al., 1999; Tishby, 2001) have been proposed.

Feature selection methods, on the other hand, choose a subset of the input components which are supposed to be relevant to solving a task and leave it to a subsequent stage of processing to combine their values in a proper way.[1]. In the following we focus on feature selection, that is, on the task of choosing a subset of "informative" input components, that is, components which are relevant to predicting the class labels. The classifier is then selected using the reduced feature

1. This combination can also be done during feature selection.

vectors as the objects' description. Therefore, only feature selection techniques address the extraction of indicator genes and cost reduction.[2].

Overview feature selection

Review articles on feature selection have been published in a special issue on relevance of the journal *Artificial Intelligence* (see Kohavi and John, 1997; Blum and Langley, 1997) and a special issue on variable and feature selection of the *Journal of Machine Learning Research* (Guyon and Elisseeff, 2003) to which we refer the reader for more details. The book of Liu and Motoda (1998) also gives an overview of feature selection.

15.2.1 Feature Selection Using Class Attributes

This subsection gives a general overview of feature selection techniques which have been used for gene selection on microarray data, whereas the next subsection focuses on the more recently developed kernel-based methods. In both sections we consider feature selection techniques which extract features according to their dependencies with the sample classes; hence we assume that the class labels are available for the training set. Methods which exploit the additional information given by the class labels are in general superior to methods not using this information; features which can be removed without changing the conditional probability of class labels with respect to all features are irrelevant to classification. Without explicit class labels feature selection must be based on the distribution of feature values, for example, on entropy or saliency measures. Those methods are not considered here.

Feature ranking and subset selection

Feature selection methods perform either feature ranking or subset selection. In feature ranking an importance value is assigned to every feature while subset selection attempts to construct an optimal subset of features. While some feature ranking methods do not consider dependencies between features, subset selection methods usually do and may even include features which have low correlations with the class label if justified by a good classification performance. The latter usually happens if dependencies between features (and not between class label and a certain feature) are important for prediction. In those cases the selection of interacting features is important, but it is also difficult to achieve (see Turney, 1993a,b).

Feature selection methods fall into one of two categories (Langley, 1994; Kohavi and John, 1997; John et al., 1994; Das, 2001; Liu and Motoda, 1998; Liu and Setiono, 1996):

- Filter methods
- Wrapper methods

2. Note that the identification of indicator genes may not be fully addressed by feature selection approaches because redundant features are avoided and not all indicator genes are extracted. However, the missing genes can be extracted by correlation analysis in a subsequent step.

Filter methods

Filter methods Filter methods extract features whose values show dependencies with class labels without explicitly relying on a predictor (classifier). One example is statistical methods which compute the statistical dependencies between class labels and features and select features where the dependencies are strong. The calculation of dependencies is based on Pearson's correlation coefficient, Wilcoxon statistics, t-statistics, Fisher's criterion or signal-to-noise ratios (see Hastie et al., 2001; Golub et al., 1999; Furey et al., 2000; Tusher et al., 2001). Statistical methods are fast and robust, but assume certain data or class distributions and cannot recognize dependencies between features. In principle, statistical methods can serve as subset selection methods if the dependency between a whole feature subset and the class label is computed. For example, the mutual information between feature sets and class labels has been considered by Koller and Sahami (1996). However the number of possible subsets increases exponentially with the number of features, which makes these approaches unattractive. Therefore, the method in Koller and Sahami (1996) is only tractable if approximations are made.

Ranking methods, statistics and information theory

The "relief" methods (Kira and Rendell, 1992; Rendell and Kira, 1992; Kononenko, 1994; Robnik-Sikonja and Kononenko, 1997) are another approach to assigning relevance values to features. Values are assigned according to the average separation of data vectors belonging to different classes minus the average separation of data points belonging to the same class. The averages are computed by randomly selecting a data point and determining its nearest data points from the same class and the opposite class. The relief methods are fast, can detect feature dependencies, but—again—do not remove redundant features.

Subset selection methods

Combinatorial search procedures are able to remove redundant features from the selected set. These methods exhaustively test all feature subsets for their ability to separate the classes, that is, whether two training vectors have the same values on the selected feature subset but different class labels. After testing, the minimal subset necessary to predict the class label is chosen (e.g., FOCUS, Almuallim and Dietterich, 1991; or the probabilistic approach in Liu and Setiono, 1996). Combinatorial search methods, however, suffer from high computational costs and can only be applied to a small number of features. They are prone to overfitting through noise, but on the other hand they will find the best solution in the noiseless case. Another feature subset selection which—like FOCUS—searches for a minimal necessary feature subset to separate the classes is based on decision trees (Cardie, 1993). The decision tree is used for separating the classes but not as a classifier. This method, however, is not applicable to small training sets because only $\log_2 m$ features are selected if m training examples are available. Since the sample size for microarray data is usually below 100, only $\log_2 100 \approx 7$ genes are typically selected. These are too few genes.

Wrapper Methods Wrapper methods (see Kohavi and John, 1997; John et al., 1994) use a classifier as the objective function for the evaluation of a subset of features. The classifier is obtained through a model selection (training) method which minimizes the classification error on the training data. The classifier is then

Wrapper methods

used to compute the prediction error on a validation set. Typical classifiers are decision trees, for example, ID3 (Quinlan, 1986), CART (Breiman et al., 1984), and C4.5 (Quinlan, 1993), or instance-based classifiers like K-nearest neighbor.

Well-known wrapper methods are the nested subset methods which are based on greedy strategies like hill-climbing (e.g., SLASH, Caruana and Freitag, 1994; and the random mutation hillclimbing—random mutation of feature presence map—described in Skalak, 1994). Nested subset methods perform either "forward selection" (Cover and Campenhout, 1977) or "backward elimination" (Marill and Green, 1963). Forward selection works in the underfitting regimen. It starts from an empty set of features and adds features step by step leading to the largest reduction of the generalization error. Backward elimination, on the other hand, works in the overfitting regimen. It starts with the set of all features and removes unimportant features step by step to maximally reduce the generalization error. The major shortcoming of these methods is that they do not consider all possible combinations of features (Cover and Campenhout, 1977). If, for example, only the XOR of two features is important, these features would not be recognized by a forward selection procedure which adds only a single feature at a time. The backward selection procedure suffers from a similar problem. Assume that one feature conveys the information of two other features and vice versa. The best strategy would be to remove these two features to obtain a minimal set, but backward selection may keep these two features and remove the single one. Another problem of backward selection is to determine good candidate features for deletion because overfitting makes it hard to distinguish between label noise fitting and true dependencies with class labels.

Forward vs. backward selection

Hill-climbing and search methods

Other search strategies are computationally more expensive but explore more possible feature sets. Such search methods include beam and bidirectional search (Siedlecki and Sklansky, 1988), best first search (Xu et al., 1989), and genetic algorithms (Vafaie and De Jong, 1992, 1993; Bala et al., 1995).

15.2.2 Kernel-Based Methods

Recently, kernel based feature selection methods which use the SVM approach have shown good performance for feature selection tasks (see, e.g., the review in Guyon and Elisseeff, 2003). Kernel-based feature selection methods are emphasized in this subsection because they are especially suited for microarray data due to the fact that they have shown good results in high-dimensional data spaces and have been successfully applied to noisy data. These methods use either one of two feature selection techniques, which were already known in the field of neural networks:

Feature selection during or after learning

- Feature selection by pruning irrelevant features after a classifier has been learned
- Adding a regularization term to the training error which penalizes the use of uninformative features during learning a classification task

Feature Selection After Learning Guyon et al. (2002) proposed a feature selection method for support vector learning of linear classifiers where the features

with the smallest squared weight values are pruned after learning is complete. This method is a special case of the "optimal brain surgeon" (OBS, Hassibi and Stork, 1993) or "optimal brain damage" (OBD, LeCun et al., 1990) techniques for dependent (in the case of OBS) or independent (in the case of OBD) feature values under the assumption that feature values have variance 1. OBS is based on a Taylor expansion of the mean squared error around its minimum, and the increase in training for a pruned feature is estimated by the Hessian.[3] Intuitively, the support vector method corresponds to projecting the normal vector of the separating hyperplane into the subspace perpendicular to the less important directions. The features for which these values are lowest are then deleted. Guyon et al. (2002) also describe an iterative version of this feature selection procedure where the feature with the smallest absolute weight is removed after each SVM optimization step. This method is then called "recursive feature elimination" (RFE) and is an example of backward elimination of features. It has recently been extended by Rakotomamonjy (2003) to nonlinear kernels. Note, however, that these methods, which prune features after learning, cannot detect redundant features and they are sensitive to outliers.

Recursive feature elimination

Feature Selection During Learning Regularization techniques have been proposed for SVMs to improve prediction performance by selecting relevant features. The first set of techniques directly favors SVMs with sparse weight vectors. This can be done by using the 1-norm in the SVM objective function, a technique known as the linear programming (LP) machine (Schölkopf and Smola, 2002; Smola et al., 1999; Frieß and Harrison, 1998). This approach leads to many zero components of the weight vector and to the removal of the corresponding features. In Bradley and Mangasarian (1998) and Bi et al. (2003), these methods are utilized together with backward elimination. In Bradley and Mangasarian (1998) the 0-norm of the weight vector is considered as an objective to select a classifier. The 0-norm counts the non-zero components of the weight vector which leads to a discrete and NP-hard optimization problem. Approximations can be made but they are sensitive to the choice of parameters (see Weston et al., 2003b) and the optimization is still computationally complex in high dimensions. Weston et al. (2003b) propose an improved approximation of the 0-norm, which reduces to a method which iteratively solves 1-norm SVMs and adjusts scaling factors for the different features. In Perkins et al. (2003) both the 0-norm and the 1- or 2-norm are used for feature selection, where the 1- or 2-norm serves for regularization and the 0-norm selects features.

1-norm SVMs

0-norm SVMs

The second set of techniques is based on the proper choice of scaling factors for the different features. Weston et al. (2000) applies scaling factors to the 2-norm SVM approach (R2W2). Two phases are performed iteratively. First the SVM is optimized and a bound for the generalization error is computed. Second, the scaling

R2W2

3. For a linear classifier the Hessian of the mean squared error is equal to the estimated covariance matrix.

factors are determined by a gradient descent method minimizing the bound. This method has the advantage that it can be extended to nonlinear kernels, where the scaling factors are put into the kernel function. On the other hand, this method is computationally expensive because two optimization problems (SVM solution and error bound) have to be solved for every iteration and the kernel matrix must be evaluated for every step. Additionally, the gradient-based optimization suffers from convergence to local optima.

Statistical methods are so far the most common choice for selecting relevant genes from microarray data (e.g., see Pomeroy et al., 2002). However, SVM-based methods have recently been applied with good success (Shipp et al., 2002).

15.3 The Potential Support Vector Machine for Feature Selection

In this section we introduce a new feature selection method based on the SVM technique (see also Hochreiter and Obermayer, 2003a). Feature selection and classification are performed simultaneously. The main differences to previous approaches are the following:

- *Sphering.* In order to judge the relevance of feature components, the variance should be normalized, that is, the data should be sphered (whitened). Therefore, an objective is formulated according to which the classifier is selected by maximizing the margin after sphering. It turns out that sphering has two additional advantages for the SVM technique. First, the derived new SVM approach is invariant to linear transformation of the data—as are the margin bounds. Second, tighter margin bounds can be obtained.
- *New constraints.* The constraints of the optimization problem are modified to ensure that the classifier is optimal with respect to the mean squared error between the classification function and the labels. In contrast to previous approaches where one constraint is associated with each of the m training examples, each constraint is now associated with one feature and the number of new constraints is equal to the number N of features.
- *Support features.* The combination of the new objective with the new constraints allows assignment of support vector weights to features, and the normal vector of the classification boundary is expanded in terms of these weights rather than in terms of support vector data points. This allows feature selection according to whether a feature is a support vector or not. As a side effect the dual optimization problem can be efficiently solved using a technique similar to sequential minimal optimization (Platt, 1999).

In summary, a classifier is selected from the set of all classifiers with minimal mean squared error which yields the largest margin after sphering the data. The new SVM removes irrelevant features which lead to a minimal increase of the mean squared

error when removed. More formally, feature selection is done by assigning support vector weights to features—the features which are support vectors are selected.

In the following subsections, we first briefly review the classic SVM. Then we introduce a new objective for achieving scale-invariant SVMs, present new constraints for correct classification, and combine the new objective and the new constraints into one framework. Finally, a summary of the new technique is given.

15.3.1 The Classic SVM

Let us consider a set of m objects, which are described by feature vectors $\mathbf{x} \in \mathbb{R}^N$, and let us represent this data set by the matrix $\mathbf{X} := (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)$. We furthermore assume that every object belongs to one of two classes, and that class membership is denoted by a binary label $y \in \{+1, -1\}$. The labels for the m objects are summarized by a label vector $\mathbf{y}$, where y_i is the label of $\mathbf{x}_i$.

The goal is to construct a linear classifier based on the feature vectors $\mathbf{x}$. In the standard SVM approach (see chapter 2) this classifier is defined by taking the sign of the classification function

$$f(\mathbf{x}) \;=\; (\mathbf{w} \cdot \mathbf{x}) \;+\; b \;, \tag{15.1}$$

where the weight vector $\mathbf{w}$ has been normalized such that the margin ρ, that is the distance between the classification boundary and the closest data point, is $\rho \;=\; \|\mathbf{w}\|^{-1}$.

Classic SVMs construct a classification function which maximizes the margin under the constraint that the training data are classified correctly:

$$\begin{aligned} &\min_{\mathbf{w},b} \;\; \frac{1}{2}\,\|\mathbf{w}\|^2 \\ &\text{subject to} \;\; y_i \;((\mathbf{w} \cdot \mathbf{x}_i) \;+\; b) \;\geq\; 1 \;\; . \end{aligned} \tag{15.2}$$

Here we assume that the data $\mathbf{X}$ with label vector $\mathbf{y}$ are linearly separable; otherwise slack variables have to be used. If the number of training examples m is larger than the Vapnik-Chervonenkis (VC) dimension h (a capacity measure for classifiers, see, e.g., Vapnik, 1998), then one obtains the following bound on the generalization error $R(f)$ of f (also called "risk of f") (Vapnik, 1998; Schölkopf and Smola, 2002):

VC dimension

$$R(f) \;\leq\; R_{\text{emp}}(f) \;+\; \sqrt{\frac{1}{m}\left(h\;\left(\ln\left(\frac{2\;m}{h}\right) \;+\; 1\right) \;-\; \left(\frac{\ln(\delta)}{4}\right)\right)} \tag{15.3}$$

Worst case bounds

which holds with probability $1 - \delta$. δ denotes the probability that a training set $\mathbf{X}$ of size m has been randomly drawn from the underlying distribution, for which the bound (15.3) does not hold. $R_{\text{emp}}(f)$ denotes the training error of f (also called the "empirical risk of f"). For the set of all linear classifiers defined on $\mathbf{X}$, for which $\rho \;\geq\; \rho_{\min}$ holds, one obtains

$$h \;\leq\; \min\left\{\left[\frac{\mathrm{R}^2}{\rho^2_{\min}}\right] \;,\; N\right\} \;+\; 1 \tag{15.4}$$

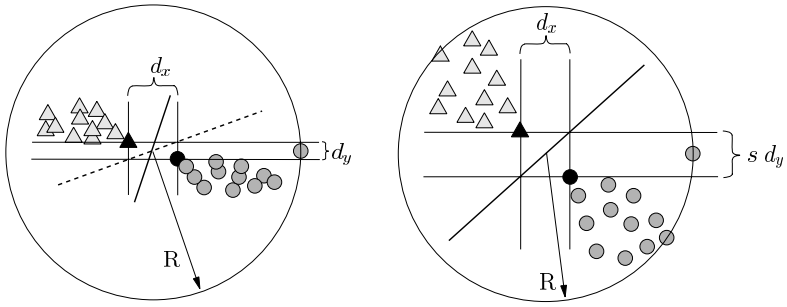


Figure 15.2 Left; Data points from two classes (*triangles* and *circles*) are separated by the largest margin hyperplane (*solid line*) according to the support vector approach. The two support vectors (*black symbols*) are separated by d_x along the horizontal and by d_y along the vertical axis, from which we obtain $\rho = \frac{1}{2}\sqrt{d_x^2 + d_y^2}$ and $\frac{\mathrm{R}^2}{\rho^2} = \frac{4\ \mathrm{R}^2}{d_x^2 + d_y^2}$. The dashed line indicates the classification boundary of the classifier shown on the right, scaled back by a factor of $\frac{1}{s}$. Right; The same data but scaled along the vertical axis by the factor s. The data points still lie within the sphere of radius R. The solid line denotes the support vector hyperplane, whose scaled version is shown on the left (*dashed line*). We obtain $\rho = \frac{1}{2}\sqrt{d_x^2 + s^2\ d_y^2}$ and $\frac{\mathrm{R}^2}{\rho^2} = \frac{4\ \mathrm{R}^2}{d_x^2 + s^2\ d_y^2}$. For $s \neq 1$ and $d_y \neq 0$ both the margin ρ and the term $\frac{\mathrm{R}^2}{\rho^2}$ change with scaling (see text for further explanation).

(see Vapnik, 1998; Schölkopf and Smola, 2002), where $[\cdot]$ denotes the integer part and R is the radius of the smallest sphere in the data space, which contains all the training data. The fact that the bounds[4] become smaller for increasing ρ and decreasing N motivates the maximum margin principle (15.2), as well as the concept of feature selection (minimizing N). Bounds on the *expected* generalization error can also be derived (cf. Vapnik, 1998; Schölkopf and Smola, 2002). They also become smaller for increasing ρ and decreasing N.

15.3.2 A Scale Invariant Objective Function

Both the selection of a classifier using the maximum margin principle and the values obtained for the bounds on the generalization error described in the last section suffer from the problem that they are not invariant under linear transformations. This problem is illustrated in figure 15.2. The figure shows a two-dimensional classification problem, where the data points from the two classes are indicated by triangles and circles. In the left figure, both classes are separated by the hyperplane with the largest margin (solid line). In the right figure, the same classification problem is shown, but scaled along the vertical axis by a factor s. Again, the solid

4. Note that these bounds can be improved using the concepts of covering numbers and the fat shattering dimension (Shawe-Taylor et al., 1996, 1998; Schölkopf and Smola, 2002).

line denotes the support vector solution, but when the classifier is scaled back to $s = 1$ (dashed line in the left figure) it no longer coincides with the original SVM solution. Therefore, the optimal hyperplane is not invariant under scaling, hence predictions of class labels may change if the data are rescaled before learning. In the legend of figure 15.2 it is shown that the ratio R^2/ρ^2, which bounds the VC dimension [see (15.4)] and determines an upper bound on the generalization error [see (15.3)] has also changed. If, however, the classifier depends on scale factors, the question arises of which scale factors should be used for classifier selection.

Here we suggest scaling the training data such that the margin ρ remains constant while the radius R of the sphere containing all training data becomes as small as possible. This scaling results in a new sphere with radius $\tilde{\mathrm{R}}$ which still contains all training data and which leads to a tight margin-based bound for the generalization error. Optimality is achieved when all directions orthogonal to the normal vector $\mathbf{w}$ are scaled to zero and $\tilde{\mathrm{R}} = \min_{t\in\mathbb{R}} \max_i |(\hat{\mathbf{w}} \cdot \mathbf{x}_i) + t| \leq \max_i |(\hat{\mathbf{w}} \cdot \mathbf{x}_i)|$, where $\hat{\mathbf{w}} := \frac{\mathbf{w}}{\|\mathbf{w}\|}$. Note that with offset b of the classification function the sphere must not be centered at the origin (Vapnik, 1998). Unfortunately, the above formulation does not lead to a manageable optimization problem. Therefore, we suggest minimizing the upper bound:

$$\frac{\tilde{\mathrm{R}}^2}{\rho^2} = \tilde{\mathrm{R}}^2 \|\mathbf{w}\|^2 \leq \max_i (\mathbf{w} \cdot \mathbf{x}_i)^2 \leq \sum_i (\mathbf{w} \cdot \mathbf{x}_i)^2 = \left\|\mathbf{X}^\top \mathbf{w}\right\|^2 . \tag{15.5}$$

New objective

In Hochreiter and Obermayer (2003b) it is shown that replacing the objective function in (15.2) by the upper bound

$$\mathbf{w}^\top \mathbf{X}\, \mathbf{X}^\top \mathbf{w} = \left\|\mathbf{X}^\top \mathbf{w}\right\|^2 , \tag{15.6}$$

of (15.5) on R^2/ρ^2 corresponds to the integration of sphering (whitening) and SVM learning into one framework. The resulting classifier is called a "sphered support vector machine" (S-SVM). Minimizing the new objective leads to normal vectors which tend to point in directions of low variance of the data. If the data have already been sphered, then the covariance matrix is given by $\mathbf{X}\, \mathbf{X}^\top = \mathbf{I}$ and we recover the classical SVM.[5] In Hochreiter and Obermayer (2003b) a new error bound based on covering numbers is derived according to the considerations above, which additionally motivates the new objective function (15.6). There it is also shown, that the new objective is well defined for cases where $\mathbf{X}\, \mathbf{X}^\top$ or $\mathbf{X}^\top \mathbf{X}$ or both are singular.

Improved error bound by sphering

Invariant under linear transformations

The new objective leads to separating hyperplanes which are invariant to linear transformations of the data. Consequently, the bounds and the performance of the derived classifier no longer depend on scale factors. Note that the kernel trick carries over to the S-SVM as shown in Hochreiter and Obermayer (2003b). The S-SVM can also be applied to kernels which are not positive definite, that is, which are not Mercer kernels (Hochreiter and Obermayer, 2002).

5. In general, however, sphering is not possible as a preprocessing step if a kernel is used.

15.3.3 New Constraints

To assign support vector weights to the feature components the m constraints enforcing correct classification have to be transformed into N constraints associated with the features. The idea of the transformation is to compute the correlation between the residual error and a feature component. If these correlations are zero, the empirical risk is minimal.

Residual error

We define a residual error r_i for a data point $\mathbf{x}_i$ as the difference between its class label y_i and the value of the classification function f, $f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + b$:

$$r_i = (\mathbf{w} \cdot \mathbf{x}_i) \; + \; b \; - \; y_i \quad . \tag{15.7}$$

For every feature component j we then compute the mixed moments σ_j,

$$\sigma_j \;=\; \frac{1}{m} \sum_{i=1}^{m} \; (\mathbf{x}_i)_j \; r_i \; , \tag{15.8}$$

between the residual error r_i and the measured values $(\mathbf{x}_i)_j$. These mixed moments σ_j should be made small (or zero). The rationale behind minimal values for σ_j is that, given quadratic loss functions, they lead to an optimal classifier. Consider the quadratic loss function

$$c(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) \;=\; \frac{1}{2} r_i^2 \tag{15.9}$$

and the empirical loss (the mean squared error)

$$R_{\text{emp}}(f_{\mathbf{w}}) \;=\; \frac{1}{m} \sum_{i=1}^{m} c\,(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) \quad . \tag{15.10}$$

The mixed moments σ_j are equal to the derivative of the empirical loss with respect to $(\mathbf{w})_j$:

$$\sigma_j \;=\; \frac{\partial R_{\text{emp}}(f)}{\partial\,(\mathbf{w})_j} \quad . \tag{15.11}$$

Constraint assures minimal empirical risk

That is, the empirical error is minimal if

$$\sigma_j \;=\; \frac{1}{m} \; \sum_{i} (\mathbf{x}_i)_j \; r_i \;=\; 0 \; . \tag{15.12}$$

Note that there exists only one minimum since the squared error is a convex function in the parameters $\mathbf{w}$.

These considerations motivate a new set of constraints:

$$\mathbf{X}\,\mathbf{r} \;=\; \mathbf{X} \; \left(\mathbf{X}^{\top} \; \mathbf{w} \; + \; b\mathbf{1} \; - \; \mathbf{y}\right) \;=\; \mathbf{0} \quad , \tag{15.13}$$

which an optimal classifier must fulfill, because

$$(\mathbf{X}\,\mathbf{r})_j \;=\; \sum_{i=1}^{m} \; (\mathbf{x}_i)_j \; r_i \;=\; m\,\sigma_j \;=\; m\,\frac{\partial R_{\text{emp}}(f)}{\partial\,(\mathbf{w})_j} \; . \tag{15.14}$$

However, measurement noise may lead to high values of σ_j which, when minimized, would lead to strong overfitting. Therefore, we introduce a "correlation threshold" ϵ which separates the noise from the signal part, and we modify the constraints in (15.13) according to

Correlation threshold ϵ

$$\begin{aligned} \mathbf{X}\ \left(\mathbf{X}^\top\ \mathbf{w}\ +b\mathbf{1}\ -\ \mathbf{y}\right)\ -\ \epsilon\ \mathbf{1}\ &\leq\ \mathbf{0}\ , \\ \mathbf{X}\ \left(\mathbf{X}^\top\ \mathbf{w}\ +b\mathbf{1}\ -\ \mathbf{y}\right)\ +\ \epsilon\ \mathbf{1}\ &\geq\ \mathbf{0}\ . \end{aligned} \tag{15.15}$$

This formulation is analogous to the ϵ-insensitive loss (Schölkopf and Smola, 2002).

If measurements of some features have larger variance then others, a global (independent of the feature j) correlation threshold ϵ cannot distinguish between high σ_j values resulting from high correlation between the r_i and $(\mathbf{x}_i)_j$ and high σ_j values resulting from large variance of the values $(\mathbf{x}_i)_j$. A global, that is, feature-independent, ϵ would lead to an undesired preference of highly varying features even if they do not convey information about the class label. Therefore, the variance of the values $(\mathbf{x}_i)_j$ should be taken into account. For example, a more appropriate measure would be Pearson's correlation coefficient

$$\hat{\sigma}_j\ =\ \frac{\sum_{i=1}^m\ \left((\mathbf{x}_i)_j\ -\ \bar{x}_j\right)\ (r_i\ -\ r)}{\sqrt{\sum_{i=1}^m\ \left((\mathbf{x}_i)_j\ -\ \bar{x}_j\right)^2}\ \sqrt{\sum_{i=1}^m\ (r_i\ -\ r)^2}}\ , \tag{15.16}$$

where $r\ =\ \frac{1}{m}\ \sum_{i=1}^m\ r_i$ is the mean residual and $\bar{x}_j\ =\ \frac{1}{m}\ \sum_{i=1}^m\ (\mathbf{x}_i)_j$ is the mean of the jth feature. In order to utilize the correlation coefficient $\hat{\sigma}_j$ for a global ϵ, we assume that the data vectors $\left((\mathbf{x}_1)_j, (\mathbf{x}_2)_j, \ldots, (\mathbf{x}_m)_j\right)$ are normalized to zero mean and unit variance:

$$\frac{1}{m}\sum_{i=1}^m\ \left((\mathbf{x}_i)_j\ -\ \bar{x}_j\right)^2\ =\ 1\ \text{ and }\ \bar{x}_j\ =\ \frac{1}{m}\ \sum_{i=1}^m\ (\mathbf{x}_i)_j\ =\ 0\ . \tag{15.17}$$

This normalization assumption is sufficient for a global ϵ because σ_j,

$$\sigma_j\ =\ \frac{1}{m}\sum_{i=1}^m\ (\mathbf{x}_i)_j\ r_i\ \ =\ \hat{\sigma}_j\ \frac{1}{\sqrt{m}}\ \|\mathbf{r}\ -\ r\mathbf{1}\|\ , \tag{15.18}$$

is linear in the correlation coefficient $\hat{\sigma}_j$ and otherwise independent of component j. If the noise is large, random correlations will occur more often, and the value of ϵ must be increased. If the strength of the measurement noise is known, the correct value of ϵ can be determined a priori. Otherwise, it takes the role of a hyperparameter and must be adapted using model selection techniques.

Besides the important interpretation of ϵ as a noise parameter, there is a second interpretation in terms of bounding the increase of the residual error when a feature is removed. If we change $\mathbf{w}$ in the direction $\mathbf{e}_j$ by an amount of β, the new residual error r_i^{new} is

ϵ bounds the error increase

$$r_i^{\text{new}}\ =\ ((\mathbf{w}+\beta\ \mathbf{e}_j)\cdot\mathbf{x}_i)\ +\ b\ -\ y_i\ \ , \tag{15.19}$$

where $\mathbf{e}_j$ is the unit vector parallel to the jth feature axis. We obtain

$$\begin{aligned}\sum_i (r_i^{\text{new}})^2 &= \sum_i \left(r_i^{\text{old}} \; + \; \beta \, (\mathbf{e}_j \cdot \mathbf{x}_i)\right)^2 \\ &= \sum_i \left(r_i^{\text{old}}\right)^2 \; + \; 2 \; \beta \; \sum_i r_i^{\text{old}} \, (\mathbf{x}_i)_j \; + \; \sum_i \beta^2 \, (\mathbf{x}_i)_j^2 \\ &= \sum_i \left(r_i^{\text{old}}\right)^2 \; + \; 2 \; \beta \;\; m \; \sigma_j \; + \; m \; \beta^2 \; .\end{aligned} \tag{15.20}$$

Because the constraints ensure that $|\sigma_j| \; m \; \leq \; \epsilon$, the increase on the residual error after the elimination the jth feature is bounded by

$$2 \; \epsilon \; |(\mathbf{w})_j| \; + \; m \; (\mathbf{w})_j^2 \; , \tag{15.21}$$

where we set $\beta = - \; (\mathbf{w})_j$.

15.3.4 The Potential SVM

Now we combine both the new objective from (15.6) and the new constraints from (15.15) and call the new procedure of selecting a classifier the *potential support vector machine (P-SVM)*. As we will see, the combination of new objective and new constraints leads to an expansion of the normal vector of the classification boundary into a sparse set of features, hence allows expressing the relevance of features via support vector weights. Combining (15.6) and (15.15) we obtain

potential SVM: Primal

$$\begin{aligned} \min_{\mathbf{w},b} \;\; & \frac{1}{2} \; \|\mathbf{X}^\top \; \mathbf{w}\|^2 \\ \text{subject to} \;\; & \mathbf{X} \; \left(\mathbf{X}^\top \; \mathbf{w} \; + b \mathbf{1} \; - \; \mathbf{y}\right) \; + \; \epsilon \; \mathbf{1} \; \geq \; \mathbf{0} \\ & \mathbf{X} \; \left(\mathbf{X}^\top \; \mathbf{w} \; + b \mathbf{1} \; - \; \mathbf{y}\right) \; - \; \epsilon \; \mathbf{1} \; \leq \; \mathbf{0} \; . \end{aligned} \tag{15.22}$$

Why ϵ?

If the row vectors of $\mathbf{X}$ are normalized to mean zero, then $\mathbf{X1} \; = \; \mathbf{0}$ and the term containing b vanishes. The parameter ϵ serves two important purposes:

- Large values of ϵ lead to a sparse expansion of the weight vector through the support features.
- If $\mathbf{X} \, \mathbf{X}^\top$ is singular and $\mathbf{w}$ is not uniquely determined, ϵ enforces a unique solution, which is characterized by the most sparse representation through features.

The interpretation of ϵ as a sparseness property is known from support vector regression (Schölkopf and Smola, 2002).

Optimization is usually performed using the Wolfe dual of (15.22) given by (see appendix A)

Potential SVM: Dual

$$\begin{aligned} \min_{\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-} \;\; & \frac{1}{2} \left(\boldsymbol{\alpha}^+ \; - \; \boldsymbol{\alpha}^-\right)^\top \; \mathbf{X} \, \mathbf{X}^\top \; \left(\boldsymbol{\alpha}^+ \; - \; \boldsymbol{\alpha}^-\right) \; - \\ & \mathbf{y}^\top \, \mathbf{X}^\top \; \left(\boldsymbol{\alpha}^+ \; - \; \boldsymbol{\alpha}^-\right) \; + \; \epsilon \; \mathbf{1}^\top \left(\boldsymbol{\alpha}^+ \; + \; \boldsymbol{\alpha}^-\right) \\ \text{subject to} \;\; & \mathbf{1}^\top \, \mathbf{X}^\top \; \left(\boldsymbol{\alpha}^+ \; - \; \boldsymbol{\alpha}^-\right) \; = \; 0 \; , \\ & \mathbf{0} \; \leq \; \boldsymbol{\alpha}^+ \; , \;\; \mathbf{0} \; \leq \; \boldsymbol{\alpha}^- \;\; , \end{aligned} \tag{15.23}$$

where the quantities $\boldsymbol{\alpha} = (\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-)$ denote the Lagrange parameters. $\mathbf{1}^\top \mathbf{X}^\top (\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-) = 0$ is automatically satisfied if the row vectors of $\mathbf{X}$ are normalized to zero mean. The dual problem is solved by a sequential minimal optimization (SMO) technique (see Platt, 1999). A fast solver for the P-SVM is described in Hochreiter and Obermayer (2003b), where advantage can be taken of the fact that the equality constraint vanishes for zero mean row vectors of $\mathbf{X}$. The SMO technique is important if the number of features is large because the optimization problem of the P-SVM is quadratic in the number of features rather than in the number of data points. In contrast to standard SVMs with a linear kernel it is the correlation matrix $\mathbf{X}\,\mathbf{X}^\top$ and *not* the Gram matrix $\mathbf{X}^\top\,\mathbf{X}$ which enters the SVM objective.

Fast optimization by SMO

Finally, the classification function f has to be constructed using the optimal values of the Lagrange parameters $\boldsymbol{\alpha}$. In appendix A,we show that

$$\mathbf{w} = \boldsymbol{\alpha} \ . \tag{15.24}$$

In contrast to the standard SVM expansion of $\mathbf{w}$ by its support vectors $\mathbf{x}$, the weight vector $\mathbf{w}$ is now expanded into a sparse set of feature components which serve as the support vectors in this case. The value for b can be computed from the condition that the average residual error r is equal to zero:

Computing b

$$b = -\frac{1}{m} \sum_{i=1}^{m} ((\mathbf{w} \cdot \mathbf{x}_i) - y_i) \ . \tag{15.25}$$

Note that b is chosen so that

$$\frac{\partial R_{\mathrm{emp}}(f)}{\partial b} = \frac{1}{m} \sum_i r_i = b + \frac{1}{m} \sum_i ((\mathbf{w} \cdot \mathbf{x}_i) - y_i) = 0 \ . \tag{15.26}$$

That means b takes on its optimal value for minimization of the empirical risk (as was already ensured for $\mathbf{w}$ through the constraints). If the row vectors of $\mathbf{X}$ are normalized to zero, we obtain

$$b = \frac{1}{m} \sum_{i=1}^{m} y_i \ . \tag{15.27}$$

The classification function is then given by

$$f(\mathbf{u}) = (\mathbf{w} \cdot \mathbf{u}) + b = \sum_{j=1}^{n} \alpha_j \ (\mathbf{u} \cdot \mathbf{e}_j) + b = \sum_{j=1}^{n} \alpha_j \ (\mathbf{u})_j + b \ . \tag{15.28}$$

The classifier based on (15.28) depends on the weighting coefficients α_j and b, which were determined during optimization, and on the measured values $(\mathbf{u})_j$ of the selected features for the object to be classified. The weighting coefficients α_j can be interpreted as class indicators, because they separate the features into features which are relevant for class 1 and class -1, according to the sign of $\alpha_j = \alpha_j^+ - \alpha_j^-$. If the value of ϵ is large enough during learning, the expansion (15.24) of the weight vector contains only a few "most informative" features, hence most of the components of $\mathbf{w}$ are zero. The other features are discarded because of being

Class indicators

Sparseness

Feature selection

too noisy or not conveying information about the class labels. Sparseness can be attributed to the term $\epsilon\, \mathbf{1}^\top (\boldsymbol{\alpha}^+ + \boldsymbol{\alpha}^-)$ (or $\epsilon\, \|\boldsymbol{\alpha}\|_1$) in the dual objective function (15.23). For large enough values of ϵ, this term pushes all α_j toward zero except for the features most relevant to classification.

One may wonder whether the P-SVM is similar to the 1-norm SVM because sparseness of $\mathbf{w}$ is enforced through $\epsilon\, \|\boldsymbol{\alpha}\|_1$. However in contrast to the 1-norm SVM, the P-SVM still contains a quadratic part which enforces a large Euclidean margin on the training data. The 1-norm term originates from the ϵ-insensitive loss in the primal optimization problem. The P-SVM is in this sense similar to support vector regression (SVR) (Schölkopf and Smola, 2002), where the vector $\boldsymbol{\alpha}$ is also regularized by a quadratic and a 1-norm part. However, all terms of the P-SVM are different from the corresponding terms in the SVR: the quadratic matrix, the linear term, and the constraints. In contrast to both the 1-norm SVM and the SVR the value ϵ which weights the sparseness has a noise interpretation (measurement noise) and ϵ can be used to bound the residual error if a feature component is deleted.

15.3.5 Summary of the P-SVM and Its Application to Microarray Data

The P-SVM is a method for selecting a classification function, where the classification boundary depends on a small number of "relevant" features. The method can be used for feature selection, and it can also be used for the subsequent prediction of class labels in a classification task. The optimal P-SVM classifier is a classifier with minimal empirical risk but with the largest margin after sphering the data. Feature selection can be interpreted as removing features from the optimal classifier but bounding the increase in mean squared error through the value ϵ.

The first observation is that optimization (i.e., the selection of the proper values for $\boldsymbol{\alpha}$ and b) only involves the measurement matrix $\mathbf{X}$ and the label vector $\mathbf{y}$. In order to apply the P-SVM method to the analysis of microarray data, we identify the objects with samples, the features with genes, and the matrix $\mathbf{X}$ with the matrix of expression values. Due to the term $\mathbf{X}\,\mathbf{X}^\top$ in the dual objective, the optimization problem is well defined for measurement matrices $\mathbf{X}$ of expression values. From a conceptual point of view, however, it is advantageous to interpret the matrix $\mathbf{X}$ of observations (here: expression values) itself as a dot product matrix whose values emerge as a result of the application of a measurement kernel. This view is taken by Hochreiter and Obermayer (2003a,b) and briefly summarized in Appendix B.

The second observation is that an evaluation of the classifier for new samples i only involves the measurement of its expression values $(\mathbf{x}_i)_j$ for the selected "support genes" j. The number of support genes depends on the value of a noise parameter, the correlation threshold ϵ. If the value of ϵ is large during learning, only a few "most informative" genes are kept. The other genes are discarded because of being too noisy or not conveying information about the class labels.

The set of all genes for which the weighting coefficients α_j are non-zero (the set of support genes) is the selected feature set. The size of this set is controlled by the

value of ϵ, and if the P-SVM is applied for feature selection, the value of ϵ should be large.

15.4 The Gene Selection Protocol

In this section we describe the protocol for extracting meaningful genes from a given set of expression values for the purpose of predicting labels of the sample classes. The protocol includes data preprocessing, the proper normalization of the expression values, the feature selection and ranking steps, and the final construction of the predictor. We use the protocol together with our feature selection procedure, which was described in section 15.3. The protocol, however, can also be applied to other feature selection or ranking methods.

Additional information by labels

Note that our feature selection method requires class labels which must be supplied together with the expression values of the microarray experiment. When this technique is applied to the classification of tumor samples in subsection 15.5.2 we are provided with binary class labels, but real values associated with the different samples may also be used. For the following, however, we assume that the task is to select features for classification and that m labeled samples are given for training the classifier.

15.4.1 Description of the Protocol

Gene selection protocol

1. Expression values vs. ratios. Before data analysis starts it is necessary to choose an appropriate representation of the data. Common representations are based on the ratio $T_j = \frac{R_j}{G_j}$ of expression values between the value R_j (red) of a gene j in the sample to analyze and the value G_j (green) in the control sample, and the log ratio $L_j = \log_2(T_j)$. We, however, suggest using the original expression values R_j because our experimental findings on different data sets (e.g., the GIST data from Allander et al., 2001) showed increased classification performance when the original values were used.

Present call

2. Present call. The present call is usually the first step in the analysis of microarray data. During this step genes are identified for which the confidence is high that they are actually expressed in at least one of the samples. Genes for which this confidence is low are excluded from further processing in order to suppress noise.

For this purpose an error model has to be constructed for the expression values or their ratios (sometimes before, sometimes after averaging across multiple measurements of the same sample; see Tseng et al., 2001; Schuchhardt et al., 2000; Kerr et al., 2000; Hartemink et al., 2001). This error model accounts for both measurement-specific noise (e.g., background fluctuations), which affects all expression values in a similar way, and gene specific noise (for example the binding efficiency of the dye), which affects expression values for different genes in a different way. Using this error model one assigns a p-value, which gives the probability

that the observed measurement is produced by noise, to every measurement of an expression level. If the p-value is smaller than a threshold q_1 (typical values are 5%, 2%, or 1%), the expression level is marked "reliable". If this happens for a minimum number q_2 (typical values range from 3 to 20) of samples, the corresponding gene is selected and a so-called present call has been made.

Normalization

3. Normalization. Before further processing, the expression values are normalized to mean zero and unit variance across all training samples and separately for every gene. Normalization accounts for the fact that expression values may differ by orders of magnitudes between genes and allows assessing the importance of genes, including genes with small expression values. Sometimes more advanced normalization techniques are used (Schuchhardt et al., 2000; Hill et al., 2001; Durbin et al., 2002; Yang et al., 2002; Huber et al., 2002).

Gene ranking, determining hyperparameters and number of genes

4. Gene ranking and gene selection. Here we assume that a feature selection method has been chosen where the size of the set of selected genes is controlled by a hyperparameter which we call ϵ in the following. Although we propose to use the P-SVM method, any other features selection method can be used with this gene selection protocol.

In this step we perform two loops: an "inner loop" and an "outer loop" (the leave-one-out loop). The inner loop serves two purposes. It ranks features if only a subset method like the P-SVM is available and it makes feature selection more robust against variations due to initial conditions of the selection method. The outer loop also serves two purposes. It makes the selection robust against outlier samples and allows determination of the optimal number of selected genes together with the optimal values of hyperparameters for the later prediction of class labels. In order to do this, a predictor must be constructed. Here we suggest using a ν-SVM where the value of ν is optimized by the outer loop. In order to implement the outer (leave-one-out) loop, m different sets of samples of size $m-1$ are constructed by leaving out one sample for validation. For each of the m sets of reduced size, we perform gene selection and ranking using the following "inner loop."

Inner loop

Inner loop. The subset selection method is applied multiple times to every reduced set of samples for different values of ϵ. For every set of samples multiple sets of genes of different size are obtained, one for every value of ϵ. If the value of ϵ is large, the number of selected genes is small, and vice versa. The inner loop starts with values of ϵ which are fairly large in order to obtain a few genes only. Gradually the value is reduced to obtain more genes per run. Genes obtained for the largest value of ϵ obtain the highest rank; the second highest rank is given to genes which additionally appear for the second largest value of ϵ, and so on. The values of ϵ are constant across sample sets. The minimal value should be chosen such that the number of extracted genes is approximately the total number m of samples. The maximal value should be chosen such that approximately five to ten genes are selected. The other values are distributed uniformly between these extreme values. In the numerical experiments in section 15.5 a total of three to five different values were used.

Outer loop

Outer loop. The results of the inner loops are then combined across the m different sets of samples. A final ranking of genes is obtained according to how often genes are selected in the m leave-one-out runs of the inner loop. If a gene is selected in many leave-one-out runs, it is ranked high; otherwise it is ranked low. Genes which are selected equally often are ranked according to the average of their rank determined by the inner loops. The advantage of the leave-one-out procedure is that a high correlation between expression values and class labels induced by a single sample is scaled down if the according sample is removed. This makes the procedure more robust against outliers.

The outer loop is also used for selecting an optimal number of genes and other hyperparameters. For this purpose, ν-SVMs are trained on each of the m sets of samples for different values of the hyperparameter ν and the number F of high-ranking genes (ranking is obtained by the inner loop). Then the average error is calculated on the left-out samples. Since the leave-one-out error as a function of the number F of selected genes is noisy, the leave-one-out error for F is replaced by the average of the leave-one-out errors for F, $F + a$, and $F - a$. Then the values of the hyperparameter ν and the number of genes F which give rise to the lowest error are selected. This completes the feature selection procedure.

15.4.2 Comments on the Protocol and on Gene Selection

Corrections to the outer, leave-one-out loop. The samples which were removed from the data in the outer loop when constructing the m reduced subsets for the gene ranking should not be considered for the present call and for determining the normalization parameters. Both steps should be done individually for each of the m sets of sample, otherwise feature or hyperparameter selection may not be optimal.

Computational costs. The feature selection protocol requires $m \times n_\epsilon$ feature selection runs, where n_ϵ is the number of different values of the ϵ parameter. However the computational effort is justified by the increased robustness against correlation by chance (see next item) and the elimination of single sample correlations.

Correlation by chance

Correlations by chance. "Correlations by chance" refers to the fact that noise induced spurious correlations between genes and class labels may appear for a small sample size if the level of noise is high. If the number of selected genes is small compared to the total number of probes (genes) on the chip, spurious correlations may become a serious problem. The Monte-Carlo simulations of van't Veer et al. (2002) on randomly chosen expression values for a data set of 78 samples and 5000 genes resulted in 36 "genes" which had noise-induced correlation coefficients larger than .3. In order to avoid large negative effects of the above mentioned spurious correlations the number of selected genes should not be too small, and one should extract a few tens of genes rather than a few genes only to decrease the influence of single spurious correlated genes. The random correlation effect can also be reduced by increasing q_2, the minimum number of "reliable" expression values for making a present call. This avoids the selection of genes for which too few samples contribute

Many genes are better than few genes

to the correlation measure. However as explained in the next paragraph, too many genes should be avoided as well.

Redundancy. Redundant sets of genes, that is, sets of genes with correlated expression patterns, should be avoided in order to obtain good machine learning results (Jäger et al., 2003). Selection of too many genes with redundant information may lead to low generalization performance (cf. section 15.1). The P-SVM described in section 15.3 extracts a sparse set of genes, hence reduces redundancy. Another reason for avoiding redundancy is that not all causes which imply the conditions may be recognized. This may happen if the set has to be kept small while redundant genes are included (redundant genes indicate the same cause; see experiments in subsection 15.5.1). Reducing redundancy does not preclude the extraction of coexpressed clusters of genes: coregulated genes can be extracted in a subsequent processing step, for example, based on classic statistical analysis.

Redundancy

Finally, one may wonder why redundant information does not help to decrease the noise level of otherwise informative genes. Empirically, one finds that nonredundant feature selection methods (P-SVM and R2W2) outperform feature selection methods which include redundant genes (Fisher correlation and RFE); see subsection 15.5.1. It seems as if the detrimental effects of a larger number of features are stronger.

15.4.3 Classification of Samples

In order to construct a predictor for the class labels of new samples a classifier is trained on all the m samples using the optimal set of genes and the optimal value of the hyperparameter (here: ν, cf. subsection 15.4.1, step 4). The generalization performance of the classifier can again be estimated using a cross-validation procedure. This procedure must involve performing the full gene selection procedure including all preprocessing steps (for example normalization and feature selection) separately on all m cross-validation subsets. Otherwise a bias is introduced in the estimate. Note that this also requires performing the "inner loop" of step 4 on sets of $m - 2$ samples.

Before the classifier is applied to new data, the expression values for the new sample must be scaled according to the parameters derived from the training set. As a consequence we may observe expression values which are larger than the ones which occur in the training data. We set the expression values exceeding the maximal value in the training set to this maximal value. With this procedure we may underestimate certain expression levels, but the robustness against unexpected deviations from the training data is increased.

15.5 Experiments

15.5.1 Toy Data

We compare different methods on data analogous to, but more difficult than the ones used in Weston et al. (2000). Compared to the data in Weston et al. (2000), the number of features is 10 times larger, the ratio of the number of relevant features to the number of all features is smaller, and the noise in the data (measured by the misclassification rate on data vectors where all irrelevant features are set to zero) is larger too. The methods which are chosen for comparison are the Fisher score, recursive feature elimination (RFE) (Guyon et al., 2002), R2W2 according to Weston et al. (2000), and the P-SVM. The benchmark methods have all been successfully applied to microarray data: the Fisher score in Pomeroy et al. (2002) and van't Veer et al. (2002), RFE in Guyon et al. (2002), and R2W2 in Shipp et al. (2002).

For the toy experiments we simulate microarray data by assuming two or more causes ("modes") for the class labels. Each mode is characterized by a few indicators which means that a cause is reflected by an expression pattern across a few genes, for example, genes belonging to the same pathway. Most features have no dependencies with the label. All features are very noisy, and only a few samples are given, because microarray data typically suffer from small sample sizes.

Weston Data 1 We randomly chose 600 data points (samples) with probabilities .5 from class 1 ($y_i = 1$) and .5 from class 2 ($y_i = -1$). One hundred data points are available for feature selection and training and the remaining 500 data points are used for testing the classifier. Next, we generated the 2000 attributes which simulate expression values of 2000 genes. Each data point was generated according to one of two modes to simulate two class-determining causes for every sample.

Mode 1 was chosen with probability .7 and mode 2 with probability .3. In mode 1 the first 10 features indicate mode 1 and are generated according to $(\mathbf{x}_i)_l \sim y_i \cdot \mathrm{N}(l, 10)\,,\ 1 \le l \le 10$, where $\mathrm{N}(\mu, \sigma)$ denotes a normal distribution with mean μ and standard deviation σ. The features from 11 to 20 are chosen according to $(\mathbf{x}_i)_l \sim \mathrm{N}(0, 10)\,,\ 11 \le l \le 20$. In mode 2 the features from 11 to 20 indicate mode 2 and are generated according to $(\mathbf{x}_i)_l \sim y_i \cdot \mathrm{N}(l - 10, 10)\,,\ 11 \le l \le 20$. The first 10 features are drawn from $(\mathbf{x}_i)_l \sim \mathrm{N}(0, 10)\,,\ 1 \le l \le 10$. The remaining 1980 features are chosen according to $(\mathbf{x}_i)_l \sim \mathrm{N}(0, 20)\,,\ 21 \le l \le 2000$, for both modes.

Table 15.1 shows the fraction of misclassification on the test set which is averaged over 10 different runs with different training and test sets. Features were selected using the Fisher score, RFE (Guyon et al., 2002), R2W2 (Weston et al., 2000), and the P-SVM method. Numbers are reported for classifiers using the 5, 10, 15, 20, and 30 top–ranked features. With these features we trained a classic C-SVM on the training set and validated the performance on a test set with 500 data points. The

Table 15.1 Classification performance for the Weston data 1. The values are the fractions of misclassification averaged over 10 runs on different test sets for classifiers trained on the selected features. The original data have 2000 features of which only 20 are relevant. Features 1 to 10 were class indicators in 70% and features 11 to 20 in 30% of the data points. The table shows the results using the top-ranked 5, 10, 15, 20, and 30 features. The methods are Fisher score, RFE, R2W2, and the P-SVM. The new P-SVM, performed best in all cases.

features	5	10	15	20	30
Fisher	0.21	0.23	0.26	0.26	0.28
RFE	0.26	0.28	0.28	0.28	0.27
R2W2	0.23	0.24	0.24	0.23	0.24
P-SVM	0.21	0.20	0.22	0.22	0.23

hyperparameter C was selected through five fold cross-validation on the training set from the set $\{0.01, 0.1, 1, 10, 100\}$ (0.1 was chosen in most cases) for all methods. To ensure a fair comparison of the methods the hyperparameter C was not determined in the outer loop of our protocol for the P-SVM. Because we neither imposed label noise nor generated extreme outliers in our data, the SVM was not sensitive to the hyperparameter choice and the C-SVM performed as well as the ν-SVM which we proposed in the protocol.

The success of feature selection depends on how many noisy, irrelevant features are wrongly selected and whether all modes which influence classification performance are represented sufficiently well. The result of a C-SVM for using all features is 0.37 (no feature selection) and for using the relevant 20 features (the perfect selection) is 0.11. The P-SVM shows the best results in all cases (see table 15.1).

Weston Data 2 Here we extend the number of modes to five to make the task more difficult. Each mode was chosen with equal probability .2. For every mode r, $1 \leq r \leq 5$, we draw the values for the 4 associated features. We first choose a mode r, then we draw the feature values for the 4 associated features according to $(\mathbf{x}_i)_l \sim y_i \cdot \mathrm{N}(2, 0.5\,\tau)$, where $1 \leq \tau \leq 4$ and $l = (r-1) \cdot 5 + \tau$. The remaining features from 1 to 20 (i.e., excluding the features associated with r) are chosen according to $(\mathbf{x}_i)_l \sim \mathrm{N}(0, 1)$. The remaining 1980 features are for all modes always chosen according to $(\mathbf{x}_i)_l \sim \mathrm{N}(0, 20)$, $21 \leq l \leq 2000$. All other parameters and the hyperparameter selection scheme were similar to the previous experiments (Weston data 1). This data set is more challenging because there is a high chance of missing indicators for one mode, especially if the set of selected features is small. Missing a mode leads to differences in performance.

Table 15.2 shows the fractions of misclassification on the test set averaged over 10 different combinations of training and test set. It is instructive to compare the results of table 15.2 with the values obtained for the 20 relevant features (perfect selection), which leads to a fractional error of 0.10, and for all 2000 features (no

Table 15.2 Classification performance for the Weston data 2. Parameters and procedures are described in the legend of table 15.1.

features	5	10	15	20	30
Fisher	0.31	0.28	0.26	0.25	0.26
RFE	0.33	0.32	0.32	0.31	0.32
R2W2	0.29	0.28	0.28	0.27	0.27
P-SVM	0.28	0.23	0.24	0.24	0.26

Table 15.3 Numbers of the top 30 selected features for a typical single trial, listed according to their rank. Relevant features are in boldface.

P-SVM:	**7**	837	**2**	**18**	1248	**5**	**6**	**12**	**20**	**14**
	1562	980	664	1110	**11**	1404	1822	668	525	**9**
	80	1205	997	1228	1331	289	1605	621	1277	1987
R2W2:	837	**2**	980	**7**	**20**	**11**	1277	**6**	45	**5**
	18	1822	**12**	621	398	664	289	**14**	1110	587
	1605	1833	1331	1248	1752	525	1060	1443	820	997
Fisher:	980	**7**	**5**	837	**6**	**18**	1562	**12**	**2**	837
	20	1248	**8**	1404	**14**	1110	**11**	1228	80	664
	1987	1275	1331	668	263	640	621	1954	1774	1605
RFE:	837	**7**	1987	1277	**2**	753	**20**	1110	1774	997
	219	1636	**12**	398	**6**	1472	536	820	**18**	314
	974	525	**14**	877	621	1516	540	654	1331	664

selection), which leads to a fractional error of 0.38. Feature selection improves the classification result but does not quite reach the performance of the "perfect selection" case because not all relevant genes were selected. R2W2 with the weighing coefficients instead of selecting features has an error of 0.26, which means R2W2 in the nonselection mode is better than in the selection mode. P-SVM shows the best results in all cases.

In table 15.3 we listed the numbers of the top 30 selected features for a typical single trial according to their rank. P-SVM found 11, R2W2 9, Fisher statistic 10, and RFE 7 relevant features (numbers in boldface). All other features are spurious. All five modes were detected by P-SVM, R2W2, Fisher statistics, and RFE using the 10, 18, 15, and 23 most highly ranked features. P-SVM detected indicator genes corresponding to all five modes using the smallest features set from all the methods tested, which explains in part the better performance of classifiers based on the P-SVM feature set.

15.5.2 Microarray Data

Data Sets In this subsection we apply the P-SVM to the DNA microarray data published in Pomeroy et al. (2002),Shipp et al. (2002), and van't Veer et al. (2002).

1. *Brain tumor data set* (Pomeroy et al., 2002). In our first data set embryonal tumors in the central nervous system are investigated. The response to therapy for the malignant brain tumor medulloblastoma should be predicted. Patients have a highly variable response to therapy, which made it difficult for classic methods to predict the therapy outcome. A good machine-based prognosis, however, is highly desirable. Negative prognoses may indicate the necessity of an alternative therapy while positive prognoses may lead to a therapy with reduced toxicity.
Data are provided (supplementary to Pomeroy et al., 2002) for 60 samples of human tissue taken from patients with different brain tumors of the medulloblastoma kind before treatment. The patients were treated in a similar way by chemotherapy and radiation. The clinical follow-up was monitored and the samples were labeled according to treatment outcome. From the frozen tumor samples RNA was isolated and hybridized to an array containing 7129 probes.
The data were generated by the Affimetrix software and numbers denote "perfect match minus mismatch." Perfect match probes are oligonucleotides which are the probes with the highest hybridization efficiency (the identifying base sequence) for a cDNA, and mismatch probes are oligonucleotides with a small difference from the perfect match probe (one base in the middle of the identifying base sequence changed). Therefore, the specialization and the efficiency of the probe can be normalized by subtracting the expression value of the mismatch probe from the probe's expression value. For more details see Pomeroy et al. (2002).

2. *Lymphoma data set* (Shipp et al., 2002). Lymphoma tumors (diffuse large B-cell lymphoma, DLBCL) show a positive response to therapy in fewer than 50% of cases. Previous methods were not sufficient to reliably predict treatment outcome, hence new approaches are necessary. Good predictions would allow identification of high-risk patients, closer observation and monitoring, and would certainly improve existing treatments.
Samples and clinical follow-ups on 58 DLBCL patients are available. For all patients the chemotherapy is the same and the labels denote the treatment outcome: positive or negative. From the frozen tumor samples RNA was isolated and hybridized to an array containing 7129 probes resulting in an 58×7129 matrix of "perfect match minus mismatch." For more details see Shipp et al. (2002).

3. *Breast cancer data set* (van't Veer et al., 2002). Seventy percent to 80% of breast cancer patients receiving chemotherapy or hormone therapy survive without treatment (van't Veer et al., 2002), because metastasis appears in only 20% to 30% of cases. Because therapy has strong side effects, it would be important to predict beforehand whether a patient would benefit from a particular therapy or not. Therefore, tumor samples were analyzed using the DNA microarray technique to search for gene expression patterns indicating the development of metastasis

and the need for stronger medication. Clinical indicators, however, failed to predict treatment outcome. The data set is a collection of 78 patients and expression values of 24,481 genes. All patients were treated with modified radical mastectomy or breast-conserving treatment. The treatment outcome was monitored and the tumor samples were labeled according to whether the outcome was positive or negative. The data were given as log expression ratios; for more details see van't Veer et al. (2002).

Common Setting in All Following Experiments We normalized the rows of the data matrix $\mathbf{X}$ to mean 0 and variance 1. In step 4 of our protocol we used $a = 5$ for the first and third, and $a = 3$ for the second experiment for estimating the optimal number of features because the number of features was smaller in the second experiment.

To classify the tissue samples after selecting the relevant genes, we applied a *linear* ν-SVM (Schölkopf and Smola, 2002). We found that the choice of the classifier does not matter much (also C-SVM and K-nearest neighbor worked) but the ν-SVM was the most robust against variations in the hyperparameter ν which allowed a simpler optimization scheme in the outer loop of the protocol: ν was chosen from the set $\{0.2, 0.3, 0.4, 0.5\}$. For all ν-SVM classifiers we fixed the threshold value b to 0, because initial experiments showed that these reduced sets of classifiers led to better generalization performance compared to the full set ($b \neq 0$). Table 15.4 summarizes the parameters used in the experiments.

Benchmark Methods We compared the result of the P-SVM method in combination with the ν-SVM to the results reported in the literature. Therefore, for each data set, the compared methods are different. For the ν-SVM and the C-SVM, we used LIBSVM (Chang and Lin, 2001) whereas the P-SVM was implemented in C.

Results for the Brain Tumor Data Set The data set from Pomeroy et al. (2002) was processed according to the protocol from section 15.4, except for step 2 because of the missing p-values.

For the P-SVM method, the optimal number of features was 45 (average over the leave-one-out runs). Table 15.5 shows the number of misclassifications obtained from a leave-one-out cross-validation procedure. The P-SVM is compared with the "TrkC"-gene classification (one gene classification), R2W2, "weighted voting" classification (the sum of the features weighted by their correlation to class labels according to the signal-to-noise statistics), K-nearest neighbor, and combined SVM-TrkC-KNN classifier. Feature selection was based on the correlation of features with classes using signal-to-noise statistics (Golub et al., 1999) for K-nearest neighbor and weighted voting. The other feature selection method is called SPLASH, developed by Califano et al. (1999). SPLASH is a greedy subset selection method (wrapper method) and the subsequent classifier is a likelihood ratio classifier (LRC) based on density estimation for each gene. For the R2W2 SVM technique (Weston et al., 2000), see section 15.2.2. The results show that the P-SVM method clearly

Table 15.4 Summary of parameter values used in the numerical experiments. Top, The first column (data set) gives the number of the data set (1, brain tumor; 2, lymphoma; 3, breast cancer). The second column (samples) reports the number of tumor samples (patients). The third column (genes) gives the number of probes (genes) in the original data. The fourth column (n_ϵ) shows the number n_ϵ of ϵ runs in our protocol. The fifth column (ϵ) lists the ϵ values for the runs. Bottom, The second column (step 2) tells whether step 2 of our protocol is performed, that is, whether the p-values were available. The third column (q_1) shows the p-value threshold for step 1 of our protocol. The fourth column (q_2) gives the minimal number of samples which must possess a p-value below the threshold in step 1 of our protocol to select the gene. The fifth column (N) is the number of features after step 1 of our protocol. The sixth column (a) lists the number of features which are subtracted and added to the actual feature number to build an average. The seventh column (F) shows the average number of features used for classification.

data set	samples	genes	n_ϵ	ϵ
1	60	7129	3	0.25, 0.15, 0.05
2	58	7129	3	0.23, 0.13, 0.03
3	78	24481	4	0.1, 0.07, 0.03, 0.01

data set	step 2	q_1	q_2	N	a	F
1	no	–	–	7129	5	45
2	no	–	–	7129	3	18
3	yes	0.02	20	3623	5	30

outperforms standard methods—the number of misclassifications is down by a factor of 3.

Results for the Lymphoma Data Set The data set from Shipp et al. (2002) was processed according to the protocol from section 15.4, except for step 2 because of the missing p-values. For the P-SVM the optimal number of features was 18.

Table 15.6 summarizes the results. The P-SVM is compared with weighted voting, K-nearest neighbor, and the R2W2 technique. The signal-to-noise statistics was used to select feature for weighted voting and KNN. The new P-SVM selected more features than the selection methods taken from Shipp et al. (2002). The increased number of features in this experiment was not too surprising because sometimes "many genes are better than a few genes" to reduce the impact of "correlations by chance" (see subsection 15.4.2). The P-SVM method yields comparable to slightly better results than the best methods from Shipp et al. (2002).

Table 15.5 Brain tumor data set: comparison of different approaches to prediction of therapy outcome based on the DNA microarray data (for explanation, see text). The table shows the leave-one-out error given by the number of wrong classifications (E) for a given number of selected features (F). For R2W2 * means that there is no "number of features" (R2W2 scales features and does not select features). For the P-SVM/ν-SVM the protocol determined $\nu = 0.4$. Features were selected using signal-to-noise-statistics (statistics), R2W2 (Weston et al., 2000), SPLASH (Califano et al., 1999), and P-SVM. Data with statistical feature selection are provided for TrkC—gene classification, weighted voting, K-nearest neighbor (KNN), combined SVM/TrkC/KNN (Comb). For SPLASH the classifier is a likelihood ratio classifier (LRC). The ν-SVM is used as a classifier after feature selection with P-SVM. Except for our method (P-SVM/ν-SVM), results were taken from Pomeroy et al. (2002).

Feature Selection / Classification	# F	# E
TrkC (one gene)	1	20
SPLASH / LRC		15
R2W2	*	15
statistics / weighted voting		14
statistics / KNN	8	13
Comb		12
P-SVM / ν-SVM	45	4

Results for the Breast Cancer Data Set Before further processing of the data set from van't Veer et al. (2002) the log-ratios of the expression values were transformed according to

$$S_j = \text{sgn}\,(L_j)\ 2^{|L_j|} = \begin{cases} \frac{R_j}{G_j} & \text{for } R_j \geq G_j \\ -\frac{G_j}{R_j} & \text{otherwise .} \end{cases} \tag{15.29}$$

This transformation was performed to scale up the ratios into magnitudes of the original R_j (see Step 1 of the protocol). Because p-values were given, step 2 of our protocol was performed and we set the parameters q_1 and q_2 (see table 15.4) to pick between 3000 and 8000 genes (however, we did not optimize these values); 3623 were selected after step 2 for further processing. For the P-SVM the optimal number of features was 30.

In van't Veer et al. (2002) the results for different classification threshold values are published in the supplementary information report. That allowed us to compare classifiers according to the receiver operating characteristic (ROC) curve. The ROC curve consists of points whose x-components (distance to the left) denote the false-positive rate (class 2 misclassification rate), that is, wrongly positive classified negatives divided by the overall number of negatives. The y-components denote the true-positive rate, that is, correctly classified positives divided by the overall number of positives. Note that $(1-y)$ (distance to the top) is the false-negative rate

Table 15.6 Lymphoma data set: comparison of different approaches to prediction of therapy outcome based on the DNA microarray data (for explanation, see text). The columns are as in table 15.5. The outer loop of the protocol yielded $\nu = 0.5$ for the P-SVM/ν-SVM method. Feature selection is done by signal-to-noise-statistics (statistic), R2W2, and the P-SVM. The classifiers are K-nearest neighbor (KNN), weighted voting, and R2W2. Except for P-SVM, results were taken from Shipp et al. (2002).

Feature Selection / Classification	# F	# E
statistic / KNN	8	16
statistic / weighted voting	13	14
R2W2	*	13
P-SVM / ν-SVM	18	12

(class 1 misclassification rate) and that $n\,x + p\,(1-y)$ is the overall misclassification rate, where n is the fraction of class 2 (negative) examples and p the fraction of class 1 (positive) examples. For this experiment we observe $n = 0.44$ and $p = 0.56$, therefore, the overall misclassification rate is approximately $x + (1 - y)$. A high-performance classifier has a ROC curve which is close to the left upper corner ($x = 0$ and $y = 1$—no misclassification). The ROC curve gives more information on the quality of the classifier, especially if it is required to work in different regimens, for example, under the requirement that class 1 or class 2 misclassifications should be below a given threshold. The ROC allows, for example, optimization for (a) the selection of patients for adjuvant therapy where negative therapy outcome should not be misclassified, that is, a small false-positive rate required (small x-values should have large y-values: the curve steeply increases at the left hand side); (b) the selection of patients for alternative treatment where positive treatment outcome should not be misclassified, that is, a small false-negative rate required (large y-values should have small x-values: the curve should not decrease starting from the right upper corner); (c) the selection of good indicator genes (indicated by both the minimal misclassification error given by the minimal distance $x + (1 - y)$ of the ROC curve to the left upper corner and the area under the ROC curve).

Table 15.7 reports the results for the breast cancer data set. The ROC curves are shown in figure 15.3 where the threshold b of the ν-SVM classifier was varied to produce the ROC curve for the P-SVM/ν-SVM method. For comparison these figures also contain the weighted voting result from van't Veer et al. (2002) (supplementary information). Item (a) is the goal described in van't Veer et al. (2002) (distance of the left part of the ROC curve to the top), where the poor-prognosis patients should be recognized. In contrast to (a), in (b) positive-prognosis patients should be recognized (distance of the top of the ROC curve to the left). The ROC curve judges all the regimens between (a) and (b). For (c) we suggest two indicators, the minimum leave-one-out error (indicated by the distance $x + (1 - y)$ of the ROC curve to the left upper corner) and the area under the ROC curve, where both indicators must use all genes in an optimal way. For (a) the poor-prognosis indica-

Table 15.7 Breast cancer data set: comparison of different approaches to predict therapy outcome based on the DNA microarray data (for explanation, see text). Features are selected by Fisher statistics (statistics) and the P-SVM. The classifiers are weighted voting and ν-SVM where weighted voting results were taken from van't Veer et al. (2002). The number F of selected features, the number E of minimal misclassifications over the threshold range, and the area under the ROC curve are shown. The protocol chose $\nu = 0.2$ for the ν-SVM.

Feature Selection / Classification	# F	min. # E	ROC area
statistics / weighted voting	70	20	0.77
P-SVM / ν-SVM	30	12	0.88

tors are more important and in (b) the positive-prognosis indicators are the most relevant. The P-SVM results are comparable with van't Veer et al. (2002) for (a), but the results are better for (b) and (c). Overall, the P-SVM method performed better than weighted voting in van't Veer et al. (2002) which is expressed by the larger values for the ROC areas. Larger values of the area under the ROC curve (ROC area) mean higher performance where the minimum is 0.5 and the maximum 1.0.

Table 15.7 shows that the P-SVM method identified a smaller number of genes. Here a small number of genes is desirable because we already discarded genes with present calls based on less than 20 reliable values, which reduces the risk of random correlations.

15.6 Summary

We have introduced a new feature selection method based on the SVM technique and applied it to the analysis of DNA microarray data. In contrast to previous approaches, features become support vectors and determine the classification boundary. This allows selection and ranking of features by the support vector weights. Because the set of support vectors is sparse and avoids redundancy, the P-SVM approach is preferred over statistical methods which cannot recognize redundant information. We described a data analysis protocol for the extraction of relevant genes from microarray data which can be used with the P-SVM as well as with other feature selection techniques. We compared our feature selection approach on toy problems with state-of-the-art feature selection techniques and showed that our method gave the best results. Finally, we applied the P-SVM method to three data sets where the outcome of chemo- or radiation therapy for cancer or tumors is predicted on the basis of gene expression profiles obtained from the microarray technique. The P-SVMs outperform previously used algorithms due to an improved selection of relevant genes.

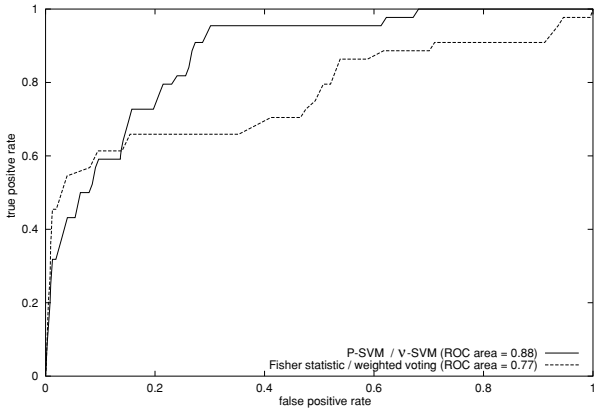


Figure 15.3 Breast cancer data set: the receiver operating characteristic (ROC) curve is shown for the P-SVM feature selection followed by a ν-SVM classifier (*solid line*) and for the weighted voting approach of van't Veer et al. (2002) (*dotted line*). The number of selected features was F=30 for the P-SVM. The P-SVM outperformed the weighted voting approach of van't Veer et al. (2002) except for small numbers of false positives (*left*).

Acknowledgments

We thank Cyril Minoux, Raman Sanyal, Merlyn Albery-Speyer, and Christoph Büscher for their help with the numerical simulations. This work was funded by the DFG (SFB 618).

Appendix A: Derivation of the Dual Optimization Problem for the P-SVM

The primal optimization problem of the P-SVM is

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2} \|\mathbf{X}^\top \mathbf{w}\|^2 \\ \text{subject to} \quad & \mathbf{X} \left(\mathbf{X}^\top \mathbf{w} + b\mathbf{1} - \mathbf{y}\right) + \epsilon \mathbf{1} \geq \mathbf{0} \\ & \mathbf{X} \left(\mathbf{X}^\top \mathbf{w} + b\mathbf{1} - \mathbf{y}\right) - \epsilon \mathbf{1} \leq \mathbf{0} \ . \end{aligned} \tag{15.30}$$

Following standard techniques of constrained optimization, we now derive the dual formulation of the optimization problem. The Lagrangian L is given by

$$\begin{aligned} L = \frac{1}{2}\, \mathbf{w}^\top\, \mathbf{X}\, \mathbf{X}^\top\, \mathbf{w}\; - \\ \left(\boldsymbol{\alpha}^+\right)^\top\; \left(\mathbf{X}\left(\mathbf{X}^\top\, \mathbf{w}\; + b\mathbf{1}\; -\; \mathbf{y}\right)\; +\; \epsilon\, \mathbf{1}\right)\; + \\ \left(\boldsymbol{\alpha}^-\right)^\top\; \left(\mathbf{X}\left(\mathbf{X}^\top\, \mathbf{w}\; + b\mathbf{1}\; -\; \mathbf{y}\right)\; -\; \epsilon\, \mathbf{1}\right)\;\; , \end{aligned} \tag{15.31}$$

where the vectors $\boldsymbol{\alpha}^+ \;\geq\; \mathbf{0}$ and $\boldsymbol{\alpha}^- \;\geq\; \mathbf{0}$ are the Lagrange multipliers for the constraints in (15.30). The optimality conditions (Schölkopf and Smola, 2002) require that the following derivatives with respect to the primal variables of the Lagrangian L are zero:

$$\begin{aligned} \nabla_{\mathbf{w}} L = \mathbf{X}\, \mathbf{X}^\top\, \mathbf{w}\; -\; \mathbf{X}\, \mathbf{X}^\top\, \boldsymbol{\alpha}\; =\; \mathbf{0}\;, \\ \frac{\partial L}{\partial b} = \mathbf{1}^\top\, \mathbf{X}^\top\, \boldsymbol{\alpha}\; =\; 0\;\; , \end{aligned} \tag{15.32}$$

where we used the abbreviation $\boldsymbol{\alpha} = \boldsymbol{\alpha}^+ \;-\; \boldsymbol{\alpha}^-$ ($\alpha_i = \alpha_i^+ \;-\; \alpha_i^-$). In order to ensure the first condition $\mathbf{X}\, \mathbf{X}^\top\, \mathbf{w} \;=\; \mathbf{X}\, \mathbf{X}^\top\, \boldsymbol{\alpha}$ we set

$$\mathbf{w} \;=\; \boldsymbol{\alpha}\;\; . \tag{15.33}$$

We then obtain the dual optimization problem of the P-SVM:

$$\begin{aligned} \min_{\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-}\;\; & \frac{1}{2}\left(\boldsymbol{\alpha}^+ \;-\; \boldsymbol{\alpha}^-\right)^\top\; \mathbf{X}\, \mathbf{X}^\top\; \left(\boldsymbol{\alpha}^+ \;-\; \boldsymbol{\alpha}^-\right)\; - \\ & \mathbf{y}^\top\, \mathbf{X}^\top\; \left(\boldsymbol{\alpha}^+ \;-\; \boldsymbol{\alpha}^-\right)\; +\; \epsilon\, \mathbf{1}^\top \left(\boldsymbol{\alpha}^+ \;+\; \boldsymbol{\alpha}^-\right) \\ \text{subject to}\;\; & \mathbf{1}^\top\, \mathbf{X}^\top\; \left(\boldsymbol{\alpha}^+ \;-\; \boldsymbol{\alpha}^-\right)\; =\; 0\;, \\ & \mathbf{0}\; \leq\; \boldsymbol{\alpha}^+\;,\;\; \mathbf{0}\; \leq\; \boldsymbol{\alpha}^-\;\; . \end{aligned} \tag{15.34}$$

Normalization of $\mathbf{X}$ to zero mean during preprocessing automatically leads to the satisfaction of $\mathbf{1}^\top\, \mathbf{X}^\top\; \left(\boldsymbol{\alpha}^+ \;-\; \boldsymbol{\alpha}^-\right)\; =\; 0$.

If vectors $\mathbf{u} \;\neq\; \mathbf{0}$ exist for which $\mathbf{X}^\top \mathbf{u} \;=\; 0$ holds, then the solution of (15.33) and of the primal optimization problem (15.30) is not unique (in the primal problem $\mathbf{w}$ appears only in context $\mathbf{X}^\top \mathbf{w}$). For positive values of ϵ, however, this degeneracy does not matter and a vector $\mathbf{w}$ is chosen that is most sparse in its components, that is, which has the largest number of zero components. The sparseness is due to $\mathbf{w} \;=\; \boldsymbol{\alpha}$ and the dual problem eqs. (15.34), where $\boldsymbol{\alpha}$ appears only in context $\mathbf{X}^\top \boldsymbol{\alpha}$ except for the ϵ-part which enforces sparseness.

Appendix B: Measurements of Complex Features

Here we consider the case that objects are fully described by a feature vector $\mathbf{x}$, but that we have measurement devices at hand that do not allow us to measure all of its individual components. Instead we assume that a measurement apparatus allows us to determine the values of a limited set of $\tilde{N}$ complex features $\mathbf{v}$. The complex

Complex features

features $\mathbf{v}$ are linear combinations of the elementary features $(\mathbf{x})_l$, $1 \leq l \leq N$, and the value of a complex feature j for an object i is given by the dot product

$$K_{ij} = (\mathbf{x}_i \cdot \mathbf{v}_j) \quad . \tag{15.35}$$

Measurement matrix

If we define the matrix $\mathbf{V} := (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{\tilde{N}})$, our (incomplete) knowledge about the set $\mathbf{X}$ of objects can be summarized by the measurement or data matrix $\mathbf{K}$,

$$\mathbf{K} \;=\; \mathbf{X}^\top \, \mathbf{V} \quad . \tag{15.36}$$

For an application to microarray data, for example, we would identify the measured matrix $\mathbf{K}$ with the matrix of expression values obtained by a microarray experiment.

The complex features $\mathbf{v}$ span a subspace of the original feature space, but we do not require them to be orthogonal, normalized, or linearly independent. Due to the measurements, all objects are now described by an $\tilde{N}$-dimensional feature vector $(K_{i1}, K_{i2}, \ldots, K_{i\tilde{N}}) \;=\; ((\mathbf{x}_i \cdot \mathbf{v}_1), (\mathbf{x}_i \cdot \mathbf{v}_2), \ldots, (\mathbf{x}_i \cdot \mathbf{v}_{\tilde{N}}))$. If the number $\tilde{N}$ of different measurements is large, overfitting may occur. In order to obtain good generalization performance, feature selection must be performed on the set $\mathbf{V}$ of complex features.

Using the same line of arguments as in subsection 15.3.3, the following constraints can be derived:

$$\begin{aligned} &\mathbf{K}^\top \left(\mathbf{X}^\top \, \mathbf{w} \; + b\mathbf{1} \; - \; \mathbf{y} \right) \; - \; \epsilon \, \mathbf{1} \;\leq\; \mathbf{0} \; , \\ &\mathbf{K}^\top \left(\mathbf{X}^\top \, \mathbf{w} \; + b\mathbf{1} \; - \; \mathbf{y} \right) \; + \; \epsilon \, \mathbf{1} \;\geq\; \mathbf{0} \quad . \end{aligned} \tag{15.37}$$

Again we normalize the vectors which correspond to single genes,

$$\sum_{i=1}^{m} K_{ij} \;=\; 0 \;\text{ and }\; \frac{1}{m} \sum_{i=1}^{m} K_{ij}^2 \;=\; 1 \; , \tag{15.38}$$

and—together with the P-SVM objective of subsection 15.3.2—we obtain the primal optimization problem:

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \, \| \mathbf{X}^\top \, \mathbf{w} \|^2 \\ \text{subject to} \quad & \mathbf{K}^\top \left(\mathbf{X}^\top \, \mathbf{w} \; + b\mathbf{1} \; - \; \mathbf{y} \right) \; + \; \epsilon \, \mathbf{1} \;\geq\; \mathbf{0} \\ & \mathbf{K}^\top \left(\mathbf{X}^\top \, \mathbf{w} \; + b\mathbf{1} \; - \; \mathbf{y} \right) \; - \; \epsilon \, \mathbf{1} \;\leq\; \mathbf{0} \quad . \end{aligned} \tag{15.39}$$

The corresponding dual formulation is

$$\begin{aligned} \min_{\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-} \quad & \frac{1}{2} \left(\boldsymbol{\alpha}^+ \; - \; \boldsymbol{\alpha}^- \right)^\top \, \mathbf{K}^\top \, \mathbf{K} \, \left(\boldsymbol{\alpha}^+ \; - \; \boldsymbol{\alpha}^- \right) \; - \\ & \mathbf{y}^\top \, \mathbf{K} \, \left(\boldsymbol{\alpha}^+ \; - \; \boldsymbol{\alpha}^- \right) \; + \; \epsilon \, \mathbf{1}^\top \left(\boldsymbol{\alpha}^+ \; + \; \boldsymbol{\alpha}^- \right) \\ \text{subject to} \quad & \mathbf{1}^\top \, \mathbf{K} \, \left(\boldsymbol{\alpha}^+ \; - \; \boldsymbol{\alpha}^- \right) \;=\; 0 \; , \\ & \mathbf{0} \;\leq\; \boldsymbol{\alpha}^+ \; , \;\; \mathbf{0} \;\leq\; \boldsymbol{\alpha}^- \quad , \end{aligned} \tag{15.40}$$

where the normal vector $\mathbf{w}$ has now been expanded with respect to the complex features $\mathbf{v}$,

$$\mathbf{w} \;=\; \mathbf{V}\,\boldsymbol{\alpha} \quad . \tag{15.41}$$

The offset is again obtained by

$$b \;=\; -\frac{1}{m}\,\sum_{i=1}^{m}\left((\mathbf{w}\cdot\mathbf{x}_i) \;-\; y_i\right) \;, \tag{15.42}$$

from which we obtain the classification function

$$f(\mathbf{u}) \;=\; (\mathbf{w}\cdot\mathbf{u}) \;+\; b \;=\; \sum_{j=1}^{\tilde{N}}\alpha_j\;(\mathbf{u}\cdot\mathbf{v}_j) \;+\; b \;=\; \sum_{j=1}^{\tilde{N}}\alpha_j\;K_{i_u j} + \; b\;. \tag{15.43}$$

Note that $\mathbf{K}$ is neither required to be positive definite nor square, because only the quadratic part $\mathbf{K}^\top\mathbf{K}$ appears in the objective function of (15.40). Therefore, we may consider $\mathbf{K}$ as the Gram matrix of a kernel which is not positive definite, that is, a kernel which is not a Mercer kernel. Indeed, it has been shown that kernels which are not positive definite can be used for classification without any loss of generalization performance (Hochreiter and Obermayer, 2003b).

The new interpretation allows data to be treated in matrix form, where the matrix entries express the relationships between two sets of objects ("row objects" and "column objects"), and also allows application of machine learning methods, like classification, regression, or clustering, to this data. The matrix originates from a dot product of representations of these objects in some feature space. An example of such matrix data is the drug-gene matrix in Scherf et al. (2000), where a drug-cell matrix and a gene-cell matrix (expression values from a microarray experiment) are multiplied to obtain a drug-gene matrix. In this example the matrices $\mathbf{X}$ and $\mathbf{V}$ can be identified; however, the proposed framework allows these matrices to be related not only by a plain dot product but by some kernel evaluation. In this case our new interpretation holds as long as the kernel represents a dot product in some space (see kernel/dot product considerations below).

Matrix data

A special case of data in matrix form occurs if the row objects are identical to the column objects. This case is called "pairwise data," and the data matrix is usually interpreted as a similarity matrix $\mathbf{K}$. The advantage of the interpretation put forward in this appendix is that the P-SVM framework can still be applied by setting $\mathbf{V} \;=\; \mathbf{X}$. Pairwise data are common in bioinformatics, for example, when considering the similarity measures for protein sequences (Lipman and Pearson, 1985), functional similarities of proteins (Sigrist et al., 2002; Falquet et al., 2002), chromosome location similarities of genes (Cremer et al., 1993; Lu et al., 1994), or coexpression data for genes (Heyer et al., 1999).

Pairwise data

One issue, however, still remains open. Under what conditions is it possible to interpret a matrix of measured values as a dot product matrix $\mathbf{K}$? There is no full answer to this question from a theoretical viewpoint; practical applications have to confirm (or disprove) the chosen ansatz and data model. However, the question

whether it is possible to describe a measurement by a dot product can be replaced by the question whether or not the following three conditions hold (see Hochreiter and Obermayer, 2003a):

1. Column objects ("samples") to classify are elements of a Hilbert space H_1, that is, given a basis, these objects can be described by (possibly infinite dimensional) vectors.

2. Row objects ("complex features") are elements of a Hilbert space H_2, that is, given a basis, these features can be described by (possibly infinite dimensional) vectors.

Measurement device as kernel

3. The measurement process can be expressed via the evaluation of a kernel.

Condition (3) equates the evaluation of a kernel as known from standard SVMs with physical measurements. As the kernel matrix is measured, no model selection has to be performed w.r.t. the kernel.

References

S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. San Francisco, Morgan Kaufmann, 2000.

A.V. Aho and M.J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6):333 – 340, June 1975.

M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

S. Akaho. A kernel method for canonical correlation analysis. In *Proceedings of the 2000 Workshop on Information-Based Induction Sciences (IBIS2000),* 17–18 July 2000, Izu, Japan, pages 123–128, 2001.

J. Albert and S. Chib. Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association*, 88:669–679, 1993.

B. Alberts, D. Bray, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Essential Cell Biology: An Introduction to the Molecular Biology of the Cell*. New York, Garland Science Publishing, 1998.

B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. New York, Garland Science Publishing, 4th edition, 2002.

S. V. Allander, N. N. Nupponen, M. Ringner, G. Hostetter, G. W. Maher, N. Goldberger, Y. Chen, J. Carpten, A. G. Elkahloun, and P. S. Meltzer. Gastrointestinal stromal tumors with KIT mutations exhibit a remarkably homogeneous gene expression profile. *Cancer Research*, 61:8624–8628, 2001.

E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

H. Almuallim and T. G. Dietterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 547–552, Anaheim, CA, AAAI Press, 1991.

U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack, and A. J. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences of the United States of America*, 96:6745–6750, 1999.

S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines. In T. Faucett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning*, pages 3–10, Menlo Park, CA, AAAI Press, 2003.

S. Amari. Information geometry of the EM and em algorithms for neural networks. *Neural Networks*, 9:1379–1408, 1995.

S.-I. Amari and H. Nagaoka. *Methods of Information Geometry*. Oxford, UK, Oxford University Press, 2000.

C. Ambroise and G. J. McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Academy of Sciences of the United States of America*, 99(10):6562–6566, 2002.

A. Amir, M. Farach, Z. Galil, R. Giancarlo, and K. Park. Dynamic dictionary matching. *Journal of Computer and System Science*, 49(2):208–222, October 1994.

D. C. Anderson, W. Li, D. G. Payan, and W. S. Noble. A new algorithm for the evaluation of shotgun peptide sequencing in proteomics: Support vector machine classification of peptide MS/MS spectra and SEQUEST scores. *Journal of Proteome Research*, 2(2):137–146, 2003.

C.B. Anfinsen, C.J. Epstein, and R.F. Goldberger. The genetic control of tertiary protein structure: studies with model systems. *Cold Spring Harbor Symposia on Quantitative Biology*, 28:439–449, 1963.

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

T. K. Attwood, M. E. Beck, D. R. Flower, P. Scordis, and J. N Selley. The PRINTS protein fingerprint database in its fifth year. *Nucleic Acids Research*, 26(1):304–308, 1998.

F. R. Bach and M. I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1–48, 2002.

C. Bahlmann, B. Haasdonk, and H. Burkhardt. On-line handwriting recognition with support vector machines – a kernel approach. In *Proceedings of the 8th International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 49–54, Niagara-on-the-Lake, Canada, August 6-8, 2002.

W. Bains and G. Smith. A novel method for nucleic acid sequence determination. *Journal of Theoretical Biology*, 135:303–307, 1988.

A. Bairoch. The PROSITE database, its status in 1995. *Nucleic Acids Research*, 24:189–196, 1995.

J. Bala, K. A. D. Jong, J. Haung, H. Vafaie, and H. Wechsler. Hybrid learning using genetic algorithms and decision trees for pattern classification. In C. S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 719–724. San Francisco, Morgan Kaufmann, 1995.

P. Baldi and S. Brunak. *Bioinformatics: the Machine Learning Approach*. Cambridge, MA, MIT Press, 2001.

P. Baldi, S. Brunak, Y. Chauvin, C.A. Andersen, and H. Nielsen. Assessing the accuracy of prediction algorithms for classification: An overview. *Bioinformatics*, 16(5):412–424, 2000.

P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri. Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15(11): 937–946, 1999.

P. Baldi, Y. Chauvin, T. Hunkapiller, and M. A. McClure. Hidden Markov models of biological primary sequence information. *Proceedings of the National Academy of Sciences of the United States of America*, 91(3):1059–1063, 1994.

R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Philadelphia, Society for Industorial and Applied Mathematics (SIAM), second edition, 1994.

P. L. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods—Support Vector Learning*, pages 43–54, Cambridge, MA, MIT Press, 1999.

N. Beerenwinkel, M. Daumer, M. Oette, K. Korn, D. Hoffmann, R. Kaiser, L. Lengauer, J. Selbig, and H. Walter. Geno2pheno: Estimating phenotypic drug resistance from HIV-1 genotypes. *Nucleic Acids Research*, 31(13):3850–3855, 2003.

M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 585–591. Cambridge, MA, MIT Press, 2002.

M. Belkin and P. Niyogi. Using manifold structure for partially labelled classification. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. Cambridge, MA, MIT Press, 2003.

A. J. Bell and T. J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995.

R. E. Bellman. *Adaptive Control Processes*. Princeton, NJ, Princeton University Press, 1961.

A. Ben-Dor, L. Bruhn, N. Friedman, I. Nachman, M. Schummer, and Z. Yakhini. Tissue classification with gene expression profiles. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology (RE-*

COMB), New York, ACM Press, 2000.

A. Ben-Hur and D. Brutlag. Remote homology detection: A motif based approach. *Bioinformatics*, 19(Suppl. 1):i26–i33, 2003.

K. P. Bennett. Combining support vector and mathematical programming methods for induction. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods—SV Learning*, pages 307–326, Cambridge, MA, MIT Press, 1999.

D.A. Benson, M.S. Boguski, D.J. Lipman, J. Ostell, B.F. Ouellette, B.A. Rapp, and D.L. Wheeler. Genbank. *Nucleic Acids Research*, 27:12–17, 1999.

C. Berg, J. P. R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups: Theory of Positive Definite and Related Functions*. New York, Springer Verlag, 1984.

H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

J. Bi, K. Bennett, M. Embrechts, C. Breneman, and M. Song. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, 3:1229–1243, 2003. Special issue on variable and feature selection.

S. D. Black and D. R. Mould. Development of hydrophobicity parameters to analyze proteins which bear post- or cotranslational modifications. *Analytical Biochemistry*, 193:72–82, 1991.

A. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245–271, 1997.

J. R. Bock and D. A. Gough. Predicting protein-protein interactions from primary structure. *Bioinformatics*, 17:455–460, 2001.

B. E. Boser, I. M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, ACM Press, 1992.

S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. Philadelphia, Society for Industorial and Applied Mathematics (SIAM), 1994.

S. Boyd and L. Vandenberghe. Convex optimization. Course notes for EE364, Stanford University, Stanford, CA. Available from `http://www.stanford.edu/class/ee364`, August 2001.

P. S. Bradley and O. L. Mangasarian. Feature selection via concave minimization and support vector machines. In J. Shavlik, editor, *Machine Learning Proceedings of the Fifteenth International Conference (ICML '98)*, pages 82–90, San Francisco, Morgan Kaufmann, 1998.

M. Branca. Putting gene arrays to the test. *Science*, 300:238, 2003.

E.J. Bredensteiner and K.P. Bennett. Multicategory classification by support vector machines. *Computational Optimization and Applications*, 12(1/3):53–79, 1999.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Belmont, CA, Wadsworth International Group, 1984.

M. P. S. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. S. Furey, Jr. M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene expression data using support vector machines. *Proceedings of the National Academy of Sciences of the United States of America*, 97(1):262–267, 2000.

C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268:78–94, 1997.

C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

M. Burset, I.A. Seledtsov, and V.V. Solovyev. Splice DB: Database of canonical and non-canonical mammalian splice sites. *Nucleic Acids Research*, 29(1):255–259, 2000.

D. Cai, A. Delcher, B. Kao, and S. Kasif. Modeling splice sites with Bayes networks. *Bioinformatics*, 16(2):152–158, 2000.

Y.-D. Cai, X.-J. Liu, X.-B. Xu, and G.-P. Zhou. Support vector machines for predicting protein structural class. *BMC Bioinformatics*, 2(3), 2001.

A. Califano, G. Stolovitzky, and Y. Tu. Analysis of gene expression microarrays for phenotype classification. In T. Lengauer, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, H.-W. Mewes, and R. Zimmer, editors, *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 75–85, Menlo Park, CA, AAAI Press, 1999.

C. Cardie. Using decision trees to improve case–based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 25–32. San Francisco, Morgan Kaufmann, 1993.

J.-F. Cardoso and A. Souloumiac. Blind beamforming for non Gaussian signals. *IEE Proceedings-F*, 140(6):362–370, 1993.

R. J. Carter, I. Dubchak, and S. R. Holbrook. A computational approach to identify genes for functional RNAs in genomic sequences. *Nucleic Acids Research*, 29(19): 3928–3938, 2001.

R. Caruana and D. Freitag. Greedy attribute selection. In *International Conference on Machine Learning*, pages 28–36, 1994.

C. Chang and C. Lin. LIBSVM: A library for support vector machines, 2001. Available from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/index.html`.

W. I. Chang and E. L. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica*, 12(4/5):327–344, 1994.

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–159, 2002.

C. P. Chen and B. Rost. State-of-the-art in membrane protein prediction. *Applied Bioinformatics*, 1(1):21–35, 2002.

K. C. Chou and D. Elrod. Protein subcellular location prediction. *Protein Engineering*, 12:107–118, 1999.

S. Chu, J. DeRisi, M. Eisen, J. Mulholland, D. Botstein, P. Brown, and I. Herskowitz. The transcriptional program of sporulation in budding yeast. *Science*, 282:699–705, 1998.

J.S. Chuang and D. Roth. Splice site prediction using a sparse network of winnows. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, February 2001.

Fan R. K. Chung. *Spectral Graph Theory*. Number 92 in Regional Conference Series in Mathematics. American Mathematical Society, Providence, RI, 1997.

Fan R. K. Chung and Shing-Tung Yau. Coverings, heat kernels and spanning trees. *Electronic Journal of Combinatorics*, 6, 1999.

M. Collins and N. Duffy. Convolution kernels for natural language. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 625–632, Cambridge, MA, MIT Press, 2002.

T. C. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. Cambridge, MA, MIT Press, 1990.

C. Cortes, P. Haffner, and M. Mohri. Rational kernels. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 601–608. Cambridge, MA, MIT Press, 2003.

T. M. Cover and J. M. V. Campenhout. On the possible orderings in the measurement selection problem. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(9):657–661, 1977.

K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

T. Cremer, A. Kurz, R. Zirbel, S. Dietzel, B. Rinke, E. Schröck, M. R. Speichel, U. Mathieu, A. Jauch, P. Emmerich, H. Schertan, T. Ried, C. Cremer, and P. Lichter. Role of chromosome territories in the functional compartmentalization of the cell nucleus. *Cold Spring Harbor Symposia on Quantitative Biology*, 58: 777–792, 1993.

N. Cristianini, A. Elisseeff, J. Shawe-Taylor, and J. Kandola. On kernel target alignment. Technical Report NC-TR-01-099, NeuroCOLT2, 2001.

N. Cristianini, H. Lodhi, and J. Shawe-Taylor. Latent semantic kernels. *Journal of Intelligent Information Systems (JJIS)*, 18(2-3):127–152, 2002a.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge, UK, Cambridge University Press, 2000.

N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola. On kernel-target alignment. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 367–373. Cambridge, MA, MIT Press, 2002b.

J.A. Cuff and G.J. Barton. Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. *Proteins*, 34:508–519, 1999.

S. Das. Filters, wrappers and a boosting-based hybrid for feature selection. In *Proceedings of the 18th International Conference on Machine Learning*, pages 74–81. San Francisco, Morgan Kaufmann, 2001.

M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A model of evolutionary change in proteins. In M.O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, pages 345–358. Silver Spring, MD, National Biomedical Research Foundation, 1978.

L. De Raedt and S. Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 853–862. San Francisco, Morgan Kaufmann, 2001.

S. Degroeve, B. De Baets, Y. Van de Peer, and P. Rouz. Feature subset selection for splice site prediction. *Bioinformatics*, 18:S75–S83, 2002.

A.L. Delcher, D. Harmon, S. Kasif, O. White, and S.L. Salzberg. Improved microbial gene identification with GLIMMER. *Nucleic Acids Research*, 27(23):4636–4641, 1999.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39:1–22, 1977.

M. Deng, T. Chen, and F. Sun. An integrated probabilistic model for functional prediction of proteins. In *Proceedings of the Seventh Annual International Conference on Computational Biology (RECOMB),* April 10-13, 2003, Berlin, Germany, pages 95–103, 2003a.

M. Deng, F. Sun, and T. Chen. Assessment of the reliability of protein-protein interactions and protein function prediction. In *Eighth Pacific Symposium on Biocomputing (PSB)*, pages 140–151, Kauai, Hawaii, 2003b.

J. L. DeRisi, V. R. Iyer, and P. O. Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278:680–686, 1997.

J.P. Derrick and D.B. Wigley. The third IgG-binding domain from streptococcal protein G: An analysis by X-ray crystallography of the structure alone and in a complex with Fab. *Journal of Molecular Biology*, 243(5):906–918, 1994.

E. Didiot. Conception et mise en œuvre de m-svm dédiées au traitement de séquences biologiques. Master's thesis, DEA informatique de Lorraine, France, 2003.

C. Ding and I. Dubchak. Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics*, 17(4):349–358, 2001.

A. Drawid and M. Gerstein. A Bayesian system integrating expression data with sequence patterns for localizing proteins: comprehensive application to the yeast genome. *Journal of Molecular Biology*, 301:1059–1075, 2000.

R. Drmanac, I. Labat, I. Brukner, and R. Crkvenjakov. Sequencing of megabase plus DNA by hybridization: theory of the method. *Genomics*, 4:114–128, 1989.

H. Drucker, C.J.C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 155–161. Cambridge, MA, MIT Press, 1997.

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. New York, Wiley, 1973.

B. Durbin, J. Hardin, D. Hawkins, and D. M. Rocke. A variance-stabilizing transformation for gene-expression microarray data. *Bioinformatics*, 18(Supplement 1):105–110, 2002.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis—Probabilistic Models of Proteins and Nucleic Acids*. Cambridge, UK, Cambridge University Press, 1998.

S. R. Eddy. Multiple alignment using hidden Markov models. In C. Rawlings, D. Clark, R. Altman, L. Hunter, T. Lengauer, and S. Wodak, editors, *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, pages 114–120. Menlo Park, CA, AAAI Press, 1995.

M. Eisen, P. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences of the United States of America*, 95:14863–14868, 1998.

A. Elisseeff. *Etude de la complexité et contrôle de la capacité des systèmes d'apprentissage : SVM multi-classe, réseaux de régularisation et réseaux de neurones multicouches*. PhD thesis, ENS Lyon, France, 2000.

The *C. elegans* Sequencing Consortium. Genome sequence of the nematode *Caenorhabditis elegans*. A platform for investigating biology. *Science*, 282:2012–2018, 1998.

J. K. Eng, A. L. McCormack, and J. R. Yates. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society for Mass Spectrometry*, 5:976–989, 1994.

D. M. Engleman, T. A. Steitz, and A. Goldman. Identifying nonpolar transbilayer helices in amino acid sequences of membrane proteins. *Annual Review of Biophysics and Biomolecular Structure*, 15:321–353, 1986.

M.D. Ermolaeva, O. White, and S.L. Salzberg. Prediction of operons in microbial genomes. *Nucleic Acids Research*, 29:1216–1221, 2001.

E. Eskin, W. S. Noble, Y. Singer, and S. Snir. A unified approach for sequence prediction using sparse sequence models. Technical report, Hebrew University, Jerusalem, 2003.

L. Falquet, M. Pagni, P. Bucher, N. Hulo, C. J. Sigrist, K. Hofmann, and A. Bairoch. The PROSITE database, its status in 2002. *Nucleic Acids Research*, 30:235–238, 2002.

M. A. T. Figueiredo. Adaptive sparseness for supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25, 2003.

M. A. T. Figueiredo and A. K. Jain. Bayesian learning of sparse classifiers. In *2001 Conference on Computer Vision and Pattern Recognition (CVPR 2001)*. Piscataway, NJ, IEEE Press, December 2001.

R. Fletcher. *Practical Methods of Optimization*. New York, Wiley, 1989.

M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

Y. Freund and R. Shapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

J. Friedman, T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. Discussion of "consistency in boosting". *Annals of Statistics*, 32(1), 2004.

J. H. Friedman. Another approach to polychotomous classification. Technical report, Department of Statistics and Stanford Linear Accelerator Center, Stanford University, Stanford, CA, 1996.

J. H. Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

J. H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American Statistical Association*, 76(376):817–823, December 1981.

J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, C-23(9):881–890, 1974.

T.-T. Frieß and R. F. Harrison. Linear programming support vector machines for pattern classification and regression estimation and the set reduction algorithm. TR RR-706, University of Sheffield, Sheffield, UK, 1998.

G. Fung and O. Mangasarian. A feature selection Newton method for support vector machine classification. Technical report, Data Mining Institute, University of Wisconsin, Madison, September 2002.

T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, and D. Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.

C. Fyfe and P. L. Lai. ICA using kernel canonical correlation analysis. In *Proceedings of International Workshop on Independent Component Analysis and Blind Signal Separation (ICA2000)*, pages 279–284, Helsinki, 2000.

A. Gammerman, V. Vovk, and V. Vapnik. Learning by transduction. In G. F. Cooper and S. Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 148–155. San Francisco, Morgan Kaufmann, 1998.

T. Gärtner. Exponential and geometric kernels for graphs. In *NIPS Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*, 2002. Available from http://mlg.anu.edu.au/unrealdata/.

T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In B. Schölkopf and M. Warmuth, editors, *Proceedings of the 16th Annual Conference on Computational Learning Theory and the Seventh Annual Workshop on Kernel Machines*, volume 2777 of *Lecture Notes in Computer Science*, Heidelberg, Springer-Verlag, 2003.

O. Gascuel and J.L. Golmard. A simple method for predicting the secondary structure of globular proteins: implications and accuracy. *Computer Applications in the Biosciences*, 4(3):357–365, 1988.

H. Ge, Z. Liu, G. Church, and M. Vidal. Correlation between transcriptome and interactome mapping data from *Saccharomyces cerevisiae*. *Nature Genetics*, 29: 482–486, 2001.

C. Geourjon and G. Deléage. SOPMA: Significant improvements in protein secondary structure prediction by consensus prediction from multiple alignments. *Computer Applications in the Biosciences*, 11(6):681–684, 1995.

D. Gerhold, T. Rushmore, and C. T. Caskey. DNA chips: Promising toys have become powerful tools. *Trends in Biochemical Sciences*, 24(5):168–173, 1999.

R. Giegerich and S. Kurtz. From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19(3):331–353, 1997.

M. Girolami. Mercer kernel based clustering in feature space. *IEEE Transactions on Neural Networks*, 13, 2002.

F. Girosi. An equivalence between sparse approximation and support vector machines. *Neural Computation*, 10(6):1455–1480, 1998.

F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. Baltimore, Johns Hopkins University Press, 1996.

T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, 1999.

S. M. Gomez, W. S. Noble, and A. Rzhetsky. Learning to predict protein-protein interactions. *Bioinformatics*, 19:1875–1881, 2003.

J. Gorodkin, R.B. Lyngso, and G.D. Stormo. A mini-greedy algorithm for faster structural RNA stem-loop search. In H. Matsuda, S. Miyano, T. Takagi, and L. Wong, editors, *Genome Informatics 2001*, pages 184–193. Tokyo, Universal

Academy Press, 2001.

S. Goto, Y. Okuno, M. Hattori, T. Nishioka, and M. Kanehisa. LIGAND: Database of chemical compounds and reactions in biological pathways. *Nucleic Acids Research*, 30:402–404, 2002.

T. Graepel, R. Herbrich, P. Bollmann-Sdorra, and K. Obermayer. Classification on pairwise proximity data. In M.S. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 438–444. Cambridge, MA, MIT Press, 1999a.

T. Graepel, R. Herbrich, B. Schölkopf, A. J. Smola, P. L. Bartlett, K. Müller, K. Obermayer, and R. C. Williamson. Classification on proximity data with LP-machines. In *Ninth International Conference on Artificial Neural Networks*, Conference Publications No. 470, pages 304–309, London, Institution of Electrical Engineers (IEE), 1999b.

T. Graepel and K. Obermayer. Fuzzy topographic kernel clustering. In *Proceedings of the Fifth GI Workshop Fuzzy Neuro Systems*, pages 90–97, Munich, Germany, March 19–20, 1998.

M. Gribskov, R. Lüthy, and D. Eisenberg. Profile analysis. *Methods in Enzymology*, 183:146–159, 1990.

M. Gribskov, A. D. McLachlan, and D. Eisenberg. Profile analysis: Detection of distantly related proteins. *Proceedings of the National Academy of Sciences of the United States of America*, 84:4355–4358, 1987.

M. Gribskov and N. L. Robinson. Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching. *Computers and Chemistry*, 20(1):25–33, 1996.

A. Grigoriev. A relationship between gene expression and protein interactions on the proteome scale: Analysis of the bacteriophage T7 and the yeast *Saccharomyces cerevisiae*. *Nucleic Acids Research*, 29:3513–3519, 2001.

W. N. Grundy, T. L. Bailey, C. P. Elkan, and M. E. Baker. Meta-MEME: Motif-based hidden Markov models of protein families. *Computer Applications in the Biosciences*, 13(4):397–406, 1997.

Y. Guermeur. *Combinaison de classifieurs statistiques, application à la prédiction de la structure secondaire des protéines*. PhD thesis, Université Paris 6, 1997.

Y. Guermeur. Combining discriminant models with new multi-class SVMs. *Pattern Analysis and Applications*, 5(2):168–179, 2002.

Y. Guermeur, A. Elisseeff, and D. Zelus. A comparative study of multi-class support vector machines in the unifying framework of large margin classifiers. *Applied Stochastic Models in Business and Industry*, 2003. (under revision).

Y. Guermeur, G. Pollastri, A. Elisseeff, D. Zelus, H. Paugam-Moisy, and P. Baldi. Combining protein secondary structure prediction models with ensemble methods of optimal complexity. *Neurocomputing*, 56C:305–327, 2004.

T. Gururaja, W. Li, W. S. Noble, D. G. Payan, and D. C. Anderson. Multiple functional categories of proteins identified in an in vitro cellular ubiquitin affinity extract using shotgun peptide sequencing. *Journal of Proteome Research*, 2:383–393, 2003.

L. Gurvits. A note on a scale-sensitive dimension of linear bounded functionals in Banach spaces. *Theoretical Computer Science*, 261(1):81–90, 2001.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003. Special issue on variable and feature selection.

I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

D. Hanisch, A. Zien, R. Zimmer, and T. Lengauer. Co-clustering of biological networks and gene expression data. *Bioinformatics*, 18:S145–S154, 2002.

J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36, 1982.

S. Harmeling, A. Ziehe, M. Kawanabe, B. Blankertz, and K.-R. Müller. Nonlinear blind source separation using kernel feature spaces. In T.-W. Lee, editor, *Proceedings of the International Workshop on Independent Component Analysis and Blind Signal Separation (ICA2001)*, pages 102–107, 2001.

A. Hartemink, D. Gifford, T. Jaakkola, and R. Young. Maximum likelihood estimation of optimal scaling factors for expression array normalization. In *Microarrays: Optical Technologies and Informatics, Proceedings of the International Society for Optical Engineering (SPIE)*, volume 4266, pages 132–140, 2001.

B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In J. D. Cowan S. J. Hanson and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 164–171. San Mateo, CA: Morgan Kaufmann, 1993.

T. Hastie, A. Buja, and R. Tibshirani. Penalized discriminant analysis. *Annals of Statistics*, 23:73–102, 1995.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. New York, Springer Verlag, 2001.

D. Haussler. Computational genefinding. *Trends in Biochemical Sciences,* Supplementary Guide to Bioinformatics, pages 12–15, 1998.

D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, Department of Computer Science, University of California at Santa Cruz, 1999.

S. Haykin. *Neural Networks : A Comprehensive Foundation.* New York, Macmillan, second edition, 1998.

S.M. Hebsgaard, P.G. Korning, N. Tolstrup, J. Engelbrecht, P. Rouze, and S. Brunak. Splice site recognition in *A. thaliana* DNA by combining local and

global sequence information. *Nucleic Acids Research*, 24:3439–3452, 1996.

C. Helma, R.D. King, S. Kramer, and A. Srinivasan. The predictive toxicology challenge 2000-2001. *Bioinformatics*, 17(1):107–108, 2001.

S. Henikoff and J. G. Henikoff. Automated assembly of protein blocks for database searching. *Nucleic Acids Research*, 19(23):6565–6572, 1991.

S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89:10915–10919, 1992.

R. Herbrich. *Learning Kernel Classifiers: Theory and Algorithms*. MIT Press, 2002.

R. Herbrich, T. Graepel, and C. Campbell. Bayes point machines. *Journal of Machine Learning Research*, 1:245–279, August 2001.

R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 115–132, Cambridge, MA, MIT Press, 2000.

L. J. Heyer, S. Kruglyak, and S. Yooseph. Exploring expression data: identification and analysis of coexpressed genes. *Genome Research*, 11:1106–1115, 1999.

A. A. Hill, E. L. Brown, M. Z. Whitley, G. Tucker-Kellogg, C. P. Hunter, and D. K. Slonim. Evaluation of normalization procedures for oligonucleotide array data based on spiked cRNA controls. *Genome Biology*, 3(1):research0055.1–0055.13, 2001.

S. Hochreiter and K. Obermayer. Classification of pairwise proximity data with support vectors. In Y. LeCun and Y. Bengio, editors, *The Learning Workshop*. Snowbird, UT, Computational & Biological Learning Society, 2002.

S. Hochreiter and K. Obermayer. Feature selection and classification on matrix data: From large margins to small covering numbers. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 889–896. Cambridge, MA, MIT Press, 2003a.

S. Hochreiter and K. Obermayer. Feature selection and matrix data. Technical report, Technische Universität Berlin, Fakultät für Elektrotechnik und Informatik, 2003b.

S. Hochreiter and J. Schmidhuber. LOCOCODE performs nonlinear ICA without knowing the number of sources. In J.-F. Cardoso, C. Jutten, and P. Loubaton, editors, *Proceedings of the First International Workshop on Independent Component Analysis and Signal Separation,* Aussois, France, pages 149–154, 1999.

L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233(1):123–38, 1993.

I. Holmes and W. J. Bruno. Finding regulatory elements using joint likelihoods for sequence and expression profile data. In *Proceedings of the Ninth International Conference on Intelligent Systems for Molecular Biology*, pages 202–210, 2000.

J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Reading, MA, Addison-Wesley, 1979.

T. P. Hopp and K. R. Woods. Prediction of protein antigenic determinants from amino acid sequences. *Proceedings of the National Academy of Sciences of the United States of America*, 78:3824–3828, 1981.

H. Hotelling. Relation between two sets of variates. *Biometrika*, 28:321–377, 1936.

C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, 2003.

C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13:415–425, 2002.

S. Hua and Z. Sun. A novel method of protein secondary structure prediction with high segment overlap measure: Support vector machine approach. *Journal of Molecular Biology*, 308:397–407, 2001a.

S. Hua and Z. Sun. Support vector machine approach for protein subcellular localization prediction. *Bioinformatics*, 17(8):721–728, 2001b.

P. J. Huber. Projection pursuit. *Annals of Statistics*, 13(2):435–475, 1985.

W. Huber, A. von Heydebreck, H. Sültmann, A. Poustka, and M. Vingron. Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics*, 18(Supplement 1):96–104, 2002.

R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: Extension and analysis of the basic method. *Computer Applications in the Biosciences*, 12 (2):95–107, 1996.

A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. New York, Wiley, 2001.

S. Ikeda, S. Amari, and H. Nakahara. Convergence of the wake-sleep algorithm. In M.S. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 239–245. Cambridge, MA, MIT Press, 1999.

A. Inokuchi, T. Washio, and H. Motoda. An a priori-based algorithm for mining frequent substructures from graph data. In D.A. Zighed, H.J. Komorowski, and J.M. Zytkow, editors, *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, volume 1910 of *Lecture Notes in Computer Science*, pages 13–23, Heidelberg, Springer-Verlag, 2000.

T. Ito, T. Chiba, R. Ozawa, M. Yoshida, M. Hattori, and Y. Sakaki. A comprehensive two-hybrid analysis to explore the yeast protein interactome. *Proceedings of the National Academy of Sciences of the United States of America*, 98(8): 4569–4574, 2001.

T. Ito, K. Takemoto, H. Mori, and T. Gojobori. Evolutionary instability of operon structures disclosed by sequence comparisons of complete microbial genomes. *Molular Biology and Evolution*, 16:332–346, 1999.

T. Jaakkola, M. Diekhans, and D. Haussler. Using the Fisher kernel method to detect remote protein homologies. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158, Menlo Park, CA, AAAI Press, 1999.

T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2): 95–114, 2000.

T.S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In M.S. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 487–493. Cambridge, MA, MIT Press, 1999.

J. E. Jackson. *A User's Guide to Principal Components*. Wiley, New York, 1991.

J. Jäger, R. Sengupta, and W. L. Ruzzo. Improved gene selection for classification of microarrays. In *Biocomputing—Proceedings of the 2003 Pacific Symposium*, pages 53–64, Kauai, Hawaii, 2003.

R. Jansen, N. Lan, J. Qian, and M. Gerstein. Integration of genomic datasets to predict protein complexes in yeast. *Journal of Structural and Functional Genomics*, 2:71–81, 2002.

T. Jebara and R. Kondor. Bhattacharyya and expected likelihood kernels. In B. Schölkopf and M. Warmuth, editors, *Proceedings of the 16th Annual Conference on Computational Learning Theory and the Seventh Annual Workshop on Kernel Machines*, volume 2777 of *Lecture Notes in Computer Science*. Heidelberg, Springer-Verlag, 2003.

T. Joachims. *Learning to Classify Text Using Support Vector Machines: Methods, Theory, and Algorithms*. The Kluwer International Series in Engineering and Computer Science. Boston, Kluwer Academic Publishers, 2002.

G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the 11th International Conference on Machine Learning*, pages 121–129. San Francisco, Morgan Kaufmann, 1994.

R.A. Johnson and D.W. Wichern. *Applied Multivariate Statistical Analysis*. Paramus, NJ, Prentice Hall, 1998.

I.T. Jolliffe. *Principal Component Analysis*. New York, Springer-Verlag, 1986.

D.T. Jones. Protein Secondary Structure Prediction Based on Position-specific Scoring Matrices. *J. Mol. Biol.*, 292:195–202, 1999.

C. Jutten and J. Herault. Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24(1):1–10, 1991.

W. Kabsch and C. Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12): 2577–2637, 1983.

M. Kanehisa, S. Goto, S. Kawashima, and A. Nakaya. The KEGG databases at genomenet. *Nucleic Acids Research*, 30:42–46, 2002.

R. Karchin, K. Karplus, and D. Haussler. Classifying G-protein coupled receptors with support vector machines. *Bioinformatics*, 18:147–159, 2002.

K. Karplus, C. Barrett, and R. Hughey. Hidden Markov models for detecting remote protein homologies. *Bioinformatics*, 14(10):846–56, 1998.

M. Karplus and G.A. Petsko. Molecular dynamics simulations in biology. *Nature*, 347:631–639, 1990.

H. Kashima and A. Inokuchi. Kernels for graph classification. In *IEEE ICDM Workshop on Active Mining*. Maebashi, Japan, 2002.

H. Kashima and T. Koyanagi. Kernels for semi-structured data. In C. Sammut and A.G. Hoffmann, editors, *Proceedings of the 19th International Conference on Machine Learning*, pages 291–298. San Francisco, Morgan Kaufmann, 2002.

H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In T. Faucett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning*, pages 321–328, Menlo Park, CA, AAAI Press, 2003.

T. Kawabata and K. Nishikawa. Protein tertiary structure comparison using the Markov transition model of evolution. *Proteins*, 41:108–122, 2000.

W.J. Kent and A.M. Zahler. The Intronerator: Exploring introns and alternative splicing in C. elegans. *Nucleic Acids Research*, 28(1):91–93, 2000.

M. K. Kerr, M. Martin, and G. A. Churchill. Analysis of variance for gene expression microarray data. *Journal of Computational Biology*, 7:819–837, 2000.

J. Khan, J. S. Wei, M. Ringner, L. H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. R. Antonescu, C. Peterson, and P. S. Meltzer. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature Medicine*, 7(6):673–679, 2001.

G.S. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33:82–95, 1971.

T. Kin, K. Tsuda, and K. Asai. Marginalized kernels for RNA sequence data analysis. In R.H. Lathtop, K. Nakai, S. Miyano, T. Takagi, and M. Kanehisa, editors, *Genome Informatics 2002*, pages 112–122. Tokyo, Universal Academic Press, 2002.

K. Kira and L. A. Rendell. The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 129–134. Cambridge, MA, MIT Press, 1992.

J. Kittler. Feature selection and extraction. In T. Y. Young and K.-S. Fu, editors, *Handbook of Pattern Recognition and Image Processing*, pages 59–83. New York, Academic Press, 1986.

S. Knudsen. *A Biologist's Guide to Analysis of DNA Microarray Data*. New York, Wiley, 2002.

D. E. Knuth. *The Art of Computer Programming: Fundamental Algorithms,* second edition, volume 1. Reading, Massachusetts, Addison-Wesley, 1998.

R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

D. Koller and M. Sahami. Toward optimal feature selection. In *Proceedings of the 13th International Conference on Machine Learning*, pages 284–292. San Francisco, Morgan Kaufmann, 1996.

R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In C. Sammut and A. G. Hoffmann, editors, *Machine Learning, Proceedings of the 19th International Conference (ICML 2002)*, pages 315–322. San Francisco, Morgan Kaufmann, 2002.

I. Kononenko. Estimating attributes: Analysis and extensions of Relief. In F. Bergadano and L. De Raedt, editors, *Machine Learning: ECML-94*. Berlin, Springer Verlag, 1994.

S. Kramer and L. De Raedt. Feature construction with version spaces for biochemical application. In *Proceedings of the 18th International Conference on Machine Learning*, pages 258–265. San Francisco, Morgan Kaufmann, 2001.

U. Kreßel. Pairwise classification and support vector machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods—Support Vector Learning*, pages 255–268, Cambridge, MA, MIT Press, 1999.

B. Krishnapuram, L. Carin, and A. Hartemink. Joint classifier and feature optimization for comprehensive cancer diagnosis using gene expression data. *Journal of Computational Biology*, 2004.

B. Krishnapuram, A. Hartemink, L. Carin, and M. A. T. Figueiredo. An EM algorithm for joint feature selection and classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2003. Submitted for publication.

A. Krogh, M. Brown, I. Mian, K. Sjolander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

A. Krogh, B. Larsson, G. von Heijne, and E. L. L. Sonnhammer. Predicting transmembrane protein topology with a hidden markov model: Application to complete genomes. *Journal of Molecular Biology*, 305(3):567–580, 2001.

M. Kuss and T. Graepel. The geometry of kernel canonical correlation analysis. Technical Report 108, Max-Planck-Institut für biologische Kybernetik, Tübingen, Germany, 2003.

J. Kyte and R. F. Doolittle. A simple method for displaying the hydropathic character of a protein. *Journal of Molecular Biology*, 157:105–132, 1982.

J. Lafferty and G. Lebanon. Information diffusion kernels. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. Cambridge, MA, MIT Press, 2003.

G. R. G. Lanckriet, T. De Bie, N. Cristianini, M. I. Jordan, and W. S. Noble. A framework for genomic data fusion and its application to membrane protein prediction. Technical Report 03-1273, University of California, Berkeley, Division of Computer Science, 2003.

G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. I. Jordan. Learning the kernel matrix with semi-definite programming. In C. Sammut and A. Hoffmann, editors, *Proceedings of the 19th International Conference on Machine Learning*, pages 323–330. San Francisco, Morgan Kaufmann, 2002.

G. R. G. Lanckriet, M. Deng, N. Cristianini, M. I. Jordan, and W. S. Noble. Kernel-based data fusion and its application to protein function prediction in yeast. In *Proceedings of the Pacific Symposium on Biocomputing (PSB)*, Big Island of Hawaii, Hawaii, January 6-10, 2004.

P. Langley. Selection of relevant features in machine learning. In *AAAI Fall Symposium on Relevance*, pages 140–144, 1994.

Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. San Mateo, CA: Morgan Kaufmann, 1990.

Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines. Technical Report 1040, Department of Statistics, University of Wisconsin, Madison, 2001.

E. Leopold and J. Kindermann. Text categorization with support vector machines: How to represent text in input space? *Machine Learning*, 46(3):423–444, March 2002.

C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In R.B. Altman, A.K. Dunker, L. Hunter, K. Lauerdale, and T.E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575. River Edge, NJ, World Scientific, 2002.

C. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. Submitted for publication, 2003a.

C. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for SVM protein classification. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 1417–1424. Cambridge, MA, MIT Press, 2003b.

C. Leslie and R. Kuang. Fast kernels for inexact string matching. In B. Schölkopf and M. Warmuth, editors, *Proceedings of the 16th Annual Conference on Learning Theory (COLT) and 7th Annual Workshop on Kernel Machines*, Lecture Notes in Artificial Intelligence Vol. 2777, pages 114–128, Heidelberg, Springer Verlag, 2003.

S. Leurgans, R. Moyeed, and B. Silverman. Canonical correlation analysis when the data are curves. *Journal of the Royal Statistical Society*, B55:725–740, 1993.

J.M. Levin and J. Garnier. Improvements in a secondary structure prediction method based on a search for local sequence homologies and its use as a model

building tool. *Biochimica et Biophysica Acta*, 955:283–295, 1988.

J.M. Levin, B. Robson, and J. Garnier. An algorithm for secondary structure determination in proteins based on sequence similarity. *FEBS Letters*, 205(2): 303–308, 1986.

B. Lewin. *Genes VII*. New York, Oxford University Press, 2000.

Y. Li, C. Campbell, and M. Tipping. Bayesian automatic relevance determination algorithms for classifying gene expression data. *Bioinformatics*, 18:1332–1339, 2002.

L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In G. Myers, S. Hannenhalli, D. Sankoff, S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the Sixth Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 225–232, New York, ACM Press, 2002.

L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for detecting remote protein evolutionary and structural relationships. *Journal of Computational Biology*, 2003. In preparation.

D. Lipman and W. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.

D. Lipman and W. Pearson. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States of America*, 85:2444–2448, 1988.

H. Liu and H. Motoda. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Boston, Kluwer Academic Publishers, 1998.

H. Liu and R. Setiono. A probabilistic approach to feature selection—a filter solution. In *Proceedings of the 13th International Conference on Machine Learning*, pages 319–327. San Francisco, Morgan Kaufmann, 1996.

D. Lockhart, H. Dong, M. Byrne, M. Follettie, M. Gallo, M. Chee, M. Mittmann, C. Wang, M. Kobayashi, H. Horton, and E. Brown. Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature Biotechnology*, 14 (12):1675–1680, 1996.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

B. Logan, P. Moreno, B. Suzek, Z. Weng, and S. Kasif. A study of remote homology detection. Technical report, COMPAQ Cambridge Research Laboratory, Cambridge, MA, June 2001. Available from http://www.hpl.hp.com/techreports/Compaq-DEC/CRL-2001-5.html.

Q. Lu, L. L. Wallrath, and S. C. R. Elgin. Nucleosome positioning and gene regulation. *Journal of Cellular Biochemistry*, 55:83–92, 1994.

R. B. Lyngsø, C. N. S. Pedersen, and H. Nielsen. Metrics and similarity measures for hidden Markov models. In *Proceedings of the Seventh International Conference*

on Intelligent Systems for Molecular Biology, pages 178–186, Menlo Park, CA, AAAI Press, 1999.

Y. Lysov, V. Florent'ev, A. Khorlin, K. Khrapko, V. Shik, and A. Mirzabekov. DNA sequencing by hybridization with oligonucleotides. *Doklady Academi Nauk*, 303:1508–1511, 1988.

D. J. C. MacKay. Introduction to Gaussian processes. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, pages 133–165. Berlin, Springer-Verlag, 1998.

O. Mangasarian. Arbitrary-norm separating plane. *Operations Research Letters*, 24(1-2):15–23, 1999.

O. L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations Research*, 13:444–452, 1965.

C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. Cambridge, MA, MIT Press, 1999.

E. M. Marcotte, M. Pellegrini, M. J. Thompson, T. O. Yeates, and D. Eisenberg. A combined algorithm for genome-wide prediction of protein function. *Nature*, 402(6757):83–86, 1999.

T. Marill and D. M. Green. On the effectiveness of receptors in recognition systems. *IEEE Transactions on Information Theory*, 9:11–17, 1963.

D. Mattera, F. Palmieri, and S. Haykin. Simple and robust methods for support vector expansions. *IEEE Transactions on Neural Networks*, 10(5):1038–1047, 1999.

B.W. Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta*, 405:442–451, 1975.

E. Mayoraz and E. Alpaydin. Support vector machines for multi-class classification. Technical Report 98-06, The Dalle Molle Institute for Perceptual Artificial Intelligence (IDIAP), Matigny, Switzerland, 1998.

E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, April 1976.

H. W. Mewes, D. Frishman, C. Gruber, B. Geier, D. Haase, A. Kaps, K. Lemcke, G. Mannhaupt, F. Pfeiffer, C Schüller, S. Stocker, and B. Weil. MIPS: a database for genomes and protein sequences. *Nucleic Acids Research*, 28(1):37–40, 2000.

C. A. Micchelli. Algebraic aspects of interpolation. *Proceedings of Symposia in Applied Mathematics*, 36:81–102, 1986.

F. Model, P. Adorján, A. Olek, and C. Piepenbrock. Feature selection for DNA methylation based cancer classification. *Bioinformatics*, 17(Supplement 1):S157–S164, 2001.

E. J. Moler, M. L. Chow, and I. S. Mian. Analysis of molecular profile data using generative and discriminative methods. *Physiological Genomics*, 4:109–126, 2000.

R. E. Moore, M. K. Young, and T. D. Lee. Qscore: An algorithm for evaluating SEQUEST database search results. *Journal of the American Society for Mass Spectrometry*, 13(4):378–386, 2002.

R. Mrowka, W. Lieberneister, and D. Holste. Does mapping reveal correlation between gene expression and protein-protein interaction? *Nature Genetics*, 33: 15–16, 2003.

S. Mukherjee, P. Tamayo, D. Slonim, A. Verri, T. Golub, J. Mesirov, and T. Poggio. Support vector machine classification of microarray data. Technical Report AI Memo 1677, Massachusetts Institute of Technology, Cambridge, MA, 1999.

S. Mukherjee and V. Vapnik. Multivariate density estimation: An SVM approach. Technical Report AI Memo 1653, Massachusetts Institute of Technology, Cambridge, MA, 1999.

K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12 (2):181–201, 2001.

A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.

E. Myasnikova, A. Samsonova, M. Samsonova, and J. Reinitz. Support vector regression applied to the determination of the developmental age of a *Drosophila* embryo from its segmentation gene expression patterns. *Bioinformatics*, 18:S87–S95, 2002.

A. Nakaya, S. Goto, and M. Kanehisa. Extraction of correlated gene clusters by multiple graph comparison. In H. Matsuda, S. Miyano, T. Takagi, and L. Wong, editors, *Genome Informatics 2001*, pages 44–53. Tokyo, Universal Academy Press, 2001.

G. Navarro and M. Raffinot. Fast regular expression search. In J.S. Vitter and C.D. Zaroliagis, editors, *Algorithm Engineering, 3rd International Workshop, WAE '99*, volume 1668 of *Lecture Notes in Computer Science*, pages 198–212. Heidelberg, Springer Verlag, 1999.

S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

Y. Nesterov and A. Nemirovsky. *Interior Point Polynomial Methods in Convex Programming: Theory and Applications*. Philadelphia, PA, Society for Industorial and Applied Mathematics (SIAM), 1994.

A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 849–856, Cambridge, MA, MIT Press, 2002.

C. O'Donovan, M.J. Martin, A. Gattiker, E. Gasteiger, A. Bairoch, and R. Apweiler. High-quality protein knowledge resource: Swiss-Prot and TrEMBL. *Briefings in Bioinformatics*, 3:275–284, 2002.

H. Ogata, W. Fujibuchi, S. Goto, and M. Kanehisa. A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic Acids Research*, 28:4021–4028, 2000.

A. Ohara, N. Suda, and S. Amari. Dualistic differential geometry of positive definite matrices and its applications to related problems. *Linear Algebra and Its Applications*, pages 31–53, 1996.

E. Oja. A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273, 1982.

J. Park, K. Karplus, C. Barrett, R. Hughey, D. Haussler, T. Hubbard, and C. Chothia. Sequence comparisons using multiple sequences detect three times as many remote homologues as pairwise methods. *Journal of Molecular Biology*, 284(4):1201–1210, 1998.

E. Parzen. Extraction and detection problems and reproducing kernel Hilbert spaces. *Journal of the Society for Industrial and Applied Mathematics. Series A, On control*, 1:35–62, 1962.

P. Pavlidis, T. S. Furey, M. Liberto, D. Haussler, and W. N. Grundy. Promoter region-based classification of genes. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 151–163. River Edge, NJ, World Scientific, 2001a.

P. Pavlidis, J. Weston, J. Cai, and W. N. Grundy. Gene functional classification from heterogeneous data. In *Proceedings of the Fifth Annual International Conference on Computational Biology (RECOMB)*, pages 249–255. New York, ACM Press, 2001b.

P. Pavlidis, J. Weston, J. Cai, and W. S. Noble. Learning gene functional classifications from multiple data types. *Journal of Computational Biology*, 9(2):401–411, 2002.

W. R. Pearson. Rapid and sensitive sequence comparisions with FASTP and FASTA. *Methods in Enzymology*, 183:63–98, 1990.

M. Pellegrini, E. M. Marcotte, M. J. Thompson, D. Eisenberg, and T. O. Yeates. Assigning protein functions by comparative genome analysis: protein phylogenetic profiles. *Proceedings of the National Academy of Sciences of the United States of America*, 96(8):4285–4288, 1999.

S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3: 1333–1356, 2003. Special issue on variable and feature selection.

M. Pertea, X. Lin, and S.L. Salzberg. Genesplicer: a new computational method for splice site prediction. *Nucleic Acids Research*, 29(5):1185–1190, 2001.

T.N. Petersen, C. Lundegaard, M. Nielsen, H. Bohr, J. Bohr, S. Brunak, G.P. Gippert, and O. Lund. Prediction of protein secondary structure at 80% accuracy.

Proteins, 41(1):17–20, 2000.

J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods—Support Vector Learning*, pages 185–208, Cambridge, MA, MIT Press, 1999.

J. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*. Cambridge, MA, MIT Press, 2001.

T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9), September 1990.

G. Pollastri, D. Przybylski, B. Rost, and P. Baldi. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins*, 47(2):228–235, 2002.

S. L. Pomeroy, P. Tamayo, M. Gaasenbeek, L. M. Sturla, M. Angelo, M. E. McLaughlin, J. Y. H. Kim, L. C. Goumnerova, P. M. Black, C. Lau, J. C. Allen, D. Zagzag, J. M. Olson, T. Curran, C. Wetmore, J. A. Biegel, T. Poggio, S. Mukherjee, R. Rifkin, A. Califano, G. Stolovitzky, D. N. Louis, J. P. Mesirov, E. S. Lander, and T. R. Golub. Prediction of central nervous system embryonal tumour outcome based on gene expression. *Nature*, 415(6870):436–442, 2002.

N.J. Proudfoot, A. Furger, and M.J. Dye. Integrating mrna processing with transcription. *Cell*, 108:501–512, February 2002.

N. Qian and T.J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, 202:865–884, 1988.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, Morgan Kaufmann, 1993.

A. Rakotomamonjy. Variable selection using SVM based criteria. *Journal of Machine Learning Research*, 3:1357–1370, 2003.

S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C. H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. P. Mesirov, T. Poggio, W. Gerald, M. Loda, E. S. Lander, and T. R. Golub. Multiclass cancer diagnosis using tumor gene expression signatures. *Proceedings of the National Academy of Sciences of the United States of America*, 98(26):15149–15154, 2001.

S. Rampone. Recognition of splice junctions on DNA sequences by BRAIN learning algorithm. *Bioinformatics*, 14(8):676–684, 1998.

G. Rätsch and S. Sonnenburg. A new splice form predictor for *C. elegans* . Technical report, Max-Planck Institute for Biological Cybernetics, 2004. In preparation.

M.G. Reese, F. H. Eeckman, D. Kulp, and D. Haussler. Improved splice site detection in Genie. *Journal of Computational Biology*, 4:311–323, 1997.

L. A. Rendell and K. Kira. A practical approach to feature selection. In D.H. Sleeman and P. Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning*, pages 249–256. San Francisco, Morgan Kaufmann, 1992.

S. Riis and A. Krogh. Improving prediction of protein secondary structure using structured neural networks and multiple sequence alignments. *Journal of Computational Biology*, 3:163–183, 1996.

M. Robnik-Sikonja and I. Kononenko. An adaptation of Relief for attribute estimation in regression. In D.H. Fisher, editor, *Proceedings of the 14th International Conference on Machine Learning*, pages 296–304. San Francisco, Morgan Kaufmann, 1997.

R. Rosipal and L. J. Trejo. Kernel partial least squares regression in reproducing kernel Hilbert space. *Journal of Machine Learning Research*, 2:97–123, 2001.

S. Rosset, J. Zhu, and T. Hastie. Margin maximizing loss functions. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS) 15*. Cambridge, MA, MIT Press, 2003.

B. Rost. Review: Protein secondary structure prediction continues to rise. *Journal of Structural Biology*, 134(2):204–218, 2001.

B. Rost and S. O'Donoghue. Sisyphus and prediction of protein structure. *Computer Applications in the Biosciences*, 13:345–356, 1997.

B. Rost and C. Sander. Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*, 232:584–599, 1993.

B. Rost, C. Sander, and R. Schneider. Redefining the goals of protein secondary structure prediction. *Journal of Molecular Biology*, 235:13–26, 1994.

V. Roth. The generalized LASSO. *IEEE Transactions on Neural Networks*, 2003. In preparation.

V. Roth, J. Laub, J.M. Buhmann, and K.-R. Müller. Going metric: Denoising pairwise data. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 817–824. Cambridge, MA, MIT Press, 2003.

W.J. Rugh. *Linear System Theory*. Paramus, NJ, Prentice Hall, 1995.

P. Ruján and M. Marchand. Computing the Bayes kernel classifier. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 329–347, Cambridge, MA, MIT Press, 2000.

M. Rychetsky, J. Shawe-Taylor, and M. Glesner. Direct Bayes point machines. In P. Langley, editor, *Proceedings of the 17th International Conference on Machine Learning*, San Francisco, Morgan Kaufmann, 2000.

Y. Sakakibara, M. Brown, R. Hughey, I.S Mian, K. Sjölander, R.C. Underwood, and D. Haussler. Stochastic context free grammars for tRNA modeling. *Nucleic Acids Research*, 22:5112–5120, 1994.

H. Salgado, G. Moreno-Hagelsieb, T.F. Smith, and J. Collado-Vides. Operons in *Escherichia coli:* Genomic analysis and prediction. *Proceedings of the National Academy of Sciences of the United States of America*, 97:6652–6657, 2000.

G. Salton. *Automatic Text Processing*. Addison-Wesley, Massachusetts, 1989.

S. Salzberg, A.L. Delcher, K.H. Fasman, and J. Henderson. A decision tree system for finding genes in DNA. *Journal of Computational Biology*, 5(4):667–680, 1998.

C. Sander and R. Schneider. Database of homology derived protein structures and the structural meaning of sequence alignment. *Proteins*, 9:56–68, 1991.

R.A. Sayle and E.J. Milner-White. Rasmol: Biomolecular graphics for all. *Trends in Biochemical Sciences*, 20(9):374–376, 1995.

M. Schena, D. Shalon, R.W. Davis, and P.O. Brown. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270: 467–70, 1995.

U. Scherf, D. T. Ross, M. Waltham, L. H. Smith, J. K. Lee, L. Tanabe, K. W. Kohn, W. C. Reinhold, T. G. Myers, D. T. Andrews, D. A. Scudiero, M. B. Eisen, E. A. Sausville, Y. Pommier, D. Botstein, P. O. Brown, and J. N. Weinstein. A gene expression database for the molecular pharmacology of cancer. *Nature Genetics*, 24(3):236–244, 2000.

M. Schmidt and H. Gish. Speaker identification via support vector classifiers. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP 96)*, pages 105–108, Atlanta, May 1996.

B. Schölkopf. *Support vector learning*. Munich, Oldenbourg Verlag, 1997.

B. Schölkopf, C.J.C. Burges, and V.N. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery & Data Mining*. Menlo Park, CA, AAAI Press, 1995.

B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, 2001.

B. Schölkopf, A. Smola, R.C. Williamson, and P.L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

B. Schölkopf and A. J. Smola. *Learning with Kernels*. Cambridge, MA, MIT Press, 2002.

B. Schölkopf, A. J. Smola, and K.-R. Müller. Kernel principal component analysis. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 327–352. MIT Press, 1999.

B. Schölkopf, A.J. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

B. Schölkopf, J. Weston, E. Eskin, C. Leslie, and W. S. Noble. A kernel approach for learning from almost orthogonal patterns. In T. Elomaa, H. Mannila, and

H. Toivonen, editors, *Proceedings of ECML 2002, 13th European Conference on Machine Learning, Helsinki, Finland, August 19-23, 2002*, volume 2430 of *Lecture Notes in Computer Science*, pages 511–528. Heidelberg, Springer Verlag, 2002.

C. M. Schubert. Microarray to be used as routine clinical screen. *Nature Medicine*, 9(1):9, 2003.

J. Schuchhardt, D. Beule, A. Malik, E. Wolski, H. Eickhoff, H. Lehrach, and H. Herzel. Normalization strategies for cDNA microarrays. *Nucleic Acids Research*, 28(10):E47, 2000.

I. Schur. Bemerkungen zur Theorie der beschränkten Bilinearformen mit unendlich vielen Veränderlichen. *Journal für die Reine und Angewandte Mathematik*, 140: 1–29, 1911.

R. M. Schwartz and M. O. Dayhoff. *Atlas of Protein Sequence and Structure*, pages 353–358. Silver Spring, MD, National Biomedical Research Foundation, 1978.

M. Seeger. Bayesian model selection for support vector machines, Gaussian processes, and other kernel classifiers. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems (NIPS) 12*. Cambridge, MA, MIT Press, 2000.

M. Seeger. Covariance kernels from Bayesian generative models. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, MIT Press, 2002.

N. H. Segal, P. Pavlidis, C. R. Antonescu, R. G. Maki, W. S. Noble, J. M. Woodruff, J. J. Lewis, M. F. Brennan, A. N. Houghton, and C. Cordon-Cardo. Classification and subtype prediction of soft tissue sarcoma by functional genomics and support vector machine analysis. *American Journal of Pathology*, 169:691–700, 2003a.

N. H. Segal, P. Pavlidis, W. S. Noble, C. R. Antonescu, A. Viale, U. V. Wesley, K. Busam, H. Gallardo, D. DeSantis, M. F. Brennan, C. Cordon-Cardo, J. D. Wolchok, and A. N. Houghton. Classification of clear cell sarcoma as melanoma of soft parts by genomic profiling. *Journal of Clinical Oncology*, 21:1775–1781, 2003b.

J. Shawe-Taylor, P. L. Bartlett, R. Williamson, and M. Anthony. A framework for structural risk minimization. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, pages 68–76, New York, Association for Computing Machinery, 1996.

J. Shawe-Taylor, P. L. Bartlett, R. C. Williamson, and M. Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940, 1998.

H. Shimodaira, K.-I. Noma, M. Nakai, and S. Sagayama. Dynamic time-alignment kernel in support vector machine. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 921–928, Cambridge, MA, MIT Press, 2002.

M. A. Shipp, K. N. Ross, P. Tamayo, A. P. Weng, R. C. T. Aguiar J. L. Kutok, M. Gaasenbeek, M. Angelo, M. Reich, T. S. Ray G. S. Pinkus, M. A. Koval, K. W. Last, A. Norton, J. Mesirov T. A. Lister, D. S. Neuberg, E. S. Lander, J. C. Aster, and T. R. Golub. Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature Medicine*, 8(1):68–74, 2002.

W. Siedlecki and J. Sklansky. On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2(2):197–220, 1988.

C. J. Sigrist, L. Cerutti, N. Hulo, A. Gattiker, L. Falquet, M. Pagni, A. Bairoch, and P. Bucher. PROSITE: A documented database using patterns and profiles as motif descriptors. *Brief Bioinformatics*, 3:265–274, 2002.

D. B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the 11th International Conference on Machine Learning*, pages 293–301. San Francisco, Morgan Kaufmann, 1994.

T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

A. J. Smola, T. Frieß, and B. Schölkopf. Semiparametric support vector and linear programming machines. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 585–591, Cambridge, MA, MIT Press, 1999.

A. J. Smola and R. Kondor. Kernels and regularization on graphs. In B. Schölkopf and M. Warmuth, editors, *Proceedings of the 16th Annual Conference on Learning Theory (COLT) and 7th Annual Workshop on Kernel Machines*, volume 2777 of *Lecture Notes in Artificial Intelligence*, pages 144–158. Heidelberg, Springer Verlag, 2003.

A. J. Smola, B. Schölkopf, and K.-R. Müller. The connection between regularization operators and support vector kernels. *Neural Networks*, 11:637–649, 1998.

A.J. Smola and S.V.N. Vishwanathan. Hilbert space embeddings in dynamical systems. In *Proceedings of the 13th IFAC Symposium on System Identification*, pages 760–767. IFAC, August 2003.

S. Sonnenburg. New Methods for Splice Site Recognition. Master's thesis, Humboldt University, Berlin, Germany, 2002. supervised by K.-R. Müller, H.-D. Burkhard, and G. Rätsch.

S. Sonnenburg, G. Rätsch, A. Jagota, and K.-R. Müller. New methods for splice site recognition. In J.R. Dorronsoro, editor, *Artificial Neural Networks—ICANN 2002*, pages 329–336. Heidelberg, Springer Verlag, 2002.

E. Sonnhammer, S. Eddy, and R. Durbin. Pfam: A comprehensive database of protein domain families based on seed alignments. *Proteins*, 28(3):405–420, 1997.

E. Southern. United Kingdom patent application GB8810400, 1988.

P. T. Spellman, G. Sherlock, M. Q. Zhang, V. R. Iyer, K. Anders, M. B. Eisen, P. O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of

cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. *Mol. Biol. Cell*, 9:3273–3297, 1998.

A. Srinivasan, S. Muggleton, R. D. King, and M. Sternberg. Theories for mutagenicity: a study of first-order and feature based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.

C. J. Stone. Optimal rates of convergence for nonparametric estimators. *Annals of Statistics*, 8(6):1348–1360, 1980.

J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999. Special issue on interior point methods (CD supplement with software).

Y. Su, T. M. Mural, V. Pavlovic, M. Schaffer, and S. Kasif. RankGene: Identification of diagnostic genes based on expression data. *Bioinformatics*, 2003. In preparation.

J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. River Edge, NJ, World Scientific, 2002.

E. Takimoto and M. K. Warmuth. Predicting nearly as well as the best pruning of a planar decision graph. In *Proceedings of the Tenth International Conference on Algorithmic Learning Theory—ALT '99*, volume 1720 of *Lecture Notes in Artificial Intelligence*, pages 335–346. Heidelberg, Springer-Verlag, 1999.

E. Takimoto and M. K. Warmuth. Path kernels and multiplicative updates. In J. Kivinen and R.H. Sloan, editors, *Proceedings of the 15th Annual Conference on Computational Learning Theory*, pages 74–89. Heidelberg, Springer Verlag, 2002.

A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18:S136–S144, 2002.

R. Tibshirani. Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society (B)*, 58:267–288, 1996.

R. J. Tibshirani and B. Efron. Pre-validation and inference in microarrays. *Statistical Applications in Genetics and Molecular Biology*, 1(1):1–18, 2002.

A. N. Tikhonov and V. Y. Arsenin. *Solution of Ill–Posed Problems*. Washington, DC, Winston, 1977.

M.E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

N. Tishby. Relevant coding and information bottlenecks: A principled approach to multivariate feature selection. In *NIPS'2001 Workshop on Variable and Feature Selection*, Whistler, Canada, 2001.

N. Tishby, F.C. Pereira, and W. Bialek. The information bottleneck method. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control and Computing*, pages 368–377, 1999.

M. Tomita. Whole-cell simulation: A grand challenge of the 21st century. *Trends in Biochemical Sciences*, 19(6):205–210, 2001.

G. Tseng, M. Oh, L. Rohlin, J. Liao, and W. Wong. Issues in cDNA microarray analysis: quality filtering, channel normalization, models of variations and assessment of gene effects. *Nucleic Acids Research*, 29:2549–2557, 2001.

K. Tsuda. Support vector classification with asymmetric kernel function. In M. Verleysen, editor, *Proceedings of the Seventh European Symposium on Artificial Neural Networks*, pages 183–188, 1999.

K. Tsuda, S. Akaho, and K. Asai. The em algorithm for kernel matrix completion with auxiliary data. *Journal of Machine Learning Research*, 4:67–81, May 2003.

K. Tsuda, S. Akaho, M. Kawanabe, and K.-R. Müller. Asymptotic properties of the Fisher kernel. *Neural Computation*, 16:115–137, 2004.

K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K.R. Müller. A new discriminative kernel from probabilistic models. *Neural Computation*, 14:2397–2414, 2002a.

K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 18:S268–S275, 2002b.

P. D. Turney. Exploiting context when learning to classify. In P. Brazdil, editor, *Proceedings of the European Conference on Machine Learning*, pages 402–407. Heidelberg, Springer Verlag, 1993a. Available from ftp://ai.iit.nrc.ca/pub/ksl-papers/NRC-35058.ps.Z.

P. D. Turney. Robust classification with context-sensitive features. In P. Chung, G. L. Lovegrove, and M. Ali, editors, *Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 268–276. Williston, VA, Gordon and Breach Publishing, 1993b. Available from ftp://ai.iit.nrc.ca/pub/ksl-papers/NRC-35074.ps.Z.

V. G. Tusher, R. Tibshirani, and G. Chu. Significance analysis of microarrays applied to the ionizing radiation response. *Proceedings of the National Academy of Sciences of the United States of America*, 98:5116–5121, 2001.

E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

H. Vafaie and K. De Jong. Genetic algorithms as a tool for feature selection in machine learing. In *Proceedings of the Fourth Conference on Tools for Artificial Intelligence*, pages 200–203. Los Alamitos, CA, IEEE Computer Society, 1992.

H. Vafaie and K. De Jong. Robust feature selection algorithms. In *Proceedings of the Fifth Conference on Tools for Artificial Intelligence*, pages 356–363. Los Alamitos, CA, IEEE Computer Society, 1993.

L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1): 49–95, 1996.

L. J. van't Veer, H. Dai, M. J. van de Vijver, Y. D. He, A. A. Hart, M. Mao, H. L. Peterse, K. van der Kooy, M. J. Marton, A. T. Witteveen, G. J. Schreiber, R. M. Kerkhoven, C. Roberts, P. S. Linsley, R. Bernards, and S. H. Friend. Gene

expression profiling predicts clinical outcome of breast cancer. *Nature*, 415:530–536, 2002.

V. N. Vapnik. *The Nature of Statistical Learning Theory*. New York, Springer Verlag, 1995. ISBN 0-387-94559-8.

V. N. Vapnik. *Statistical Learning Theory*. Adaptive and Learning Systems for Signal Processing, Communications, and Control. New York, Wiley, 1998.

V.N. Vapnik and A.Y. Chervonenkis. Uniform convergence of frequencies of occurence of events to their probabilities. *Doklady Akademii nauk SSSR*, 181: 915–918, 1968.

V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16(2):264–280, 1971.

V.N. Vapnik and A.Y. Chervonenkis. *Theory of Pattern Recognition [In Russian]*. Moscow, Nauka, 1974.

V. E. Velculescu, L. Zhang, B. Vogelstein, and K. W. Kinzler. Serial analysis of gene expression. *Science*, 270:484–487, 1995.

J.-P. Vert. Support vector machine prediction of signal peptide cleavage site using a new class of kernels for strings. In R. B. Altman, A. K. Dunker, L. Hunter, K. Lauerdale, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 649–660. World Scientific, 2002a.

J.-P. Vert. A tree kernel to analyze phylogenetic profiles. *Bioinformatics*, 18:S276–S284, 2002b.

J.-P. Vert and M. Kanehisa. Graph-driven features extraction from microarray data. Technical report, Arxiv, June 2002.

J.-P. Vert and M. Kanehisa. Extracting active pathways from gene expression data. *Bioinformatics*, 19:238ii–234ii, 2003a.

J.-P. Vert and M. Kanehisa. Graph-driven features extraction from microarray data using diffusion kernels and kernel CCA. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 1425–1432. Cambridge, MA, MIT Press, 2003b.

R. Vert. Designing a M-SVM Kernel for Protein Secondary Structure Prediction. Master's thesis, DEA informatique de Lorraine, France, 2002c.

S. V. N. Vishwanathan. *Kernel Methods: Fast Algorithms and Real Life Applications*. PhD thesis, Indian Institute of Science, Bangalore, India, November 2002.

S. V. N. Vishwanathan and A. J. Smola. Fast kernels for string and tree matching. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. Cambridge, MA, MIT Press, 2003.

G. von Heijne. A new method for predicting signal sequence cleavage sites. *Nucleic Acids Research*, 14:4683–4690, 1986.

C. von Mering, R. Krause, B. Snel, M. Cornell, S. G. Olivier, S. Fields, and P. Bork. Comparative assessment of large-scale data sets of protein-protein interactions. *Nature*, 417:399–403, 2002.

G. Wahba. Soft and hard classification by reproducing kernel Hilbert space methods. *Proceedings of the National Academy of Sciences of the United States of America*, 99(26):16524 – 16530, 2002.

J. C. Wallace and S. Henikoff. PATMAT: a searching and extraction program for sequence, pattern and block queries and databases. *Computer Applications in the Biosciences*, 8:249–254, 1992.

D.M. Walters, R. Russ, H.J. Knackmuss, and P.E. Rouviere. High-density sampling of a bacterial operon using mRNA differential display. *Gene*, 273(2):305–315, 2001.

D. G. Wang, J.-B. Fan, and C.-J. Siao. Large-scale identification, mapping, and genotyping of single-nucleotide polymorphisms in the human genome. *Science*, 280:1077–1082, 1998.

M. S. Waterman, J. Joyce, and M. Eggert. Computer alignment of sequences. In *Phylogenetic Analysis of DNA Sequences*, pages 59–72. Oxford, UK, Oxford University Press, 1991.

C. Watkins. Dynamic alignment kernels. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 39–50, Cambridge, MA, 2000. MIT Press.

P. Weiner. Linear pattern matching algorithms. In *Proceedings of the IEEE* 14^{th} *Annual Symposium on Switching and Automata Theory*, pages 1–11, The University of Iowa, 1973. IEEE.

H. L. Weinert, editor. *Reproducing Kernel Hilbert Spaces—Applications in Statistical Signal Processing*. Stroudsburg, PA, Hutchinson Ross, 1982.

Y. Weiss. Segmentation using eigenvectors: a unifying view. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 975–982. Los Alamitos, CA, IEEE Computer Society, 1999.

J. Weston. Leave-one-out support vector machines. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 727–733. San Francisco, Morgan Kaufmann, 1999.

J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. Vapnik. Kernel dependency estimation. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 873–880. Cambridge, MA, MIT Press, 2003a.

J. Weston, A. Elisseeff, B. Schölkopf, and M. Tipping. Use of the zero-norm with linear models and kernel methods. *Journal of Machine Learning Research*, 3: 1439–1461, 2003b. Special issue on variable and feature selection.

J. Weston, A. Gammerman, M. Stitson, V.N. Vapnik, V. Vovk, and C. Watkins. Support vector density estimation. In B. Schölkopf, C.J.C. Burges, and A.J.

Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 293–305. MIT Press, Cambridge, MA, 1999.

J. Weston and R. Herbrich. Adaptive margin support vector machines. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 281–296, Cambridge, MA, 2000. MIT Press.

J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for SVMs. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. Cambridge, MA, MIT Press, 2000.

J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, Egham, TW20 0EX, UK, 1998.

J. Weston and C. Watkins. Multi-class support vector machines. In M. Verleysen, editor, *Proceedings of the Seventh European Symposium on Artificial Neural Networks*. Brussels, D Facto, 1999.

C. K. I. Williams. Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In M.I. Jordan, editor, *Learning and Inference in Graphical Models*. Boston, Kluwer Academic Publishers, 1998.

C. K. I. Williams and D. Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(20), 1998.

R. C. Williamson, A. J. Smola, and B. Schölkopf. Generalization performance of regularization networks and support vector machines via entropy numbers of compact operators. *IEEE Transactions on Information Theory*, 47(6):2516–2532, 2001.

M. Xiong, W. Li, J. Zhao, L. Jin, and E. Boerwinkle. Feature (gene) selection in gene expression-based tumor classification. *Molecular Genetics and Metabolism*, 73(3):239–247, 2001.

L. Xu, P. Zan, and T. Chang. Best first strategy for feature selection. In *Ninth International Conference on Pattern Recognition*, pages 706–708. Los Alamitos, CA, IEEE Computer Society, 1989.

T. Yada, M. Nakao, Y. Totoki, and K. Nakai. Modeling and predicting transcriptional units of *Escherichia coli* genes using hidden Markov models. *Bioinformatics*, 15:987–993, 2001.

Y. Yamanishi, J.-P. Vert, A. Nakaya, and M. Kanehisa. Extraction of correlated gene clusters from multiple genomic data by generalized kernel canonical correlation analysis. *Bioinformatics*, 19:i323–i330, 2003.

Y. H. Yang, S. Dudoit, P. Luu, D. M. Lin, V. Peng, J. Ngai, and T. P. Speed. Normalization for cDNA microarray data: A robust composite method addressing single and multiple slide systematic variation. *Nucleic Acids Research*, 30(4):e15, 2002.

C. Yeang, S. Ramaswamy, P. Tamayo, S. Mukherjee, R. R. Rifkin, M. Angelo, M. Reich, E. Lander, J. Mesirov, and T. Golub. Molecular classification of multiple tumor types. *Bioinformatics*, 17(Supplement 1):S316–S322, 2001.

N. Zavaljevski, F.J. Stevens, and J. Reifman. Support vector machines with selective kernel scaling for protein classification and identification of key amino acid positions. *Bioinformatics*, 18(5):689–696, 2002.

A. Zemla, Č. Venclovas, K. Fidelis, and B. Rost. A modified definition of Sov, a segment-based measure for proteinsecondary structure prediction assessment. *Proteins*, 34(2):220–223, 1999.

X. Zhang, J.P. Mesirov, and D.L. Waltz. Hybrid system for protein secondary structure prediction. *Journal of Molecular Biology*, 225:1049–1063, 1992.

Y. Zheng, J.D. Szustakowski, L. Fortnow, R.J. Roberts, and S. Kasif. Computational identification of operons in microbial genomes. *Genome Research*, 12(8): 1221–1230, 2002.

J. Zhu and T. Hastie. Classification of gene microarrays by penalized logistic regression. *Biostatistics*, 2003. In preparation.

A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K.-R. Müller. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 16(9):799–807, September 2000.

Contributors

Tatsuya Akutsu
Bioinformatics Center
Institute for Chemical Research
Kyoto University
Uji 611-0011, Japan
takutsu@kuicr.kyoto-u.ac.jp

Lawrence Carin
Department of Electrical Engineering, Duke University
Box 90291, Durham, NC 27708, USA
lcarin@ee.duke.edu

Nello Cristianini
Department of Statistics
University of California Davis
Davis, CA 95616, USA
nello@support-vector.net

Eleazar Eskin
Department of Computer Science and Engineering
University of California, San Diego
eeskin@cs.ucsd.edu

Yann Guermeur
LORIA - CNRS
Campus Scientifique, BP 239
54506 Vandœuvre-lès-Nancy cedex, France
Yann.Guermeur@loria.fr

Alexander Hartemink
Department of Computer Science, Duke University
Box 90129, Durham, NC 27708, USA
amink@cs.duke.edu

Sepp Hochreiter
Technische Universität Berlin
Fakultät für Elektrotechnik und Informatik
Franklinstraße 28/29
10587 Berlin, Germany
hochreit@cs.tu-berlin.de

Akihiro Inokuchi
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502, Japan
inokuchi@jp.ibm.com

Michael I. Jordan
Computer Science Division and Department of Statistics
University of California Berkeley
Berkeley, CA 94720, USA
jordan@cs.berkeley.edu

Minoru Kanehisa
Bioinformatics Center,
Institute for Chemical Research,
Kyoto University, Gokasho, Uji,
Kyoto 611-0011, Japan
kanehisa@kuicr.kyoto-u.ac.jp

Hisashi Kashima
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502, Japan
hkashima@jp.ibm.com

Tsuyoshi Kato
AIST Computational Biology Research Center
2-43, Aomi, Koto-ku, 135-0064 Tokyo, Japan
kato-tsuyoshi@aist.go.jp

Taishin Kin
AIST Computational Biology Research Center
2-43, Aomi, Koto-ku, 135-0064 Tokyo, Japan
taishin@cbrc.jp

Risi Kondor
Columbia University, Department of Computer Science
Mail Code 0401, 1214 Amsterdam Avenue,
New York, NY 10027, USA
risi@cs.columbia.edu

Balaji Krishnapuram
Department of Electrical Engineering, Duke University
Box 90291, Durham, NC 27708, USA
balaji@ee.duke.edu

Rui Kuang
Columbia University
1214 Amsterdam Ave, New York NY 10027, USA
rkuang@cs.columbia.edu

Gert R. G. Lanckriet
Department of Electrical Engineering and Computer Science
University of California Berkeley
Berkeley, CA 94720, USA
gert@cs.berkeley.edu

Christina Leslie
Columbia University
1214 Amsterdam Ave, New York NY 10027, USA
cleslie@cs.columbia.edu

Alain Lifchitz
LIP6 - CNRS
8, rue du Capitaine Scott
75015 Paris, France
Alain.Lifchitz@lip6.fr

William Stafford Noble
Department of Genome Sciences
University of Washington
Seattle, WA, 98195, USA
noble@gs.washington.edu

Klaus Obermayer
Technische Universität Berlin
Fakultät für Elektrotechnik und Informatik
Franklinstraße 28/29
10587 Berlin, Germany
oby@cs.tu-berlin.de

Gunnar Rätsch
Max Planck Institute for Biological Cybernetics
Spemannstr. 38, 72076, Tübingen, Germany
Fraunhofer FIRST
Kekuléstr. 7, 12489 Berlin, Germany
Gunnar.Raetsch@tuebingen.mpg.de

Hiroto Saigo
Bioinformatics Center
Institute for Chemical Research
Kyoto University
Uji 611-0011, Japan
hiroto@kuicr.kyoto-u.ac.jp

Bernhard Schölkopf
Max Planck Institute for Biological Cybernetics
Spemannstr. 38, 72076 Tübingen, Germany
bernhard.schoelkopf@tuebingen.mpg.de

Alexander Johannes Smola
Machine Learning Group, RSISE
The Australian National University
Canberra, ACT 0200, Australia
Alex.Smola@anu.edu.au

Sören Sonnenburg
Fraunhofer FIRST
Kekuléstr. 7, 12489 Berlin, Germany
Soeren.Sonnenburg@first.fraunhofer.de

Koji Tsuda
Max Planck Institute for Biological Cybernetics
Spemannstr. 38, 72076 Tübingen, Germany
AIST Computational Biology Research Center
2-43, Aomi, Koto-ku, 135-0064 Tokyo, Japan
koji.tsuda@tuebingen.mpg.de

Jean-Philippe Vert
Ecole des Mines de Paris
Centre de Géostatistique
35 rue Saint-Honoré
77300 Fontainebleau, France
Jean-Philippe.Vert@mines.org

Régis Vert
LRI, Bâtiment 490
Université Paris-Sud
91405 Orsay cedex, France
Regis.Vert@lri.fr

S.V.N. Vishwanathan
Machine Learning Program
National ICT Australia
Canberra, ACT 0200, Australia
vishy@axiom.anu.edu.au

Yoshihiro Yamanishi
Bioinformatics Center,
Institute for Chemical Research,
Kyoto University, Gokasho, Uji,
Kyoto 611-0011, Japan
yoshi@kuicr.kyoto-u.ac.jp

Alexander Zien
Max Planck Institute for Biological Cybernetics
Spemannstr. 38, 72076 Tübingen, Germany
alexander.zien@tuebingen.mpg.de

Index