Machine Learning for Genomic Sequence Analysis

- Dissertation -

vorgelegt von

Dipl. Inform.
Sören Sonnenburg

aus Berlin

Von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin

Technische Universität Berlin

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
— Dr. rer. nat. —

genehmigte Dissertation

Promotionsaussschuß

- Vorsitzender: Prof. Dr. Thomas Magedanz
- Berichter: Dr. Gunnar Rätsch
- Berichter: Prof. Dr. Klaus-Robert Müller

Tag der wissenschaftlichen Aussprache: 19. Dezember 2008

Berlin 2009
D83

Acknowledgements Above all, I would like to thank Dr. Gunnar Rätsch and Prof. Dr. Klaus-Robert Müller for their guidance and inexhaustible support, without which writing this thesis would not have been possible. All of the work in this thesis has been done at the Fraunhofer Institute FIRST in Berlin and the Friedrich Miescher Laboratory in Tübingen. I very much enjoyed the inspiring atmosphere in the IDA group headed by K.-R. Müller and in G. Rätsch's group in the FML. Much of what we have achieved was only possible in a joint effort. As the various fruitful within-group collaborations expose — we are a team.

I would like to thank Fabio De Bona, Lydia Bothe, Vojtech Franc, Sebastian Henschel, Motoaki Kawanabe, Cheng Soon Ong, Petra Philips, Konrad Rieck, Reiner Schulz, Gabriele Schweikert, Christin Schäfer, Christian Widmer and Alexander Zien for reading the draft, helpful discussions and moral support.

I acknowledge the support from all members of the IDA group at TU Berlin and Fraunhofer FIRST and the members of the Machine Learning in Computational Biology group at the Friedrich Miescher Laboratory, especially for the tolerance of letting me run the large number of compute-jobs that were required to perform the experiments. Finally, I would like to thank the system administrators Roger Holst, Andre Noll, Andreas Schulz, Rolf Schulz and Sebastian Stark — your support is very much appreciated.

This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence (IST-2002-506778), and by the Learning and Inference Platform of the Max Planck and Fraunhofer Societies.

To my parents.

Preface

When I came into first contact with Machine Learning at a seminar on Support Vector Machines (SVMs) held by Klaus-Robert Müller and Bernhard Schölkopf in 1999, I became utterly excited about this subject. During that seminar, I gave an introductory talk on SVMs to an all-knowing audience, but it took a while before I seriously picked up the SVM subject in 2002. For my student research project, Gunnar Rätsch suggested Hidden Markov Models (HMMs) and their application to Bioinformatics. The idea was to find genes, but as this problem is too complex to start with, I started with the sub-problem of recognising splice sites. HMMs using a certain predefined topology performed quite well. However, we managed to improve on this using string kernels (derived from HMMs) and Support Vector Machines (Sonnenburg et al., 2002, Tsuda et al., 2002b). Unfortunately, training SVMs with these string kernels was computationally very costly. Even though we had a sample of more than 100,000 data points, we could barely afford training on 10,000 instances on a, at that time, cutting-edge Compaq Alpha compute cluster. In contrast to HMMs, the resulting trained SVM classifiers were not easily accessible. After a research break of about 2 years during which I became the group's system administrator, I started work on exactly these — still valid — topics. All of this research was application driven: improving the splicing signal (and later, other signals) detection in order to construct a gene finder that is applicable on a genomic scale.
To this end, I worked on computationally efficient string kernels suitable for detecting biological signals such as the aforementioned splicing signal and on large scale algorithms for SVMs with string kernels. The results of these efforts are presented in this doctoral thesis in Chapter 2 and 3. Trying to improve the classification performance of a string kernel SVM using the so-called weighted degree kernel, I investigated multiple kernel learning, which turned out to be useful to understand the learnt SVM classifier. Having learnt lessons about the implied string kernel feature spaces (by working on large scale learning) and since there was still room for improvements in understanding SVMs, I continued to research this subject. I consider the resulting Positional Oligomer Importance Matrices (POIMs) a big step forward toward a full understanding of string kernel based classifiers. Both concepts are described in Chapter 4.
Since my research was application motivated, all algorithms were implemented in the very versatile SHOGUN toolbox that I developed in parallel (cf. Appendix C). Equipped with this tool-set, several signal detection problems, like splice site recognition and transcription start site recognition, were tackled. The work on large scale learning with string kernels enabled us to train string kernel SVMs on 10,000,000 instances, obtaining state-of-the-art accuracy, and to apply the trained SVMs to billions of instances. At the same time, I used POIMs to understand the complex SVM decision rule. These improved signal detectors were finally integrated to *accurately* predict gene structures (Rätsch et al., 2007), as outlined in Chapter 6 and are used in a full gene finding system (Schweikert et al., 2008) outperforming state-of-the-art methods.

Zusammenfassung

Die Entwicklung neuer Sequenziertechnologien ebnete den Weg für kosteneffiziente Genomsequenzierung. Allein im Jahr 2008 werden etwa 250 neue Genome sequenziert. Es ist offensichtlich, dass diese gewaltigen Mengen an Daten effektive und genaue computergestützte Methoden zur Sequenzanalyse erfordern. Diese werden benötigt, um eines der wichtigsten Probleme der Bioinformatik zu lösen: die akkurate Lokalisation von Genen auf der DNA. In dieser Arbeit werden auf Basis von Support Vector Machines (SVMs) genaueste genomische Signalerkenner entwickelt, die in Gensuchmaschinen verwendet werden können. Die Arbeit untergliedert sich in folgende Themenschwerpunkte:

String-Kerne Es wurden String-Kerne zur Detektion von Signalen auf dem Genom entwickelt und erweitert. Die Kerne haben eine in der Länge der Eingabesequenzen nur lineare Berechnungskomplexität und sind für eine Vielzahl von Problemen verwendbar. Dadurch gestaltet sich die Sequenzanalyse sehr effektiv: Mit nur geringem Vorwissen ist es möglich, geeignete String-Kernkombinationen auszuwählen, die hohe Erkennungsraten ermöglichen.

Large-Scale-Lernen Das Training von SVMs war bisher zu rechenintensiv, um auf Daten genomischer Größe angewendet zu werden. Mithilfe der in dieser Arbeit entwickelter Large-Scale-Lernmethoden ist es nun in kurzer Zeit möglich, string-kern-basierte SVMs auf bis zu zehn Millionen Sequenzen zu trainieren und auf über sechs Milliarden Sequenzpositionen vorherzusagen. Der entwickelte `linadd`-Ansatz beschleunigt die Berechnung von Linearkombinationen von String-Kernen, die bereits in linearer Zeit berechenbar sind. Dieser Ansatz ermöglicht den Verzicht auf einen Kern-Cache beim SVM-Training und führt somit zu einer drastischen Reduktion des Speicheraufwands.

Interpretierbarkeit Ein häufig kritisierter Nachteil von SVMs mit komplexen Kernen ist, dass ihre Entscheidungsregeln für den Menschen schwer zu verstehen sind. In dieser Arbeit wird die „Black Box" der SVM-Klassifikatoren „geöffnet", indem zwei Konzepte entwickeln werden, die zu ihrem Verständnis beitragen: Multiple Kernel Learning (MKL) und Positional Oligomer Importance Matrices (POIMs). MKL ermöglicht die Interpretation von SVMs mit beliebigen Kernen und kann dazu verwendet werden heterogene Datenquellen zu fusionieren. POIMs sind besonders gut für String-Kerne geeignet, um die diskriminativsten Sequenzmotive zu bestimmen.

Genom-Sequenzanalyse In dieser Arbeit werden SVMs mit neuentwickelten, sehr präzisen String-Kernen zur Detektion einer Vielzahl von Genom-Signalen, zum Beispiel der Transkriptionsstart- oder Spleißstellen, angewendet. Dabei werden unerreichte Genauigkeiten erzielt. Unter Verwendung von POIMs wurden die trainierten Klassifikatoren analysiert und unter anderem viele bereits bekannte Sequenzmuster gefunden. Die verbesserten Signaldetektoren wurden dann zur genauen Genstrukturvorhersage verwendet. Die Doktorarbeit schließt mit einem Ausblick, wie in dieser Arbeit entwickelte Methoden die Basis für eine vollständige Gensuchmaschine legen.

Keywords: Support Vector Machine, Interpretierbarkeit, String-Kerne, Large-Scale-Lernen, Multiple Kernel Learning, Positional Oligomer Importance Matrices, Spleißstellenerkennung, Transkriptionsstartstellenerkennung, Gensuche

Summary

With the development of novel sequencing technologies, the way has been paved for cost efficient, high-throughput whole genome sequencing. In the year 2008 alone, about 250 genomes will have been sequenced. It is self-evident that the handling of this wealth of data requires efficient and accurate computational methods for sequence analysis. They are needed to tackle one of the most important problems in computational biology: the localisation of genes on DNA. In this thesis, I describe the development of state-of-the-art genomic signal detectors based on Support Vector Machines (SVM) that can be used in gene finding systems. The main contributions of this work can be summarized as follows:

String Kernels We have developed and extended string kernels so that they are particularly well suited for the detection of genomic signals. These kernels are computationally very efficient — they have linear effort with respect to the length of the input sequences — and are applicable to a wide range of signal detection problems. Only little prior knowledge is needed to select a suitable string kernel combination for use in a classifier that delivers a high recognition accuracy.

Large Scale Learning The training of SVMs used to be too computationally demanding to be applicable to datasets of *genomic scale*. We have developed large scale learning methods that enable the training of string kernel based SVMs using up to ten million instances and the application of the trained classifiers to six billions of instances within reasonable time. The proposed `linadd` approach speeds up the computation of linear combinations of already linear time string kernels. Due to its high efficiency, there is no need for kernel caches in SVM training. This leads to drastically reduced memory requirements.

Interpretability An often criticised downside of SVMs with complex kernels is that it is very hard for humans to understand the learnt decision rules and to derive insight from them. We have opened the "black box" of SVM classifiers by developing two concepts helpful for their understanding: Multiple Kernel Learning (MKL) and Positional Oligomer Importance Matrices (POIMs). While MKL algorithms work with arbitrary kernels and are also useful in fusing heterogeneous data sources, POIMs are especially well suited for string kernels and the identification of the most discriminative sequence motifs.

Genomic Sequence Analysis We have applied SVMs using novel string kernels to the detection of various genomic signals, like the transcription start and splice sites, outperforming state-of-the-art methods. Using POIMs, we analysed the trained classifiers, demonstrating the fidelity of POIMs by identifying, among others, many previously known functional sequence motifs. Finally, we have used the improved signal detectors to accurately predict gene structures. This thesis concludes with an outlook of how this framework prepares the ground for a fully fledged gene finding system outcompeting prior state-of-the-art methods.

Keywords: Support Vector Machine, Interpretability, String Kernel, Large Scale Learning, Multiple Kernel Learning, Positional Oligomer Importance Matrices, Splice Site Recognition, Transcription Start Site Prediction, Gene Finding

Contents

1 Introduction

With the sequencing of entire genomes, important insight into gene functions and genetic variation has been gained over the last decades. As novel sequencing technologies are rapidly evolving, the way will be paved for cost efficient, high-throughput whole genome sequencing, which is going to provide the community with massive amounts of sequences. It is self-evident that the handling of this wealth of data will require efficient and accurate computational methods for sequence analysis. Among the various tools in computational genetic research, gene prediction remains one of the most prominent tasks, as recent competitions have further emphasised (e.g., Bajic et al. (2006), Guigo et al. (2006), Stein et al. (2007)). Accurate gene prediction is of prime importance for the creation and improvement of annotations of recently sequenced genomes (Rätsch et al., 2007, Bernal et al., 2007, Schweikert et al., 2008). In the light of new data related to natural variation (e.g., Hinds et al. (2005), The International HapMap Consortium (2005), Clark et al. (2007)), the importance of accurate computational gene finding gains increasing importance since it helps to understand the effects of polymorphisms on the gene products. In this work we focus on improving genomic signal detectors so as to increase their prediction accuracy beyond the current state-of-the-art. These predictions can then be used to develop more accurate gene finding systems (Schweikert et al., 2008).

Organisation of the Thesis Following a short introduction into genomic sequence analysis (cf. Section 1.1), we introduce Support Vector Machines (cf. Section 1.2), advanced sequence kernels (cf. Chapter 2) to learn to distinguish true signal sites from decoy sites. SVMs were not previously applicable to problems of genomic scale. Training on millions of examples or the application of the trained classifier to billions of instances was computationally far too demanding even for modern compute cluster systems. To address this problem, we have developed computationally very efficient SVM learning methods that are applicable on very large data sets (cf. Chapter 3). Another major drawback of SVMs was their "black box" nature, i.e., their decision function was typically hard to understand. In Chapter 4 we describe two novel methods that can help to gain insights into the learnt decision function. The first approach based on *Multiple Kernel Learning* (Section 4.1) aids understanding of SVMs even with general kernels, while the second approach called *Positional Oligomer Importance Matrices* (Section 4.2) enables pin-pointing of the most discriminating sequence motifs from string kernel based SVMs. We show that the newly developed signal detectors based on string kernel SVMs operate with very high previously unobtainable accuracy (Chapter 5). Finally, we show how these predictors can be integrated to predict gene structures with half of the error rate of the best competing methods (Section 6.1). We conclude with an outline of the remaining steps towards a fully fledged gene finding system (Section 6.2) and an outlook (Section 7).

1.1 Biological Background

A brief history Almost 150 years ago in 1865, Gregor Johann Mendel defined the basic nature of genes, starting a new branch in biology, *Genetics* (cf. Table 1.1). In 1869, Johannes Friedrich Miescher discovered a previously unknown substance in cell nuclei, *deoxyribonucleic acid (DNA)* (Dahm, 2005). In 1944, Oswald Theodore Avery showed chemically that DNA is the substance that is genetically inherited (Lewin, 2000). In 1976, the *ribonucleic acid (RNA)* sequence of the bacteriophage MS2 was the first genome to be fully sequenced (Fiers et al., 1976). Coding for only 3 genes, the MS2 genome is composed of 3,569 nucleotides (nt), a major challenge for the sequencing technologies available at the time. With the help of ribonuclease and electrophoresis, RNA was divided into small fragments of sizes between 50 and 250 nt that could be sequenced (Jou et al., 1972). The fragment sequences were then manually assembled. A year later, the first DNA-based genome, belonging to the virus Phage Φ-X174, was fully sequenced by a group around Frederick Sanger using the plus-minus sequencing method (Sanger and Coulson, 1975, Sanger et al., 1977). In contrast to previous approaches that were based on degrading DNA, this method synthesises DNA sequences (Goujon, 2001). It was this work and its extension to the chain terminator method (Sanger et al., 1977) that earned Sanger his second Noble prize. Still, Sanger had to manually assemble the 5,386 bases of the virus genome, containing 11 genes.
With the invention of shotgun sequencing where DNA is randomly fragmented into small pieces of about 2kb that are individually sequenced, and subsequent whole genome shotgun sequencing techniques, many organisms have been completely sequenced. Among them are: Baker's yeast in 1996 (Goffeau et al., 1996, 12 million base-pairs, more than 6,000 genes), the 97Mb genome with 19,000 genes of the nematode *Caenorhabditis elegans* completed at the end of 1998 (The C. *elegans* Sequencing Consortium, 1998), and the human genome first draft (The International Human Genome Mapping Consortium, 2001) and completed sequences (Gregory et al., 2006). The many reads/fragments generated by these project were mostly assembled computationally, relying on overlaps between reads to generate larger contiguous sequences or contigs. While the press featured headlines such as "human genome deciphered", today we merely know the DNA sequence of a few individuals of our species. Genomic annotation data, e.g., for the location of genes, do exist, but they are neither complete nor necessarily accurate. Even for the much simpler roundworm *C. elegans* it is currently estimated that about 10%-50% of the annotated genes contain errors (Merrihew et al., 2008, Rätsch et al., 2007).
In 2007, the genome of the first human individual (that of Craig Venter) was sequenced at a cost of about $100 million using shotgun sequencing (Levy et al., 2007). Among the recently developed techniques is pyro sequencing (Ronaghi et al., 1996) where the addition of nucleotides during the synthesis of DNA results in the emission of light at nucleotide-specific wave lengths so that at each step, the added nucleotide can be determined. This scheme has been massively parallelised in 454 sequencing (Wikipedia, 2008a). Using this technique, the complete genome of James Watson was sequenced for less than $1 million (Wheeler et al., 2008). The most recent development of "next-generation sequencing techniques" (e.g., Marusina, 2006) enables the sequencing of large human-scale genomes within weeks at a cost of less than $100,000 possible with Solexa and ABI solid platforms (Hayden, 2008, Illumina, 2008) and will inevitably lead to the "$1000 genome". Since more than 850 genomes will be completely sequenced by the end of 2008 (Liolios et al., 2007) and more than 4,000 genome projects have been registered

as of September 19, 2008,[1] the demand for methods to *accurately* and automatically determine the locations of genes has never been higher. This work focuses on methods to accurately detect properties of genes, like the gene start and end, and the protein coding regions (exons) contained in eukaryotic genes, in order to aid *gene finding.*

Genetic Background For those unfamiliar with genetics, a short overview will be given in the following. We are exclusively dealing with the class of eukaryotes, i.e., organisms that have nuclei and a cytoskeleton in their cells. All "higher" organisms are eukaryotic, e.g., humans as well as *C. elegans* belong to this class while, e.g., bacteria belong to the class of prokaryotes. The word "eukaryote" means "true nuclei", i.e., the nuclei contain genetic information that is organised into discrete chromosomes and contained within a membrane-bounded compartment. In principle, [2] the whole set of chromosomes, containing all of the organism's genes and regulatory elements, is referred to as *the genome.* For example, the human genome consists of 23 chromosome pairs, where the pair 23 determines the sex (females have two X chromosomes; males have one X and one Y chromosome).

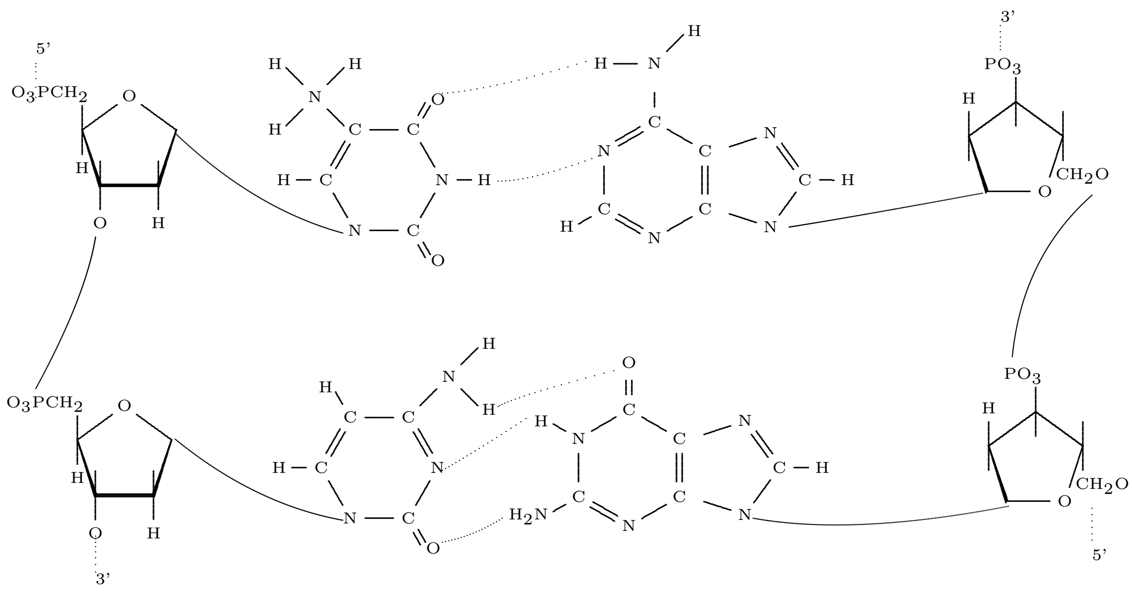


Figure 1.1: Schematic illustration of the four bases adenine (A), cytosine (C), guanine (G), thymine (T) and their pairing using hydrogen bonds. A sequence of only the two base pairs T-A (top) and C-G (bottom) is shown. In the outermost parts, one can see the sugar molecules, which form the backbone of DNA. The strands are antiparallel, i.e., one strand runs in $5' \rightarrow 3'$ direction, while the other runs $3' \rightarrow 5'$.

Each chromosome is composed of a single double-stranded DNA molecule. Single-stranded DNA consists of alternating pentose (sugar) and phosphate residues with a base attached to each pentose. Either one of the bases Adenine, Thymine, Guanine or Cytosine is linked to a sugar at position 1 (cf. Figure 1.1). They are usually referred to by their initial letters. The name DNA is derived from the sugar-component in DNA (2-deoxyribose), the "backbone" of DNA. Metaphorically speaking, double-stranded DNA looks like a rope ladder twisted into a double helix. The ropes to either side of the ladder correspond to the sugar backbones of the two DNA strands, while the "steps"

[1]See `http://www.genomesonline.org/gold.cgi`

[2]To be correct, the DNA contained in chloroplasts, mitochondria, and plasmids does also belong to the genome.

1865	Genes are particulate factors (Gregor Mendel)
1869	Discovery of DNA by Friedrich Miescher (Dahm, 2005)
1903	Chromosomes are hereditary units (Walter Sutton)
1910	Genes lie on chromosomes
1913	Chromosomes contain linear arrays of genes
1927	Mutations are physical changes in genes
1931	Recombination is caused by crossing over
1944	DNA is genetic material (Avery et al., 1944)
1945	A gene codes for a protein
1953	Double helix structure of DNA found (Watson and Crick, 1953)
1956	First successful DNA synthesis (Kornberg, 1991)
1958	DNA replicates semiconservatively (Meselson and Stahl, 1958)
1961	Genetic code is composed of triplets (Crick et al., 1961, Nirenberg, 2004)
1972	Nucleotide sequence of first gene known (Jou et al., 1972)
1976	First RNA-based genome (Bacteriophage MS2) sequenced (Fiers et al., 1976)
1977	First DNA-based genome sequenced (Bacteriophage ΦX174) (Sanger et al., 1977)
1983	Polymerase Chain Reaction (PCR; Mullis, 2004)
1986	Applied Biosystems markets first automated DNA sequencing machine
1996	Baker's yeast genome sequenced (Goffeau et al., 1996)
1998	Completion of the genome of *Caenorhabditis Elegans* (The C. *elegans* Sequencing Consortium, 1998)
2000	Working Draft of the human genome announced (The International Human Genome Mapping Consortium, 2001)
2002	Draft sequence of the mouse genome announced (Mouse Genome Sequencing Consortium, 2002)
2003	Genome of Bacteriophage ΦX174 chemically synthesised (Smith et al., 2003)
2004	454 high throughput pyro-sequencing (Wikipedia, 2008a)
2006	Completion of the DNA sequencing of the human genome announced (Gregory et al., 2006)
2007	454 Sequencing allows 100 megabases to be sequenced per 7-hour run at cost of less than \$10,000 (Wikipedia, 2008a)
2007	Individual human genome sequenced: of Craig Venter (Levy et al., 2007) and James Watson (Wheeler et al., 2008)
2007	More than 500 genomes sequenced completely (Liolios et al., 2007)
2007	$>90\%$ of the genome is transcribed (Kapranov et al., 2007)
2008	More than 850 genomes are completely sequenced (Liolios et al., 2007)
open	Number and location of human genes determined
open	Function and protein structure of protein coding genes deciphered
open	Function and structure of RNA coding genes deciphered
open	...

Table 1.1: A brief history of genetics. This table is based on (Lewin, 2000) and (Wikipedia, 2008b). A more exhaustive list of sequenced organisms can be found at (Wikipedia, 2007).

of the ladder are formed by pairs of bases with hydrogen bonds between them. Most importantly, only the base Adenine may pair with Thymine and only Guanine may pair with Cytosine, leading to the sequence of bases in the one strand to be the reverse complement of the other. The helix is constructed by linking the 5' position of one pentose ring to the 3' position of the next pentose ring via a phosphate group. As a result, one end of the chain has a free 3' group, while the other has a free 5' group. As a convention, the sequence of bases in strand are written in the 5' to 3' direction. Due to the complementary base pairing, the strands are anti-parallel, i.e., one strand is in $5' \rightarrow 3'$ and the other in $3' \rightarrow 5'$ direction, cf. Figure 1.1. 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 1.2.

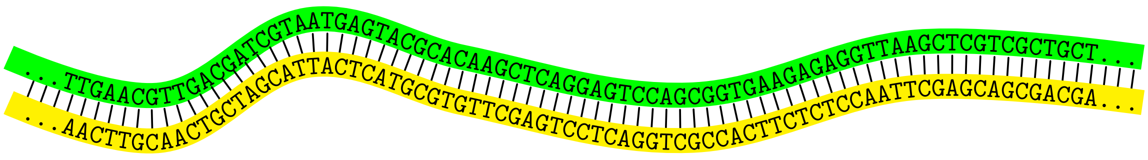

...TTGAACGTTGACGATCGTAATGAGTACGCACAAGCTCAGGAGTCCAGCGGTGAAGAGAGGTTAAGCTCGTCGCTGCT...
...AACTTGCAACTGCTAGCATTACTCATGCGTGTTCGAGTCCTCAGGTCGCCACTTCTCTCCAATTCGAGCAGCGACGA...

Figure 1.2: 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 paired with Thymine, and Guanine with Cytosine.

Each chromosome consists of genetic and non-genetic regions. In many genomes, including human, the latter make up most of the chromosome.[3] But where on the chromosome are genes located and what is a gene in the first place? A gene can be defined as:[4]

Definition 1.1 (Gene (Pearson, 2006))**.** *A locatable region of genomic sequence, corresponding to a unit of inheritance, which is associated with regulatory regions, transcribed regions and/or other functional sequence regions.*

Note that this definition covers not only protein but also RNA coding genes. While the question about the location of genes has not been sufficiently answered yet, it is known that multiple stages are involved in the expression of protein coding genes (i.e., the process of synthesising proteins from genes, cf. Lewin (2000)). These steps are carried out mostly sequentially, but are known to interfere with each other.

1. Activation of gene structure
2. Initiation of transcription
3. Processing of the transcript
4. Postprocessing
5. Transport of mRNA to cytoplasm
6. Translation of mRNA

Genes are "activated" in a tissue-specific fashion. A gene may be active in cells composing one kind of tissue, but "inactive" in other kinds of tissue. When the precondition – the gene is active – is fulfilled, it can be transcribed, that is, a copy of the gene that

[3]The function of the non-genetic regions, i.e., the regions that do not code for proteins, is not well understood. They are likely responsible for regulatory control.

[4]The definition of the term "gene" is still being discussed, cf., `http://www.genomicglossaries.com/content/gene_def.asp`

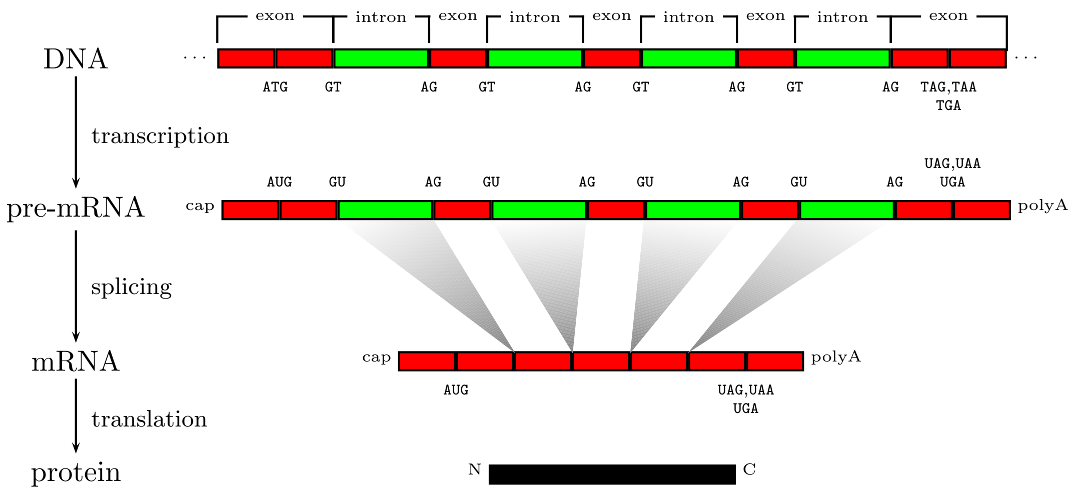


Figure 1.3: The major steps in protein synthesis. See text for details. Idea is taken from (Chuang and Roth, 2001, Lewin, 2000)

is encoded on only one of the two strands of the double-stranded DNA (the coding strand) is synthesized. The copy is not DNA, but single-stranded precursor messenger ribonucleic acid (pre-mRNA). The chemistry of RNA requires that the role of Thymine is taken over by the base Uracil (U). For convenience, we will use Uracil and Thymine synonymously. Transcription starts when an enzyme called *RNA polymerase* binds to the *promoter*, a special region located *upstream*[5] of the first base pair that is transcribed into pre-mRNA. From this *Transcription Start Site*, RNA polymerase moves *downstream* in $5' \rightarrow 3'$ direction, continuously synthesising pre-mRNA until a termination sequence is reached. Shortly after the start of transcription, a cap is added to the $5'$ end of the pre-mRNA. At the end of transcription, the $3'$ end is cut off at the *Cleavage Site* and a polyA tail (a long sequence of ≈ 250 adenine nucleotides) is appended (Zhao et al., 1999). Both the cap and tail prevent degradation of the RNA by exonucleases. The process of $3'$ end formation promotes transcription termination and transport of the mRNA from the nucleus into the cytoplasm (Zhao et al., 1999). In the "postprocessing" phase, the pre-mRNA is transformed into mRNA. One necessary step in this process of obtaining mature mRNA is called *Splicing*. The coding sequence of many eukaryotic genes is "interrupted" by non-coding regions called *introns*. Such a multi-exonic gene starts with an exon and is then interrupted by an *intron*, followed by another exon, intron etc. until it ends in an exon. The $5'$ end of an intron (exon-intron boundary) is called *Donor Splice Site* and almost always contains a `GU` consensus. The $3'$ end of an intron (intron-exon boundary) is called *Acceptor Splice Site* and almost always contains an `AG` consensus. These short consensus motifs are not sufficient for accurate splice site detection. After the pre-mRNA has been spliced to form mRNA, the splicing product may be transported from the nucleus to the cytoplasm. There, during translation of protein coding genes, the mRNA is read as *codons*, i.e., as triplets of nucleotides. Hence, there are three different *reading frames*, i.e., ways of reading triplets of RNA (one for each of the possible start positions $0, 1$ or 2). 61 of the $4^3 = 64$ possible codons code for 20 amino acids, while the remaining 3 (`UAG,UAA,UGA`) are termination codons, which mark the end of the protein-coding sequence. The translation

[5]Upstream means closer to the 5' end, while downstream means closer to the 3' end.

begins at a *translation initiation site* (TIS) that almost always is the start codon `AUG`. However, only a minority of all `AUG` codons, representing the amino acid methionine, actually signals a TIS. When a stop codon is reached, translation is terminated at the *Translation Termination Site* and the set of amino acids, the result of the translation process, form a long polypeptide, the *protein.* Figure 1.3 schematically summarises the entire process of protein expression.

1.1.1 Sequence Analysis Problems

Detecting any of the above mentioned *signals*, i.e., the transcription start, translation start, donor and acceptor splice site as well as translation end, polyA signal and cleavage site, can be formulated as a sequence classification task. In computational gene finding, these *signals* are often interspersed with variable length regions, like exons and introns. While the latter are recognised by methods called *content* sensors, the former can be recognised by methods named *signal* sensors (Haussler, 1998). For example, 5th-order Markov chains are very common content sensors for modelling eukaryotic introns and exons. Inhomogeneous Markov chains and decision trees are common methods for signal detectors. The following paragraphs summarise the most important signal and content sensors for gene finding.

Transcription Start Site Prediction One of the key elements to recognizing genic regions in the genome is to identify the promoter site. Methods for finding transcription start sites (TSS) of RNA-polymerase II transcribed genes often exploit their association with CpG islands (i.e., regions with a higher than expected frequency of Cytosine-Guanine dinucleotides, e.g., Gardiner-Garden and Frommer (1987)) overlapping or near the TSS (Down and Hubbard, 2002, Davuluri et al., 2001). For further approaches, see the discussion in Section 5.4.

Translation Initiation Site Prediction Translation converts part of the nucleotide sequence of mRNA into the sequence of amino acids comprising a protein. Not the whole length of mRNA is translated, but only the coding region (CDS), defined by the translation initiation site (begins with start `ATG` codon) and stop ({`TAA`,`TAG`,`TGA`}) codons. Among the methods for detecting translation initiation sites are markov chains, neural networks and support vector machines (e.g., Pedersen and Nielsen, 1997, Zien et al., 2000, Saeys et al., 2007).

Splice Site Prediction Splice sites mark the boundaries between potentially protein-coding exons on the one hand and on the other hand, introns that are excised from pre-mRNAs after transcription. Most splice sites are so-called *canonical splice sites* that are characterised by the presence of the dimers `GT` and `AG` at the donor and acceptor sites, respectively. However, the occurrence of the dimer is not sufficient for a splice site. The classification task for splice site sensors therefore consists in discriminating true splice sites from decoy sites that also exhibit the consensus dimers. Often *Positional Weight Matrices* (PWMs) or *Inhomogeneous Markov Chains* (IMCs) are used to detect splice sites. For further approaches, see the discussion in Section 5.3.

Recognising Segment Content Content sensors are used to recognise the typical sequence composition of the individual segments. For each segment type (intergenic, inter-cistronic, untranslated region, coding exon, intron) one may design a specific content sensor. Markov Chains (Durbin et al., 1998) are among the most frequently used content sensors.

Translation End Site Prediction The coding region (CDS) ends with one of the ({TAA,TAG,TGA}) stop codons. Additional sequence motifs around the translation end aid in localising this site.

Polyadenylation Signal Prediction Polyadenylation (polyA) is a process by which the $3'$ ends of pre-mRNA molecules are transformed. PolyA comprises the addition of around 50-250 Adenosines to the $3'$ end of the cleaved pre-mRNA. Relative to the cleavage site, there are an upstream element consisting of a highly conserved AATAAA (or similar) hexamer called the polyadenylation signal (PAS), and a downstream element often described as a poorly conserved GT- or T-rich sequence Liu et al. (2003), Philips et al. (2008).

Cleavage Site Prediction The cleavage site is located approximately 10-25 base pairs downstream of the polyadenylation signal. Around 10-30 nucleotides downstream of the cleavage site, a U-rich motif is located (Chen et al., 1995). Since the cleavage and polyadenylation signal sites are close, prediction of the polyadenylation signal and the cleavage site is tightly coupled.

Summary of the Signal Detection Problems The different signal detection problems can roughly be categorised according to the spatial arrangement of the relevant features:

1. very localised – position dependent
2. localised but variable – position semi-dependent
3. very variable – position independent

Splice sites are very localised due to high selective pressure. A mis-detection often introduces frameshifts, which impose early termination of transcription (many stop codons occur in non-coding frames). As a result, methods employing positional information like inhomogeneous markov chains perform well on that task. Transcription start sites and many of the other signals are more weakly localised and characterised by many positionally variable weak motifs. For content sensors, the localisation of the motif does not matter. Only its occurrence (frequency) does. Since signal and content sensors alone perform poorly, integrated models, such as Generalised Hidden Markov Models (GHMMs Kulp et al., 1996) are used to combine both methods. Integrated models may capture "higher correlations" between different signals and can incorporate constraints and structure. For example, prior knowledge, like "an intron is always followed by an exon"; "the distance of the promoter to the translation start is at least a certain number of bases", etc. can be incorporated into such models.

In this thesis, we focus on *methods* to improve the signal and content sensors for the task of gene finding.

1.2 Machine Learning

In Machine Learning, the task of classification is to find a rule, which based on external observations, assigns an object to one of several classes (e.g., Müller et al., 2001, Ben-Hur et al., 2008). In the simplest case of real valued vectors representing objects and only two classes, one possible formalisation of this problem is to estimate a function (a classifier) $f : \mathbb{R}^D \to \{-1, +1\}$ that assigns a label $\hat{y} \in \{-1, +1\}$ to each object $\mathbf{x} \in \mathbb{R}^D$. To estimate this function, N labelled training examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ are given, where the assumption is made that the pairs of training data are generated independent and identically distributed according to an unknown probability distribution $\Pr[\mathbf{x}, y]$. One hopes that the classifier f performs well on unseen examples, i.e., that the estimated label $f(\mathbf{x}) = \hat{y}$ is equal to the true label y for an unseen sample $\mathbf{x}$, which is assumed to be generated from the same distribution $\Pr[\mathbf{x}, y]$. Thus, the goal is to find the "best" function f, i.e., the function that minimises the expected error (risk)

$$R[f] = \int l(f(\mathbf{x}), y) d\Pr[\mathbf{x}, y].$$

Here l denotes a loss function. Its arguments are the predicted value $\hat{y} = f(\mathbf{x})$ and the true label y. In classification, the $0-1$ loss is usually used, which can be defined as

$$l(f(\mathbf{x}), y) = \begin{cases} 0, & f(\mathbf{x}) = y \\ 1, & \text{otherwise} \end{cases}.$$

When looking at the risk functional, one recognises that the risk depends on the *unknown* probability distribution $\Pr[\mathbf{x}, y]$. All one can do is to estimate the function f based on the available information, i.e., from a *finite* number of training examples. There are several methods of estimation to accomplish this. They are called inductive principles (Cherkassky and Mulier, 1998). For example, one simple principle is to approximate the risk functional by the *empirical risk*

$$R_{emp} = \frac{1}{N} \sum_{i=1}^{N} l(f(\mathbf{x}_i), y_i).$$

Given a class of functions $f \in F$ described by a set of parameters, one could now choose the best f by using the one that minimises the empirical risk. Doing so we get an estimate of the decision boundary. This type of learning the decision function f is called *discriminative learning*, since we were given positive and negative examples and used these to derive a function f that separates these two classes. However, in the case that the set of functions F can capture arbitrary decision boundaries, one could suffer from *overfitting*. On the other hand, F might contain functions that are too simple and hence, incapable of solving the classification problem. This dilemma is explained best by an example. Consider the task is to classify trees and non-trees.[6] When a botanist with photographic memory has memorised a number of trees, he will not realise that a newly presented tree is a tree since the number of its leaves will differ compared to any of the memorised trees. On the other hand, the botanist's lazy brother declares that if something is green it is a tree and thus, mistakes a frog for a tree. It is therefore crucial to choose F appropriately.

1.2.1 Generative Models

Instead of learning a discriminative decision function, one could estimate the class conditional probability distribution $\Pr[\mathbf{x}|Y = +1, \boldsymbol{\theta}_+]$ and $\Pr[\mathbf{x}|Y = -1, \boldsymbol{\theta}_-]$ using generative models $\boldsymbol{\theta}_+$ and $\boldsymbol{\theta}_-$ and the class prior $\alpha := \Pr[Y = +1|\alpha]$ directly. This approach is an example of *generative or descriptive learning*. Using the class conditional likelihoods and the class prior, we define $\boldsymbol{\theta} := (\boldsymbol{\theta}_+, \boldsymbol{\theta}_-, \alpha)$. To minimize the expected error, one assigns to the object $\mathbf{x}$ the class label for which its posterior probability

$$\Pr[\hat{y}|\mathbf{x}, \boldsymbol{\theta}] = \frac{\Pr[\hat{y}|\boldsymbol{\theta}] \Pr[\mathbf{x}|\hat{y}, \boldsymbol{\theta}]}{\Pr[\mathbf{x}]}$$

[6]This example is taken from (Burges, 1998)

is largest, which, assuming the model is true, leads to the Bayes optimal decision rule

$$\begin{aligned} f(\mathbf{x}) &:= \operatorname*{argmax}_{\hat{y}\in\{-1,+1\}} \Pr[\hat{y}|\boldsymbol{\theta}]\Pr[\mathbf{x}|\hat{y},\boldsymbol{\theta}] \\ &= \operatorname{sign}\left(\Pr[Y=+1|\boldsymbol{\theta}]\Pr[\mathbf{x}|Y=+1,\boldsymbol{\theta}] - \Pr[Y=-1|\boldsymbol{\theta}]\Pr[\mathbf{x}|Y=-1,\boldsymbol{\theta}]\right). \end{aligned}$$

One may equivalently use the convenient posterior log-odds of a probabilistic model that are obtained by applying the logarithm to the posterior probability, i.e.,

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sign}\left(\log\left(\Pr[Y=+1|\mathbf{x},\boldsymbol{\theta}]\right) - \log\left(\Pr[Y=-1|\mathbf{x},\boldsymbol{\theta}]\right)\right) \\ &= \operatorname{sign}\left(\log\left(\Pr[\mathbf{x}|Y=+1,\boldsymbol{\theta}]\right) - \log\left(\Pr[\mathbf{x}|Y=-1,\boldsymbol{\theta}]\right) + b\right), \end{aligned}$$

where $b = \log\Pr[Y=+1|\boldsymbol{\theta}] - \log\Pr[Y=-1|\boldsymbol{\theta}]$ is a bias term that corresponds to the class priors. Then, when using the maximum likelihood (ML) inductive principle, one has to choose an estimate for the parameters $\widehat{\boldsymbol{\theta}} \in \boldsymbol{\Theta}'$ of the generative models $\Pr[\mathbf{x}|\widehat{\boldsymbol{\theta}}_+, Y=+1]$ and $\Pr[\mathbf{x}|\widehat{\boldsymbol{\theta}}_-, Y=-1]$ such that the likelihood functions (Cherkassky and Mulier, 1998)

$$\begin{aligned} \Pr[\mathbf{x}_{i_1},\ldots,\mathbf{x}_{i_{N_+}}|\boldsymbol{\theta}_+, y_{i_1}=+1,\ldots,y_{i_{N_+}}=+1] &= \prod_{n=1}^{N_+}\Pr[\mathbf{x}_{i_n}|\boldsymbol{\theta}_+, y_{i_n}=+1] \text{ and} \\ \Pr[\mathbf{x}_{j_1},\ldots,\mathbf{x}_{j_{N_-}}|\boldsymbol{\theta}_-, y_{j_1}=-1,\ldots,y_{j_{N-}}=-1] &= \prod_{n=1}^{N_-}\Pr[\mathbf{x}_{j_n}|\boldsymbol{\theta}_-, y_{j_n}=-1], \end{aligned}$$

are maximised independently. Here $\widehat{\boldsymbol{\theta}}$ denotes the estimate of the model parameters and $y_{i_1},\ldots,y_{i_{N_+}} = +1$ and $y_{j_1},\ldots,y_{j_{N_-}} = -1$,. The class prior can be estimated to be $\alpha = \Pr[Y=+1|\boldsymbol{\theta}] = N_+/N$. Hence, ML returns an estimate $\widehat{\boldsymbol{\theta}}$ for the true model parameters $\boldsymbol{\theta}^\star$, which maximises the above likelihood functions for the given training dataset. Under the assumption that the true distribution can be parametrised and the number of examples is large enough, the estimated probability distributions can lead to the Bayes optimal decision rule. Furthermore, one can create new examples by sampling from the class conditional distributions and use the sample density functions to define a metric or to derive a kernel function that can be used in discriminative classifiers to improve the decision function, given weak probabilistic models (Jaakkola and Haussler, 1999, Tsuda et al., 2002a).

Inhomogeneous Markov Chains In sequence analysis, one commonly used probabilistic model is that of (inhomogeneous) Markov Chains. Reasons for their widespread use are their flexibility in modelling a variety of signal and content sensors, their simplicity, easy interpretation, fast training methods, and their overall good performance on classification tasks.

Definition 1.2 (Durbin et al. (Markov Chain, 1998))**.** *Given a sequence* $\mathbf{x} := x_1,\ldots,\mathbf{x}_{l_\mathbf{x}}$ *of length* $l_\mathbf{x}$ *and random variables* $\mathbf{X} := X_1,\ldots,X_{l_\mathbf{x}}$ *taking values of* x_i*, a Markov chain of order d is defined as*

$$\Pr[\mathbf{X}=\mathbf{x}|\boldsymbol{\theta}^\pm] := \Pr[X_1=x_1,\ldots,X_d=x_d|\boldsymbol{\theta}^\pm]\prod_{i=d+1}^{l_\mathbf{x}}\Pr[X_i=x_i|X_{i-1}=x_{i-1},\ldots,X_{i-d}=x_{i-d},\boldsymbol{\theta}^\pm]. \tag{1.1}$$

A Markov Chain (MC) is a model with limited memory that approximates the class conditional likelihood where, given the last d states $X_{i-1},\ldots,X_{i-d}$*, the predicted future state* X_i *is independent of the past states* $X_{i-d-1},\ldots,X_1$ (Markov property)*. In case* *the model parameters remain constant over sequence position, i.e.,*

$$\Pr[X_{i+1} = x_{d+1}|X_i = x_1, \ldots, X_{i-d+1} = x_d, \boldsymbol{\theta}^{\pm}] = \Pr[X_i = x_{d+1}|X_{i-1} = x_1, \ldots, X_{i-d} = x_d, \boldsymbol{\theta}^{\pm}]$$

the MC is called homogeneous*, and* inhomogeneous *otherwise.*

For DNA, $\mathbf{x}$ are sequences of length $l_{\mathbf{x}}$, with symbols taken from the four letter alphabet $x_i \in \Sigma = \{A, C, G, T\}$. MC are typically defined by introducing a state for each nucleotide, i.e., $X_i \in \Sigma$. To learn a Markov chain, each factor in Equation (1.1) has to be estimated in model training, e.g., one counts how often each symbol appears at each position in the training data conditioned on every possible $x_{i-1}, \ldots, x_{i-d}$. Then, for given model parameters $\widehat{\boldsymbol{\theta}}$ we have

$$\Pr[\mathbf{X} = \mathbf{x}|\widehat{\boldsymbol{\theta}}^{\pm}] = \hat{\theta}_d^{\pm}(x_1, \ldots, x_d) \prod_{i=d+1}^{l_{\mathbf{x}}} \hat{\theta}_i^{\pm}(x_i, \ldots, x_{i-d}),$$

where $\hat{\theta}_d^{\pm}(x_1, \ldots, x_d)$ is an estimate for $\Pr[X_1 = x_1, \ldots, X_d = x_d]$ and $\hat{\theta}_i^{\pm}(x_i, \ldots, x_{i-d})$ is an estimate for $\Pr[X_i = x_i|X_{i-1} = x_{i-1}, \ldots, X_{i-d} = x_{i-d}]$. In contrast to a homogeneous model, this MC's model parameters vary over sequence position (hence the sub-index for θ). Figure 1.4 displays such an Inhomogeneous Markov Chain. Since the

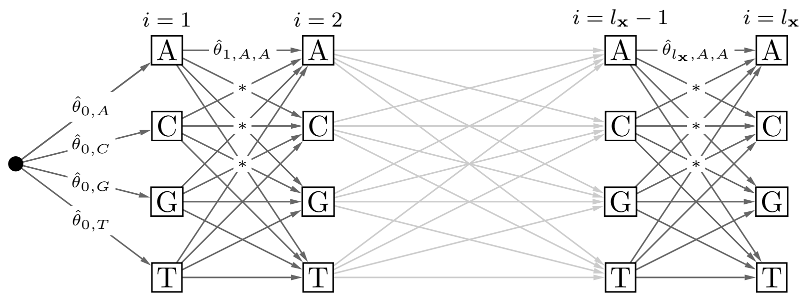


Figure 1.4: A first order Markov chain on DNA. Only $\hat{\theta}_1(A, A)$ is explicitly labelled. $\hat{\theta}_1(C, A), \hat{\theta}_1(G, A)$ and $\hat{\theta}_1(T, A)$ are indicated with stars. As the transition probabilities (model parameters $\hat{\theta}_i$) vary across the positions $i = d, \ldots, l_{\mathbf{x}}$ in the sequence, this MC is inhomogeneous. Higher order MCs can be represented as first order MCs by mapping the states A, C, G, T to the d-th cross-product $\Sigma \mapsto \Sigma \times \Sigma \times \cdots \times \Sigma = \Sigma^d$ (cf. Durbin et al. (1998)).

alphabet Σ of DNA has four letters, each model has $(l_{\mathbf{x}} - d + 1) \cdot 4^{d+1}$ parameters. Their maximum a posteriori (MAP) estimate (for pseudo-count $\pi = 0$ the maximum likelihood estimate is obtained) is given by

$$\begin{aligned}
\hat{\theta}_d(s_1, \ldots, s_d) &= \frac{1}{N + \pi}\Big(\sum_{n=1}^{N} \mathbb{I}(s_1 = x_1^n \wedge \cdots \wedge s_d = x_d^n) + \pi\Big) \\
\hat{\theta}_i(s_i, \ldots, s_{i-d}) &= \frac{\sum_{n=1}^{N} \mathbb{I}(s_i = x_i^n \wedge \cdots \wedge s_{i-d} = x_{i-d}^n) + \pi}{\sum_{n=1}^{N} \mathbb{I}(s_i = x_{i-1}^n \wedge \cdots \wedge s_{i-d} = x_{i-d}^n) + 4\pi},
\end{aligned}$$

where $\mathbb{I}(\cdot)$ is the indicator function, which evaluates to 1 if its argument is true and to 0 otherwise. The variable n enumerates over the number of observed sequences N, and π is the commonly used pseudo-count (a model parameter used to regularize the model towards lower complexity, i.e., uniform solutions, cf. Durbin et al. (1998)). Unfortunately, in the general case it is considered to be a *challenging* problem to estimate the density function of the two classes. When the true distribution is not contained in the set of possible distributions or insufficient data is available, the estimate can be very poor.

Further, descriptive methods need to estimate the *whole* distribution, regardless of its complexity. See for example Figure 1.5, where one can see two distributions separated by a plane. The distribution on the left hand side is rather simple, while the other on the right hand side is more complex. In such cases, discriminative learning techniques, which put special emphasis only on the examples close to the decision boundary may still provide good performance. In addition, the decision boundary might be *simpler*, i.e., it can be parametrised by fewer parameters. While in that case one has to estimate fewer parameters, the choice of the right set of functions F remains crucial. In this work, we will solely focus on learning discriminative methods for genomic signal detection. As we are using *Support Vector Machines* to tackle these signal detection problems, we devote the next subsection to their introduction.

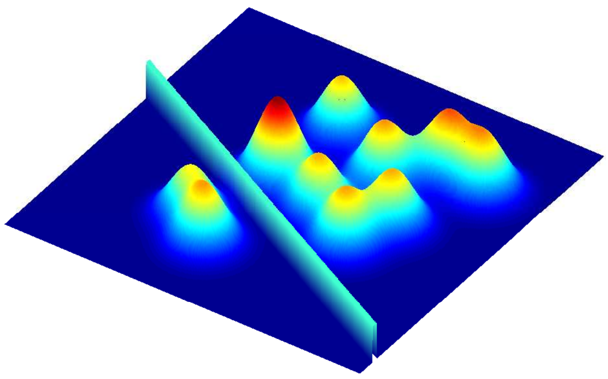


Figure 1.5: Two distributions separated by a plane. While the distribution on the left side is rather simple, the right class has a difficult structure.

1.2.2 Support Vector Machines and String Kernels

A Support Vector Machine (SVM) (Boser et al., 1992, Cortes and Vapnik, 1995, Vapnik, 1995, Schölkopf, 1997, Vapnik, 1998, Schölkopf et al., 1999, Ben-Hur et al., 2008) is a very competitive learning machine, as has been shown on several learning tasks.[7] Suppose we are given N training examples. Each training sample consists of a pair: an input vector $\mathbf{x}_i \in \mathbb{R}^D$ and the output $y_i \in \{-1, +1\}$ that tells us about the true class of the input. Our learning task is to estimate a decision function $f : \mathbf{x} \mapsto \{-1, +1\}$ from the training examples that accurately labels all inputs $\mathbf{x} \in \mathbb{R}^D$, i.e., the function correctly partitions the input space into the two classes $+1$ and -1. The SVM constructs such a decision function from the training examples $(\mathbf{x}_i, y_i)_{i=1}^N$ in the form of a linear separating hyperplane:

$$f(\mathbf{x}) = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b) \tag{1.2}$$

We want to find the classifier $f \in F$, where F is the set of linear classifiers, that minimises the risk functional. Hence, the specific learning task is to find normal weights $\mathbf{w}$ and a bias b that achieve this. Among the set of separating hyperplanes the SVM chooses the one that maximises the margin (for an explanation see Figure 1.6) between the two classes such that each training point lies on the “correct side” (Boser et al., 1992):

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 \text{ for all } i = 1, \dots, N. \tag{1.3}$$

Why is a large margin desirable?

Consider a decision function that achieves only a small margin between the two classes. Then there is little room for error when f is applied to unseen examples

[7]See Isabelle Guyon’s web page `http://www.clopinet.com/isabelle/Projects/SVM/applist.html` on applications of SVMs.

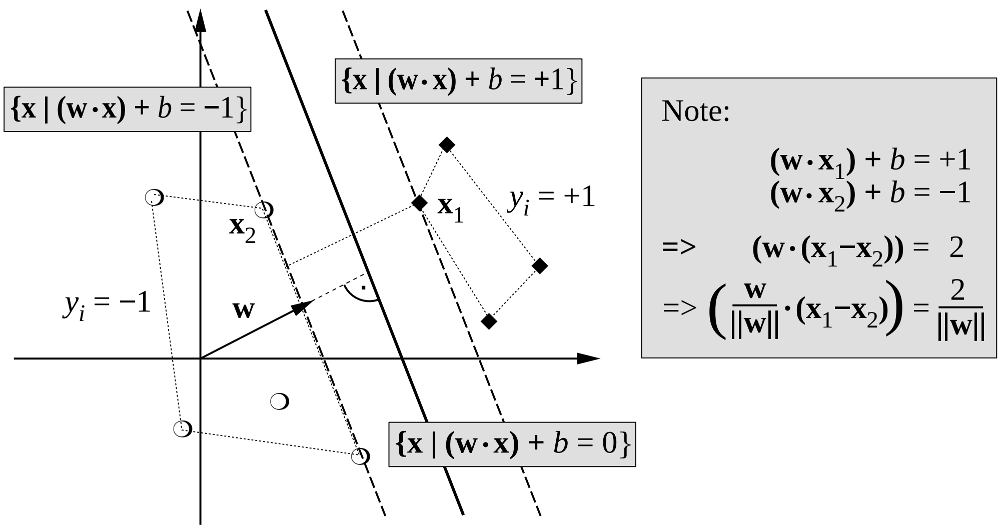


Figure 1.6: Illustration of the margin. On the left hand, you see the objects belonging to the class $y = -1$ which are separated by the linear function $\mathbf{w}\cdot\mathbf{x}+b = 0$ from the objects of the other class $y = +1$ on the right hand side. Within the dashed lines, which correspond to the hyperplanes $H_1 : \{\mathbf{x}|\mathbf{w}\cdot\mathbf{x}+b = -1\}$ and $H_2 : \{\mathbf{x}|\mathbf{w}\cdot\mathbf{x}+b = +1\}$, one finds the *margin* of size $\frac{2}{\|\mathbf{w}\|}$. The data points lying on either of the dashed lines are called support vectors. This figure is taken from (Smola et al., 2000).

$\mathbf{x}$. On the other hand, if a classifier f achieves a large margin, it will be more robust with respect to *patterns and parameters*, i.e., slightly perturbed patterns $\mathbf{x}$ that were previously far away from the separating hyperplane, will still be given the same labels. If one changes the parameters of f, i.e., $\mathbf{w}$ or b slightly, one would similarly expect (Smola et al., 2000) the same labels for examples far away from the decision boundary. For linear classifiers without bias, i.e., $b = 0$, it has been proven (Bartlett and Shawe-Taylor, 1998) that the test error is in principle[8] bounded by the sum of the fraction of training examples lying within a certain margin ρ and a more complex term proportional to $\frac{R}{\rho}$, where the latter decreases with the number of training examples (here R denotes the smallest radius of a sphere that contains all examples $\mathbf{x}$). Thus, the classifier that achieves the maximum margin for a certain number of training examples is expected to give the smallest test error. It is therefore a good idea to choose the classifier with maximum margin among the separating functions.

From Figure 1.6 we see that to maximise the margin, we have to minimise $\frac{1}{2}\|\mathbf{w}\|$ with respect to the constraints (1.3). However, since most practically relevant problems are not separable by a hyperplane, the so-called soft-margin was introduced (Cortes and Vapnik, 1995). To allow for some outliers, slack variables ξ_i, one for each example, are introduced and the problem is reformulated as follows

$$\begin{aligned} \text{minimise} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}\xi_i \\ \text{w.r.t.} \quad & \mathbf{w}\in\mathbb{R}^D, b\in\mathbb{R}, \xi\in\mathbb{R} \\ \text{subject to} \quad & y_i(\mathbf{x}_i\cdot\mathbf{w}+b) \geq 1-\xi_i \\ & \xi_i \geq 0 \qquad \forall i = 1,\ldots,N \end{aligned}$$

[8]The exact inequality is given in (Bartlett and Shawe-Taylor, 1998) but we will leave aside the details and focus on the “message” of this theorem. Note that similar bounds *on the training error* are available for perceptrons since the 1960s (Block, 1962, Novikoff, 1962).

Note that at the optimum $\sum_{i=1}^{N} \xi_i$ is an upper bound on the training error.

Kernel Functions Up to this point, all the SVM does is to construct a separating hyperplane with maximum margin *in the input space.* Since a linear classifier may not be sufficient for more complex problems, one can introduce a mapping $\Phi : \mathbb{R}^D \to \mathcal{F}$, which nonlinearly maps an input vector $\mathbf{x}$ into some higher dimensional feature space $\mathcal{F}$ (Aizerman et al., 1964, Boser et al., 1992).[9] During the training process of the SVM (and in the SVM decision function) only dot-products $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$ are calculated, i.e., $\Phi(\mathbf{x})$ is never used explicitly. We may therefore introduce a kernel function $\mathrm{k}(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$ that replaces the dot product and changes the decision function (Equation (1.2)) to

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}) + b\right).$$

By using a kernel function, one avoids calculating $\Phi(\mathbf{x})$ and instead uses the "kernel trick," i.e., computing the result directly in input space. Using kernels, SVMs are capable of learning a non-linear decision function w.r.t. the input space. However, the separation function in kernel feature space is still linear. Figure 1.7 illustrates the effect of using a kernel. Kernels enable the application of SVMs to domains other

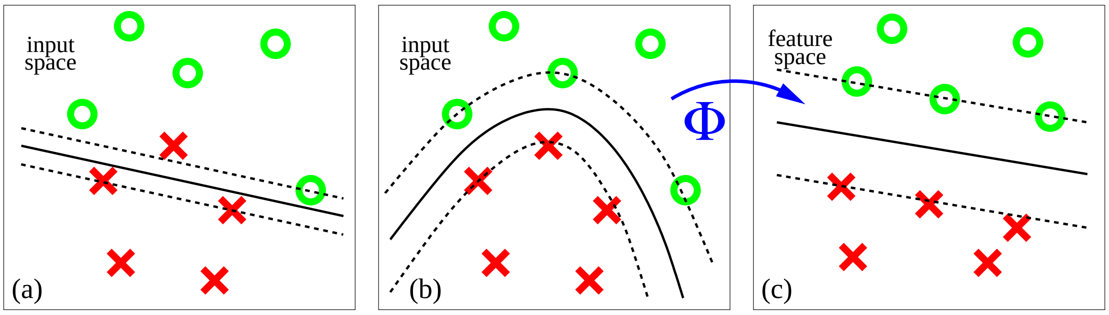


Figure 1.7: (a) A linear separation of training points is not possible without errors. (b) A nonlinear decision function is suggested. (c) The nonlinear function corresponds to a linear function in feature space. This picture is taken from (Zien et al., 2000).

than real-values inputs, like *strings.* It has been shown that all symmetric squared integrable functions that satisfy Mercer's theorem (Mercer, 1909) are valid kernel functions; for details cf. Aizerman et al. (1964), Saitoh (1988), Boser et al. (1992), Cortes and Vapnik (1995), Burges (1998), Müller et al. (2001). A variety of kernel functions are defined in Cortes and Vapnik (1995), Schölkopf (1997), Burges (1998), Jaakkola and Haussler (1999), Watkins (2000).

The choice of a good kernel function is crucial, for instance, an unsuitable kernel applied to the splice site recognition task (e.g., the RBF kernel) performs badly (Rätsch et al., 2001, Braun et al., 2008). Consequently, Chapter 2 focuses on developing appropriate string kernels for genomic signal detection problems.

[9] $\mathcal{F}$ may have infinite dimensions

2 String Kernels

Given two strings $\mathbf{x}$ and $\mathbf{x}'$, there is no single answer to the question: How similar are $\mathbf{x}$ and $\mathbf{x}'$? In contrast to vectors in $\mathbb{R}^d$ where a quantity inverse to $\|\mathbf{x}-\mathbf{x}'\|$ can be used, similarity of strings can be expressed in a variety of ways – each accounting and emphasizing different features and aspects.
Let us start by defining a few terms:

Definition 2.1. *A String (Document or Sequence)* $\mathbf{x}$ *is defined as* $\mathbf{x} \in \Sigma^*$*, where* $\Sigma^* := \cup_{n=0}^{\infty}\Sigma^n$ *denotes the Kleene closure over all symbols from finite alphabet* Σ*. The length of a string is defined as* $l_{\mathbf{x}} := |\mathbf{x}|$*.*

Example 2.2. *An extract of a spam email:*

```
Working on your company's image?  Start with a visual identity
a key to the first good impression.  We are here to help you!

We'll take part in buildinq a positive visual image of
your company by creatinq an outstandinq loqo, presentable
stationery items and professional website.  These marketinq
tools will significantIy contribute to success of your
business.
```

In this case, the alphabet is $\Sigma = \{A,\ldots,Z,a,\ldots,z,0,\ldots,9,!,?,\text{“},'',.,\_\}$*. The letters in curly brackets form the alphabet set, where '_' denotes a blank.*

Example 2.3. *A DNA sequence (*$\Sigma = \{A,C,G,T\}$*):*

```
CAGGAGACGACGACAGGTATGATGATGAGCGACGACATATATGATGAATGTA
```

Using these definitions we are now able to define similarity measures for use with kernel machines on strings, the so-called *string kernels*. In general, there are two major types of string kernels, first the ones that are directly defined on strings, and second kernels that are defined on generative models (like hidden Markov models, e.g. Jaakkola et al., 2000, Tsuda et al., 2002b,c), or by using appropriately defined scores (for instance alignment scores; e.g. Liao and Noble, 2002, Vert et al., 2004). These latter similarity measures are directly working on strings but learnt or, respectively, defined independently beforehand. The following subsections will cover string kernels of the first type Bag-of-words (Salton, 1979, Joachims, 1998), n-gram (Damashek, 1995, Joachims, 1999), Locality Improved (Zien et al., 2000), Subsequence (Lodhi et al., 2002), Spectrum (Leslie et al., 2002), Weighted Degree kernel (Rätsch and Sonnenburg, 2004), Weighted Degree kernel with *shifts* (Rätsch et al., 2005) and also the Fisher (Jaakkola and Haussler, 1999) and TOP kernel (Tsuda et al., 2002b) both of which of the latter type. We will see later (in Section 2.4.3) how the Fisher and TOP kernel are related to the Spectrum and Weighted Degree kernels. For related work that is not directly covered[1] by this work the reader is referred to Haussler (1999), Lodhi et al. (2002), Leslie et al. (2003a), Leslie and Kuang (2004), Schölkopf et al. (2004), Cortes et al. (2004).

[1]Though the `linadd` optimisation trick presented later in this chapter is - in some cases - also applicable.

2.1 Bag-of-words Kernel, n-gram Kernel and Spectrum Kernel

In information retrieval, a classic way to characterise a text document is to represent the text by the unordered set of words it contains – the *bag of words* (Salton, 1979). The text document is split at word boundaries into words using a set of delimiter symbols, such as space, comma and period. Note that in this representation the ordering of words (e.g., in a sentence) will not be taken into account. The feature space $\mathcal{F}$ consists of all possible words and a document $\mathbf{x}$ is mapped to a sparse vector $\Phi(\mathbf{x}) \in \mathcal{F}$, so that $\Phi_i(\mathbf{x}) = 1$ if the word represented by index i is contained in the document. Further alternatives for mapping $\mathbf{x}$ into a feature space $\mathcal{F}$ correspond to associating $\Phi_i(\mathbf{x})$ with frequencies or counts of contained words.

Definition 2.4 (Bag-of-Words-Kernel). *With feature map $\Phi_i(\mathbf{x})$ the bag-of-words kernel is computed as the inner product in $\mathcal{F}$*

$$k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}') = \sum_{i \in words} \Phi_i(\mathbf{x}) \Phi_i(\mathbf{x}'). \tag{2.1}$$

In practice, computing the kernel boils down to counting the number of words common to both documents and can thus be computed very efficiently. Another common but similar approach is to characterise a document by contained n-grams – substrings of n consecutive characters, including word boundaries – where n is fixed beforehand (Suen, 1979, Damashek, 1995). The corresponding feature space $\mathcal{F}$ is spanned by all possible strings of length n. Here no dependencies other than the consecutive n characters are taken into account, which, however, might contain more than one word. The kernel is computed as in Equation (2.1) with the *bag of words* containing all n-grams appearing in a document. Note that the n-gram kernel can – to some extent – cope with mismatches, as for instance a single mismatch only affects n neighboring n-grams, while keeping further surrounding ones intact.
In bioinformatics problems one often does not know any natural word boundary, which is why $n-$gram (in bioinformatics terminology $n-$mer) based kernels are used. Note that the $n-$gram kernel is *position independent* and can be used to compare documents of different length.

Spectrum Kernel The spectrum kernel (Leslie et al., 2002) implements the n-gram or bag-of-words kernel (Joachims, 1998) as originally defined for text classification in the context of biological sequence analysis. The idea is to count how often a d-mer (bioinformatics terminology for d-gram, a contiguous string of length d) is contained in the sequences $\mathbf{x}$ and $\mathbf{x}'$. Summing up the product of these counts for every possible d-mer (note that there are exponentially many) gives rise to the kernel value, which formally is defined as follows:

Definition 2.5 (Spectrum Kernel). *Let Σ be an alphabet and $\boldsymbol{u} \in \Sigma^d$ a d-mer and $\#\boldsymbol{u}(\mathbf{x})$ the number of occurrences of $\boldsymbol{u}$ in $\mathbf{x}$. Subsequently, the spectrum kernel is defined as:*

$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = \sum_{\boldsymbol{u} \in \Sigma^d} \#\boldsymbol{u}(\mathbf{x}) \#\boldsymbol{u}(\mathbf{x}'). \tag{2.2}$$

Spectrum-like kernels cannot extract any positional information from the sequence, which goes beyond the d-mer length. It is well suited for describing the content of a sequence but is less suitable for instance for analyzing signals where motifs may appear in a certain order or at specific positions. However, spectrum-like kernels are capable of dealing with sequences with varying length.

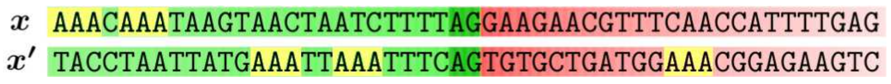

x AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
x' TACCTAATTATGAAATTAAATTTCAGTGTGCTGATGGAAACGGAGAAGTC

Figure 2.1: Given two sequences $\mathbf{x}$ and $\mathbf{x}'$ the spectrum kernel is computed by summing up products of counts of k-mers common to both sequences.

Definition 2.6 (Weighted Spectrum Kernel)**.** *Given non-negative weights β_k, the Weighted Spectrum kernel is defined as*

$$\mathrm{k}_{wspec}(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{d} \beta_k \, \mathrm{k}_k^{spec}(\mathbf{x}, \mathbf{x}').$$

The spectrum kernel can be efficiently computed in $\mathcal{O}(d(l_{\mathbf{x}} + l_{\mathbf{x}'}))$ using tries (Leslie et al., 2002) and $\mathcal{O}(l_{\mathbf{x}} + l_{\mathbf{x}'})$ using suffix trees (Vishwanathan and Smola, 2003), where $l_{\mathbf{x}}$ and $l_{\mathbf{x}'}$ denote the length of sequence $\mathbf{x}$ and $\mathbf{x}'$. An easier and less complex way to compute the kernel for two sequences $\mathbf{x}$ and $\mathbf{x}'$ is to separately extract and sort the $(l_{\mathbf{x}} + l_{\mathbf{x}'})$ d-mers in each sequence, which can be done in a preprocessing step. Then, one iterates over all d-mers of sequences $\mathbf{x}$ and $\mathbf{x}'$ simultaneously, counts which d-mers appear in both sequences and finally sums up the product of their counts. For small alphabets and d-gram lengths individual d-mers can be stored in fixed-size variables, e.g., DNA d-mers of length $d \leq 16$ can be efficiently represented as a 32-bit integer values. The ability to store d-mers in fixed-bit variables or even CPU registers greatly improves performance, as only a single CPU instruction is necessary to compare or index a d-mer. The computational complexity of the kernel computation is $\mathcal{O}(l_{\mathbf{x}} + l_{\mathbf{x}'})$ omitting the preprocessing step. Finally, note that this representation allows efficient computation of the *Weighted Spectrum kernel* in $\mathcal{O}(d(l_{\mathbf{x}} + l_{\mathbf{x}'}))$ without requiring additional storage.

Spectrum Kernel with Mismatches When considering long d-mers, the probability that exactly the same d-mer appears in another sequence drops to zero very fast. Therefore, it can be advantageous (depending on the problem at hand) to consider not only exact matches but also matches with a few mismatching positions.

Definition 2.7 (Spectrum Kernel with Mismatches)**.** *Leslie et al. (2003b) proposed the following kernel:*

$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = \Phi_m(\mathbf{x}) \cdot \Phi_m(\mathbf{x}')$$

where $\Phi_m(\mathbf{x}) = \sum_{\boldsymbol{u} \in \mathbf{x}} \Phi_m(\boldsymbol{u})$, $\Phi_m(\boldsymbol{u}) = (\phi_\sigma(\boldsymbol{u}))_{\sigma \in \Sigma^d}$ *and* $\phi_\sigma(\boldsymbol{u}) = 1$ *if σ mismatches with $\boldsymbol{u}$ in at most m positions and* $\phi_\sigma(\boldsymbol{u}) = 0$ *otherwise.*

This kernel is equivalent to

$$\mathrm{k}_{2m}(\mathbf{x}, \mathbf{x}') = \sum_{\boldsymbol{u} \in \mathbf{x}} \sum_{\boldsymbol{u}' \in \mathbf{x}'} \Delta_{2m}(\boldsymbol{u}, \boldsymbol{u}'), \tag{2.3}$$

where $\Delta_{2m}(\boldsymbol{u}, \boldsymbol{u}') = 1$ if $\boldsymbol{u}$ mismatches $\boldsymbol{u}'$ in at most $2m$ positions. Note that if $m = 1$ then one already considers matches of k-mers that mismatch in *two* positions.[2] Leslie et al. (2003b) proposed a suffix trie based algorithm that computes a single kernel element in $\mathcal{O}(d^{m+1}|\Sigma|^m(|\mathbf{x}| + |\mathbf{x}'|))$.

[2]By using the formulation Equation (2.3) one may of course also consider the case with at most one mismatch (i.e., $m = \frac{1}{2}$). While this kernel is empirically positive definite, it is theoretically not clear whether it always has this property.

2.2 Linear and Polynomial String Kernels

As the standard linear kernel $\mathrm{k}(\mathbf{x}, \mathbf{x}') = \mathbf{x}\cdot\mathbf{x}'$ and polynomial kernel $\mathrm{k}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}\cdot\mathbf{x}'+c)^d$, are only defined for numerical inputs $\mathbf{x}$ and $\mathbf{x}'$ they cannot be directly used on strings. One way to numerically represent a string and thus making it applicable to kernels working on numerical inputs, is to embed it in a binary feature space. Then, a string is represented as a binary feature vector

$$\Phi_b(\mathbf{x}) := \begin{pmatrix} \Phi_b(x_1) \\ \Phi_b(x_2) \\ \vdots \\ \Phi_b(x_D) \end{pmatrix}$$

where $\Phi_b(x_i)$ is a vector of dimensionality $|\Sigma|$ with all entries zero except dimension $\mathcal{D}(x_i)$. Here $\mathcal{D} : \Sigma \mapsto 1 \dots |\Sigma|$ enumerates all possible elements in the alphabet assigning an index. Thus, the resulting feature space has cardinality $|\Sigma|l_{\mathbf{x}}$ and, as each symbol is assigned a new dimension, this is a one-to-one mapping for $c \in \Sigma$. This procedure works also for higher-order mappings, i.e., mapping each $n-$tuple $\Sigma^n \mapsto \Sigma'$ and then applying the same binary feature space transformation while still only $l_{\mathbf{x}}$ components of $\Phi_b(\mathbf{x})$ are non-zero. Using the embedding Φ_b one may finally use kernels defined on numerical inputs, e.g., the linear kernel is now computed as $\mathrm{k}(\mathbf{x}, \mathbf{x}') = \Phi_b(\mathbf{x}) \cdot \Phi(\mathbf{x}')$. Note that the inner product Equation (2.1) using this feature space creates a *position dependent* kernel and that this kernel can be computed as a sum of $l_{\mathbf{x}}$ n-gram kernels each of which working only on a single position of the string $\mathbf{x}$. Finally, note that using a *homogeneous* polynomial kernel of degree n creates a position dependent string kernel that considers *any* not necessarily consecutive substring of length n. This leads to the idea of the Locality Improved kernel (Zien et al., 2000) and is also closely related to the Subsequence kernel (Lodhi et al., 2002). However, the polynomial kernel is computationally much more efficient, as the kernel is of effort $\mathcal{O}(l_{\mathbf{x}})$, but is limited to comparing strings of the same length.

Subsequence Kernel In contrast to the $n-$gram kernel, the subsequence kernel (Lodhi et al., 2002) not only takes into account matches of n *consecutive* characters, but matches of any “subsequence” of n characters of a string that has the same ordering. As now matches may spread over a large region and may include a large number of gaps, the match is penalised by the length of the matching region. Formally, a string $\boldsymbol{u} \in \Sigma^n$ is called subsequence of $\mathbf{x}$ if there exists a sequence $\mathbf{s} = s_1, \dots, s_n$ with $s_i \in \{1, \dots, l_{\mathbf{x}}\}$ and $s_i < s_{i+1}$, where the subsequence is of length $l_{\mathbf{s}} := s_n - s_1 + 1$ and $x_{s_1} \dots x_{s_n} = \boldsymbol{u}$. Furthermore, let $S^{\mathbf{x}}_{\boldsymbol{u}}$ denote the set of all possible subsequences of $\boldsymbol{u}$ in $\mathbf{x}$.

Definition 2.8 (Subsequence Kernel). *The subsequence kernel is defined as the inner product of feature maps* $(\Phi_{\boldsymbol{u}})_{\boldsymbol{u}\in\Sigma^n}$ *with* $\Phi_{\boldsymbol{u}}(\mathbf{x}) := \sum_{\mathbf{s}\in S^{\mathbf{x}}_{\boldsymbol{u}}} \lambda^{l_{\mathbf{s}}}$*, which is*

$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = \sum_{\boldsymbol{u}\in\Sigma^n} \sum_{\mathbf{s}\in S^{\mathbf{x}}_{\boldsymbol{u}}} \sum_{\mathbf{s}'\in S^{\mathbf{x}'}_{\boldsymbol{u}}} \lambda^{l_{\mathbf{s}}+l_{\mathbf{s}'}},$$

where $0 \le \lambda \le 1$ *is a decay term penalizing long subsequences and thus long gaps.*

In Lodhi et al. (2002), a dynamic programming formulation with effort $\mathcal{O}(nl_{\mathbf{x}}l_{\mathbf{x}'})$ was proposed to compute this kernel. Note that this kernel uses a subset of the features of a homogeneous polynomial string kernel, as it considers matches of the same ordering. However, in contrast to the polynomial kernel the subsequence kernel can handle strings of different lengths and does not impose an artificial binomial weighting over inputs.

Locality Improved Kernel The Locality Improved kernel was inspired by the polynomial kernel, putting emphasise on local features. It has successfully been applied to tasks as “Recognition of Translation Initiation Sites” and “Splice Site Recognition”, where *local* information between the observation symbols seems to be highly important (Zien et al., 2000, Sonnenburg et al., 2002).

Definition 2.9. *The Locality Improved kernel (Zien et al., 2000) (LIK) for two sequences* $\mathbf{x}$ *and* $\mathbf{x}'$ *is defined as*

$$\mathrm{k}(\mathbf{x},\mathbf{x}') = \left(\sum_{i=1}^{l_\mathbf{x}} \left(\sum_{j=-l}^{+l} w_j I_{i+j}(\mathbf{x},\mathbf{x}') \right)^{d_1} \right)^{d_2},$$

where

$$I_i(\mathbf{x},\mathbf{x}') = \begin{cases} 1, & x_i = x'_i \\ 0, & \textit{otherwise} \end{cases}$$

Comparing this to the polynomial kernel where $\mathbf{x}$ and $\mathbf{x}'$ are binary vectors

$$\mathrm{k}(\mathbf{x},\mathbf{x}') = (\mathbf{x}\cdot\mathbf{x}')^d = \left(\sum_{i=1}^{l_\mathbf{x}} x_i \cdot x'_i \right)^d = \left(\sum_{i=1}^{l_\mathbf{x}} I_i(\mathbf{x},\mathbf{x}') \right)^d$$

one clearly sees that the LIK is an extension to the polynomial kernel. For two sequences $\mathbf{x}$ and $\mathbf{x}'$, it computes the weighted sum of match functions for subsets of size $2l+1$ pairs of bases in each sequence (shown grey in Figure 2.2), which is illustrated as computing the binary dot product $(.,.)$. Then, these sums are taken to the power of d_1. As mixed term products may occur in the result, even more correlations can be taken into account. Finally, correlations between these windows can be utilised by adding the results from all patches and taken their sum to the power of d_2, which leads to the LIK. While the parameter d_2 is in one to one correspondence to d and can be used to adjust the amount of information gained from relations *between* the inner windows (global information), the parameters l and d_1 can be used to adjust the amount and kind of local information that is to be used, i.e., the relations in each of the *windows* of size $2l+1$. Here l controls the size of the window and d_1 how many terms are taken into account.

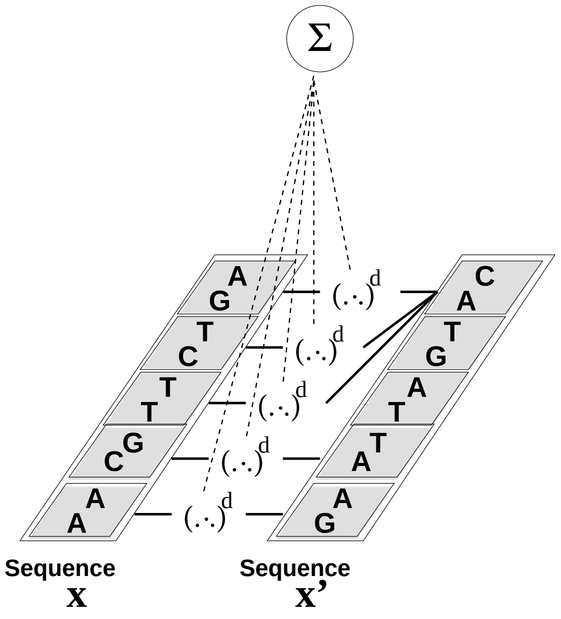


Figure 2.2: The principle of how the Locality Improved kernel works. The sequences $\mathbf{x}$ and $\mathbf{x}'$ consist of only two symbols for simplicity. See text for explanation. This picture is based on (Schölkopf and Smola, 2002).

2.3 Weighted Degree Kernel

The so-called *weighted degree* (WD) kernel (Rätsch and Sonnenburg, 2004) efficiently computes similarities between sequences while taking positional information of k-mers into account. The main idea of the WD kernel is to count the (exact) co-occurrences of k-mers at corresponding positions in the two sequences to be compared.

Definition 2.10 (Weighted Degree kernel)**.** *The* WD kernel *of order d compares two sequences $\mathbf{x}_i$ and $\mathbf{x}_j$ of length $l_{\mathbf{x}}$ by summing all contributions of k-mer matches of lengths $k \in \{1, \dots, d\}$, weighted by coefficients β_k:*

$$\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{d} \beta_k \sum_{l=1}^{l_{\mathbf{x}}-k+1} \mathbb{I}(\boldsymbol{u}_{k,l}(\mathbf{x}_i) = \boldsymbol{u}_{k,l}(\mathbf{x}_j)). \tag{2.4}$$

Here, $\boldsymbol{u}_{k,l}(\mathbf{x})$ is the string of length k starting at position l of the sequence $\mathbf{x}$ and $\mathbb{I}(\cdot)$ is the indicator function that evaluates to 1 *when its argument is* true *and to* 0 *otherwise. For the weighting coefficients, Rätsch and Sonnenburg (2004) proposed to use*

$$\beta_k = 2\frac{d-k+1}{d(d+1)}. \tag{2.5}$$

Matching substrings are thus rewarded with a score depending on the length of the substring. The computational complexity of the WD kernel is $\mathcal{O}(dl_{\mathbf{x}})$ as can be directly seen from Equation (2.4).
Note that although in our case $\beta_{k+1} < \beta_k$, longer matches nevertheless contribute more strongly than shorter ones: this is due to the fact that each long match also implies several short matches, adding to the value of Equation (2.4). Exploiting this knowledge allows for a $\mathcal{O}(l_{\mathbf{x}})$ reformulation of the kernel using "block-weights" as has been done in Sonnenburg et al. (2005b) and will be discussed below.

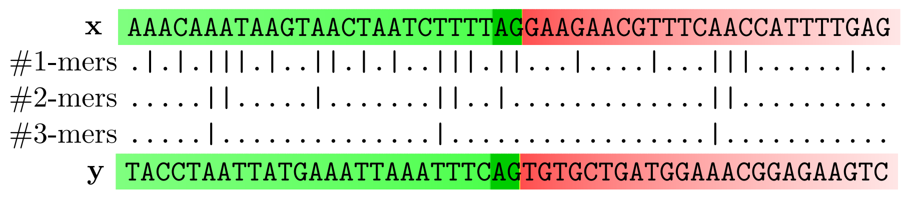

x
AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
#1-mers
#2-mers
#3-mers
y
TACCTAATTATGAAATTAAATTTCAGTGTGCTGATGGAAACGGAGAAGTC

Figure 2.3: Example degree $d = 3 : \mathrm{k}(\mathbf{x}, \mathbf{x}') = \beta_1 \cdot 21 + \beta_2 \cdot 8 + \beta_3 \cdot 3$

In the weighting scheme (Equation (2.5)) higher-order matches seem to get lower weights, which appears counter-intuitive. Note, however, that a k-mer contains two $(k-1)$-mers, three $(k-2)$-mers etc. Hence, a block of length k contains $k-b+1$ blocks of length b. We can make use of this finding and reformulate the kernel. Instead of counting all matches of length $1, 2, \dots, d$ one moves along the sequence only weighting the longest matching block (and not the smaller ones contained within, c.f. Figure 2.4) using different weights $\boldsymbol{w}$, which can be computed from the original weights as follows: For matches of length $B \leq d$ the "block weights" w_B for the weighting (cf. Equation (2.5)) are given by

$$w_B \quad = \quad \sum_{b=1}^{B} m(b)\frac{2(d-b+1)}{d(d+1)} = \sum_{b=1}^{B}(B+1-b)\frac{2(d-b+1)}{d(d+1)} = \frac{B(-B^2+3d\cdot B+3d+1)}{3d(d+1)},$$

where $m(b)$ is the number of times blocks of length b fit within blocks of length B. When the length of the matching block is larger than the maximal degree, i.e., $B > d$, the block weights are given by:

$$w_B = \sum_{b=1}^{B} m(b) \frac{2(d-b+1)}{d(d+1)} = \sum_{i=1}^{d} (k+1-i) \frac{2(d-i+1)}{d(d+1)} = \frac{3B-d+1}{3}$$

To compute the kernel, one determines the longest matches between the sequences $\mathbf{x}$ and $\mathbf{x}'$ and adds up their corresponding weights. This requires only $l_{\mathbf{x}}$ steps reducing

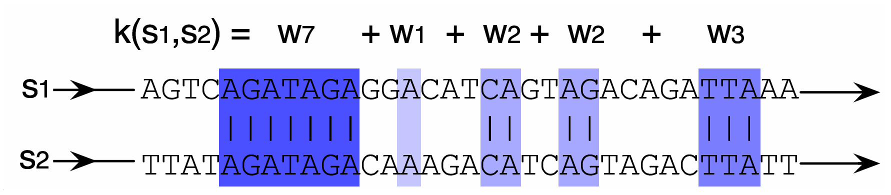

k(S1,S2) = W7 + W1 + W2 + W2 + W3
S1
AGTCAGATAGAGGACATCAGTAGACAGATTAAA
S2
TTATAGATAGACAAAGACATCAGTAGACTTATT

Figure 2.4: Given two sequences $\mathbf{x}_1$ and $\mathbf{x}_2$ of equal length, the kernel consists of a weighted sum to which each match in the sequences makes a contribution w_B depending on its length B, where longer maximal matches contribute more significantly. Figure taken from Sonnenburg et al. (2007a).

the computational complexity to $\mathcal{O}(l_{\mathbf{x}})$. For illustration, Figure 2.5 displays the weighting w_B for different block lengths B at fixed d: longer maximal matching blocks get increased weights; while the first few weights up to $b = d$ increase quadratically, higher-order weights increase only linearly. Note that the WD kernel can be understood as a Spectrum kernel where the k-mers starting at different positions are treated independently of each other. It therefore is very position dependent and does not tolerate any mismatches or positional "shift". Moreover, the WD kernel does not only consider substrings of length exactly d, but also substrings of all shorter matches. Hence, the feature space for each position has $\sum_{k=1}^{d} |\Sigma|^k = \frac{|\Sigma|^{d+1}-1}{|\Sigma|-1} - 1$ dimensions and is additionally duplicated $l_{\mathbf{x}}$ times (leading to $\mathcal{O}(l_{\mathbf{x}}|\Sigma|^d)$ dimensions).

Weighted Degree Kernel with Mismatches In this paragraph we briefly discuss an extension of the WD kernel that considers mismatching k-mers.

Definition 2.11 (Weighted Degree Kernel with Mismatches)**.** *The WD mismatch kernel is defined as*

$$\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{d} \sum_{m=0}^{M} \beta_{k,m} \sum_{l=1}^{l_{\mathbf{x}}-k+1} \mathbb{I}(\boldsymbol{u}_{k,l}(\mathbf{x}_i) \neq_m \boldsymbol{u}_{k,l}(\mathbf{x}_j)),$$

where $\boldsymbol{u} \neq_m \boldsymbol{u}'$ evaluates to true if and only if there are exactly m mismatches between $\boldsymbol{u}$ and $\boldsymbol{u}'$. When considering $\mathrm{k}(\boldsymbol{u}, \boldsymbol{u}')$ as a function of $\boldsymbol{u}'$, then one would wish that full matches are fully counted while mismatching $\boldsymbol{u}'$ sequences should be less influential, in particular for a large number of mismatches. If we choose $\beta_{k,m} = \beta_k / \left(\binom{k}{m} (|\Sigma|-1)^m\right)$ for $k > m$ and zero otherwise, then an m-mismatch gets the full weight divided by the number of possible m-mismatching k-mers.

Note that this kernel can be implemented such that its computation only needs $\mathcal{O}(l_{\mathbf{x}}d)$ operations (instead of $\mathcal{O}(MLd)$).
As the WD mismatch kernel computation remains costly (computationally and memory-

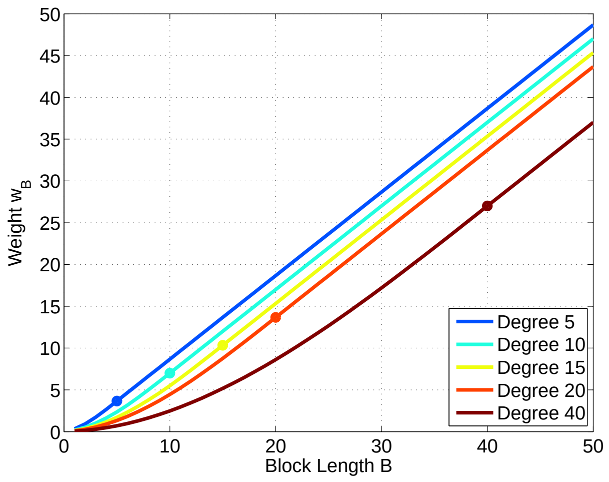


Figure 2.5: How the block weights in the Weighting Degree kernel are chosen. The figure shows the weighting for a sequence length $l_{\mathbf{x}} = 50$ for various degree $d \in \{5, 10, 15, 20, 40\}$. The circle marks the transition from polynomial to linear growth in terms of block weights.

wise), one may ask the question how much the classical WD kernel is affected by mismatches. Let $\mathbf{x}$ be a sequence of length $l_{\mathbf{x}}$. Then considering the block formulation using the standard WD-kernel weighting, a sequence $\mathbf{x}'$ with a mismatch to sequence $\mathbf{x}$ at position p splits a match of size $l_{\mathbf{x}}$ into matches of length $l_{\mathbf{x}1} := p-1$ and $l_{\mathbf{x}2} := l_{\mathbf{x}} - p$. Investigating the relative loss in kernel value $\Delta\,\mathrm{k}_p := \frac{\mathrm{k}(\mathbf{x},\mathbf{x})-\mathrm{k}(\mathbf{x},\mathbf{x}')}{\mathrm{k}(\mathbf{x},\mathbf{x})}$ it becomes clear that there are two cases. A mismatch either splits the full match into long matches $l_{\mathbf{x}1,2} \geq d$ or creates at least one shorter $l_{\mathbf{x}i} \leq d$ match. If both blocks $l_{\mathbf{x}i}$ are large, the mismatch-induced change is a constant $\Delta\,\mathrm{k}_p = \frac{d-1}{3} / \,\mathrm{k}(\mathbf{x}, \mathbf{x})$. If one of the blocks is short, $\Delta\,\mathrm{k}_p$ is a polynomial function w.r.t. p, whereas a mismatch in the middle of the sequence $\mathbf{x}$, at $\frac{l_{\mathbf{x}}-1}{2}$, causes the maximum loss in kernel value.
Figure 2.6 displays several facets of the influence of mismatches: For a fixed sequence length $l_{\mathbf{x}} = 20$, one may note that if the order d of the WD kernel is low, mismatches have almost no effect (constant loss that scales linear with the number of mismatches). When increasing d, the standard WD kernel weighting imposes a prior over the impact of the mismatch position. The prior arguably does what one would naturally expect: The impact is largest around the center of the sequence and decays towards the sequence boundaries. The higher the order the more mismatches will be punished and the less uniform but peaked the prior. Another aspect is the maximum loss occurring for varying sequence lengths $l_{\mathbf{x}}$. One would naturally expect the relative loss to increase inversely proportional to the length of the match. For example, if $l_{\mathbf{x}} = 1$ then the loss is 100% and in long sequences a few mismatches should not cause a strong change in kernel value. This is exactly the case: If $d = 1$ then the loss is proportional to $\Delta\,\mathrm{k}_p = \frac{c_1}{p+c_2} + c_3$ (for c_i =const). For larger d, the mismatch cost is overall higher and at the same time the mismatch costs for matches of similar length become more similar. Finally, one may note that using the block formulation of the kernel, arbitrary positive weights w_B may be selected to adjust mismatch costs to the application in mind. It is however not obvious whether there exists a corresponding weighting β_k in the original WD kernel formulation.[3]

[3]Nevertheless, the block formulation of the WD kernel using any positive weighting is still a valid

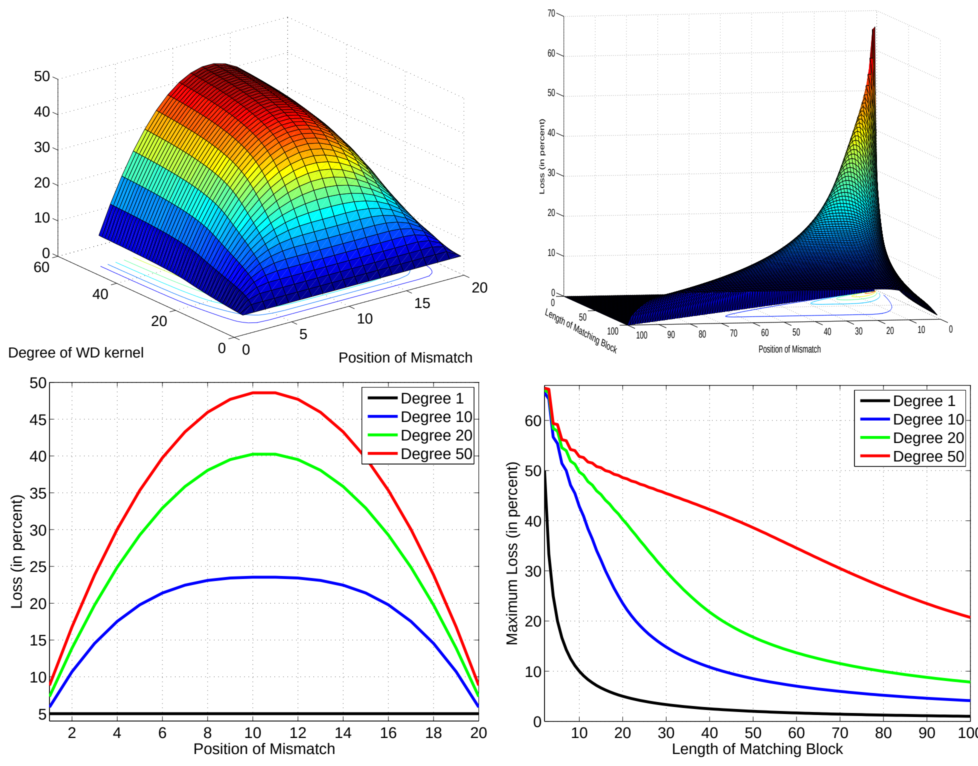


Figure 2.6: Influence of mismatches for the Weighted Degree kernel. Given two strings $\mathbf{x}$ and $\mathbf{x}'$ of same length the relative loss in kernel value is shown for various sequence lengths, different order d and different mismatch locations p. **(top left)** In this figure, the relative loss in kernel value is investigated w.r.t. the order and position of the mismatch for $l_{\mathbf{x}} = 20$. In case the order of the kernel is low, the mismatch has little impact. For larger order, the impact is peaked if the mismatch is located at the central position of $\mathbf{x}'$. **(bottom left)** 2D-representation of the top left figure, highlighting only order 1, 10, 20 and 50. **(top right)** The effect of a mismatch w.r.t. the length $\mathbf{x}$ of the maximal matching block and the position p of the mismatch is investigated for fixed order $d = 20$. The relative loss increases when the length of the matching block is low. Maximum loss occurs if the mismatch happens in the middle of the sequence. **(bottom right)** Depicted is the maximal loss for several orders 1,10,20 and 50 for varying sequence lengths $l_{\mathbf{x}}$. The maximal loss is induced by introduction of a mismatch in the middle of the matching block. Depending on the order of the kernel the relative loss decays with $\frac{1}{p}$ for low order and decays almost linearly for large order.

Weighted Degree Shift Kernel The WD kernel works well for problems where the position of motifs are approximately constant in the sequence or when sufficiently many training examples are available, as it is the case for splice site detection, where the splice factor binding sites appear almost at the same positions relative to the splice site. However, if for instance the sequence is shifted by only 1 nt (cf. Figure 2.4), then potentially existing matches will not be found anymore. We therefore extend the WD kernel to find sequence motifs, which are less precisely localised. Our proposed kernel lies in-between the completely position dependent WD kernel and kernels like the spectrum kernel (Leslie et al., 2002) that does not use positional information. Using β_j as before, let γ_l be a weighting over the position in the sequence, $\delta_s = 1/(2(s+1))$ the weight assigned to shifts (in either direction) of extent s, and let $S(l)$ determine the shift range at position l.[4]

kernel in a potentially much higher dimensional feature space of all k-mers of length $k = 1 \ldots l_{\mathbf{x}}$.

[4]Note that we could have already used γ_l in the position dependent WD kernel described before.

Definition 2.12 (Weighted Degree Shift Kernel)**.** *The kernel with shifts is defined as*

$$\mathrm{k}(\mathbf{x}_i,\mathbf{x}_j) = \sum_{k=1}^{d} \beta_k \sum_{l=1}^{l_{\mathbf{x}}-k+1} \gamma_l \sum_{\substack{s=0 \\ s+l\leq l_{\mathbf{x}}}}^{S(l)} \delta_s \, \mu_{k,l,s,\mathbf{x}_i,\mathbf{x}_j}, \tag{2.6}$$

$$\mu_{k,l,s,\mathbf{x}_i,\mathbf{x}_j} = \mathbb{I}(\boldsymbol{u}_{k,l+s}(\mathbf{x}_i) = \boldsymbol{u}_{k,l}(\mathbf{x}_j)) + \mathbb{I}(\boldsymbol{u}_{k,l}(\mathbf{x}_i) = \boldsymbol{u}_{k,l+s}(\mathbf{x}_j)).$$

In our applications, $S(l)$ is at most 40, hence the computational complexity of the kernel with shifts is only higher by a factor of at most 40.

From a mathematical points of view, it is important to ask the question whether this kernel is positive definite. If not, then the underlying SVM theory may not be applicable and optimisation algorithms may fail. Suppose T is a shift operator, and Φ is the map associated with the zero-shift kernel k. The kernel

$$\tilde{\mathrm{k}}(\mathbf{x},\mathbf{x}') := (\Phi(\mathbf{x}) + \Phi(T\mathbf{x})) \cdot (\Phi(\mathbf{x}') + \Phi(T\mathbf{x}'))$$

is trivially positive definite. On the other hand, we have

$$\begin{aligned}\tilde{\mathrm{k}}(\mathbf{x},\mathbf{x}') &= \Phi(\mathbf{x})\cdot\Phi(\mathbf{x}') + \Phi(T\mathbf{x})\cdot\Phi(T\mathbf{x}') + \Phi(T\mathbf{x})\cdot\Phi(\mathbf{x}') + \Phi(\mathbf{x})\cdot\Phi(T\mathbf{x}') \\ &= \mathrm{k}(\mathbf{x},\mathbf{x}') + \mathrm{k}(T\mathbf{x},T\mathbf{x}') + \mathrm{k}(T\mathbf{x},\mathbf{x}') + \mathrm{k}(\mathbf{x},T\mathbf{x}').\end{aligned}$$

Assuming that we may discard edge effects, $\mathrm{k}(T\mathbf{x},T\mathbf{x}')$ is identical to $\mathrm{k}(\mathbf{x},\mathbf{x}')$; we then know that $2\,\mathrm{k}(\mathbf{x},\mathbf{x}') + \mathrm{k}(T\mathbf{x},\mathbf{x}') + \mathrm{k}(\mathbf{x},T\mathbf{x}')$ is positive definite. Our kernel (Equation (2.6)), however, is a linear combination with positive coefficients of kernels of this type, albeit multiplied with different constants δ_s. The above arguments show that if δ_0 is at least twice as large as the sum of the remaining δ_s, the kernel function will be positive definite. In our experiments detailed below, δ_0 does not in all cases satisfy this condition. Nevertheless, we have always found the kernel matrix to be positive definite on the given training data, i.e., leading to positive definite matrices, and thus posing no difficulties for the SVM optimiser.

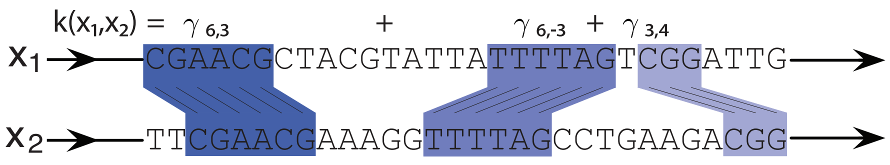

k(x1,x2) = γ 6,3 + γ 6,-3 + γ 3,4
x1 CGAACGCTACGTATTATTTTAGTCGGATTG
x2 TTCGAACGAAAGGTTTTAGCCTGAAGACGG

Figure 2.7: Given two sequences $\mathbf{x}_1$ and $\mathbf{x}_2$ of equal length, the WD kernel with shift consists of a weighted sum to which each maximal match in the sequences makes a contribution $\gamma_{k,p}$ depending on its length k and relative position s, where long matches at the same position contribute most significantly. The γ's can be computed from the β's and δ's in Equation (2.6). The spectrum kernel is based on a similar idea, but it only considers substrings of a fixed length and the contributions are independent of the relative positions of the matches to each other. Figure taken from Sonnenburg et al. (2007a).

Oligo Kernel Meinicke et al. (2004) proposed a so-called "oligo kernel" that is closely related to our extended WD kernel: for each possible k-mer (k fixed in advance), one scans the sequence and generates a numeric sequence whose entries characterise the degree of match between the given k-mer and the sequence (at the corresponding location). To achieve a certain amount of positional invariance, the numeric sequence is convolved with a Gaussian. The convolved sequences are concatenated to give the feature space representation of the original sequence. Meinicke et al. (2004) write down the dot product between two such feature space vectors, and observe that its evaluation does not require summing over all possible k-mers occurring in the two sequences, but only over those that actually appear in *both* sequences. By construction as a dot product in some feature space, their kernel is positive definite.

Definition 2.13 (Oligo Kernel)**.** *The Oligo kernel is defined as*

$$k(\mathbf{x},\mathbf{x}') = \sqrt{\pi}\sigma \sum_{\boldsymbol{u}\in\Sigma^k} \sum_{p\in S_{\boldsymbol{u}}^{\mathbf{x}}} \sum_{q\in S_{\boldsymbol{u}}^{\mathbf{x}'}} e^{-\frac{1}{4\sigma^2}(p-q)^2},$$

where $0 \leq \sigma$ *is a smoothing parameter,* $\boldsymbol{u}$ *is a* $k-$*mer and* $S_{\boldsymbol{u}}^{\mathbf{x}}$ *is the set of positions within sequence* $\mathbf{x}$ *at which* $\boldsymbol{u}$ *occurs as a substring.*

This kernel has the disadvantage that it can be very slow (effort $\mathcal{O}(l_{\mathbf{x}}l_{\mathbf{x}'})$) when k-mers appear often in the sequences (e.g., for short k-mers). Additionally, it only considers k-mers of a fixed length instead of a mixture of k-mers, typically leading to inferior results for large k (which seem necessary in many bioinformatics applications).

2.4 Fisher and TOP kernel

Bioinformatics is currently dominated by graphical models, such as hidden markov models (e.g., Rabiner, 1989) for obvious reasons: Most importantly prior knowledge can be directly integrated into the model, e.g., different states describe different contexts, like the nucleotide distribution in an exon or intron and transitions between the states describe potential context switches (e.g., exons and introns are alternating). Graphical models describe the underlying class distribution. When one is only interested in *discriminating* classes it may be of advantage to only learn the (simpler) discriminating surface, instead of modelling each class distribution. While one would want to use SVMs to learn such discriminating surface, one also wants to make use of the prior knowledge contained in the fine tuned graphical models. The Fisher (Jaakkola and Haussler, 1999, Jaakkola et al., 2000) and TOP (Tsuda et al., 2002a,b) kernel were invented to achieve this goal. Both of which are very universal as they can be defined on arbitrary graphical models, which is why Fisher or TOP kernels derived from specific graphical model closely resemble some of the kernels mentioned above: The Spectrum, Locality Improved and Subsequence kernel. It even motivated the Weighted Degree kernels.

2.4.1 Fisher Kernel

In contrast to other kernels, the Fisher kernel (FK) and TOP kernel, are derived from a probabilistic model. Used in conjunction with SVMs, they are known to increase classification performance w.r.t. the probabilistic model if the true distribution can be parametrised by the probabilistic model.
The Fisher kernel (Jaakkola and Haussler, 1999, Jaakkola et al., 2000) is defined on some generative model $\Pr[\mathbf{x}|\boldsymbol{\theta}]$ (like HMMs). It is derived from the marginal distribution

$$\Pr[\mathbf{x}|\boldsymbol{\theta}] = \Pr[\mathbf{x}|\boldsymbol{\theta}, y=+1]\Pr[y=+1] + \Pr[\mathbf{x}|\boldsymbol{\theta}, y=-1]\Pr[y=-1] :$$

Definition 2.14. *The Fisher kernel is defined as*

$$\mathrm{k}(\mathbf{x},\mathbf{x}') = \mathbf{s}_{FK}(\mathbf{x},\boldsymbol{\theta})^\top \mathbf{Z}^{-1}(\boldsymbol{\theta})\mathbf{s}_{FK}(\mathbf{x}',\boldsymbol{\theta}),$$

where $\mathbf{s}_{FK}$ *is the Fisher score*

$$\mathbf{s}_{FK}(\mathbf{x},\boldsymbol{\theta}) = \left(\partial_{\theta_1}\log p(\mathbf{x}|\boldsymbol{\theta}),\ldots,\partial_{\theta_p}\log p(\mathbf{x}|\boldsymbol{\theta})\right)^\top = \nabla_{\boldsymbol{\theta}}\log p(\mathbf{x},\boldsymbol{\theta}),$$

and $\mathbf{Z}$ *is the Fisher information matrix* $\mathbf{Z}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}}\left[\mathbf{s}_{FK}(\mathbf{x},\boldsymbol{\theta})\mathbf{s}_{FK}(\mathbf{x},\boldsymbol{\theta})^\top \,\middle|\, \boldsymbol{\theta}\right]$ *, which is sometimes omitted in practice, i.e.,* $\mathbf{Z} = \mathbf{1}_N$*, or approximated as* $Z_{ij} = \delta_{ij}\sigma_i^2$*, where* σ_i *is the variance in dimension i Schölkopf and Smola (2002).*

The Fisher kernel uses a local metric, defined on the generative model, to compare two sequences $\mathbf{x}$ and $\mathbf{x}'$. As an advantage over the LIK where the sequences have to have the same length (since it counts positional matches), the FK can – depending on the underlying probabilistic model – deal with sequences of arbitrary length, making it the kernel of choice in many applications like speech recognition and DNA analysis.
The Fisher kernel was originally only defined based on the generative model describing the positive class $\Pr[\mathbf{x}|\theta^+]$ (Jaakkola et al., 2000). To explicitly make use of discriminative information, the TOP kernel (Tsuda et al., 2002a,b) was derived from the tangent vectors of posterior log-odds and it was shown to outperform the FK on certain tasks.

2.4.2 TOP Kernel

Let us consider that we have trained two generative models, one for positive samples $\Pr[\mathbf{x}|\boldsymbol{\theta}_+]$ and one for the negative class $\Pr[\mathbf{x}|\boldsymbol{\theta}_-]$.[5]
Recall the Bayes decision for the optimal model $\Pr[\mathbf{x},y|\boldsymbol{\theta}^\star]$:

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sign}\left(\Pr[y=+1|\mathbf{x},\boldsymbol{\theta}^\star] - \Pr[y=-1|\mathbf{x},\boldsymbol{\theta}^\star]\right) \\ &= \operatorname{sign}\left(\frac{1}{\Pr[\mathbf{x}|\boldsymbol{\theta}^\star]}\cdot\left(\alpha\Pr[\mathbf{x}|y=+1,\boldsymbol{\theta}^\star] - (1-\alpha)\Pr[\mathbf{x}|y=-1,\boldsymbol{\theta}^\star]\right)\right) \\ &= \operatorname{sign}\left(\alpha\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_+] - (1-\alpha)\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_-]\right) \end{aligned}$$

Here α is the prior for the positive class conditional distribution, i.e.,
$\alpha = \Pr[y=+1|\boldsymbol{\theta}^\star_+]$. When not taking the difference but the quotient of the posterior probabilities this decision function remains equivalent (except for the pathological case where the negative distribution is zero):

$$f(\mathbf{x}) = \begin{cases} -1, & \frac{\alpha\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_+]}{(1-\alpha)\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_-]} < 1 \\ +1, & \frac{\alpha\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_+]}{(1-\alpha)\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_-]} > 1 \end{cases}.$$

Taking the logarithm of this quotient (we need to assume that both distributions are nonzero) leads to the still equivalent formulation of the posterior log odds

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sign}\left(\log\left(\frac{\alpha\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_+]}{(1-\alpha)\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_-]}\right)\right) \\ &= \operatorname{sign}\left(\log(\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_+]) - \log(\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_-]) + \log\left(\frac{\alpha}{1-\alpha}\right)\right) \\ &= \operatorname{sign}\left(\log(\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_+]) - \log(\Pr[\mathbf{x}|\boldsymbol{\theta}^\star_-]) + b\right), \end{aligned}$$

where b is the bias term.

[5]This is a slightly different notation, which is introduced to underline two separately trained generative models. We usually use $\boldsymbol{\theta} = (\boldsymbol{\theta}_+,\boldsymbol{\theta}_-,\alpha)$ and $\Pr[\mathbf{x}|y=c,\boldsymbol{\theta}] = \Pr[\mathbf{x}|\boldsymbol{\theta}_c]$, where $c \in \{+1,-1\}$.

We are interested in the decision boundary $v(\mathbf{x}, \boldsymbol{\theta}^\star) = 0$, where

$$v(\mathbf{x}, \boldsymbol{\theta}) := \log(\Pr[\mathbf{x}|\boldsymbol{\theta}_+]) - \log(\Pr[\mathbf{x}|\boldsymbol{\theta}_-]) + b.$$

However, $\boldsymbol{\theta}^\star$ is not known. We use Taylor approximation to estimate the decision boundary

$$\begin{aligned} v(\mathbf{x}, \boldsymbol{\theta}^\star) &\approx v(\mathbf{x}, \widehat{\boldsymbol{\theta}}) + \sum_{i=1}^{p} \partial_{\theta_i} v(\mathbf{x}, \widehat{\boldsymbol{\theta}})(\theta_i^\star - \hat{\theta}_i) \\ &= \mathbf{s}_{TOP}(\widehat{\boldsymbol{\theta}}, \mathbf{x}) \cdot \mathbf{w} \end{aligned}$$

where

$$\begin{aligned} \mathbf{s}_{TOP}(\widehat{\boldsymbol{\theta}}, \mathbf{x}) &:= (v(\mathbf{x}, \widehat{\boldsymbol{\theta}}), \partial_{\theta_1} v(\mathbf{x}, \widehat{\boldsymbol{\theta}}), \ldots, \partial_{\theta_p} v(\mathbf{x}, \widehat{\boldsymbol{\theta}}))) \\ \mathbf{w} &:= (1, \boldsymbol{\theta}_1^\star - \hat{\theta}_1, \cdots, \boldsymbol{\theta}_p^\star - \hat{\theta}_p)^\top, \end{aligned}$$

and $\widehat{\boldsymbol{\theta}}$ is an estimate for $\boldsymbol{\theta}^\star$ obtained, e.g., by maximum likelihood learning. While we can compute $\mathbf{s}_{TOP}(\widehat{\boldsymbol{\theta}}, \mathbf{x})$, we still find the unknown model parameters $\boldsymbol{\theta}^\star$ in $\mathbf{w}$. We can use, e.g., SVMs (or any other linear estimator) to estimate $\mathbf{w}$.

Definition 2.15. *Since the* Tangent vector Of the Posterior log-odds *(TOP) constitutes the main part of the feature vector* $\mathbf{s}_{TOP}(\widehat{\boldsymbol{\theta}}, \mathbf{x})$ *the inner product*

$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = \mathbf{s}_{TOP}(\widehat{\boldsymbol{\theta}}, \mathbf{x})^\top \mathbf{s}_{TOP}(\widehat{\boldsymbol{\theta}}, \mathbf{x}')$$

is called TOP-Kernel *(Tsuda et al., 2002a).*

The TOP kernel, while applicable to the same type of problems, empirically outperforms the FK as observed in (Smith and Gales, 2002, Tsuda et al., 2002a). Furthermore, it was shown that the convergence rate when combined with an *optimal* linear classifier (which is not available in practice) is $\mathcal{O}(N^{-1})$. This shows the existence of a very efficient linear boundary in the TOP feature space.

2.4.3 Relation to Spectrum and Weighted Degree Kernel

The Fisher kernel framework is very generic, as it can be defined on any graphical model. It may therefore not be too surprising that many (if not all) of the above string kernels can be derived using the FK framework and a specific probabilistic model. In the following paragraphs, we derive probabilistic models for the Spectrum and WD kernel for which the corresponding FK for a certain setting of model parameters can be shown to be equivalent.

Probabilistic Model for the Spectrum Kernel In this section we define a rather simple single state model closely resembling a higher-order Markov Chain and show that the Fisher kernel derived from this graphical model leads to the Spectrum kernel when a uniform distribution over parameters is assumed. Let us consider a single state probabilistic model, with $m := |\Sigma|^d$ emission parameters (one for each possible $d-$mer $\boldsymbol{u} \in \Sigma^d$) and likelihood

$$\begin{aligned}\Pr[\mathbf{x}|\boldsymbol{\theta}] &:= \prod_{i=1}^{l_{\mathbf{x}}-d+1} \Pr[x_{i,\ldots,i+d-1}|\boldsymbol{\theta}] \\ &:= \prod_{i=1}^{l_{\mathbf{x}}-d+1} \theta_{x_{i,\ldots,i+d-1}}. \end{aligned} \tag{2.7}$$

Note that a different approximation to the full joint probability $\Pr[\mathbf{x}|\boldsymbol{\theta}]$ is used: Instead of the normally in Markov Chains of order $d-1$ usually used conditional likelihood $\Pr[x_{i+d-1}|x_{i+d-2,\ldots,i}, \boldsymbol{\theta}]$ the joint probability $\Pr[x_{i,\ldots,i+d-1}|\boldsymbol{\theta}]$ was chosen to represent $\Pr[\mathbf{x}|\boldsymbol{\theta}]$.[6] Figure 2.8 sketches the resulting model.

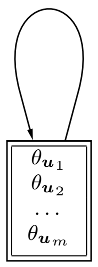

$\theta_{\boldsymbol{u}_1}$
$\theta_{\boldsymbol{u}_2}$
...
$\theta_{\boldsymbol{u}_m}$

Figure 2.8: Single state Hidden Markov like model with $m := |\Sigma|^d$ emissions (one for each possible $d-$mer $\boldsymbol{u} \in \Sigma^d$.

Using the definition Equation (2.7) we may now derive the Fisher kernel scores $\mathbf{s}_{FK} = \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}, \boldsymbol{\theta})$.

$$\begin{aligned}\partial_{\theta_{\boldsymbol{u}}} \log(\Pr[\mathbf{x}|\boldsymbol{\theta}]) &= \partial_{\theta_{\boldsymbol{u}}} \log\left(\prod_{i=1}^{l_{\mathbf{x}}-d+1} \theta_{x_{i,\ldots,i+d-1}}\right) \\ &= \partial_{\theta_{\boldsymbol{u}}} \sum_{i=1}^{l_{\mathbf{x}}-d+1} \log\left(\theta_{x_{i,\ldots,i+d-1}}\right) \\ &= \partial_{\theta_{\boldsymbol{u}}} \#(\boldsymbol{u} \in \mathbf{x}) \cdot \log \theta_u \\ &= \#(\boldsymbol{u} \in \mathbf{x}) \cdot \frac{1}{\theta_u} \end{aligned} \tag{2.8}$$

Here $\#(\boldsymbol{u} \in \mathbf{x})$ denotes the number of occurrences of the d-mer $\boldsymbol{u}$ in $\mathbf{x}$. If all model parameters were uniform, i.e., $\forall \boldsymbol{u} : \theta_{\boldsymbol{u}} = 1$ the inner product of fisher scores as computed in Equation (2.8) fully recovers the spectrum kernel Equation (2.2). Note that instead of the Fisher information matrix, the identity matrix $\mathbf{Z} = 1_m$ was used. Also to be exact $\forall \boldsymbol{u} : \theta_{\boldsymbol{u}} = c$ with $c = 1$ violates $\sum_{\boldsymbol{u} \in \Sigma^d} \theta_{\boldsymbol{u}} = 1$. A $c \neq 1$ introduced factor though vanishes when using standard kernel normalisation $\mathrm{k}(\mathbf{x}, \mathbf{x}') = \frac{\mathrm{k}'(\mathbf{x},\mathbf{x}')}{\mathrm{k}'(\mathbf{x},\mathbf{x})\,\mathrm{k}'(\mathbf{x}',\mathbf{x}')}$.

Probabilistic Model for the WD Kernel Similarly, one may define a probabilistic model for the WD kernel. Again a parameter for each possible emission is introduced. In contrast to the spectrum kernel, the emission parameter is position-dependent and order $1 \ldots d$ are simultaneously used. Let $\theta_{k,i\boldsymbol{u}}$ be the model parameter for a $k-$mer $\boldsymbol{u}$

[6]Mathematically, this model might not be a valid probability distribution.

starting at position i. We then define

$$\begin{aligned}
\Pr[\mathbf{x}|\boldsymbol{\theta}] &:= \prod_{k=1}^{d} \prod_{i=1}^{l_{\mathbf{x}}-k+1} \Pr[x_{i,\dots,i+k-1}|\boldsymbol{\theta}] \\
&:= \prod_{k=1}^{d} \prod_{i=1}^{l_{\mathbf{x}}-k+1} \theta_{i,\, x_{i,\dots,i+d-1}}. \qquad (2.9)
\end{aligned}$$

as the WD kernel associated probabilistic model.[7]
Figure 2.9 shows a Hidden Markov like probabilistic model for a fixed order k only. Again computing derivatives of the log likelihood we obtain the following Fisher scores

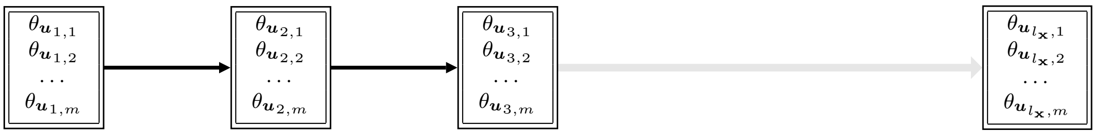

$\theta_{\boldsymbol{u}_{1,1}}$
$\theta_{\boldsymbol{u}_{1,2}}$
$\dots$
$\theta_{\boldsymbol{u}_{1,m}}$
$\theta_{\boldsymbol{u}_{2,1}}$
$\theta_{\boldsymbol{u}_{2,2}}$
$\dots$
$\theta_{\boldsymbol{u}_{2,m}}$
$\theta_{\boldsymbol{u}_{3,1}}$
$\theta_{\boldsymbol{u}_{3,2}}$
$\dots$
$\theta_{\boldsymbol{u}_{3,m}}$
$\theta_{\boldsymbol{u}_{l_{\mathbf{x}},1}}$
$\theta_{\boldsymbol{u}_{l_{\mathbf{x}},2}}$
$\dots$
$\theta_{\boldsymbol{u}_{l_{\mathbf{x}},m}}$

Figure 2.9: Hidden Markov like model M_k with $m := l_{\mathbf{x}}|\Sigma|^k$ emission parameters (one for each possible $k-$mer $\boldsymbol{u} \in \Sigma^k$ and position p. Deriving the Fisher kernel of model M_k, recovers a single order k of the WD kernel. Finally, deriving the Fisher kernel from the product model $M := \prod_{k=1}^{d} M_k$ leads to the WD kernel (using a certain parameter setting $\theta_{p,\boldsymbol{u}}$).

$$\begin{aligned}
\partial \boldsymbol{u}_p \log \Pr[\mathbf{x}|\boldsymbol{\theta}] &= \partial \boldsymbol{u}_p \log \prod_{k=1}^{d} \prod_{i=1}^{l_{\mathbf{x}}-k+1} \\
&= \partial \boldsymbol{u}_p \sum_{k=1}^{d} \sum_{i=1}^{l_{\mathbf{x}}-k+1} \log(\theta_{k,\, x_{i,\dots,i+d-1}}) \\
&= \frac{1}{\theta_{|u_p|,u_p}}. \qquad (2.10)
\end{aligned}$$

Here $\boldsymbol{u}_p$ is a substring starting a position p of length $|\boldsymbol{u}_p|$. The quantity in Equation (2.10) is zero if $\boldsymbol{u}_p \neq x_{p,\dots,p+|\boldsymbol{u}_p|-1}$. Finally, using a constant weighting for each $k-$mer length $|\boldsymbol{u}_p| = k$, i.e., for all $k = 1 \dots d$:

$$\theta_{k,u_p} := \sqrt{\frac{1}{\beta_k}},\ \forall \boldsymbol{u}_p \in \Sigma^k$$

one recovers the weighted degree kernel Equation (2.4).

2.5 Summary

The string kernels revisited in this section all differ in computational cost and the complexity of their inherent feature space. Some of their properties are summarised in Table 2.1. While some kernels like the polynomial, locality improved and subsequence string kernel impose an extremely complex feature space that may consider all possible (gappy) subsequences of a maximum length others like the n-gram (spectrum) kernel and weighted degree kernel are tuned to only count matching n-grams. In practice, a too rich feature space may contain orders of magnitude more nuisance dimensions and thus may make the problem unnecessarily hard even for relatively robust SVM

[7]For this analysis, we ignore that this model might not be a valid probability distribution.

Kernel	$l_{\mathbf{x}} \neq l_{\mathbf{x}'}$	Requires Generative Model	Uses positional information	Local or Global?	Computational Complexity
linear	no	no	yes	local	$\mathcal{O}(l_{\mathbf{x}})$
polynomial	no	no	yes	global	$\mathcal{O}(l_{\mathbf{x}})$
locality improved	no	no	yes	local/global	$\mathcal{O}(l \cdot l_{\mathbf{x}})$
sub-sequence	yes	no	yes	global	$\mathcal{O}(n l_{\mathbf{x}} l_{\mathbf{x}'})$
n-gram/Spectrum	yes	no	no	global	$\mathcal{O}(l_{\mathbf{x}})$
WD	no	no	yes	local	$\mathcal{O}(l_{\mathbf{x}})$
WD *with shifts*	no	no	yes	local/global	$\mathcal{O}(s \cdot l_{\mathbf{x}})$
Oligo	yes	no	yes	local/global	$\mathcal{O}(l_{\mathbf{x}} l_{\mathbf{x}'})$
TOP	yes/no	yes	yes/no	local/global	depends
Fisher	yes/no	yes	yes/no	local/global	depends

Table 2.1: This table summarises different properties of the discussed string kernels, like whether the kernel is capable of dealing with sequences of different length ($l_{\mathbf{x}} \neq l_{\mathbf{x}'}$), whether the kernel requires a probabilistic model from which it will be derived, whether it uses positional information (i.e., order of n-gram matters), whether it uses local or global information (i.e., information only from neighboring characters or globally using the whole string) and finally the kernels computational complexity..

classifiers. An additional downside of highly complex kernels is the often also very high computational complexity, which may drastically limit the number of applications. Overall, there is a trade-off between expressive complexity (e.g., a string kernel explicitly modelling mismatches), the number of data points it can handle (computational complexity) and the number of data points available and needed in a specific application: For example, if enough data points are available, mismatches could in principle be learnt by a simpler kernel when all variants of the mismatch are found in training. It is therefore crucial to fine tune the string kernel to the particular application in mind. In genomic sequence analysis, the relatively simple spectrum and WD kernels perform favourably. Furthermore, their computational complexity is linear in the length of the input sequences, although there are some significant differences: the spectrum kernel requires $\mathcal{O}(l_{\mathbf{x}} + l_{\mathbf{x}'})$, the WD kernel $\mathcal{O}(l)$ and the WDS kernel is most demanding with $\mathcal{O}(lS)$. However, even linear time string kernels are computationally intractable when working with genomic size datasets (size $\gg 10^6$). As there is no obvious way to further speed up single kernel computations, one can try to exploit the inherent structure of learning algorithms using string kernels. One can particularly benefit from the fact that these algorithms often need to compute linear combinations of kernel elements during training and testing, which can be significantly sped up by exploiting the sparsity of the representation in feature space. In Chapter 3, we will discuss methods to represent sparse feature maps and give algorithmic details on how one can make use of the sparse feature space to accelerate evaluations of linear combinations of string kernels, which will tremendously speed up SVM training and testing.

3 Large Scale Learning with String Kernels

In applications of bioinformatics and text processing, such as splice site recognition and spam detection, large amounts of training sequences are available and needed to achieve sufficiently high prediction performance on classification or regression tasks. Although kernel-based methods such as SVMs often achieve state-of-the-art results, training and evaluation times may be prohibitively large. When single kernel computation time is already linear (w.r.t. the input sequences), it seems difficult to achieve further speed ups.

In this section, we describe an efficient technique for computing *linear combinations of string kernels* using sparse data structures such as *explicit maps*, *sorted arrays* and *suffix tries, trees or arrays* (Sonnenburg et al., 2007a). As computing linear combinations of kernels make up the dominant part of SVM training and evaluation, speeding up their computation is essential. We show that the recently proposed and successfully used *linear time* string kernels, like the *Spectrum kernel* (Leslie et al., 2002) and the *Weighted Degree kernel* (Rätsch and Sonnenburg, 2004) can be accelerated in SVM training by factors of 7 and 60 times, respectively, while requiring considerably less memory. Our method allows us to train *string kernel* SVMs on sets as large as 10 million sequences (Sonnenburg et al., 2007b). Moreover, using these techniques the evaluation on new sequences is often several thousand times faster, allowing us to apply the classifiers on genome-sized data sets with seven billion test examples (Sonnenburg et al., 2006b). This chapter is largely based on Sonnenburg et al. (2007a).

3.1 Sparse Feature Maps

The string kernels introduced in the previous section share two important properties: (a) the mapping Φ is explicit, so that elements in the feature space $\mathcal{F}$ can be accessed directly, and (b) mapped examples $\Phi(\mathbf{x})$ are very sparse in comparison to the huge dimensionality of $\mathcal{F}$. In the following sections, we illustrate how these properties can be exploited to efficiently store and compute sparse feature vectors.

3.1.1 Efficient Storage of Sparse Weights

The considered string kernels correspond to very large feature spaces, for instance DNA d-mers of order ten span a feature space of over 1 million dimensions. However, most dimensions in the feature space are always zero since only a few of the many different d-mers actually appear in the training sequences, and furthermore a sequence $\mathbf{x}$ can only comprise at most $l_{\mathbf{x}}$ unique d-mers.

In this section, we briefly discuss four efficient *data structures* for sparse representation of sequences supporting the basic operations: `clear`, `add` and `lookup`. We assume that the elements of a sparse vector $\boldsymbol{v}$ are indexed by some index set $\mathcal{U}$ (for sequences, e.g., $\mathcal{U} = \Sigma^d$). The first operation `clear` sets $\boldsymbol{v}$ to zero. The `add` operation increases either the weight of a dimension of $\boldsymbol{v}$ for an element $\boldsymbol{u} \in \mathcal{U}$ by some amount or increases a set of weights in $\boldsymbol{v}$ corresponding to all d-mers present in a given sequence $\mathbf{x}$. Similar to `add`, the `lookup` operation either requests the value of a particular component $v_{\boldsymbol{u}}$ of $\boldsymbol{v}$ or returns a set of values matching all d-mers in a provided sequence $\mathbf{x}$. The latter two operations need to be performed as quickly as possible.

Explicit Map If the dimensionality of the feature space is small enough, then one might consider keeping the whole vector $\boldsymbol{v}$ in memory and to perform direct operations on its elements. In this case, each `add` or `lookup` operation on single elements takes $\mathcal{O}(1)$ time.[1] The approach, however, has expensive memory requirements ($\mathcal{O}(|\Sigma|^d)$), but is highly efficient and best suited for instance for the spectrum kernel on DNA sequences with $d \leq 14$ and on protein sequences with $d \leq 6$.

Sorted Arrays and Hash Tables More memory efficient but computationally more expensive are sorted arrays of index-value pairs $(\boldsymbol{u}, v_{\boldsymbol{u}})$. Assuming $l_{\mathbf{x}}$ indices of a sequence $\mathbf{x}$ are given and sorted in advance, one can efficiently `add` or `lookup` a single $v_{\boldsymbol{u}}$ for a corresponding $\boldsymbol{u}$ by employing a binary search procedure with $\mathcal{O}(\log l_{\mathbf{x}})$ run-time. Given a sequence $\mathbf{x}'$ to look up all contained d-mers at once, one may sort the d-mers of $\mathbf{x}'$ in advance and then simultaneously traverse the arrays of $\mathbf{x}$ and $\mathbf{x}'$ to determine which elements appear in $\mathbf{x}'$. This procedure results in $\mathcal{O}(l_{\mathbf{x}} + l_{\mathbf{x}'})$ operations – omitting the sorting of the second array – instead of $\mathcal{O}(l_{\mathbf{x}'} \log l_{\mathbf{x}})$. The approach is well suited for cases where $l_{\mathbf{x}}$ and $l_{\mathbf{x}'}$ are of comparable size, as for instance for computations of single spectrum kernel elements (Leslie et al., 2003b). If $l_{\mathbf{x}} \gg l_{\mathbf{x}'}$, then the binary search procedure should be preferred.

A trade-off between the efficiency of explicit maps and the low memory requirements of sorted arrays can be achieved by storing the index-value pairs $(\boldsymbol{u}, v_{\boldsymbol{u}})$ in a hash table, where $\boldsymbol{u}$ is hashed to a bin in the hash table containing $v_{\boldsymbol{u}}$ (Rieck et al., 2006). Both operations `add` and `lookup` for $\boldsymbol{u} \in \Sigma^d$ can be carried out in $\mathcal{O}(1)$ time in best-case – however the worst-case run-time is $\mathcal{O}(\log l_{\mathbf{x}})$, if all $\boldsymbol{u}$ of a sequence $\mathbf{x}$ are mapped to the same bin. The two opposed runtime bounds suggest that the hash table size has to be chosen very carefully in advance and also strongly depends on the lengths of the considered sequences, which makes the sorted array approach more practicable in terms of run-time requirements.

Tries Another way of organizing the non-zero elements are *tries* (Fredkin, 1960, Knuth, 1973): The idea is to use a tree with at most $|\Sigma|$ siblings of depth d. The leaves store a single value: the element $v_{\boldsymbol{u}}$, where $\boldsymbol{u} \in \Sigma^d$ is a d-mer and the path to the leaf corresponds to $\boldsymbol{u}$. To `add` an element to a trie, one needs $\mathcal{O}(d)$ in order to create the necessary nodes on the way from the root to a leave. Similar to the `add` operation, a `lookup` takes $\mathcal{O}(d)$ time in the worst-case, however, with growing d the probability for an arbitrary $\boldsymbol{u}$ to be present in a trie decreases exponentially, so that a logarithmic run-time of $\mathcal{O}(\log_{|\Sigma|} d)$ can be expected for large d. Note that the worst-case computational complexity of both operations is independent of the number of d-mers/elements stored in the tree.

Tries need considerably more storage than sorted arrays (for instance storing edges in nodes usually requires using hash tables or balanced tree maps), however, tries are useful for the previously discussed WD kernel. Here we not only have to lookup one substring

[1]More precisely, it is $\log d$, but for small enough d (which we have to assume anyway) the computational effort is exactly one memory access.

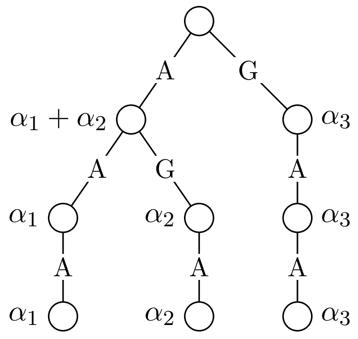
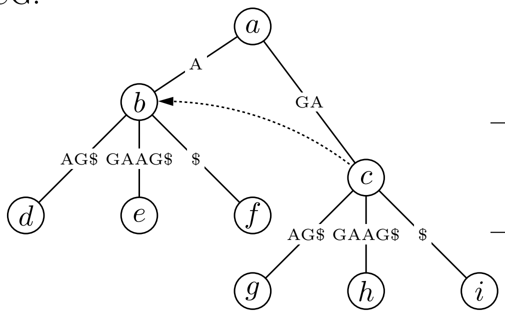


Figure 3.1: Trie containing the 3-mers AAA, AGA, GAA with weights α_1, α_2, α_3. Additionally, the figure displays resulting weights at inner nodes. Figure taken from Sonnenburg et al. (2007a).

$\boldsymbol{u} \in \Sigma^d$, but also all prefixes of $\boldsymbol{u}$. For sorted arrays this amounts to d separate `lookup` operations, while for tries all prefixes of $\boldsymbol{u}$ are already known when the bottom of the trie is reached. In this case, the trie has to store aggregated weights in internal nodes (Sonnenburg et al., 2006a, Rieck et al., 2006). This is illustrated for the WD kernel in Figure 3.1.

Suffix Trees and Matching Statistics A fourth alternative for efficient storage of sparse weights and string kernel computation builds on two data structures: suffix trees and matching statistics (Vishwanathan and Smola, 2003). A *suffix tree* $S_{\mathbf{x}}$ is a compact representation of a trie, which stores all suffixes of a sequence $\mathbf{x}$ in $\mathcal{O}(l_{\mathbf{x}})$ space and allows efficient retrieval of arbitrary sub-sequences of $\mathbf{x}$ (Gusfield, 1997). A *matching statistic* $M_{\mathbf{x}'}$ for a suffix tree $S_{\mathbf{x}}$ is defined by two vectors v and c of length $l_{\mathbf{x}'}$, where v_i reflects the length of the longest substring of $\mathbf{x}$ matching $\mathbf{x}'$ at position i and c_i is a corresponding node in $S_{\mathbf{x}}$ (Chang and Lawler, 1994). As an example, Figure 3.2 shows a suffix tree $S_{\mathbf{x}}$ and a matching statistic $M_{\mathbf{x}'}$ for the sequences $\mathbf{x}$ = GAGAAG and $\mathbf{x}'$ = GAACG.



(a) Suffix tree $S_{\mathbf{x}}$

$\mathbf{x}'$	G	A	A	C	G
v_i	3	2	1	0	1
c_i	g	d	b	a	c

(b) Matching statistic $M_{\mathbf{x}'}$

Figure 3.2: Suffix tree $S_{\mathbf{x}}$ for the sequence $\mathbf{x}$ = GAGAAG and matching statistic $M_{\mathbf{x}'}$ for $\mathbf{x}'$ = GAACG matched against $S_{\mathbf{x}}$. A sentinel symbol \$ has been added to $\mathbf{x}$, s.t. that all leaves correspond to suffixes. Figure taken from Sonnenburg et al. (2007a).

By traversing $S_{\mathbf{x}}$ and looping over $M_{\mathbf{x}'}$ in parallel, a variety of string kernels $\mathrm{k}(\mathbf{x}, \mathbf{x}')$ – including the spectrum and WD kernel – can be computed using $\mathcal{O}(l_{\mathbf{x}} + l_{\mathbf{x}'})$ run-time. A detailed discussion of the corresponding algorithms, weighting schemes and extensions is given in Vishwanathan and Smola (2004), Rieck et al. (2007). The approach can be further extended to support the operations `clear`, `add` and `lookup`. In contrast to sorted arrays and tries, these operations are favorably performed on the domain of *sequences* instead of single d-mers to ensure linear run-time. Starting with an empty suffix tree

S obtained using `clear`, the `add` operation is realised by appending sequences and implicitly contained d-mers to S using an online construction algorithm, (e.g. Ukkonen, 1995). To avoid matches over multiple sequences, each sequence $\mathbf{x}_i$ is delimited by a sentinel symbol $\$_i \notin \Sigma$. Given S, the `lookup` operation for a sequence $\mathbf{x}'$ is performed by calculating $M_{\mathbf{x}'}$, so that a kernel computation can be carried out in $\mathcal{O}(l_{\mathbf{x}'})$ run-time using S and $M_{\mathbf{x}'}$.
In practice however, suffix trees introduce a crucial overhead in storage space due to the high complexity of the data structure, which makes memory preserving data structures such as sorted arrays more attractive, especially on small alphabets. Recently, an alternative, suffix arrays, have been proposed to reduce the memory requirements, still n input symbols results in at least $19n$ bytes of allocated memory independent of the considered alphabet (Teo and Vishwanathan, 2006)

	Explicit map	Sorted arrays	Tries	Suffix trees
`clear` of $\boldsymbol{v}$	$\mathcal{O}(\|\Sigma\|^d)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`add` of all $\boldsymbol{u}$ from $\mathbf{x}$ to $\boldsymbol{v}$	$\mathcal{O}(l_{\mathbf{x}})$	$\mathcal{O}(l_{\mathbf{x}} \log l_{\mathbf{x}})$	$\mathcal{O}(l_{\mathbf{x}} d)$	$\mathcal{O}(l_{\mathbf{x}})$
`lookup` of all $\boldsymbol{u}$ from $\mathbf{x}'$ in $\boldsymbol{v}$	$\mathcal{O}(l_{\mathbf{x}'})$	$\mathcal{O}(l_{\mathbf{x}} + l_{\mathbf{x}'})$	$\mathcal{O}(l_{\mathbf{x}'} d)$	$\mathcal{O}(l_{\mathbf{x}'})$

Table 3.1: Comparison of worst-case run-times for multiple calls of `clear`, `add` and `lookup` on a sparse vector $\boldsymbol{v}$ using explicit maps, sorted arrays, tries and suffix trees. Table taken from Sonnenburg et al. (2007a).

Summary Table 3.1 coarsely summarises the worst-case run-times for multiple calls of `clear`, `add` and `lookup` using the previously introduced data structures. From the provided run-time bounds, it is obvious that the explicit map representation is favorable if the considered alphabet Σ and order d is sufficiently small – for instance as in several application of DNA analysis where $|\Sigma| = 4$ and $d \leq 6$. For larger alphabets and higher d, the sorted array approach is more attractive in practice. Other than tries and suffix trees, the sorted array approach is much easier to implement and its memory requirements are easier to estimate. If either Σ or d can get arbitrarily large, suffix trees are the data structure of choice as they operate linear in sequence lengths independent of Σ and d, however, as mentioned earlier there is a large overhead in storage space due to the complexity of the suffix tree structure. The trie-based approach may not seem suitable for large scale learning in comparison to the other methods, but the per-node augmentation of tries with additional values such as aggregated weights shown in Figure 3.1 can drastically speed up computation of complex string kernels such as the WD kernel, which can not efficiently be mapped to other approaches.

3.1.2 Speeding up Linear Combinations of Kernels

One of the most time consuming operation appearing in kernel machine training and evaluation is computing output, which manifests as computing linear combinations of kernels

$$g_i = \sum_{j \in J} \alpha_j y_j \, \mathrm{k}(\mathbf{x}_j, \mathbf{x}_i) \quad i = 1 \ldots N. \tag{3.1}$$

In this section we develop the `linadd` technique, which speeds up computing linear combinations of already linear time string kernels. We apply this technique to speeding up chunking based support vector machines training (cf. Section 3.1.3) and kernel machine evaluation. They key idea of `linadd` is to split the computations of Equation (3.1) into two steps. In a first step the normal $\mathbf{w}$ is computed and stored in one of the sparse data structures mentioned above, which can be efficiently done using the using the `clear` and `add` operations. In a second step the output for all N examples is computed using `lookup` operations, more formally exploiting $\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ one may rewrite Equation (3.1) as

$$\begin{aligned} g_i &= \sum_{j \in J} \alpha_j y_j \,\mathrm{k}(\mathbf{x}_j, \mathbf{x}_i) \\ &= \left(\sum_{j \in I} \alpha_j y_j \Phi(\mathbf{x}_j) \right) \cdot \Phi(\mathbf{x}_i) \\ &= \mathbf{w} \cdot \Phi(\mathbf{x}_i) \forall i = 1 \ldots N. \end{aligned} \tag{3.2}$$

The explicit representation of $\mathbf{w} = \sum_{j \in I} \alpha_j y_j \Phi(\mathbf{x}_j)$ may depending on the `clear`, `add` and `lookup` costs (cf. Table 3.1) lead to speedups of up to factor $|J|$. As for large scale learning task, i.e., huge N most time is spend on in `lookup` operations one may gain additional speedups by parallelizing $g_i = \mathbf{w} \cdot \Phi(\mathbf{x}_i)$, $\forall i = 1 \ldots N$ as done in Section 3.2. Note that computing $\Phi(\mathbf{x}_i)$ is not a costly operation as for many string kernels it boils down to extracting $k-$mers of $\mathbf{x}_i$. However, with an increased size of the index set J $\mathbf{w}$ may – depending on the string kernel even when using the sparse data structures – quickly become prohibitively large. This, especially holds for the WD kernel with shifts (cf. Equation (2.6)) when a large degree and shift is used. However, in practice J is small for SVM training (Section 3.1.3) and only becomes large on computing outputs. To reduce the memory requirements, one may construct a sparse data structure $\mathbf{w}_d$ on only a subset of the features appearing in $\mathbf{w}$

$$\begin{aligned} g_i &= \mathbf{w} \cdot \Phi(\mathbf{x}_i) \\ &= \sum_{d=1}^{D} \mathbf{w}_d \Phi_d(\mathbf{x}_i) \quad \forall i = 1 \ldots N. \end{aligned} \tag{3.3}$$

To be efficient it is necessary that the output can be computed in `batch`es, i.e., all examples $\mathbf{x}_i, \ldots \mathbf{x}_N$ are available and thus $\mathbf{w}_d$ needs to be constructed only once. For example, the WD kernels feature space can be represented in $l_\mathbf{x}$ tries, each trie corresponding to one position in the sequence. To compute output in memory efficient `batch`es, it is therefore natural to start with $\mathbf{g} = \mathbf{0}$ and to construct only a single trie via which $\mathbf{g}$ is updated. Furthermore, especially more complex and thus computationally more demanding kernels, such as the Mismatch Spectrum, Mismatch WD, Weighted Spectrum and WD kernel with shifts, especially profit from `linadd`. For the mismatch kernels the idea is to add for each $\boldsymbol{u} \in \mathbf{x}$ all $\binom{d}{2m}(|\Sigma| - 1)^{2m}$ oligomers of length d to the sparse data structure, which mismatch with $\boldsymbol{u}$ in at most $2m$ positions, while the `lookup` costs remain the same. Note, however, that the resulting data structure may become huge for larger m, i.e., only at the expense of increased memory usage we achieve a considerable speedup. However, if the alphabet and d are small as is the case in genomic sequence analysis where a 4 character DNA alphabet and usually $d < 8$ are sufficient, one may use the explicit map representation. In this case, the lookup operation for N sequences only takes $\mathcal{O}(NL)$ even for the Weighted Spectrum and Mismatch Spectrum kernel. Similarly, for the WD kernel with *shifts* one may add all sequences to the tries shifted by $s = 0 \ldots S$. In this context it becomes clear that a positional scoring $s(\mathbf{x}) := \sum_{k=1}^{K} \sum_{i=1}^{l-k+1} w(x_i, \ldots, x_{i+k-1})$ of $k-$mers starting at position i of order $k = 1 \ldots K$ is of high expressive power and able to model most of the currently existing string kernel imposed feature spaces. Based on this observation, we will in Section 4.2 develop means to visualise such a weighting system and extract discriminating motifs motifs.

3.1.3 Speeding up SVM Training

As it is not feasible to use standard optimisation toolboxes for solving large scale SVM training problem, decomposition techniques are used in practice. Most chunking algorithms work by first selecting Q variables (working set $W \subseteq \{1, \dots, N\}$, $Q := |W|$) based on the current solution and then solve the reduced problem with respect to the working set variables. These two steps are repeated until some optimality conditions are satisfied (see Algorithm 3.1 and e.g., Joachims (1998)). For selecting the

Algorithm 3.1 SVM Chunking Algorithm

```
while optimality conditions are violated do
  select $Q$ variables for the working set.
  solve reduced problem on the working set.
end while
```

working set and checking the termination criteria in each iteration, the vector $\mathbf{g}$ with $g_i = \sum_{j=1}^N \alpha_j y_j \,\mathrm{k}(x_i, x_j)$, $\quad i = 1, \dots, N$ is usually needed. Computing $\mathbf{g}$ from scratch in every iteration requires $\mathcal{O}(N^2)$ kernel computations. To avoid recomputation of $\mathbf{g}$ one typically starts with $\mathbf{g} = \mathbf{0}$ and only computes updates of $\mathbf{g}$ on the working set W

$$g_i \leftarrow g_i^{old} + \sum_{j \in W} (\alpha_j - \alpha_j^{old}) y_j \,\mathrm{k}(x_i, x_j), \quad \forall i = 1, \dots, N.$$

As a result the effort decreases to $\mathcal{O}(QN)$ kernel computations, which can be further speed up by using kernel caching (e.g. Joachims, 1998). However, kernel caching is not efficient enough for large scale problems[2] and thus most time is spend computing kernel rows for the updates of $\mathbf{g}$ on the working set W. Note however that this update as well as computing the Q kernel rows can be easily parallelised; cf. Section 3.3.
Exploiting $\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ and $\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \Phi(\mathbf{x}_i)$ we can rewrite the update rule as

$$g_i \leftarrow g_i^{old} + \sum_{j \in W} (\alpha_j - \alpha_j^{old}) y_j \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \rangle = g_i^{old} + \mathbf{w}^W \cdot \Phi(\mathbf{x}_i), \qquad (3.4)$$

where $\mathbf{w}^W = \sum_{j \in W} (\alpha_j - \alpha_j^{old}) y_j \Phi(\mathbf{x}_j)$ is the normal (update) vector on the working set.
If the kernel feature map can be computed explicitly and is sparse (as discussed before), then computing the update in (3.4) can be accelerated. One only needs to compute and store $\mathbf{w}^W$ (using the `clear` and $\sum_{q \in W} |\{\Phi_j(\mathbf{x}_q) \neq 0\}|$ `add` operations) and performing the scalar product $\mathbf{w}^W \cdot \Phi(\mathbf{x}_i)$ (using $|\{\Phi_j(\mathbf{x}_i) \neq 0\}|$ `lookup` operations).
Depending on the kernel, the way the sparse vectors are stored Section 3.1.1 and on the sparseness of the feature vectors, the speedup can be quite drastic. For instance, for the WD kernel one kernel computation requires $\mathcal{O}(Ld)$ operations (L is the length of the sequence). Hence, computing (3.4) N times requires $\mathrm{O}(NQLd)$ operations. When using tries, then one needs QL `add` operations (each $\mathcal{O}(d)$) and NL `lookup` operations (each $\mathcal{O}(d)$). Therefore, only $\mathcal{O}(QLd + NLd)$ basic operations are needed in total. When N

[2]For instance, when using a million examples one can only fit 268 rows into 1 GB. Moreover, caching 268 rows is insufficient when for instance having many thousands of active variables.

is large enough, it leads to a speedup by a factor of Q. Finally, note that kernel caching is no longer required and as Q is small in practice (e.g., $Q = 42$) the resulting trie has rather few leaves and thus only needs little storage.
The pseudo-code of our `linadd` SVM chunking algorithm is given in Algorithm 3.2.

Algorithm 3.2 Outline of the chunking algorithm that exploits the fast computations of linear combinations of kernels (e.g., by tries).

```
{INITIALIZATION}
$g_i = 0$, $\alpha_i = 0$ for $i = 1, \ldots, N$
{LOOP UNTIL CONVERGENCE}
for $t = 1, 2, \ldots$ do
    Check optimality conditions and stop if optimal
    select working set W based on $\mathbf{g}$ and $\boldsymbol{\alpha}$, store $\boldsymbol{\alpha}^{old} = \boldsymbol{\alpha}$
    solve reduced problem $W$ and update $\boldsymbol{\alpha}$

    clear $\mathbf{w}$
    $\mathbf{w} \leftarrow \mathbf{w} + (\alpha_j - \alpha_j^{old}) y_j \Phi(\mathbf{x}_j)$ for all $j \in W$ (using add)
    update $g_i = g_i + \mathbf{w} \cdot \Phi(\mathbf{x}_i)$ for all $i = 1, \ldots, N$ (using lookup)
end for
```

3.2 A Simple Parallel Chunking Algorithm

As still most time is spent in evaluating $g(\mathbf{x})$ for all training examples, further speedups are gained when parallelizing the evaluation of $g(\mathbf{x})$. When using the `linadd` algorithm, one first constructs the trie (or any of the other possible more appropriate data structures) and then performs parallel `lookup` operations using several CPUs (e.g., using shared memory or several copies of the data structure on separate computing nodes). We have implemented this algorithm based on multiple *threads* (using shared memory) and have gained reasonable speedups (see next section).
Note that this part of the computations is almost ideal to distribute to many CPUs, as only the updated $\boldsymbol{\alpha}$ (or $\mathbf{w}$ depending on the communication costs and size) have to be transfered before each CPU computes a large chunk $I_k \subset \{1, \ldots, N\}$ of

$$h_i^{(k)} = \mathbf{w} \cdot \Phi(\mathbf{x}_i), \quad \forall i \in I_k, \quad \forall k = 1, \ldots, N, \text{ where } (I_1 \cup \cdots \cup I_n) = (1, \ldots, N)$$

that is transfered to a master node that finally computes $\mathbf{g} \leftarrow \mathbf{g} + \boldsymbol{h}$, as illustrated in Algorithm 3.3.

3.3 Benchmarking SVM Training Time

Experimental Setup To demonstrate the effect of the proposed `linadd` optimisation (Algorithm 3.2) for single, four and eight CPUs, we applied each of the algorithms to a *human* splice site data set[3], comparing it to the original WD formulation. The splice data set contains 159,771 true acceptor splice site sequences and 14,868,555 decoys, leading to a total of 15,028,326 sequences each 141 base pairs in length. It was generated following a procedure similar to the one in Sonnenburg et al. (2005a) for *C. elegans*, which however contained "only" 1,026,036 examples. Note that the dataset is very unbalanced as 98.94% of the examples are negatively labelled. We are using this data set in

[3]The splice dataset can be downloaded from `http://www.fml.tuebingen.mpg.de/raetsch/projects/lsmkl`

Algorithm 3.3 Outline of the parallel chunking algorithm that exploits the fast computations of linear combinations of kernels.

```
{ Master node }
{INITIALIZATION}
$g_i = 0$, $\alpha_i = 0$ for $i = 1, \dots, N$
{LOOP UNTIL CONVERGENCE}
for $t = 1, 2, \dots$ do
  Check optimality conditions and stop if optimal
  select working set W based on $\mathbf{g}$ and $\boldsymbol{\alpha}$, store $\boldsymbol{\alpha}^{old} = \boldsymbol{\alpha}$
  solve reduced problem $W$ and update $\boldsymbol{\alpha}$
  transfer to Slave nodes: $\alpha_j - \alpha_j^{old}$ for all $j \in W$
  fetch from $n$ Slave nodes: $\boldsymbol{h} = (\boldsymbol{h}^{(1)}, \dots, \boldsymbol{h}^{(n)})$
  update $g_i = g_i + h_i$ for all $i = 1, \dots, N$
end for
signal convergence to slave nodes

{ Slave nodes }
{LOOP UNTIL CONVERGENCE}
while not converged do
  fetch from Master node $\alpha_j - \alpha_j^{old}$ for all $j \in W$
  clear w
  $\mathbf{w} \leftarrow \mathbf{w} + (\alpha_j - \alpha_j^{old}) y_j \Phi(\mathbf{x}_j)$ for all $j \in W$ (using add)
  node $k$ computes $h_i^{(k)} = \mathbf{w} \cdot \Phi(\mathbf{x}_i)$
      for all $i = (k-1)\frac{N}{n}, \dots, k\frac{N}{n} - 1$ (using lookup)
  transfer to master: $\boldsymbol{h}^{(k)}$
end while
```

all benchmark experiments and trained SVMs using the `SHOGUN` machine learning toolbox (cf. Appendix C), which contains a modified version of SVM$^{\text{light}}$ (Joachims, 1999) on 500, 1,000, 5,000, 10,000, 30,000, 50,000, 100,000, 200,000, 500,000, 1,000,000, 2,000,000, 5,000,000 and 10,000,000 randomly sub-sampled examples and measured the time needed in SVM training. For classification performance evaluation, we always use the same remaining 5,028,326 examples as a test data set. We set the degree parameter to $d = 20$ for the WD kernel and to $d = 8$ for the spectrum kernel fixing the SVMs regularisation parameter to $C = 5$. SVM$^{\text{light}}$'s subproblem size (parameter `qpsize`) and convergence criterion (parameter `epsilon`) were set to $Q = 112$ and $\epsilon = 10^{-5}$, respectively, while a kernel cache of 1GB was used for all kernels except the precomputed kernel and algorithms using the `linadd`-SMO extension for which the kernel-cache was disabled. Later on, we measure whether changing the quadratic subproblem size Q influences SVM training time. Experiments were performed on a PC powered by *eight* 2.4GHz AMD Opteron(tm) processors running Linux. We measured the training time for each of the algorithms (single, quad or eight CPU version) and data set sizes.
The obtained training times for the different SVM algorithms are displayed in Table 3.2 and in Figure 3.3. First, SVMs were trained using standard SVM$^{\text{light}}$ with the Weighted Degree Kernel precomputed (*WDPre*), the standard WD kernel (*WD1*) and the precomputed (*SpecPre*) and standard spectrum kernel (*Spec*). Then, SVMs utilizing the `linadd` extension[4] were trained using the WD (*LinWD*) and spectrum (*LinSpec*)

[4]More precisely the `linadd` and $\mathcal{O}(L)$ block formulation of the WD kernel as proposed in Sonnenburg et al. (2005b) was used.

kernel. Finally, SVMs were trained on four and eight CPUs using the parallel version of the `linadd` algorithm (*LinWD4, LinWD8*). *WD4* and *WD8* demonstrate the effect of a simple parallelisation strategy where the computation of kernel rows and updates on the working set are parallelised, which works with *any* kernel.
The training times obtained when precomputing the kernel matrix (which includes the time needed to precompute the full kernel matrix) is lower when no more than $1,000$ examples are used. Note that this is a direct cause of the relatively large subproblem size $Q = 112$. The picture is different for, say, $Q = 42$ (data not shown) where the *WDPre* training time is in all cases larger than the times obtained using the original WD kernel demonstrating the effectiveness of SVM$^{\text{light}}$'s kernel cache. The overhead of constructing a trie on $Q = 112$ examples becomes even more visible: only starting from 50,000 examples `linadd` optimisation becomes more efficient than the original WD kernel algorithm as the kernel cache cannot hold all kernel elements anymore.[5] Thus, it would be appropriate to lower the chunking size Q as seen in Table 3.4.

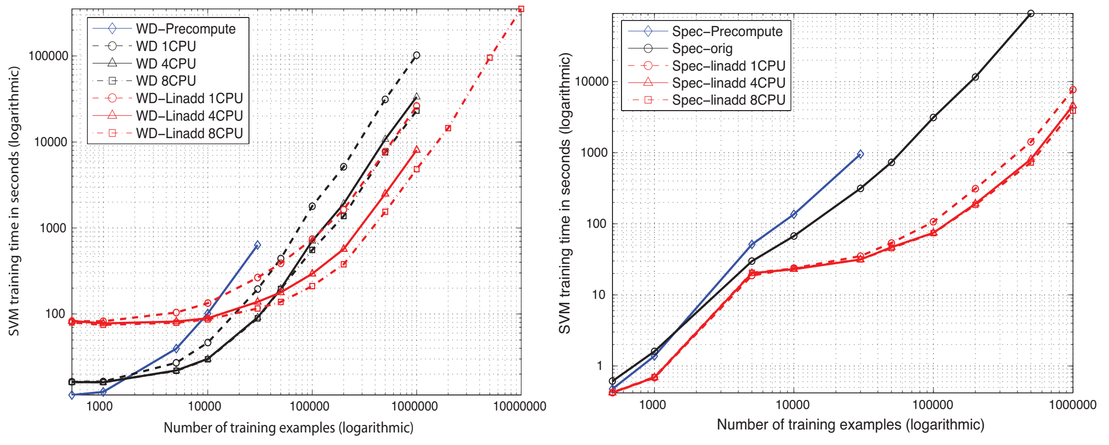


Figure 3.3: Comparison of the running time of the different SVM training algorithms using the weighted degree kernel. Note that as this is a log-log plot small appearing distances are large for larger N and that each slope corresponds to a different exponent. In the upper figure the Weighted Degree kernel training times are measured, the lower figure displays Spectrum kernel training times. Figure taken from Sonnenburg et al. (2007a).

The `linadd` formulation outperforms the original WD kernel by a factor of 3.9 on a million examples. The picture is similar for the spectrum kernel, here speedups of factor 64 on $500,000$ examples are reached, which stems from the fact that explicit maps (and not tries as in the WD kernel case as discussed in Section 3.1.1) could be used. This lead to a `lookup` cost of $\mathcal{O}(1)$ and a dramatically reduced map construction time. For that reason, the parallelisation effort benefits the WD kernel more than the Spectrum kernel: on one million examples the parallelisation using 4 CPUs (8 CPUs) leads to a speedup of factor 3.25 (5.42) for the WD kernel, but only 1.67 (1.97) for the Spectrum kernel. Thus, parallelisation will help more if the kernel computation is slow. Training with the original WD kernel with a sample size of $1,000,000$ takes about 28 hours, the `linadd` version still requires 7 hours while with the 8 CPU parallel implementation only about 6 hours and in conjunction with the `linadd` optimisation a single hour and 20 minutes are needed. Finally, training on 10 million examples takes about 4 days. Note that this data set is already $2.1GB$ in size.

[5]When single precision 4-byte floating point numbers are used, caching all kernel elements is possible when training with up to 16384 examples.

N	*WDPre*	*WD1*	*WD4*	*WD8*	*LinWD1*	*LinWD4*	*LinWD8*
500	12	17	17	17	83	83	80
1,000	13	17	17	17	83	78	75
5,000	40	28	23	22	105	82	80
10,000	102	47	31	30	134	90	87
30,000	636	195	92	90	266	139	116
50,000	-	441	197	196	389	179	139
100,000	-	1,794	708	557	740	294	212
200,000	-	5,153	1,915	1,380	1,631	569	379
500,000	-	31,320	10,749	7,588	7,757	2,498	1,544
1,000,000	-	102,384	33,432	23,127	26,190	8,053	4,835
2,000,000	-	-	-	-	-	-	14,493
5,000,000	-	-	-	-	-	-	95,518
10,000,000	-	-	-	-	-	-	353,227

N	*SpecPre*	*Spec*	*LinSpec1*	*LinSpec4*	*LinSpec8*
500	1	1	1	1	1
1,000	2	2	1	1	1
5,000	52	30	19	21	21
10,000	136	68	24	23	24
30,000	957	315	36	32	32
50,000	-	733	54	47	46
100,000	-	3,127	107	75	74
200,000	-	11,564	312	192	185
500,000	-	91,075	1,420	809	728
1,000,000	-	-	7,676	4,607	3,894

Table 3.2: **(top)** Speed Comparison of the original single CPU Weighted Degree Kernel algorithm (*WD1*) in SVM$^{\text{light}}$ training, compared to the four (*WD4*)and eight (*WD8*) CPUs parallelised version, the precomputed version (Pre) and the `linadd` extension used in conjunction with the original WD kernel for 1,4 and 8 CPUs (*LinWD1, LinWD4, LinWD8*). **(bottom)** Speed Comparison of the spectrum kernel without (*Spec*) and with `linadd` (*LinSpec1, LinSpec4, LinSpec8* using 1,4 and 8 processors). *SpecPre* denotes the precomputed version. The first column shows the sample size *N* of the data set used in SVM training while the following columns display the time (measured in seconds) needed in the training phase. Table taken from Sonnenburg et al. (2007a).

Classification Performance Figure 3.4 and Table 3.3 show the classification performance in terms of classification accuracy, area under the Receiver Operator Characteristic (ROC) Curve (Metz, 1978, Fawcett, 2003) and the area under the Precision Recall Curve (PRC) (see e.g., Davis and Goadrich (2006)) of SVMs on the human splice data set for different data set sizes using the WD kernel. Recall the definition of the ROC and PRC curves: The sensitivity (or recall) is defined as the fraction of correctly classified positive examples among the total number of positive examples, i.e., it equals the true positive rate $TPR = TP/(TP{+}FN)$. Analogously, the fraction $FPR = FP/(TN{+}FP)$ of negative examples wrongly classified positive is called the false positive rate. Plotting FPR against TPR results in the Receiver Operator Characteristic Curve (ROC) Metz (1978), Fawcett (2003). Plotting the true positive rate against the positive predictive value (also precision) $PPV = TP/(FP + TP)$, i.e., the fraction of correct positive predictions among all positively predicted examples, one obtains the Precision Recall Curve (PRC) (see e.g., Davis and Goadrich (2006)). Note that as this is a very unbalanced

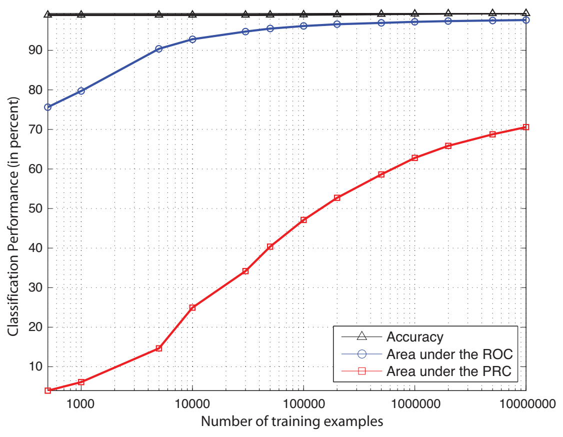


Figure 3.4: Comparison of the classification performance of the Weighted Degree kernel based SVM classifier for different training set sizes. The area under the Receiver Operator Characteristic (ROC) Curve, the area under the Precision Recall Curve (PRC) and the classification accuracy are displayed (in percent). Note that as this is a very unbalanced dataset, the accuracy and the area under the ROC curve are less meaningful than the area under the PRC. Figure taken from Sonnenburg et al. (2007a).

dataset the accuracy and the area under the ROC curve are almost meaningless, since both measures are independent of class ratios. The more sensible auPRC, however, steadily increases as more training examples are used for learning. Thus, one should train using all available data to obtain state-of-the-art results.

N	*Accuracy*	*auROC*	*auPRC*
500	98.93	75.61	3.97
1,000	98.93	79.70	6.12
5,000	98.93	90.38	14.66
10,000	98.93	92.79	24.95
30,000	98.93	94.73	34.17
50,000	98.94	95.48	40.35
100,000	98.98	96.13	47.11
200,000	99.05	96.58	52.70
500,000	99.14	96.93	58.62
1,000,000	99.21	97.20	62.80
2,000,000	99.26	97.36	65.83
5,000,000	99.31	97.52	68.76
10,000,000	99.35	97.64	70.57
10,000,000	-	96.03*	44.64*

Table 3.3: Comparison of the classification performance of the Weighted Degree kernel based SVM classifier for different training set sizes. The area under the ROC curve (*auROC*), the area under the Precision Recall Curve (*auPRC*) and the classification accuracy (*Accuracy*) are displayed (in percent). Larger values are better. A optimal classifier would achieve 100% Note that as this is a very unbalanced dataset the accuracy and the area under the ROC curve are almost meaningless. For comparison, the classification performance achieved using a 4th order Markov chain on 10 million examples (order 4 was chosen based on model selection, where order 1 to 8 using several pseudo-counts were tried) is displayed in the last row (marked *). Table taken from Sonnenburg et al. (2007a).

		Q				
N	112	12	32	42	52	72
500	83	**4**	**1**	22	68	67
1,000	83	**7**	**7**	**11**	34	60
5,000	105	**15**	**21**	33	31	68
10,000	134	**32**	**38**	54	67	97
30,000	266	**128**	**128**	**127**	160	187
50,000	389	258	**217**	242	252	309
100,000	740	696	**494**	585	573	643
200,000	1,631	1,875	1,361	**1,320**	1,417	1,610
500,000	7,757	9,411	6,558	**6,203**	6,583	7,883
1,000,000	26,190	31,145	20,831	**20,136**	21,591	24,043

Table 3.4: Influence on training time when varying the size of the quadratic program Q in SVM$^{\text{light}}$, when using the `linadd` formulation of the WD kernel. While training times do not vary dramatically, one still observes the tendency that with larger sample size a larger Q becomes optimal. The $Q = 112$ column displays the same result as column *LinWD1* in Table 3.2. Table taken from Sonnenburg et al. (2007a).

Varying SVM$^{\text{light}}$'s `qpsize` parameter As discussed in Section 3.1.3 and Algorithm 3.2, using the `linadd` algorithm for computing the output for all training examples w.r.t. to some working set can be speed up by a factor of Q (i.e., the size of the quadratic subproblems, termed `qpsize` in SVM$^{\text{light}}$). However, there is a trade-off in choosing Q as solving larger quadratic subproblems is expensive (quadratic to cubic effort). Table 3.4 shows the dependence of the computing time from Q and N. For example, the gain in speed between choosing $Q = 12$ and $Q = 42$ for 1 million of examples is 54%. Sticking with a mid-range Q (here $Q = 42$) seems to be a good idea for this task. However, a large variance can be observed, as the SVM training time depends to a large extend on which Q variables are selected in each optimisation step. For example, on the related *C. elegans* splice data set $Q = 141$ was optimal for large sample sizes while a midrange $Q = 71$ lead to the overall best performance. Nevertheless, one observes the trend that for larger training set sizes slightly larger subproblems sizes decrease the SVM training time.

3.4 Summary

This section proposes performance enhancements to make large-scale learning with string kernels and any kernel that can be written as an inner product of sparse feature vectors practical. The `linadd` algorithm (Algorithm 3.2) greatly accelerates SVM training. For the standalone SVM using the spectrum kernel, it achieves speedups of factor 60 (for the weighted degree kernel, about 7). Finally, we proposed a parallel version of the `linadd` algorithm running on a 8 CPU multiprocessor system, which lead to *additional* speedups of factor up to 4.9 for vanilla SVM training. It is future research to investigate whether training times can still be reduced by using the recent advances in training linear SVMs, like OCAS (Franc and Sonnenburg, 2008) on a re-weighted lower order WD kernel.

4 Interpretable Support Vector Machines

At the heart of many important bioinformatics problems, such as gene finding and function prediction, is the classification of biological sequences, above all of DNA and proteins. In many cases, the most accurate classifiers are obtained by training SVMs with complex sequence kernels (as presented in Chapter 2). Using the data structures from Chapter 3 they are applicable to huge datasets and achieve state-of-the-art results on, for instance, transcription start (Sonnenburg et al., 2006b) or splice site (Sonnenburg et al., 2007b) detection problems. However, an often criticised downside of SVMs with complex kernels is that it is very hard for humans to understand the learnt decision rules and to derive biological insights from them. To close this gap, we introduce two concepts: The concept of *Multiple Kernel Learning* (MKL) and the concept of *Positional Oligomer Importance Matrices (POIMs).* MKL, which will be introduced in Section 4.1, allows learning of SVMs with multiple kernels and their associated kernel weights. This provides flexibility and reflects the fact that typical learning problems often involve multiple, heterogeneous data sources. Furthermore, as we shall see in Section 4.1, it offers an elegant way to interpret the results. This concept is general in the sense that it works with arbitrary kernels. The concept of POIMs (cf. Section 4.2.2) on the other hand was specifically developed to understand string kernel based SVMs (even though it is applicable to general $k-$mer based scoring systems) and offers a leap of quality on that domain. POIMs display the contribution of a motif computed as the expected gain of its presence. For both concepts, we develop efficient algorithms for their computation and demonstrate how they can be used to find relevant motifs for different biological phenomena in a straight-forward way. This chapter is largely based on Sonnenburg et al. (2005a, 2006a, 2008).

4.1 Multiple Kernel Learning

Recent developments in the literature on SVMs and other kernel methods have shown the need to consider multiple kernels. This provides flexibility and reflects the fact that typical learning problems often involve multiple, heterogeneous data sources. Furthermore, as we shall see below, it leads to an elegant method to interpret the results, which can lead to a deeper understanding of the application.
While this so-called "multiple kernel learning" (MKL) problem can in principle be solved via cross-validation, several recent papers have focused on more efficient methods for multiple kernel learning (Chapelle et al., 2002, Bennett et al., 2002, Grandvalet and Canu, 2003, Ong et al., 2003, Bach et al., 2004, Lanckriet et al., 2004, Bi et al., 2004).
One of the problems with kernel methods compared to other techniques is that the resulting decision function

$$g(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i \, \mathrm{k}(\mathbf{x}, \mathbf{x}_i) + b \tag{4.1}$$

is hard to interpret and, hence, is difficult to use to extract relevant knowledge about the problem at hand. One can approach this problem by considering convex combinations of K kernels, i.e.

$$\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{K} \beta_k \, \mathrm{k}_k(\mathbf{x}_i, \mathbf{x}_j) \tag{4.2}$$

with $\beta_k \geq 0$ and $\sum_{k=1}^{K} \beta_k = 1$, where each kernel k_k uses only a distinct set of features. For appropriately designed sub-kernels k_k, the optimised combination coefficients can then be used to understand which features of the examples are of importance for discrimination: if one is able to obtain an accurate classification by a *sparse* weighting β_k, then one can quite easily interpret the resulting decision function. This is an important property missing in current kernel based algorithms. Note that this is in contrast to the kernel mixture framework of Bennett et al. (2002) and Bi et al. (2004) where each kernel *and* each example are assigned an independent weight and therefore do not offer an easy way to interpret the decision function. We will illustrate that the considered MKL formulation provides useful insights and at the same time is very efficient.
We consider the framework proposed by Lanckriet et al. (2004), which results in a convex optimisation problem - a quadratically-constrained quadratic program (QCQP). This problem is more challenging than the standard SVM QP, but it can in principle be solved by general-purpose optimisation toolboxes. Since the use of such algorithms will only be feasible for small problems with few data points and kernels, Bach et al. (2004) suggested an algorithm based on sequential minimial optimisation (SMO; Platt, 1999). While the kernel learning problem is convex, it is also non-smooth, making the direct application of simple local descent algorithms such as SMO infeasible. Bach et al. (2004) therefore considered a smoothed version of the problem to which SMO can be applied.
In the first part of this section we follow a different direction: We reformulate the binary classification MKL problem (Lanckriet et al., 2004) as a *Semi-Infinite Linear Program*, which can be efficiently solved using an off-the-shelf LP solver and a standard SVM implementation (cf. Section 4.1.1 for details). In a second step, we show how the MKL formulation and the algorithm are easily generalised to a much larger class of convex loss functions (cf. Section 4.1.2). Our proposed *wrapper method* works for any kernel and many loss functions: To obtain an efficient MKL algorithm for a new loss function, it now suffices to have an LP solver and the corresponding single kernel algorithm (which is assumed to be efficient). Using this general algorithm we were able to solve MKL problems with up to 50,000 examples and 20 kernels within reasonable time.
We also consider a *Chunking* algorithm that can be considerably more efficient, since it optimises the SVM $\boldsymbol{\alpha}$ multipliers and the kernel coefficients $\boldsymbol{\beta}$ at the same time. However, for large scale problems it needs to compute and cache the K kernels separately, instead of only one kernel as in the single kernel algorithm. This becomes particularly important when the sample size N is large. If, on the other hand, the number of kernels K is large, then the amount of memory available for caching is drastically reduced and, hence, kernel caching is not effective anymore. (The same statements also apply to the SMO-like MKL algorithm proposed in Bach et al. (2004).)
The `linadd` method presented in Chapter 3 avoids kernel caching for the class of kernels where the feature map $\Phi(\mathbf{x})$ can be explicitly computed and computations with $\Phi(\mathbf{x})$ can be implemented efficiently (which is the case for the considered string kernels). In the experimental part we show that the resulting algorithm is more than 70 times faster than the plain Chunking algorithm (for 50,000 examples), even though large kernel caches were used. Similarly, `linadd` allows to solve MKL problems with up to one

million examples and 20 kernels on a real-world splice site classification problem from computational biology. We conclude this section by illustrating the usefulness of our algorithms in several examples relating to the interpretation of results and to automatic model selection. Moreover, we provide an extensive benchmark study comparing the effect of different improvements on the running time of the algorithms.
We now first derive our MKL formulation for the binary classification case and then show how it can be extended to general cost functions. In the last subsection, we will propose algorithms for solving the resulting Semi-Infinite Linear Programs (SILPs).

4.1.1 Multiple Kernel Learning for Classification using SILP

In the Multiple Kernel Learning problem for binary classification one is given N data points $(\mathbf{x}_i, y_i)$ $(y_i \in \{\pm 1\})$, where $\mathbf{x}_i$ is translated via K mappings $\Phi_k(\mathbf{x}) \mapsto \mathrm{I\!R}^{D_k}$, $k = 1, \dots, K$, from the input into K feature spaces $(\Phi_1(\mathbf{x}_i), \dots, \Phi_K(\mathbf{x}_i))$ where D_k denotes the dimensionality of the k-th feature space. Then, one solves the following optimisation problem (Bach et al., 2004), which is equivalent to the linear SVM for $K = 1$:[1]

MKL Primal for Classification

$$\begin{array}{ll} \min & \dfrac{1}{2}\left(\displaystyle\sum_{k=1}^{K} \|\mathbf{w}_k\|_2\right)^2 + C\displaystyle\sum_{i=1}^{N} \xi_i \\ \text{w.r.t.} & \mathbf{w}_k \in \mathrm{I\!R}^{D_k}, \boldsymbol{\xi} \in \mathrm{I\!R}^N, b \in \mathrm{I\!R}, \\ \text{s.t.} & \xi_i \geq 0 \text{ and } y_i\left(\displaystyle\sum_{k=1}^{K} \mathbf{w}_k \cdot \Phi_k(\mathbf{x}_i) + b\right) \geq 1 - \xi_i, \ \ \forall i = 1, \dots, N \end{array} \tag{4.3}$$

Note that the problem's solution can be written as $\mathbf{w}_k = \beta_k \mathbf{w}'_k$ with $\beta_k \geq 0$, $\forall k = 1, \dots, K$ and $\sum_{k=1}^{K} \beta_k = 1$ (Bach et al., 2004). Note that therefore the ℓ_1-norm of $\boldsymbol{\beta}$ is constrained to one, while one is penalizing the ℓ_2-norm of $\mathbf{w}_k$ in each block k separately. The idea is that ℓ_1-norm constrained or penalised variables tend to have sparse optimal solutions, while ℓ_2-norm penalised variables do not (e.g. Rätsch, 2001, Chapter 5.2). Thus, the above optimisation problem offers the possibility to find sparse solutions on the block level with non-sparse solutions within the blocks.

Bach et al. (2004) derived the dual for problem (4.3). A more general derivation of the dual can be found in Appendix A.1. Taking their problem (D_K), squaring the constraints on gamma, multiplying the constraints by $\frac{1}{2}$ and finally substituting $\frac{1}{2}\gamma^2 \mapsto \gamma$ leads to the following *equivalent* multiple kernel learning dual:

[1]We assume $\mathrm{tr}(K_k) = 1$, $k = 1, \dots, K$ and set d_j in Bach et al. (2004) to one.

MKL Dual for Classification

$$\begin{array}{ll} \min & \gamma - \sum_{i=1}^{N} \alpha_i \\ \text{w.r.t.} & \gamma \in \mathbb{R}, \boldsymbol{\alpha} \in \mathbb{R}^N \\ \text{s.t.} & \mathbf{0} \leq \boldsymbol{\alpha} \leq \mathbf{1}C, \ \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \, \mathrm{k}_k(\mathbf{x}_i, \mathbf{x}_j) \leq \gamma, \ \ \forall k = 1, \ldots, K \end{array}$$

where $\mathrm{k}_k(\mathbf{x}_i, \mathbf{x}_j) = \Phi_k(\mathbf{x}_i) \cdot \Phi_k(\mathbf{x}_j)$. Note that we have one quadratic constraint per kernel ($S_k(\boldsymbol{\alpha}) \leq \gamma$). For $K = 1$, the above problem reduces to the original SVM dual. We will now move the term $-\sum_{i=1}^{N} \alpha_i$, into the constraints on γ. This can be equivalently done by adding $-\sum_{i=1}^{N} \alpha_i$ to both sides of the constraints and substituting $\gamma - \sum_{i=1}^{N} \alpha_i \mapsto \gamma$:

MKL Dual* for Classification

$$\begin{array}{ll} \min & \gamma \\ \text{w.r.t.} & \gamma \in \mathbb{R}, \boldsymbol{\alpha} \in \mathbb{R}^N \\ \text{s.t.} & \mathbf{0} \leq \boldsymbol{\alpha} \leq \mathbf{1}C, \ \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & \underbrace{\frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \, \mathrm{k}_k(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{N} \alpha_i}_{=:S_k(\boldsymbol{\alpha})} \leq \gamma, \ \ \forall k = 1, \ldots, K \end{array} \tag{4.4}$$

To solve (4.4), one may solve the following saddle point problem: minimise

$$\mathcal{L} := \gamma + \sum_{k=1}^{K} \beta_k (S_k(\boldsymbol{\alpha}) - \gamma) \tag{4.5}$$

w.r.t. $\boldsymbol{\alpha} \in \mathbb{R}^N, \gamma \in \mathbb{R}$ (with $\mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1}$ and $\sum_i \alpha_i y_i = 0$), and maximise it w.r.t. $\boldsymbol{\beta} \in \mathbb{R}^K$, where $\mathbf{0} \leq \boldsymbol{\beta}$. Setting the derivative w.r.t. to γ to zero, one obtains the constraint $\sum_{k=1}^{K} \beta_k = 1$ and (4.5) simplifies to: $\mathcal{L} = S(\boldsymbol{\alpha}, \boldsymbol{\beta}) := \sum_{k=1}^{K} \beta_k S_k(\boldsymbol{\alpha})$. While one *minimises* the objective w.r.t. $\boldsymbol{\alpha}$, at the same time one *maximises* w.r.t. the kernel weighting $\boldsymbol{\beta}$. This leads to a

Min-Max Problem

$$\begin{array}{ll} \max\limits_{\boldsymbol{\beta}} \min\limits_{\boldsymbol{\alpha}} & \sum_{k=1}^{K} \beta_k S_k(\boldsymbol{\alpha}) \\ \text{w.r.t.} & \boldsymbol{\alpha} \in \mathbb{R}^N, \ \boldsymbol{\beta} \in \mathbb{R}^K \\ \text{s.t.} & 0 \leq \boldsymbol{\alpha} \leq C \ , \ 0 \leq \boldsymbol{\beta}, \sum_{i=1}^{N} \alpha_i y_i = 0 \text{ and } \sum_{k=1}^{K} \beta_k = 1. \end{array} \tag{4.6}$$

This problem is very similar to Equation (9) in Bi et al. (2004) when "composite kernels," i.e., linear combinations of kernels are considered. There the first term of $S_k(\boldsymbol{\alpha})$ has been moved into the constraint, still $\boldsymbol{\beta}$, including the $\sum_{k=1}^K \beta_k = 1$ is missing.[2]
Assume $\boldsymbol{\alpha}^*$ were the optimal solution, then $\theta^* := S(\boldsymbol{\alpha}^*, \boldsymbol{\beta})$ would be minimal and, hence, $S(\boldsymbol{\alpha}, \boldsymbol{\beta}) \geq \theta^*$ for all $\boldsymbol{\alpha}$ (subject to the above constraints). Hence, finding a saddle-point of (4.5) is equivalent to solving the following semi-infinite linear program:

Semi-Infinite Linear Program (SILP)

$$\begin{aligned}
\max \quad & \theta && (4.7)\\
\text{w.r.t.} \quad & \theta \in \mathbb{R}, \boldsymbol{\beta} \in \mathbb{R}^K \\
\text{s.t.} \quad & \mathbf{0} \leq \boldsymbol{\beta},\ \sum_k \beta_k = 1 \text{ and } \sum_{k=1}^K \beta_k S_k(\boldsymbol{\alpha}) \geq \theta && (4.8)\\
& \text{for all } \boldsymbol{\alpha} \in \mathbb{R}^N \text{ with } \mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1} \text{ and } \sum_i y_i \alpha_i = 0
\end{aligned}$$

Note that this is a linear program, as θ and $\boldsymbol{\beta}$ are only linearly constrained. However, there are infinitely many constraints: one for each $\boldsymbol{\alpha} \in \mathbb{R}^N$ satisfying $0 \leq \boldsymbol{\alpha} \leq C$ and $\sum_{i=1}^N \alpha_i y_i = 0$. Both problems (4.6) and (4.7) have the same solution. To illustrate that, consider $\boldsymbol{\beta}$ is fixed and we minimise $\boldsymbol{\alpha}$ in (4.6). Let $\boldsymbol{\alpha}^*$ be the solution that minimises (4.6). Then, we can increase the value of θ in (4.7) as long as none of the infinitely many $\boldsymbol{\alpha}$-constraints (4.8) is violated, i.e., up to $\theta = \sum_{k=1}^K \beta_k S_k(\boldsymbol{\alpha}^*)$. On the other hand, as we increase θ for a fixed $\boldsymbol{\alpha}$ the maximizing $\boldsymbol{\beta}$ is found. We will discuss in Section 4.1.3 how to solve such semi-infinite linear programs.

4.1.2 Multiple Kernel Learning with General Cost Functions

In this section, we consider a more general class of MKL problems, where one is given an *arbitrary* strictly convex and differentiable loss function, for which we derive its MKL SILP formulation. We will then investigate in this general MKL SILP using different loss functions, in particular the soft-margin loss, the ϵ-insensitive loss and the quadratic loss.
We define the MKL primal formulation for a strictly convex and differentiable loss function $L(g(\mathbf{x}), y)$ as:

MKL Primal for Generic Loss Functions

$$\begin{aligned}
\min \quad & \frac{1}{2}\left(\sum_{k=1}^K \|\mathbf{w}_k\|\right)^2 + \sum_{i=1}^N L(g(\mathbf{x}_i), y_i) && (4.9)\\
\text{w.r.t.} \quad & \mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_K) \in \mathbb{R}^{D_1} \times \cdots \times \mathbb{R}^{D_K} \\
\text{s.t.} \quad & g(\mathbf{x}_i) = \sum_{k=1}^K \Phi_k(\mathbf{x}_i) \cdot \mathbf{w}_k + b, \ \ \forall i = 1, \ldots, N
\end{aligned}$$

[2] In Bi et al. (2004) it is argued that the approximation quality of composite kernels is inferior to mixtures of kernels where a weight is assigned per example *and* kernel as in Bennett et al. (2002). For that reason, and as no efficient methods were available to solve the composite kernel problem, they only considered mixtures of kernels and in the experimental validation used a uniform weighting in the composite kernel experiment. Also they did not consider to use composite kernels as a method to interpret the resulting classifier but looked at classification accuracy instead.

In analogy to Bach et al. (2004) we treat problem (4.9) as a second order cone program (SOCP) leading to the following dual (see Appendix A.1 for the derivation):

MKL Dual* for Generic Loss Functions

$$\begin{aligned}
\min \quad & \gamma \\
\text{w.r.t.} \quad & \gamma \in \mathbb{R},\ \boldsymbol{\alpha} \in R^N \\
\text{s.t.} \quad & \sum_{i=1}^N \alpha_i = 0 \quad \text{and} \\
\frac{1}{2}\left\|\sum_{i=1}^N \alpha_i \Phi_k(\mathbf{x}_i)\right\|_2^2 & - \sum_{i=1}^N L(L'^{-1}(\alpha_i, y_i), y_i) + \sum_{i=1}^N \alpha_i L'^{-1}(\alpha_i, y_i) \le \gamma,\ \forall k = 1, \dots, K
\end{aligned} \tag{4.10}$$

Here L'^{-1} denotes the inverse of the derivative of $L(g(\mathbf{x}), y)$ w.r.t. the prediction $g(\mathbf{x})$. To derive the SILP formulation we follow the same recipe as in Section 4.1.1: deriving the Lagrangian leads to a max-min problem formulation to be eventually reformulated as a SILP:

SILP for Generic Loss Functions

$$\begin{aligned}
\max \quad & \theta \\
\text{w.r.t.} \quad & \theta \in \mathbb{R}, \boldsymbol{\beta} \in \mathbb{R}^K \\
\text{s.t.} \quad & \mathbf{0} \le \boldsymbol{\beta}, \quad \sum_{k=1}^K \beta_k = 1 \quad \text{and} \quad \sum_{k=1}^K \beta_k S_k(\boldsymbol{\alpha}) \ge \theta,\ \forall \boldsymbol{\alpha} \in \mathbb{R}^N,\ \sum_{i=1}^N \alpha_i = 0,
\end{aligned} \tag{4.11}$$

where

$$S_k(\boldsymbol{\alpha}) = -\sum_{i=1}^N L(L'^{-1}(\alpha_i, y_i), y_i) + \sum_{i=1}^N \alpha_i L'^{-1}(\alpha_i, y_i) + \frac{1}{2}\left\|\sum_{i=1}^N \alpha_i \Phi_k(\mathbf{x}_i)\right\|_2^2.$$

We assumed that $L(x, y)$ is strictly convex and differentiable in x. Unfortunately, the soft margin and ϵ-insensitive loss do not have these properties. We therefore consider them separately in the sequel.

Soft Margin Loss We use the following loss to approximate the soft margin loss:

$$L_\sigma(x, y) = \frac{C}{\sigma} \log(1 + \exp(\sigma(1 - xy))).$$

It is easy to verify that

$$\lim_{\sigma \to \infty} L_\sigma(x, y) = C(1 - xy)_+.$$

Moreover, L_σ is strictly convex and differentiable for $\sigma < \infty$. Using this loss and assuming $y_i \in \{\pm 1\}$, we obtain (cf. Appendix A.1.3):

$$S_k(\boldsymbol{\alpha}) = -\sum_{i=1}^N \frac{C}{\sigma}\left(\log\left(\frac{Cy_i}{\alpha_i + Cy_i}\right) + \log\left(-\frac{\alpha_i}{\alpha_i + Cy_i}\right)\right) + \sum_{i=1}^N \alpha_i y_i + \frac{1}{2}\left\|\sum_{i=1}^N \alpha_i \Phi_k(\mathbf{x}_i)\right\|_2^2.$$

If $\sigma \to \infty$, then the first two terms vanish provided that $-C \leq \alpha_i \leq 0$ if $y_i = 1$ and $0 \leq \alpha_i \leq C$ if $y_i = -1$. Substituting $\alpha_i = -\tilde{\alpha}_i y_i$, we obtain

$$S_k(\tilde{\boldsymbol{\alpha}}) = -\sum_{i=1}^{N} \tilde{\alpha}_i + \frac{1}{2}\left\|\sum_{i=1}^{N} \tilde{\alpha}_i y_i \Phi_k(\mathbf{x}_i)\right\|_2^2 \text{ and } \sum_{i=1}^{N} \tilde{\alpha}_i y_i = 0,$$

with $0 \leq \tilde{\alpha}_i \leq C$ $(i = 1, \ldots, N)$, which is the same as (4.7).

One-Class Soft Margin Loss The one-class SVM soft margin (e.g. Schölkopf and Smola, 2002) is very similar to the two-class case and leads to

$$S_k(\boldsymbol{\alpha}) = \frac{1}{2}\left\|\sum_{i=1}^{N} \alpha_i \Phi_k(\mathbf{x}_i)\right\|_2^2$$

subject to $\mathbf{0} \leq \boldsymbol{\alpha} \leq \frac{1}{\nu N}\mathbf{1}$ and $\sum_{i=1}^{N} \alpha_i = 1$.

ϵ-insensitive Loss Using the same technique for the epsilon insensitive loss $L(x, y) = C(1 - |x - y|)_+$, we obtain

$$\begin{aligned} S_k(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) &= \frac{1}{2}\left\|\sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\Phi_k(\mathbf{x}_i)\right\|_2^2 - \sum_{i=1}^{N}(\alpha_i + \alpha_i^*)\epsilon - \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)y_i \\ \text{and} \quad & \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)y_i = 0, \text{ with } \mathbf{0} \leq \boldsymbol{\alpha}, \boldsymbol{\alpha}^* \leq C\mathbf{1}. \end{aligned}$$

It is easy to derive the dual problem for other loss functions such as the quadratic loss or logistic loss (see Appendix A.1.3 & A.1.3). Note that the dual SILP's only differ in the definition of S_k and the domains of the $\boldsymbol{\alpha}$'s.

4.1.3 Algorithms to solve SILPs

All *Semi-Infinite Linear Programs* considered in this work have the following structure:

$$\begin{aligned} \max \quad & \theta \\ \text{w.r.t.} \quad & \theta \in \mathbb{R}, \boldsymbol{\beta} \in \mathbb{R}^K \\ \text{s.t.} \quad & \mathbf{0} \leq \boldsymbol{\beta}, \quad \sum_{k=1}^{K} \beta_k = 1 \quad \text{and} \quad \sum_{k=1}^{K} \beta_k S_k(\boldsymbol{\alpha}) \geq \theta \text{ for all } \boldsymbol{\alpha} \in \mathcal{C}. \end{aligned} \tag{4.12}$$

They have to be optimised with respect to $\boldsymbol{\beta}$ and θ. The constraints depend on definition of S_k and therefore on the choice of the cost function. Using Theorem 5 in Rätsch et al. (2002) one can show that the above SILP has a solution if the corresponding primal is feasible and bounded (see also Hettich and Kortanek, 1993). Moreover, there is no duality gap, if $\mathcal{M} = \text{co}\{[S_1(\boldsymbol{\alpha}), \ldots, S_K(\boldsymbol{\alpha})]^\top \mid \boldsymbol{\alpha} \in \mathcal{C}\}$ is a closed set. For all loss functions considered in this section, this condition is satisfied.
We propose to use a technique called *Column Generation* to solve (4.12). The basic idea is to compute the optimal $(\boldsymbol{\beta}, \theta)$ in (4.12) for a restricted subset of constraints. It is called the *restricted master problem.* Then, a second algorithm generates a new, yet unsatisfied constraint determined by $\boldsymbol{\alpha}$. In the best case the other algorithm finds the

constraint that maximises the constraint violation for the given intermediate solution $(\boldsymbol{\beta}, \theta)$, i.e.

$$\boldsymbol{\alpha}_{\boldsymbol{\beta}} := \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathcal{C}} \sum_k \beta_k S_k(\boldsymbol{\alpha}). \tag{4.13}$$

If $\boldsymbol{\alpha}_{\boldsymbol{\beta}}$ satisfies the constraint $\sum_{k=1}^K \beta_k S_k(\boldsymbol{\alpha}_{\boldsymbol{\beta}}) \geq \theta$, then the solution $(\theta, \boldsymbol{\beta})$ is optimal. Otherwise, the constraint is added to the set of constraints and the iterations continue. Algorithm 4.1 is a special case of a set of SILP algorithms known as *exchange methods*. These methods are known to converge (cf. Theorem 7.2 in Hettich and Kortanek, 1993). However, no convergence rates for such algorithm are known.[3]
Since it is often sufficient to obtain an approximate solution, we have to define a suitable convergence criterion. Note that the problem is solved when all constraints are satisfied. Hence, it is a natural choice to use the normalised maximal constraint violation as a convergence criterion, i.e., the algorithm stops if $\epsilon \geq \epsilon_t := \left|1 - \frac{\sum_{k=1}^K \beta_k^t S_k(\boldsymbol{\alpha}^t)}{\theta^t}\right|$, where ϵ is an accuracy parameter, $(\boldsymbol{\beta}^t, \theta^t)$ is the optimal solution at iteration $t-1$ and $\boldsymbol{\alpha}^t$ corresponds to the newly found maximally violating constraint of the next iteration.
In the following paragraphs, we will formulate algorithms that alternately optimise the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

A Wrapper Algorithm The wrapper algorithm (see Algorithm 4.1) divides the problem into an inner and an outer subproblem. The solution is obtained by alternatively solving the outer problem using the results of the inner problem as input and vice versa until convergence. The outer loop constitutes the *restricted master problem*, which determines the optimal $\boldsymbol{\beta}$ for a fixed $\boldsymbol{\alpha}$ using an of-the-shelf linear optimiser. In the inner loop one has to identify unsatisfied constraints, which, fortunately, turns out to be particularly simple. Note that (4.13) is for all considered cases exactly the dual optimisation problem of the single kernel case for fixed $\boldsymbol{\beta}$. For instance, for binary classification with soft-margin loss, (4.13) reduces to the standard SVM dual using the kernel $\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \sum_k \beta_k \mathrm{k}_k(\mathbf{x}_i, \mathbf{x}_j)$:

$$\begin{aligned} v = \quad & \min_{\boldsymbol{\alpha}\in\mathbb{R}^N} \frac{1}{2}\sum_{i,j=1}^N \alpha_i\alpha_j y_i y_j \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^N \alpha_i \\ & \text{s.t.} \quad \mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1} \text{ and } \sum_{i=1}^N \alpha_i y_i = 0. \end{aligned}$$

Hence, we can use a standard SVM implementation with a single kernel to identify the most violated constraint $v \leq \theta$. Since there exists a large number of efficient algorithms to solve the single kernel problems for all sorts of cost functions, we have therefore found an easy way to extend their applicability to the problem of Multiple Kernel Learning. In addition, if the kernels are computed on-the-fly within the SVM still only a single kernel cache is required. The wrapper algorithm is very easy to implement, very generic and already reasonably fast for small to medium size problems. However, determining $\boldsymbol{\alpha}$ up to a fixed high precision even for intermediate solutions, while $\boldsymbol{\beta}$ is still far away

[3]It has been shown that solving semi-infinite problems like (4.7), using a method related to boosting (e.g. Meir and Rätsch, 2003) one requires at most $T = \mathcal{O}(\log(M)/\hat{\epsilon}^2)$ iterations, where $\hat{\epsilon}$ is the remaining constraint violation and the constants may depend on the kernels and the number of examples N (Rätsch, 2001, Rätsch and Warmuth, 2005, Warmuth et al., 2006). At least for not too small values of $\hat{\epsilon}$ this technique produces reasonably fast good approximate solutions.

Algorithm 4.1 The MKL-wrapper algorithm optimises a convex combination of K kernels and employs a linear programming solver to iteratively solve the semi-infinite linear optimisation problem (4.12). The accuracy parameter ϵ is a parameter of the algorithm. $S_k(\boldsymbol{\alpha})$ and $\mathcal{C}$ are determined by the cost function.

```
$S^0 = 1$, $\theta^1 = -\infty$, $\beta_k^1 = \frac{1}{K}$ for $k = 1, \dots, K$
for $t = 1, 2, \dots$ do
    Compute $\boldsymbol{\alpha}^t = \underset{\boldsymbol{\alpha} \in \mathcal{C}}{\operatorname{argmin}} \sum_{k=1}^{K} \beta_k^t S_k(\boldsymbol{\alpha})$ by single kernel algorithm with $\mathrm{k} = \sum_{k=1}^{K} \beta_k^t \,\mathrm{k}_k$
    $S^t = \sum_{k=1}^{K} \beta_k^t S_k^t$, where $S_k^t = S_k(\boldsymbol{\alpha}^t)$
    if $\left| 1 - \frac{S^t}{\theta^t} \right| \leq \epsilon$ then break
    $(\boldsymbol{\beta}^{t+1}, \theta^{t+1}) = \operatorname{argmax}\ \theta$
        w.r.t.  $\boldsymbol{\beta} \in \mathbb{R}^K, \theta \in \mathbb{R}$
        s.t.    $\mathbf{0} \leq \boldsymbol{\beta}$, $\sum_{k=1}^{K} \beta_k = 1$ and $\sum_{k=1}^{K} \beta_k S_k^r \geq \theta$ for $r = 1, \dots, t$
end for
```

from the global optimal is unnecessarily costly. Thus, there is room for improvement motivating the next section.

A Chunking Algorithm for Simultaneous Optimisation of α and β The goal is to simultaneously optimise $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ in SVM training. Usually it is infeasible to use standard optimisation tools (e.g., MINOS, CPLEX, LOQO) for solving even the *SVM training* problems on data sets containing more than a few thousand examples. So-called decomposition techniques as chunking (e.g., used in Joachims, 1998) overcome this limitation by exploiting the special structure of the SVM problem. The key idea of decomposition is to freeze all but a few optimisation variables (*working set*) and to solve a sequence of constant-size problems (subproblems of the SVM dual).
Here we would like to propose an extension of the chunking algorithm to optimise the kernel weights $\boldsymbol{\beta}$ and the example weights $\boldsymbol{\alpha}$ at the same time. The algorithm is motivated from an insufficiency of the wrapper algorithm described in the previous section: If the $\boldsymbol{\beta}$'s are not optimal yet, then the optimisation of the $\boldsymbol{\alpha}$'s until optimality is not necessary and therefore inefficient. It would be considerably faster if for any newly obtained $\boldsymbol{\alpha}$ in the chunking iterations, we could efficiently recompute the optimal $\boldsymbol{\beta}$ and then continue optimizing the $\boldsymbol{\alpha}$'s using the new kernel weighting.

Intermediate Recomputation of β Recomputing $\boldsymbol{\beta}$ involves solving a linear program and the problem grows with each additional $\boldsymbol{\alpha}$-induced constraint. Hence, after many iterations solving the LP may become infeasible. Fortunately, there are two facts making it still possible: (a) only a few of the added constraints remain active and one may as well remove inactive ones — this prevents the LP from growing arbitrarily and (b) for Simplex-based LP optimisers such as `CPLEX` there exists the so-called *hot-start feature*, which allows one to efficiently recompute the new solution, if for instance only a few additional constraints are added.
The SVM$^{\text{light}}$ optimiser, which we are going to modify, internally needs the output $\hat{g}_i = \sum_{j=1}^{N} \alpha_j y_j \,\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$ for all training examples $i = 1, \dots, N$ to select the next variables for optimisation (Joachims, 1999). However, if one changes the kernel weights, then

Algorithm 4.2 Outline of the MKL-Chunking algorithm for the classification case (extension to SVM$^{\text{light}}$) that optimises $\boldsymbol{\alpha}$ and the kernel weighting $\boldsymbol{\beta}$ simultaneously. The accuracy parameter ϵ and the subproblem size Q are assumed to be given to the algorithm. For simplicity, we omit the removal of inactive constraints. Furthermore, from one iteration to the next the LP only differs by one additional constraint. This can usually be exploited to save computing time for solving the LP.

```
$g_{k,i} = 0$, $\hat{g}_i = 0$, $\alpha_i = 0$, $\beta_k^1 = \frac{1}{K}$ for $k = 1, \dots, K$ and $i = 1, \dots, N$
for $t = 1, 2, \dots$ do
  Check optimality conditions and stop if optimal
  select Q suboptimal variables $i_1, \dots, i_Q$ based on $\hat{\mathbf{g}}$ and $\boldsymbol{\alpha}$
  $\boldsymbol{\alpha}^{old} = \boldsymbol{\alpha}$
  solve SVM dual with respect to the selected variables and update $\boldsymbol{\alpha}$
  $g_{k,i} = g_{k,i} + \sum_{q=1}^{Q} (\alpha_{i_q} - \alpha_{i_q}^{old}) y_{i_q} \, \mathrm{k}_k(\mathbf{x}_{i_q}, \mathbf{x}_i)$ for all $k = 1, \dots, M$ and $i = 1, \dots, N$
  for $k = 1, \dots, K$ do
    $S_k^t = \frac{1}{2} \sum_r g_{k,r} \alpha_r^t y_r - \sum_r \alpha_r^t$
  end for
  $S^t = \sum_{k=1}^{K} \beta_k^t S_k^t$
  if $\left| 1 - \frac{S^t}{\theta^t} \right| \geq \epsilon$
    $(\boldsymbol{\beta}^{t+1}, \theta^{t+1}) = \text{argmax } \theta$
        w.r.t. $\boldsymbol{\beta} \in \mathbb{R}^K, \theta \in \mathbb{R}$
        s.t.   $\mathbf{0} \leq \boldsymbol{\beta}$,  $\sum_k \beta_k = 1$ and $\sum_{k=1}^{M} \beta_k S_k^r \geq \theta$ for $r = 1, \dots, t$
  else
    $\theta^{t+1} = \theta^t$
  end if
  $\hat{g}_i = \sum_k \beta_k^{t+1} g_{k,i}$ for all $i = 1, \dots, N$
end for
```

the stored $\hat{g}_i$ values become invalid and need to be recomputed. To avoid the full recomputation one has to additionally store a $K \times N$ matrix $g_{k,i} = \sum_{j=1}^{N} \alpha_j y_j \, \mathrm{k}_k(\mathbf{x}_i, \mathbf{x}_j)$, i.e., the outputs for each kernel separately. If the $\boldsymbol{\beta}$'s change, then $\hat{g}_i$ can be quite efficiently recomputed by $\hat{g}_i = \sum_k \beta_k g_{k,i}$. We implemented the final chunking algorithm for the MKL regression and classification case and display the latter in Algorithm 4.2.

Discussion The Wrapper and the Chunking algorithm have both their merits: The Wrapper algorithm only relies on the repeated efficient computation of the single kernel solution, for which typically large scale algorithms exist. The Chunking algorithm is faster, since it exploits the intermediate $\boldsymbol{\alpha}$'s – however, it needs to compute and cache the K kernels separately (particularly important when N is large). If, on the other hand, K is large, then the amount of memory available for caching is drastically reduced and, hence, kernel caching is not effective anymore. The same statements also apply to the SMO-like MKL algorithm proposed in Bach et al. (2004). In this case, one is left with the Wrapper algorithm, unless one is able to exploit properties of the particular problem or the sub-kernels (see next section).

Speeding up in the MKL Case As elaborated in Section 4.1.3 and Algorithm 4.2, for MKL one stores K vectors $\mathbf{g}_k$, $k = 1, \dots, K$: one for each kernel to avoid full recomputation of $\hat{\mathbf{g}}$ if a kernel weight β_k is updated. Thus, to use the idea above in Algorithm 4.2 all one has to do is to store K normal vectors (e.g., tries)

$$\mathbf{w}_k^W = \sum_{j \in W} (\alpha_j - \alpha_j^{old}) y_j \Phi_k(\mathbf{x}_j), \quad k = 1, \dots, K$$

that are then used to update the $K \times N$ matrix $g_{k,i} = g_{k,i}^{old} + \mathbf{w}_k^W \cdot \Phi_k(\mathbf{x}_i)$ (for all $k = 1 \ldots K$ and $i = 1 \ldots N$) by which $\hat{g}_i = \sum_k \beta_k g_{k,i}$, (for all $i = 1 \ldots N$) is computed.

4.1.4 Estimating the Reliability of a Weighting

Finally, we want to assess the reliability of the learnt weights $\boldsymbol{\beta}$. For this purpose, we generate T bootstrap samples and rerun the whole procedure resulting in T weightings $\boldsymbol{\beta}^t$. To test the importance of a weight $\beta_{k,i}$ (and therefore the corresponding kernels for position and oligomer length) we apply the following method: We define a Bernoulli variable $X_{k,i}^t \in \{0, 1\}$, $k = 1, \ldots, d, i = 1, \ldots, L, t = 1, \ldots, T$ by

$$X_{k,i}^t = \begin{cases} 1, & \beta_{k,i}^t > \tau := \mathbf{E}_{k,i,t} X_{k,i}^t \\ 0, & \text{else} \end{cases} .$$

The sum $Z_{k,i} = \sum_{t=1}^T X_{k,i}^t$ has a binomial distribution Bin(T,p_0), p_0 unknown. We estimate p_0 with $\hat{p}_0 = \#(\beta_{k,i}^t > \tau)/T \cdot M$, i.e., the empirical probability to observe $P(X_{k,i}^t = 1)$, $\forall k, i, t$. We test whether $Z_{k,i}$ is as large as could be expected under Bin(T,$\hat{p}_0$) or larger, i.e., the null=hypothesis is $\mathcal{H}_0 : p \leq c^*$ (vs $\mathcal{H}_1 : p > c^*$). Here c^* is defined as $\hat{p}_0 + 2\mathbf{Std}_{k,i,t} X_{k,i}^t$ and can be interpreted as an upper bound of the confidence interval for p_0. This choice is taken to be adaptive to the noise level of the data and hence the (non)-sparsity of the weightings $\boldsymbol{\beta}^t$. The hypotheses are tested with a Maximum-Likelihood test on an α-level of $\alpha = 0.05$; that is c^{**} is the minimal value for that the following inequality hold:

$$0.05 = \alpha \geq P_{\mathcal{H}_0}(\text{reject } \mathcal{H}_0) = P_{\mathcal{H}_0}(Z_{k,i} > c^{**}) = \sum_{j=c^{**}}^{T} \binom{T}{j} \hat{p}_0 (1 - \hat{p}_0).$$

For further details on the test, see Mood et al. (1974) or Lehmann (1997). This test is carried out for every $\beta_{k,i}^t$. (We assume independence between the weights in one single $\boldsymbol{\beta}$, and hence assume that the test problem is the same for every $\beta_{k,i}$). If $\mathcal{H}_0$ can be rejected, the kernel learnt at position i on the k-mer is important for the detection and thus (should) contain biologically interesting knowledge about the problem at hand.

4.1.5 MKL for Knowledge Discovery

In this section, we will discuss toy examples for binary classification and regression, demonstrating that MKL can recover information about the problem at hand, followed by a brief review on problems for which MKL has been successfully used.

Classification The first example we deal with is a binary classification problem. The task is to separate two concentric classes shaped like the outline of stars. By varying the distance between the boundary of the stars, we can control the separability of the problem. Starting with a non-separable scenario with zero distance, the data quickly becomes separable as the distance between the stars increases, and the boundary needed for separation will gradually tend towards a circle. In Figure 4.1, three scatter plots of data sets with varied separation distances are displayed.
We generate several training and test sets for a wide range of distances (the radius of the inner star is fixed at 4.0, the outer stars radius is varied from $4.1 \ldots 9.9$). Each dataset contains 2,000 observations (1,000 positive and 1,000 negative) using a moderate noise level (Gaussian noise with zero mean and standard deviation 0.3). The MKL-SVM was

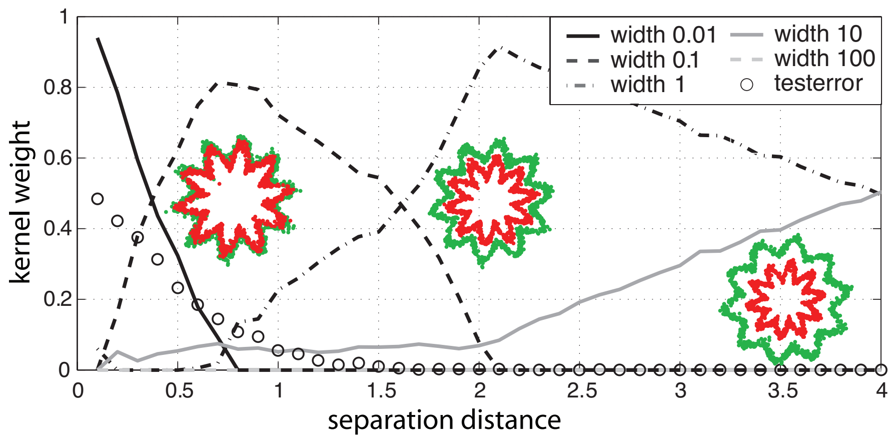


Figure 4.1: A 2-class toy problem where the dark gray star-like shape is to be distinguished from the light gray star inside the dark gray star. The distance between the dark star-like shape and the light star increases from the left to the right. Figure taken from Sonnenburg et al. (2006a).

trained for different values of the regularisation parameter C. For every value of C we averaged the test errors of all setups and choose the value of C that led to the smallest overall error ($C = 0.5$).[4]
The choice of the kernel width of the Gaussian RBF (below, denoted by RBF) kernel used for classification is expected to depend on the separation distance of the learning problem: An increased distance between the stars will correspond to a larger optimal kernel width. This effect should be visible in the results of the MKL, where we used MKL-SVMs with five RBF kernels with different widths ($2\sigma^2 \in \{0.01, 0.1, 1, 10, 100\}$). In Figure 4.1 we show the obtained kernel weightings for the five kernels and the test error (circled line) that quickly drops to zero as the problem becomes separable. Every column shows one MKL-SVM weighting. The courses of the kernel weightings reflect the development of the learning problem: as long as the problem is difficult the best separation can be obtained when using the kernel with smallest width. The low width kernel looses importance when the distance between the stars increases and larger kernel widths obtain a larger weight in MKL. Increasing the distance between the stars, kernels with greater widths are used. Note that the RBF kernel with largest width was not appropriate and thus never chosen. This illustrates that MKL can indeed recover information about the structure of the learning problem.

Regression We applied the newly derived MKL support vector regression formulation to the task of learning a sine function using three RBF-kernels with different widths ($2\sigma^2 \in \{0.005, 0.05, 0.5, 1, 10\}$). To this end, we generated several data sets with increasing frequency of the sine wave. The sample size was chosen to be 1,000. Analogous to the procedure described above we choose the value of $C = 10$, minimizing the overall test error. In Figure 4.2 exemplarily three sine waves are depicted, where the frequency increases from left to right. For every frequency, the computed weights for each kernel width are shown. One can see that MKL-SV regression switches to the width of the RBF-kernel fitting the regression problem best. In another regression experiment, we combined a linear function with two sine waves, one of lower frequency and one of high-frequency, i.e., $f(x) = \sin(ax) + \sin(bx) + cx$. Furthermore, we increase the frequency of the higher frequency sine wave, i.e., we varied a leaving b and c unchanged. The MKL

[4]Note that we are aware of the fact that the test error might be slightly underestimated.

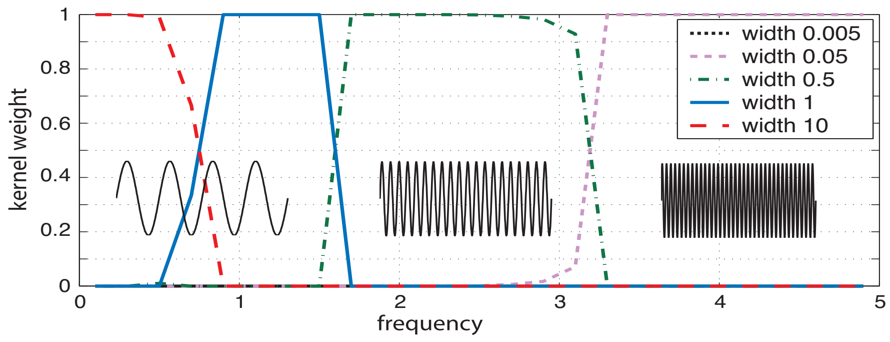


Figure 4.2: MKL Support Vector Regression for the task of learning a sine wave (please see text for details). Figure taken from Sonnenburg et al. (2006a).

weighting should show a combination of different kernels. Using ten RBF-kernels of different width (see Figure 4.3) we trained a MKL-SVR and display the learnt weights (a column in the figure). Again the sample size is 1,000 and one value for $C = 5$ is chosen via a previous experiment. The largest selected width (100) models the linear component (since RBF kernels with large widths are effectively linear) and the medium width (1) corresponds to the lower frequency sine. We varied the frequency of the high-frequency sine wave from low to high (left to right in the figure). One observes that MKL determines an appropriate combination of kernels of low and high widths, while decreasing the RBF kernel width with increased frequency. Additionally one can observe that MKL leads to sparse solutions since most of the kernel weights in Figure 4.3 are depicted in blue, that is they are zero.[5]

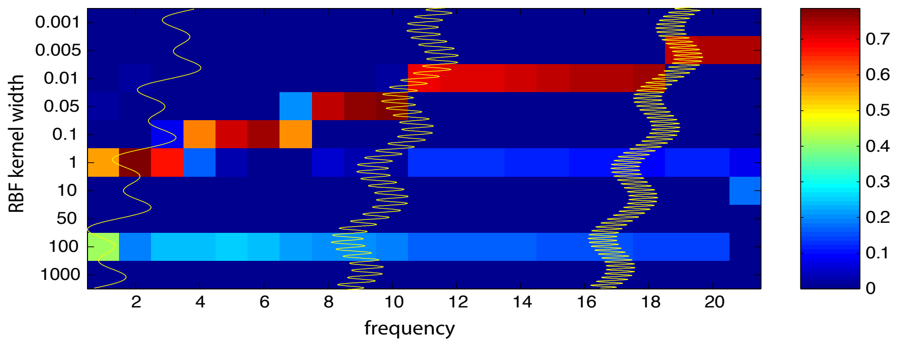


Figure 4.3: MKL support vector regression on a linear combination of three functions: $f(x) = \sin(ax) + \sin(bx) + cx$. MKL recovers that the original function is a combination of functions of low and high complexity. For more details, see text. Figure taken from Sonnenburg et al. (2006a).

MKL Learning Detects Motifs in Toy Data set As a proof of concept, we test our method on a toy data set with two hidden 7-mers (at positions 10 & 30) at four different noise levels (we used different numbers of random positions in the 7-mers that were replaced with random nucleotides; for a detailed description of the data see Appendix). We use the kernel as defined in Section 2.3 with one sub-kernel per position and oligomer length. We consider sequences of length $L = 50$ and oligomers up to length $d = 7$,

[5]The training time for MKL-SVR in this setup but with 10,000 examples was about 40 minutes, when kernel caches of size 100MB are used

leading to $M = 350$ sub-kernels. For every noise level, we train on 100 bootstrap replicates and learn the 350 WD kernel parameters in each run. On the resulting 100 weightings, we performed the reliability test (cf. Section 4.1.4). The results are shown in Figure 4.4 (columns correspond to different noise levels — increasing from left to right). Each figure shows a kernel weighting $\boldsymbol{\beta}$, where columns correspond to weights used at a certain sequence position and rows to the k-mer length used at that position. The plots in the first row show the weights that are detected to be important at a significance level of $\alpha = 0.05$ in bright (yellow) color. The likelihood for every weight to be detected by the test and thus to reject the null hypothesis $\mathcal{H}_0$ is illustrated in the plots in the second row (cf. Section 4.1.4 for details). Bright colors mean that it is more likely to reject $\mathcal{H}_0$.

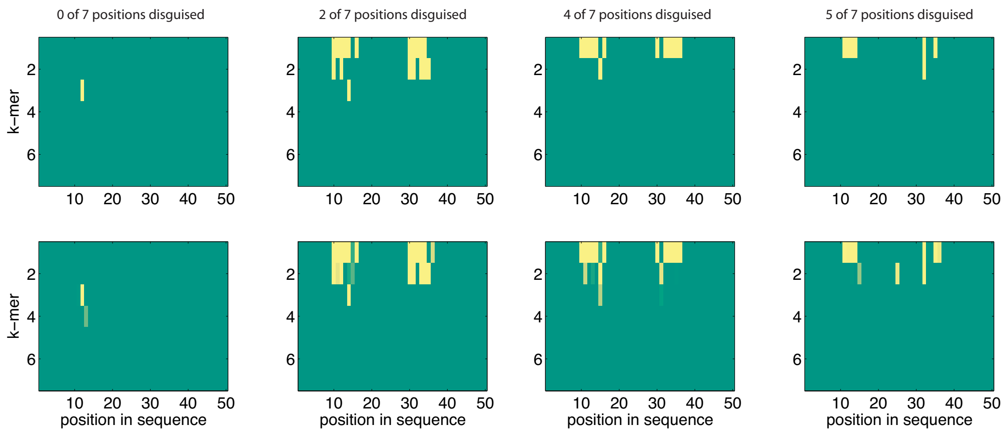


Figure 4.4: In this "figure matrix", columns correspond to the noise level, i.e., different numbers of nucleotides randomly substituted in the motif of the toy data set (cf. Appendix). Each sub-figure shows a matrix with each element corresponding to one kernel weight: columns correspond to weights used at a certain sequence position (1-50) and rows to the oligomer length used at that position (1-7). The first row of the figure matrix shows the kernel weights that are significant, while the second row depicts the likelihood of every weight to be rejected under $\mathcal{H}_0$. Figure taken from Sonnenburg et al. (2005a).

As long as the noise level does not exceed 2/7, longer matches of length 3 and 4 seem sufficient to distinguish sequences containing motifs from the rest. However, only the 3-mer is detected with the test procedure. When more nucleotides in the motifs are replaced with noise, more weights are determined to be of importance. This becomes obvious in column 3 were 4 out of 7 nucleotides within each motif were randomly replaced, but still an average ROC score of 99.6% is achieved. In the last column the ROC score drops down to 83%, but only weights in the correct range $10 \ldots 16$ and $30 \ldots 36$ are found to be significant.

Real World Applications in Bioinformatics MKL has been successfully used on real-world datasets in computational biology (Lanckriet et al., 2004, Sonnenburg et al., 2005a). It was shown to improve classification performance on the task of ribosomal and membrane protein prediction (Lanckriet et al., 2004), where a weighting over different kernels each corresponding to a different feature set was learnt. In their result, the included random channels obtained low kernel weights. However, as the data sets was small ($\approx 1,000$ examples) the kernel matrices could be precomputed and simultaneously kept in memory, which was not possible in Sonnenburg et al. (2005a), where a splice site recognition task for the worm *C. elegans* was considered. Here data is available in abundance (up to one million examples) and larger amounts are indeed needed to

obtain state of the art results (Sonnenburg et al., 2005b). On that dataset we were able to solve the classification MKL SILP for $N = 1,000,000$ examples using $K = 20$ kernels, and for $N = 10,000$ examples and $K = 550$ kernels, using the `linadd` optimisations with the weighted degree kernel. As a result we a) were able to learn the weighting $\boldsymbol{\beta}$ instead of choosing a heuristic and b) were able to use MKL as a tool for interpreting the SVM classifier as in Sonnenburg et al. (2005a), Rätsch et al. (2005).
As an example we learnt the weighting of a WD kernel of degree 20, which consist of a weighted sum of 20 sub-kernels each counting matching d-mers, for $d = 1, \ldots, 20$. The

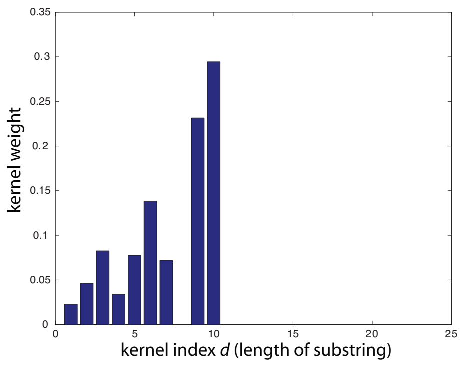


Figure 4.5: The learnt WD kernel weighting on a million of examples. Figure taken from Sonnenburg et al. (2005a).

learnt weighting is displayed in Figure 4.5 and shows a peak for 6-mers and 9&10-mers. The obtained weighting in this experiment is only partially useful for interpretation. For splice site detection, it is unlikely that k-mers of length 9 or 10 are playing the most important role. More likely to be important are substrings of length up to six. We believe that the large weights for the longest k-mers are an artifact, which comes from the fact that we are combining kernels with quite different properties, i.e., the 9th and 10th kernel leads to a combined kernel matrix that is most diagonally dominant (since the sequences are only similar to themselves but not to other sequences), which we believe is the reason for having a large weight.[6]
In Chapter 5 MKL is applied to understand splice site detection. There one kernel weight per position in the sequence is considered (or even one kernel weight per position in the sequence and k-mer length as in the example “MKL Learning Detects Motifs in Toy Data set” above.

Benchmarking MKL

Experimental Setup To demonstrate the effect of the MKL SILP formulation with and without the `linadd` extension for single, four and eight CPUs, we applied each of the algorithms to the *human* splice site data set as presented in Section 3.3. In this setup, the WD kernel weighting coefficients were learnt using Multiple Kernel Learning. The WD kernel of degree 20 consist of a weighted sum of 20 sub-kernels each counting matching d-mers, for $d = 1, \ldots, 20$. Using MKL we learnt the weighting on the splice site recognition task for one million examples as displayed in Figure 4.5 and discussed in Section 4.1.5. Focusing on a speed comparison we now show the obtained training

[6]This problem might be partially alleviated by including the identity matrix in the convex combination. However, as 2-norm soft margin SVMs can be implemented by adding a constant to the diagonal of the kernel (Cortes and Vapnik, 1995), this leads to an additional 2-norm penalisation.

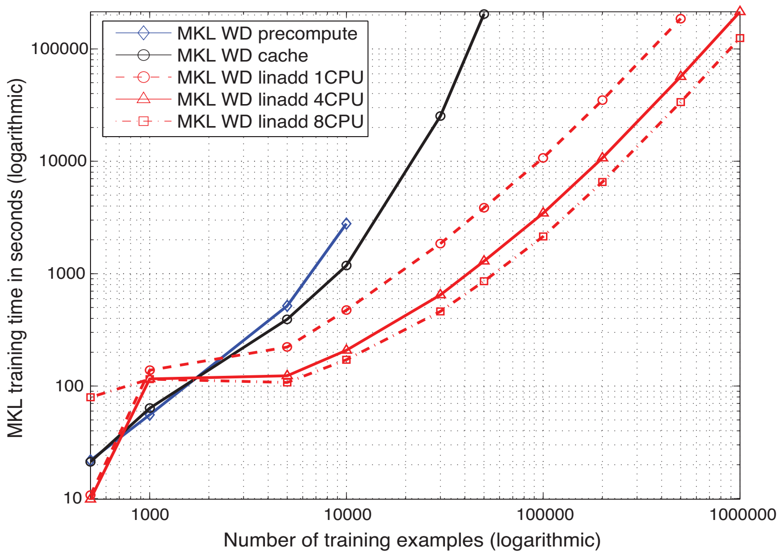


Figure 4.6: Comparison of the running time of the different MKL algorithms when used with the weighted degree kernel. Note that as this is a log-log plot, small appearing distances are large for larger N and that each slope corresponds to a different exponent. Figure taken from Sonnenburg et al. (2006a).

times for the different MKL algorithms applied to learning weightings of the WD kernel on the splice site classification task. To do so, several MKL-SVMs were trained using precomputed kernel matrices (*PreMKL*), kernel matrices, which are computed on the fly employing kernel caching (*MKL*[7]), MKL using the `linadd` extension (*LinMKL1*) and `linadd` with its parallel implementation[8] (*LinMKL4* and *LinMKL8* - on 4 and 8 CPUs). The results are displayed in Table 4.1 and in Figure 4.6. While precomputing kernel matrices seems beneficial, it cannot be applied to large scale cases (e.g., $> 10,000$ examples) due to the $\mathcal{O}(KN^2)$ memory constraints of storing the kernel matrices.[9] On-the-fly-computation of the kernel matrices is computationally extremely demanding, but since kernel caching[10] is used, it is still possible on 50,000 examples in about 57 hours. Note that no WD-kernel specific optimisations are involved here, so one expects a similar result for arbitrary kernels.
The `linadd` variants outperform the other algorithms by far (speedup factor 53 on 50,000 examples) and are still applicable to datasets of size up to one million. Note that without parallelisation MKL on one million examples would take more than a week, compared with 2.5 (2) days in the quad-CPU (eight-CPU) version. The parallel versions outperform the single processor version from the start achieving a speedup for 10,000 examples of 2.27 (2.75), quickly reaching a plateau at a speedup factor of 2.98 (4.49) at a level of $50,000$ examples and approaching a speedup factor of 3.28 (5.53) on 500,000 examples (efficiency: 82% (69%)). Note that the performance gain using 8 CPUs is relatively small, e.g., solving the QP and constructing the tree is not parallelised.

[7]Algorithm 4.2

[8]Algorithm 4.2 with the `linadd` extensions including parallelisation of Algorithm 3.3

[9]Using 20 kernels on 10,000 examples requires already 7.5GB, on 30,000 examples 67GB would be required (both using single precision floats)

[10]Each kernel has a cache of 1GB

N	*PreMKL*	*MKL*	*LinMKL1*	*LinMKL4*	*LinMKL8*
500	22	22	11	10	80
1,000	56	64	139	116	116
5,000	518	393	223	124	108
10,000	2,786	1,181	474	209	172
30,000	-	25,227	1,853	648	462
50,000	-	204,492	3,849	1292	857
100,000	-	-	10,745	3,456	2,145
200,000	-	-	34,933	10,677	6,540
500,000	-	-	185,886	56,614	33,625
1,000,000	-	-	-	214,021	124,691

Table 4.1: Speed Comparison when determining the WD kernel weight by Multiple Kernel Learning using the chunking algorithm (MKL) and MKL in conjunction with the (parallelised) `linadd` algorithm using 1, 4, and 8 processors (*LinMKL1, LinMKL4, LinMKL8*). The first column shows the sample size N of the data set used in SVM training while the following columns display the time (measured in seconds) needed in the training phase. Table taken from Sonnenburg et al. (2006a).

4.2 Positional Oligomer Importance Matrices

For many sequence classification problems, SVMs with the right choice of sequence kernels perform better than other state-of-the-art methods, as exemplified in Table 4.2. To achieve the best prediction results, it typically pays off to rather include many, potentially weak features than to manually preselect a small set of discriminative features. For instance, the SVM-based translation initiation start (TIS) signal detector `Startscan` (Saeys et al., 2007), which relies on a relatively small set of carefully designed features, shows a considerably higher error rate than an SVM with a standard kernel that implies a very high dimensional feature space (cf. Table 4.2).
The best methods in Table 4.2 are all based on SVMs that work in feature spaces that exhaustively represent the incidences of all k-mers up to a certain maximum length K using the Spectrum kernel with and without mismatches and the Weighted Degree kernel without and with shifts (cf. Chapter 2 for the definition of these kernels). As a result of our work on large scale learning (see Chapter 3) SVMs with string kernels can be trained efficiently on millions of DNA sequences even for large orders K, thereby inducing enormous feature spaces (for instance, $K = 30$ gives rise to more than $4^{30} > 10^{18}$ k-mers). Such feature spaces supply a solid basis for accurate predictions as they allow to capture complex relationships (e.g., binding site requirements). From an application point of view, however, they are yet unsatisfactory as they offer little scientific insight about the nature of these relationships. The reason is that SVM classifiers $\hat{y} = \operatorname{sign}(g(\mathbf{x}))$ employ the kernel expansion,

$$g(\mathbf{x}) \quad = \quad \sum_{i=1}^{N} \alpha_i y_i \,\mathrm{k}(\mathbf{x}_i, \mathbf{x}) + b, \tag{4.14}$$

where $(\mathbf{x}_i, y_i)_{i=1,\dots,N}$, with $y_i \in \{+1, -1\}$, are the N training examples (cf. Section 1.2). Thus, SVMs use a weighting $\boldsymbol{\alpha}$ over training examples that only indirectly relates to features. One idea to remedy this problem is to characterise input variables by their correlation with the weight vector $\boldsymbol{\alpha}$ (Üstün et al., 2007); however, the importance of features in the induced feature space remains unclear.
Partial relief is offered by multiple kernel learning (cf. Section 4.1). For appropriately

Signal Detection Problem to be solved	SVM Performance	SVM Based Approach and String Kernel Names	Performance of Competitor	Competing Approach
Transcription Start	26.2% auPRC	WDS & Spectrum (*ARTS*, cf. Section 5.4 and Sonnenburg et al., 2006b)	11.8% auPRC	RVM (`Eponine`, Down and Hubbard, 2002)
Acceptor Splice Site	54.4% auPRC	WDS (cf. Section 5.3 and Sonnenburg et al., 2007b)	16.2% auPRC	IMC (cf. Section 5.3 and Sonnenburg et al., 2007b)
Donor Splice Site	56.5% auPRC	WDS (cf. Section 5.3 and Sonnenburg et al., 2007b)	25.0% auPRC	IMC (cf. Section 5.3 and Sonnenburg et al., 2007b)
Alternative Splicing	89.7% auROC	WDS (`RASE`, Rätsch et al., 2005)	-	-
Trans-Splicing	95.2% auROC	WD (`mGene`, Schweikert et al., 2008)	-	-
Translation Initiation	10.7% Se80	WD (`mGene`, Schweikert et al., 2008)	12.5% Se80	PWM, ICM (`Startscan`, Saeys et al., 2007)

Table 4.2: Comparison of SVM performance (second column) vs. competing state-of-the-art classifiers (third column) on six different DNA signal detection problems. The chosen best SVMs employ spectrum, weighted degree (WD) or weighted degree with shift (WDS) kernels. Note that performance measures differ. AuROC denotes the area under the receiver operator characteristic curve and auPRC the area under the precision recall curve; for both, larger values correspond to better performance. Se80 is the false positive rate at a true positive rate of 80% (lower values are better). Transcription start site (TSS): Among the best TSS recognisers is the relevance vector machine (RVM) based Eponine, which is clearly outperformed by our *ARTS* TSS detector (cf. Section 5.4). Acceptor and donor splice sites: The best existing splice site detectors are SVM based (cf. Section 5.3 and Sonnenburg et al., 2007b); we therefore deliberately compare our methods to the popular inhomogeneous Markov chains (IMC), which achieve less than half of the auPRC on human splice sites. Alternative and trans-splice sites in *C. elegans*: To the best of our knowledge no other ab-initio approaches are available. Translation initiation sites: For TIS recognition we compare with Startscan (Saeys et al., 2007), which is based on positional weight matrices (PWM) and interpolated context models (ICM). Our WD-kernel SVM, trained using default settings $C = 1$, $d = 20$ (no model selection) on the dataset from Saeys et al. (2007), already performs favourably. Table taken from Sonnenburg et al. (2008).

designed sub-kernels $\mathrm{k}_m(.,.)$, the optimised kernel combination coefficients $\boldsymbol{\beta}$ can then be used to highlight the parts of an input sequence that are important for discrimination (Rätsch et al., 2006). The use of the l_1-norm constraint ($\sum_{m=1}^{M} \beta_m = 1$) causes the resulting $\boldsymbol{\beta}$ to be sparse, however at the price of discarding relevant features, which may lead to inferior performance.

An alternative approach is to keep the SVM decision function unaltered, and to find adequate ways to "mine" the decision boundary for good explanations of its high accuracy. A natural way is to compute and analyze the normal vector of the separation in feature space, $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \Phi(\mathbf{x}_i)$, where Φ is the feature mapping associated to the kernel k. This has been done, for instance, in cognitive sciences to understand the differences in human perception of pictures showing male and female faces. The resulting normal vector $\mathbf{w}$ was relatively easy to understand for humans since it can be represented as an image (Graf et al., 2006). Such approach is only feasible if there exists an explicit and manageable representation of Φ for the kernel at hand. Fortunately, as we have seen in Chapter 3, we can compute such weight vector for most string kernels, which leads to weightings over all possible k-mers. However, it seems considerably more difficult to represent such weightings in a way humans can easily understand. There have been

first attempts in this direction (Meinicke et al., 2004), but the large number of k-mers and their dependence due to overlaps at neighbouring positions still remain an obstacle in representing complex SVM decision boundaries.
In this section, we address this problem by considering new measures for k-mer based scoring schemes (such as SVMs with string kernels) useful for the understanding of complex local relationships that go beyond the well-known sequence logos. For this, we first compute the *importance* of each k-mer (up to a certain length K) at each position as its expected contribution to the total score $g(\mathbf{x})$. The resulting *Positional Oligomer Importance Matrices (POIMs)* can be used to rank and visualise k-mer based scoring schemes. Note that a ranking based on $\mathbf{w}$ is not necessarily meaningful: due to the dependencies of the features there exist $\mathbf{w}' \neq \mathbf{w}$ that implement the same classification, but yield different rankings. In contrast, our importance values are well-defined and have the desired semantics. The lowest order POIM ($p = 1$) essentially conveys the same information as is represented in a sequence logo. However, unlike sequence logos, POIMs naturally generalise to higher-order nucleotide patterns.

Organization of the Section This section is split into four parts. First, we introduce the necessary background (Section 4.2.1), then define POIMs in Section 4.2.2 and provide recursions for their efficient computation. In Section 4.2.3 we describe representations and methods to visualise POIMs. Finally, in Section 4.2.4, we use artificial data to show that POIMs easily out-compete MKL and the SVM weight $\mathbf{w}$. Note that Chapter 5 will utilize POIMs of state-of-the-art SVM-based signal detectors to exactly pin-point length, location, and typical sequences of the most discriminating motifs.

4.2.1 Linear Positional Oligomer Scoring Systems

Given an alphabet Σ, let $\mathbf{x} \in \Sigma^{l_\mathbf{x}}$ be a sequence of length $l_\mathbf{x}$. A sequence $\mathbf{y} \in \Sigma^k$ is called a k-mer or oligomer of length k. A *positional oligomer (PO)* is defined by a pair $(\mathbf{y}, i) \in \mathcal{I} := \bigcup_{k=1}^{K} \left(\Sigma^k \times \{1, \ldots, l_\mathbf{x} - k + 1\}\right)$, where $\mathbf{y}$ is the subsequence of length k and i is the position at which it begins within the sequence of length $l_\mathbf{x}$. We consider scoring systems of order K (that are based on positional oligomers of lengths $k \leq K$) defined by a *weighting function* $w : \mathcal{I} \to \mathbb{R}$. Let the *score* $s(\mathbf{x})$ be defined as a sum of PO weights:

$$s(\mathbf{x}) \quad := \quad \sum_{k=1}^{K} \sum_{i=1}^{l_\mathbf{x}-k+1} w\Big(\mathbf{x}[i]^k, i\Big) + b \ , \tag{4.15}$$

where b is a constant offset (bias), and we write $\mathbf{x}[i]^k := x_i x_{i+1} \ldots x_{i+k-1}$ to denote the substring of $\mathbf{x}$ that starts at position i and has length k. Figure 4.3 gives an example of the scoring system. Thus, if one can explicitly store the learnt normal vector $\mathbf{w}$ parametrizing the hyperplane, one can use it to rank and extract discriminating features. Many classifiers implement such a scoring system as will be shown below.

The Weighted Degree Kernel. The *weighted degree* kernel of order K (cf. Equation (2.4) and also Figure 2.3 in Section 2.3) for illustration. A feature mapping $\Phi(\mathbf{x})$ is defined by a vector representing each positional oligomer of length $\leq K$: if it is present in $\mathbf{x}$ then the vector entry is $\sqrt{\beta_k}$ and 0 otherwise. It can be easily seen that $\Phi(\mathbf{x})$ is an explicit representation of the WD kernel, i.e., $\mathrm{k}(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$. When training any kernel method the function $g(\mathbf{x})$ can be equivalently computed as $g(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) + b$, where $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \Phi(\mathbf{x}_i)$. Given the feature map of the WD kernel, it becomes

k-mer	pos. 1	pos. 2	pos. 3	pos. 4	$\cdots$
A	+0.1	-0.3	-0.2	+0.2	$\cdots$
C	0.0	-0.1	+2.4	-0.2	$\cdots$
G	+0.1	-0.7	0.0	-0.5	$\cdots$
T	-0.2	-0.2	0.1	+0.5	$\cdots$
AA	+0.1	-0.3	+0.1	0.0	$\cdots$
AC	+0.2	0.0	-0.2	+0.2	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
TT	0.0	-0.1	+1.7	-0.2	$\cdots$
AAA	+0.1	0.0	0.0	+0.1	$\cdots$
AAC	0.0	-0.1	+1.2	-0.2	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
TTT	+0.2	-0.7	0.0	0.0	$\cdots$

Table 4.3: Example of a linear scoring system

apparent that the resulting $\mathbf{w}$ is a weighting over all possible positional oligomers and, hence, $g(\mathbf{x})$ can be written in the form of Equation (4.15).

Other kernels. The above findings extend to many other string kernels from Chapter 2, the Oligomer kernel and the related *WD kernel with shift*, which additionally include displaced matches. The *(weighted) spectrum kernel* can be modeled with Equation (4.15) by using equal weights at each position $i \in \{1, \ldots, l_{\mathbf{x}}\}$ (analogously for the *spectrum kernel with mismatches*).

Markov Model of Order d. Recall from Section 1.2 that in a Markov model of order d it is assumed that each symbol in a sequence $\mathbf{x}$ is independent of all other symbols in $\mathbf{x}$ that have distance greater than d. Thus, the likelihood of $\mathbf{x}$ can be computed as a product of conditional probabilities:

$$\Pr[\mathbf{x}] = \prod_{i=1}^{d} \Pr\left[\mathbf{x}_i \,\middle|\, \mathbf{x}[1]^{i-1}\right] \cdot \prod_{i=d+1}^{l_{\mathbf{x}}} \Pr\left[\mathbf{x}_i \,\middle|\, \mathbf{x}[i-d]^{d}\right] .$$

Note that a positional weight matrix (PWM) is essentially a Markov model of order $d = 0$. The log-likelihood for $d = K - 1$ is easily expressed as a linear positional oligomer scoring system for $K-$mers as follows:

$$\log \Pr[\mathbf{x}] = \sum_{i=1}^{K-1} w\big(\mathbf{x}[1]^{i}, 1\big) + \sum_{i=K}^{l_{\mathbf{x}}} w\big(\mathbf{x}[i-K+1]^{K}, i-K+1\big) \quad ,$$

with $w\big(\mathbf{x}[1]^{i}, 1\big) = \log \Pr\left[x_i \,\middle|\, \mathbf{x}[1]^{i-1}\right]$ and $w\big(\mathbf{x}[i-K+1]^{K}, i-K+1\big) = \log \Pr\left[x_i \,\middle|\, \mathbf{x}[i-K+1]^{K-1}\right]$. A log-likelihood ratio is modeled by the difference of two such scoring systems (the posterior log-odds, cf. Section 1.2). This derivation applies to both homogeneous (position-independent) and in-homogeneous (position-dependent) Markov models. It further extends to the decision rules of mixed order models.

4.2.2 Definition of Positional Oligomer Importance Matrices

The goal of POIMs is to characterise the importance of each PO $(\mathbf{z}, j)$ for the classification. On first sight, looking at the scoring function (Equation (4.15)), it might seem that the weighting w does exactly this: $w(\mathbf{z}, j)$ seems to quantify the contribution of $(\mathbf{z}, j)$ to the score of a sequence $\mathbf{x}$. In fact, this is true whenever the POs are statistically independent. However, already for $K \geq 2$ many POs overlap with each other and thus necessarily are dependent. To assess the influence of a PO on the classification, the impact of all dependent POs has to be taken into account. For example, whenever $\mathbf{z}$ occurs at position j, the weights of all its positional substrings are also added to the score (Equation (4.15)).

To take dependencies into account, we define the *importance* of $(\mathbf{z}, j)$ as the expected increase (or decrease) of its induced score. We quantify this through the conditional expectation of $s(\mathbf{X})$ conditioned on the occurrence of the PO in $\mathbf{x}$ (i.e., on $\mathbf{X}[j] = \mathbf{z}$):[11]

$$Q(\mathbf{z}, j) \;:=\; \mathbb{E}\left[\, s(\mathbf{X}) \mid \mathbf{X}[j] = \mathbf{z} \,\right] - \mathbb{E}\left[\, s(\mathbf{X}) \,\right] \;. \tag{4.16}$$

This is the central equation of this work. When evaluated for all positions j and all p-mers $\mathbf{z}$, it results in a *positional oligomer importance matrix (POIM).*

The relevance of Equation (4.16) derives from the meaning of the score $s(\cdot)$. Higher absolute values $|s(\mathbf{x})|$ of scores $s(\mathbf{x})$ indicate higher confidence about the classifiers decision. Consequently, high $|Q(\mathbf{z}, j)|$ show that the presence of a p-mer $\mathbf{z}$ at position j in an input sequence is highly relevant for class separation. For example, with an SVM trained for splice sites a high positive score $s(x)$ suggests that $\mathbf{x}$ contains a splice site at its central location while a very negative score suggests that there were none. Thus, a PO $(\mathbf{z}, j)$ of high absolute importance $|Q(\mathbf{z}, j)|$ might be part of a splice consensus sequence, or a regulatory element (an enhancer for positive importance $Q(\mathbf{z}, j) > 0$, or a silencer for $Q(\mathbf{z}, j) < 0$).

The computation of the expectations in Equation (4.16) requires a probability distribution for the *union* of both classes. In this section, we use a zeroth-order Markov model , i.e., an independent single-symbol distribution at each position. Although this is quite simplistic, the approximation that it provides is sufficient for our applications. A generalisation to Markov models of order $d \geq 0$ can be found in Appendix A.2.

There are two strong reasons for the subtractive normalisation w.r.t. the (unconditionally) expected score. The first is conceptual: the magnitude of an expected score is hard to interpret by itself without knowing the cutoff value for the classification; it is more revealing to see how a feature would *change* the score. The second reason is about computational efficiency; this is shown next.

Efficient computation of $Q(\mathbf{z}, j)$. Naive implementation of Equation (4.16) would require a summation over all $4^{l_\mathbf{x}}$ sequences $\mathbf{x}$ of length $l_\mathbf{x}$, which is clearly intractable. The following equalities show how the computational cost can be reduced in three steps;

[11] We omit the length of the subsequence in comparisons where it is clear from the context, e.g. $\mathbf{X}[j] = \mathbf{z}$ means $\mathbf{X}[j]^{|\mathbf{z}|} = \mathbf{z}$.

for proofs and details see Appendix A.2.

$$\begin{aligned} & Q(\mathbf{z}, j) \\ = & \sum_{(\mathbf{y},i)\in\mathcal{I}} w(\mathbf{y}, i)\big[\Pr[\mathbf{X}[i] = \mathbf{y} \mid \mathbf{X}[j] = \mathbf{z}] - \Pr[\mathbf{X}[i] = \mathbf{y}]\big] \qquad (4.17) \\ = & \begin{cases} \displaystyle\sum_{(\mathbf{y},i)\in\mathcal{I}(\mathbf{z},j)} w(\mathbf{y}, i) \Pr[\mathbf{X}[i] = \mathbf{y} \mid \mathbf{X}[j] = \mathbf{z}] \\ \displaystyle -\sum_{(\mathbf{y},i)\not\perp(\mathbf{z},j)} w(\mathbf{y}, i) \Pr[\mathbf{X}[i] = \mathbf{y}] \end{cases} \qquad (4.18) \\ = & u(\mathbf{z}, j) - \sum_{\mathbf{z}'\in\Sigma^{|\mathbf{z}|}} \Pr\big[\mathrm{x}[j] = \mathbf{z}'\big]\, u(\mathbf{z}', j) \ . \qquad (4.19) \end{aligned}$$

Here u is defined by

$$u(\mathbf{z}, j) \ := \sum_{(\mathbf{y},i)\in\mathcal{I}(\mathbf{z},j)} \Pr[\mathbf{X}[i] = \mathbf{y} \mid \mathbf{X}[j] = \mathbf{z}]\, w(\mathbf{y}, i) \ , \qquad (4.20)$$

$\mathcal{I}(\mathbf{z}, j)$ is the set of features that are dependent and compatible with $(\mathbf{z}, j)$, and $\not\perp$ indicates that two POs are dependent. Two POs are compatible if they agree on all positions they share. For example, $(\texttt{TATA}, 30)$, and $(\texttt{AAA}, 32)$ are incompatible, since they share positions $\{31, 33\}$ but disagree on position 32, whereas $(\texttt{TATA}, 30)$ and $(\texttt{TACCA}, 32)$ are compatible. Note also that for Markov chains of order zero statistical dependence of two POs is equivalent with them being overlapping.

The complexity reduction works as follows. In Equation (4.17), we use the linear structure of both the scoring function and the expectation to reduce summation from all $l_{\mathbf{x}}$-mers to the set $\mathcal{I}$ of all k-mers. This is still prohibitive: for example, the number of POs for $K{=}20$ and $l_{\mathbf{x}}{=}100$ is roughly $\mathcal{O}(10^{14})$. In Equation (4.18), we exploit the facts that the terms for POs independent of $(\mathbf{z}, j)$ cancel in the difference, and that conditional probabilities of incompatible POs vanish. Finally, probability calculations lead to Equation (4.19), in which the difference is cast as a mere weighted sum of auxiliary terms u.

To compute u, we still have to access the SVM weights $\mathbf{w}$. Note that, for high orders K, the optimal $\mathbf{w}$ found by any kernel method (e.g., an SVM) is sparse as a consequence of the representer theorem (Schölkopf and Smola, 2002): the number of nonzero entries in $\mathbf{w}$ is bounded by the number of POs present in the training data, which for the spectrum and WD kernels grows linearly with the training data size. Due to this sparsity, $\mathbf{w}$ can efficiently be computed and stored in positional suffix tries (cf. Chapter 3).

Recursive algorithm for $Q(\mathbf{z}, j)$. Even with the summation over the reduced set $\mathcal{I}(\mathbf{z}, j)$ as in Equation (4.19) and (4.20), naive sequential computation of the importance of all POs of moderate order may easily be too expensive. We therefore develop a strategy to compute the entire matrix of values $Q(\mathbf{z}, j)$, for all p-mers $\mathbf{z}$ up to length P at all positions $1, \ldots L - p + 1$, which takes advantage of shared intermediate terms. This results in a recursive algorithm operating on string prefix data structures and therefore is efficient enough to be applied to real biological analysis tasks.
The crucial idea is to treat the POs in $\mathcal{I}(\mathbf{z}, j)$ separately according to their relative position to $(\mathbf{z}, j)$. To do so, we subdivide the set $\mathcal{I}(\mathbf{z}, j)$ into substrings, superstrings,

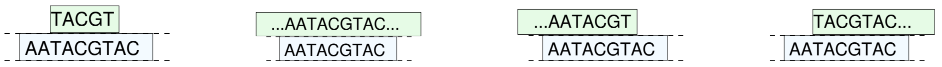


Figure 4.7: Substrings, superstrings, left partial overlaps, and right partial overlaps: definition and examples for the string AATACGTAC. Figure taken from Sonnenburg et al. (2008).

left partial overlaps, and right partial overlaps of $(\mathbf{z}, j)$:

Definition 4.1 (Substrings, Superstrings, Left and Right PO).

$$\begin{array}{rlcr}
\textit{(substrings)} & \mathcal{I}^{\vee}(\mathbf{z},j) & := & \{\, (\mathbf{y},i) \in \mathcal{I}(\mathbf{z},j) \mid i \geq j \text{ and } |\mathbf{y}| + i \leq |\mathbf{z}| + j \,\} \\
\textit{(superstrings)} & \mathcal{I}^{\wedge}(\mathbf{z},j) & := & \{\, (\mathbf{y},i) \in \mathcal{I}(\mathbf{z},j) \mid i \leq j \text{ and } |\mathbf{y}| + i \geq |\mathbf{z}| + j \,\} \\
\textit{(left p. o.)} & \mathcal{I}^{<}(\mathbf{z},j) & := & \{\, (\mathbf{y},i) \in \mathcal{I}(\mathbf{z},j) \mid i < j \text{ and } |\mathbf{y}| + i < |\mathbf{z}| + j \text{ and } |\mathbf{y}| + i - 1 \geq j \,\} \\
\textit{(right p. o.)} & \mathcal{I}^{>}(\mathbf{z},j) & := & \{\, (\mathbf{y},i) \in \mathcal{I}(\mathbf{z},j) \mid i > j \text{ and } |\mathbf{y}| + i > |\mathbf{z}| + j \text{ and } |\mathbf{z}| + j - 1 \geq i \,\}
\end{array}$$

Figure 4.7 illustrates the four cases.

The function u can be decomposed correspondingly; see Equation (4.21) in Theorem 4.2. This theorem also summarises all the previous observations in the central POIM computation Theorem 4.2. The complete derivation can be found in Appendix A.2. Once $\mathbf{w}$ is available as a suffix trie (cf. Chapter 3), the required amounts of memory and computation time for computing POIMs are dominated by the size of the output, i.e., $\mathcal{O}(|\Sigma|^p \cdot l_{\mathbf{x}})$. The recursions are implemented in the SHOGUN toolbox (cf. Appendix C) (which also offers a non-positional version for the spectrum kernel). A web interface via Galaxy is available at `http://galaxy.fml.tuebingen.mpg.de/`.

Theorem 4.2 (Efficient POIM computation for zeroth order Markov chains). *Let $\mathbf{z} \in \Sigma^p$ and $Q(\mathbf{z}, j)$ be defined as in Equation* (4.16). *Then*

$$Q(\mathbf{z},j) \;=\; u(\mathbf{z},j) - \sum_{\mathbf{z}' \in \Sigma^p} \Pr\left[\, \mathbf{X}[j] = \mathbf{z}' \,\right] u(\mathbf{z}',j) \;, \tag{4.21}$$

where u decomposes as

$$u(\mathbf{z},j) \;=\; u^{\vee}(\mathbf{z},j) + u^{\wedge}(\mathbf{z},j) + u^{<}(\mathbf{z},j) + u^{>}(\mathbf{z},j) - w(\mathbf{z},j)$$

and $u^{\vee}$, $u^{\wedge}$, $u^{<}$, $u^{>}$ are computed recursively by

$$\begin{aligned}
u^{\vee}(\sigma\mathbf{z}\tau,j) &= w_{(\sigma\mathbf{z}\tau,j)} + u^{\vee}(\sigma\mathbf{z},j) + u^{\vee}(\mathbf{z}\tau,j+1) - u^{\vee}(\mathbf{z},j+1) \quad \textit{for } \sigma,\tau \in \Sigma \\
u^{\wedge}(\mathbf{z},j) &= w_{(\mathbf{z},j)} - \sum_{(\sigma,\tau)\in\Sigma^2} \Pr\left[\,\mathbf{X}[j-1] = \sigma\,\right] \Pr\left[\,\mathbf{X}[j+p] = \tau\,\right] u^{\wedge}(\sigma\mathbf{z}\tau,j-1) \\
&\quad + \sum_{\sigma\in\Sigma} \Pr\left[\,\mathbf{X}[j-1] = \sigma\,\right] u^{\wedge}(\sigma\mathbf{z},j-1) + \sum_{\tau\in\Sigma} \Pr\left[\,\mathbf{X}[j+p] = \tau\,\right] u^{\wedge}(\mathbf{z}\tau,j) \\
u^{<}(\mathbf{z},j) &= \sum_{\sigma\in\Sigma} \Pr\left[\,\mathbf{X}[j-1] = \sigma\,\right] \sum_{l=1}^{\min\{p,K\}-1} L\left(\sigma(\mathbf{z}[1]^l), j-1\right) \\
u^{>}(\mathbf{z},j) &= \sum_{\tau\in\Sigma} \Pr\left[\,\mathbf{X}[j+p] = \tau\,\right] \sum_{l=1}^{\min\{p,K\}-1} R\left(\mathbf{z}[p-l+1]^l \tau, j+p-l\right) \;,
\end{aligned}$$

$$\begin{aligned} L(\mathbf{z}, j) &= w_{(\mathbf{z},j)} + \sum_{\sigma \in \Sigma} \Pr\left[\, \mathbf{X}\left[j-1\right] = \sigma \,\right] L(\sigma\mathbf{z}, j-1) \\ R(\mathbf{z}, j) &= w_{(\mathbf{z},j)} + \sum_{\tau \in \Sigma} \Pr\left[\, \mathbf{X}\left[j+p\right] = \tau \,\right] R(\mathbf{z}\tau, j) \ . \end{aligned}$$

This theorem enables efficient POIM computation for zeroth order Markov chains. For its derivation cf. Appendix A.2.

4.2.3 Ranking Features and Condensing Information for Visualisation

Given the POIMs, we can analyze positional oligomers according to their contributions to the scoring function. We discuss several methods to visualise the information in POIMs and to find relevant POs. In the remainder of this section, we will use the term motif either synonymously for PO, or for a set of POs that are similar (e.g., that share the same oligomer at a small range of neighbouring positions).

Ranking Tables. A simple analysis of a POIM is to sort all POs by their importance $Q(\cdot)$ or absolute importance $|Q(\cdot)|$. As argued above, the highest ranking POs are likely to be related to relevant motifs. An example of a ranking tables can be found in Table 5.7 and Figure 5.15.

POIM Plots. We can visualise an entire POIM of a fixed order $p \leq 3$ in form of a heat map: each PO $(\mathbf{z}, j)$ is represented by a cell, indexed by its position j on the x-axis and the p-mer $\mathbf{z}$ on the y-axis (e.g., in lexical ordering), and the cell is colored according to the importance value $Q(\mathbf{z}, j)$. This allows to quickly overview the importance of all possible POs at all positions. An example for $p = 1$ can be found in Figure 4.10. There also the corresponding sequence logo is shown.
However, for $p > 3$ such visualisation ceases to be useful due to the exploding number of p-mers. Similarly, ranking lists may become too long to be accessible to human processing. Therefore, we need views that aggregate over all p-mers, i.e., that show PO importance just as a function of position j and order p. Once a few interesting areas (j, p) are identified, the corresponding POs can be further scrutinised (e.g., by looking at rank lists) in a second step. In the following paragraphs we propose a two approaches for the first, summarizing step.

POIM Weight Mass. At each position j, the total importance can be computed by summing the absolute importance of all p-mers at this position, $\text{weight\_mass}_p(j) = \sum_{\mathbf{z} \in \Sigma^p} |Q(\mathbf{z}, j)|$. Several such curves for different p can be shown simultaneously in a single graph. An example can be found in Figure 4.11, first row, second column.

Differential POIMs. Here we start by taking the maximum absolute importance over all p-mers for a given length p and at a given position j. We do so for each $p \in \{1, \ldots, P\}$ and each $j \in \{1, \ldots, l_{\mathbf{x}}\}$. Then, we subtract the maximal importance of the two sets of $(p-1)$-mers covered by the highest scoring p-mer at the given position (see Figure 4.8). This results in the importance gain by considering longer POs. As we demonstrate in the next section, this allows to determine the length of relevant motifs. The first row of Figure 4.9 shows such images.

POIM Diversity. A different way to obtain an overview over a large number of p-mers for a given length p is to visualise the distribution of their importances. To do so, we

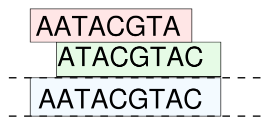


Figure 4.8: Two $(p-1)$-mers are covered by a p-mer. Figure taken from Sonnenburg et al. (2008).

$$\begin{aligned} q_{\max}^{p,j} &:= \max_{\mathbf{z}\in|\Sigma|^p} |Q(\mathbf{z},j)| \\ D(p,j) &:= q_{\max}^{p,j} - \\ &\quad \max\{q_{\max}^{p,j}, q_{\max}^{p,j+1}\} \end{aligned} \tag{4.22}$$

approximate the distribution at each position by a mixture of two normal distributions, thereby dividing them into two clusters. We set the values in the lower cluster to zero, and sort all importances in descending order. The result is depicted in a heat map just like a POIM plot. Here we do not care that individual importance values cannot be visually mapped to specific p-mers, as the focus is on understanding the distribution. An example is given in Figure 4.11, first row, third column.

p-mer Scoring Overview. Previously, the scoring and visualisation of features learnt by SVMs was performed according to the values of the weight vector $\mathbf{w}$ (Meinicke et al., 2004, Sonnenburg et al., 2007a). To show the benefit of our POIM techniques in comparison to such techniques, we also display matrices visualizing the position-wise maximum of the absolute value of *the raw weight vector* $\mathbf{w}$ over all possible p-mers at this position, $\mathrm{KS}(p,j) = \max_{\mathbf{z}\in\Sigma^p} |w(\mathbf{z},j)|$. We will also display the *Weight Plots* and the *Weight Mass* for the weight matrices SVM-$\mathbf{w}$ in the same way as for the importance matrices. Differential plots for SVM-$\mathbf{w}$ do not make sense, as the weights for different orders are not of the same order of magnitude. The second row of Figure 4.9 shows such overview images.

4.2.4 POIMs Reveal Discriminative Motifs

As a first step, we demonstrate our method on two simulations with artificial data and show that POIMs reveal motifs that remain hidden in sequence logos and SVM weights.

Two motifs at fixed positions. In our first example we re-use a toy data set of Appendix B.1: two motifs of length seven, with consensus sequences `GATTACA` and `AGTAGTG`, are planted at two fixed positions into random sequences (Sonnenburg et al., 2005a). To simulate motifs with different degrees of conservations we also mutate these consensus sequences, and then compare how well different techniques can recover them. More precisely, we generate a data set with 11,000 sequences of length $l_{\mathbf{x}} = 50$ with the following distribution: at each position the probability of the symbols $\{\mathtt{A},\mathtt{T}\}$ is 1/6 and for $\{\mathtt{C},\mathtt{G}\}$ it is 2/6. We choose 1,000 sequences to be positive examples: we plant our two POs, $(\mathtt{GATTACA}, 10)$ and $(\mathtt{AGTAGTG}, 30)$, and randomly replace s symbols in each PO with a random letter. We create four different versions by varying the mutation level $s \in \{0, 2, 4, 5\}$. Each data sets is randomly split into 1,000 training examples and 10,000 validation examples. For each s, we train an SVM with WD kernel of degree 20 exactly as in Sonnenburg et al. (2005a). We also reproduce the results of Sonnenburg et al. (2005a) obtained with a WD kernel of degree 7, in which MKL is used to obtain a sparse set of relevant pairs of position and degree.
The results can be seen in Figure 4.9. We start with the first column, which corresponds to the unmutated case $s = 0$. Due to its sparse solution, MKL (third row) characterises the positive class by a single 3-mer in the `GATTACA`. Thus, MKL fails to identify the

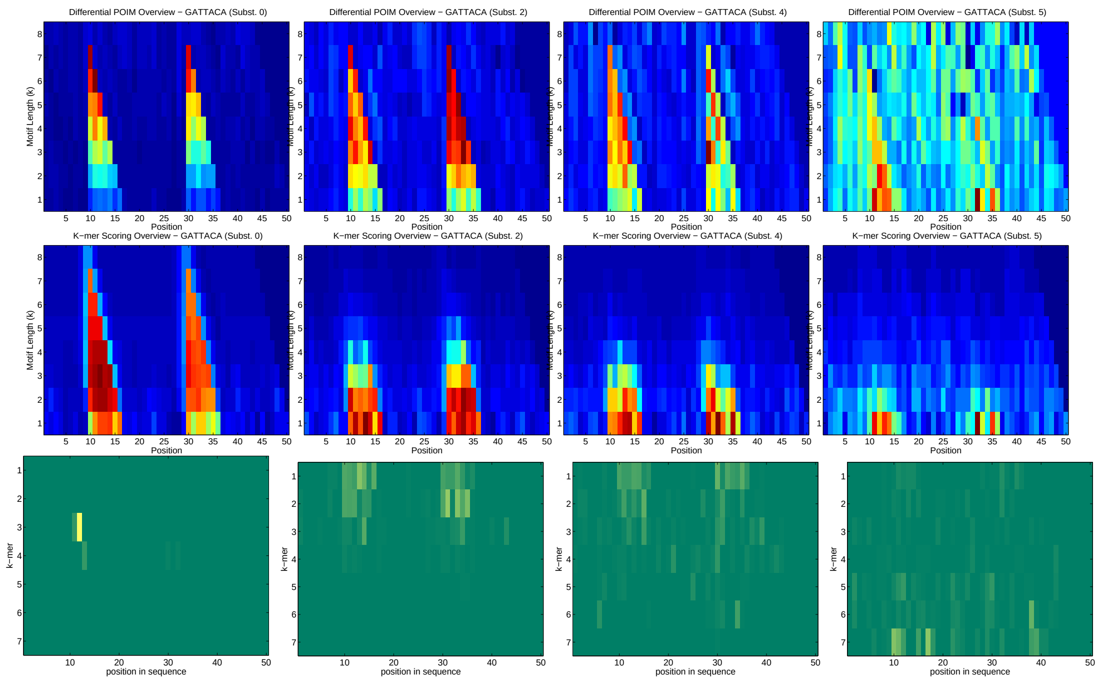


Figure 4.9: Comparison of different visualisation techniques for the fixed-position-motifs experiment. Motifs `GATTACA` and `AGTAGTG` were inserted at positions 10 and 30, respectively, with growing level of mutation (i.e., number of nucleotides randomly substituted in the motifs) from left to right. SVMs classifiers were trained to distinguish random sequences from sequences with the (mutated) motifs `GATTACA` and `AGTAGT` inserted. **(First Row)** We computed Differential POIMs (see Equation (4.22)) for up to 8-mers, from a WD-kernel SVM of order 20. Here each figure displays the importance of $k-$mer lengths (y-axis) for $k = 1 \dots 8$ at each position (x-axis) ($i = 1 \dots 50$ as a heat map. Red and yellow color denotes relevant motifs, dark blue corresponds to motifs not conveying information about the problem. $1-$mers are at the bottom of the plot, 8-mers at the top. **(Second Row)** K-mer Scoring Overview (SVM-**w**) was computed using the same setup as for differential POIMs. The SVM-**w** is again displayed as a heat map. **(Third Row)** was obtained using multiple kernel learning (averaged weighting obtained using 100 bootstrap runs (Rätsch et al., 2006)). Again the result is displayed as a heat map, but for one to seven-mers only. For a more detailed discussion, see text. Figure taken from Sonnenburg et al. (2008).

complete consensus sequences. The K-mer scoring matrices (second row) are able to identify two clusters of important oligomers at and after positions 10 and 30; however they assign highest impact to shorter oligomers. Only the differential POIMs (first row) manage to identify the exact length of the embedded POs: they do not only display that 7-mers up to $k = 7$ are important, but also that exactly 7-mers at position 10 and 30 are most important.
Moving through the columns to the left, the mutation level increases. Simultaneously the performance deteriorates, as expected. In the second column (2/7 mutations), differential POIMs still manage to identify the exact length of the embedded motifs, unlike the other approaches. For higher mutation rates even the POIMs assign more importance to shorter POs, but they continue to be closer to the truth than the two other approaches. In Figure 4.10 we show the POIM plot for 1-mers versus sequence logos for this level of mutation. As one can see, 1-mer POIMs can capture the whole information contained in sequence logos.

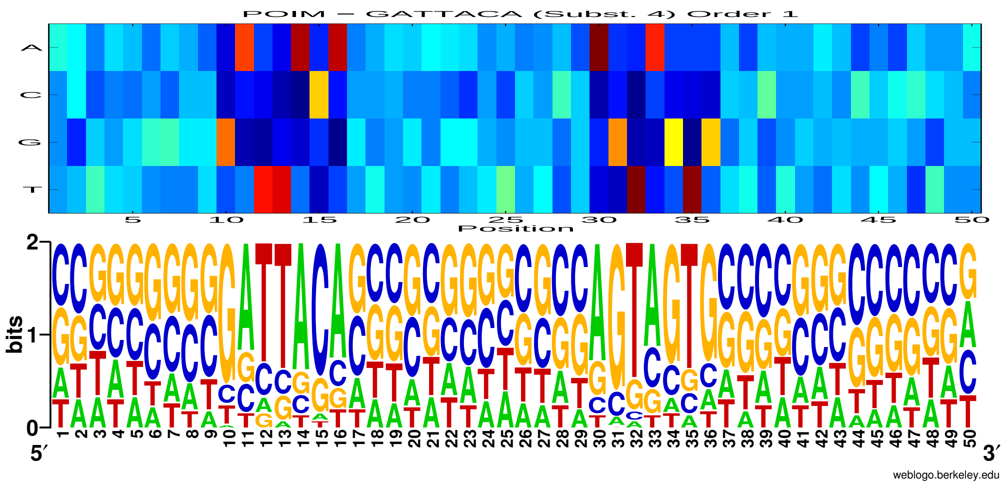


Figure 4.10: 1-mer POIM plot (**top**) vs. sequence logo (**bottom**) for motifs `GATTACA` and `AGTAGTG` at positions 10 and 30, respectively, with 4-out-of-7 mutations in the motifs. Figure taken from Sonnenburg et al. (2008).

Mutated motif at varying positions. To make our experiment more realistic, we consider motifs with positional shift. The first 5000 training and test sequences are created by drawing uniformly from $\{\mathtt{A},\mathtt{C},\mathtt{G},\mathtt{T}\}^{100}$. For half of the sequences, the positive class, we randomly insert the 7-mer `GATTACA`, with one mutation at a random position. The position j of insertion follows a normal distribution with mean 0 and standard deviation 7 (thus for 50% of the cases, $j \in [-5, +5]$). We train an SVM with WDS kernel of degree 10 with constant shift 30 to discriminate between the inseminated sequences and the uniform ones; it achieves an accuracy of 80% on the test set. As displayed in Figure 4.11, both the sequence logo and the SVM $\mathbf{w}$ fail to make the consensus and its length apparent, whereas the POIM-based techniques identify length and positional distribution. Please note that Gibbs sampling methods have been used to partially solve this problem for PWMs. Such methods can also be used in conjunction with SVMs.

4.3 Summary

Modern kernel methods with complex, oligomer-based sequence kernels are very powerful for biological sequence classification. However, until now no satisfactory tool for visualisation was available that helps to understands their complex decision surfaces. In the first part of the chapter we have proposed a simple, yet efficient algorithm to solve the multiple kernel learning problem for a large class of loss functions. The proposed method is able to exploit the existing single kernel algorithms, thereby extending their applicability. In experiments, we have illustrated that MKL for classification and regression can be useful for automatic model selection and for obtaining comprehensible information about the learning problem at hand. It would be of interest to develop and evaluate MKL algorithms for unsupervised learning such as Kernel PCA and one-class classification and to try different losses on the kernel weighting $\boldsymbol{\beta}$ (such as L_2). We proposed performance enhancements to make large scale MKL practical: the SILP wrapper, SILP chunking and (for the special case of kernels that can be written as an inner product of sparse feature vectors, e.g., string kernels) the `linadd` algorithm.For MKL, we gained a speedup of factor 53. Finally, we proposed a parallel version of the `linadd` algorithm running on a 8 CPU multiprocessor system, which lead to *additional*

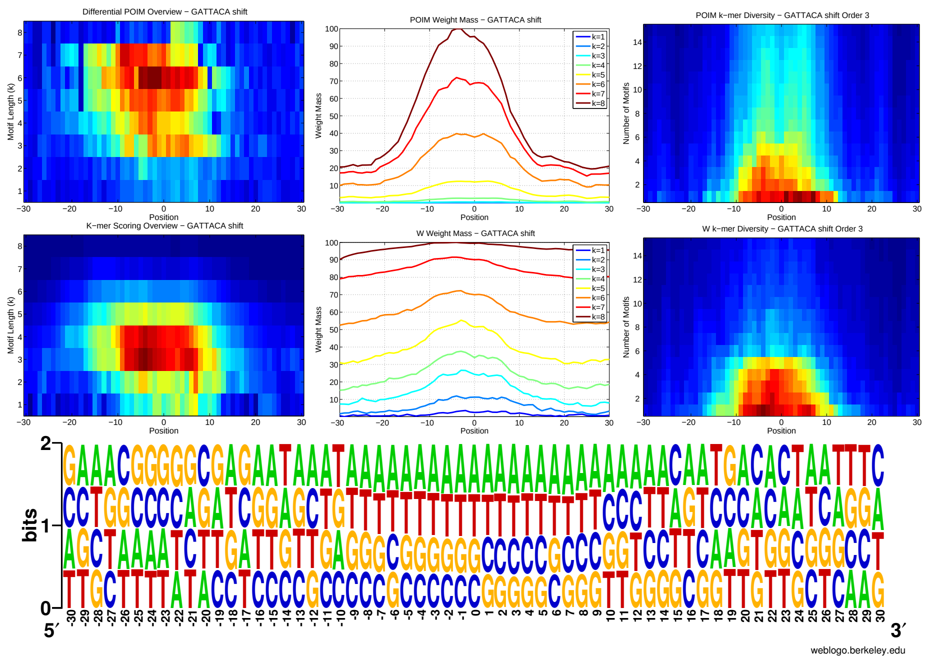


Figure 4.11: Comparison of different visualisation techniques for the varying-positions-motif experiment. The mutated motif GATTACA was inserted at positions $0+-13$ in uniformly distributed sequences. **(First row)** The first column shows the Differential POIM matrices (cf. Equation (4.22)) as a heat map, the POIM weight mass for different $k = 1\dots 8$ and the POIM k-mer diversity for $k = 3$ as a heat map ; **(Second row)** Shows the SVM-**w** overview plot as a heat map, the SVM-**w** weight mass also for $k = 1\dots 8$ and the $k-$mer diversity for $k = 3$ as a heat map; **(Third row)** sequence logo. Figure taken from Sonnenburg et al. (2008).

speedups of factor up to 5.5 for MKL. In the second part of this Chapter, we propose a solution to visualising the complex decision function of the SVM by introducing a method, which efficiently computes the *importance* of positional oligomers defined as their expected contribution to the score. In contrast to the discrimination normal vector **w**, the importance takes into account the correlation structure of all features. We illustrated on simulated data how the visualisation of positional oligomer importance matrices (POIMs) can help to identify even vaguely localised motifs where pure sequence logos will fail. In contrast to our previous MKL approach, we propose here to leave the SVM untouched for classification to retain its high accuracy, and to defer motif extraction to subsequent steps. This way, motif finding can take advantage of the SVMs power: it ensures that the truly most important POs are identified. However, MKL still proves useful when the kernel feature space is not as easily accessible as is the case with string kernels. MKL even in such cases aids interpretation of the learnt SVM classifier.
We believe that our new MKL and POIM-based ranking and visualisation algorithms are an easy to use yet very helpful analysis tool. They are freely available as part of the SHOGUN toolbox (c.f. Section C). Finally, note that it seems possible to transfer the underlying concept of POIMs to other kernels and feature spaces to aid understanding of SVMs that are used for other biological tasks, which currently is the domain of MKL.

5 Accurate Signal and Content Sensors

With the generation of whole genome sequences, important insight into gene functions and genetic variation has been gained over the last decades. As novel sequencing technologies are rapidly evolving, the way will be paved for cost efficient, high-throughput whole genome sequencing, which is going to provide the community with massive amounts of sequences. It is self-evident that the handling of this wealth of data will require efficient and accurate computational methods for sequence analysis. Among the various tools in computational genetic research, gene prediction remains one of the most prominent tasks, as recent competitions have further emphasised (e.g., Bajic et al. (2006), Stein et al. (2007)). Accurate gene prediction is of prime importance for the creation and improvement of annotations of recently sequenced genomes (Rätsch et al., 2007, Bernal et al., 2007). In the light of new data related to natural variation (e.g., Hinds et al. (2005), The International HapMap Consortium (2005), Clark et al. (2007)), the importance of accurate computational gene finding gains increasing importance since it helps to understand the effects of polymorphisms on the gene products.
Ab initio gene prediction is a highly sophisticated procedure as it mimics – in its result – the labour of several complex cellular machineries at a time: identification of the beginning and the end of a gene, as is accomplished by RNA polymerases; splicing of the nascent RNA, in the cell performed by the spliceosome; and eventually the detection of an open reading frame, as does the ribosome. The success of a gene prediction method therefore relies on the accuracy of each of these components. After a review of performance measures in Section 5.1 and a brief description of the data generation process (Section 5.2) we will discuss applications of support vector machines using the proposed string kernels to improve the detections of the signal and content sensor components, such as the detection of splice sites (Section 5.3) transcription starts (Section 5.4) and to distinguish exons from introns. Finally, Section 5.5 summarises how string kernel based SVMs can be used for a variety of signal and content sensors and gives some general guidelines on how to approach such problems using string kernel SVMs. This chapter is largely based on Sonnenburg et al. (2006b, 2007b), Schweikert et al. (2008).

5.1 Performance Measures

An important question in machine learning is the question of how to select the "best" classifier. Given a sample $\mathbf{x}_1, \ldots, \mathbf{x}_N \subset \mathcal{X}$, and a two-class classifier $f(\mathbf{x}) \mapsto \{-1, +1\}$ with outputs outputs $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)$ for $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and the true labeling $(y_1, \ldots, y_N) \in \{-1, +1\}^N$, the $N-$sample is partitioned into four partitions,

1. the positively labelled and correctly positive predicted examples (true positives),
2. the positively labelled and wrongly negative predicted examples (false negatives),
3. the negatively labelled and correctly negative predicted examples (true negatives) and

4. the negatively labelled and wrongly positive predicted examples (false positives).

The *Contingency Table* (see Table 5.1; sometimes also called confusion matrix) is a way to summarise these four types of errors (Fawcett, 2003). Here TP, FN, FP, TN denote the cardinality of the true positives et cetera on the $N-$ sample. Using these

outputs\ labeling	$y = +1$	$y = -1$	Σ
$f(x) = +1$	TP	FP	O^+
$f(x) = -1$	FN	TN	O^-
Σ	N^+	N^-	N

Table 5.1: The Contingency Table / Confusion Matrix. N denotes the sample size, $N^+ = FN + TP$ and $N^- = FP + TN$ the number of positively and negatively labelled examples, $O^+ = TP + FP$ and $O^- = FN + TN$ the number of positive and negative predictions and TP, FP, FN, TN are absolute counts of true positives, false positives, false negatives and true negatives.

four scores, several performance measures can be defined, each of which is emphasising a certain aspect of the data.

Name	**Computation**
accuracy	$\text{ACC} = \frac{TP+TN}{N}$
error rate (1-accuracy)	$\text{ERR} = \frac{FP+FN}{N}$
balanced error rate	$\text{BER} = \frac{1}{2}\left(\frac{FN}{FN+TP} + \frac{FP}{FP+TN}\right)$
weighted relative accuracy	$\text{WRACC} = \frac{TP}{TP+FN} - \frac{FP}{FP+TN}$
F1 score (harmonic mean between precision/recall)	$\text{F1} = \frac{2*TP}{2*TP+FP+FN}$
cross correlation coefficient	$\text{CC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$
sensitivity/recall	$\text{TPR} = TP/N^+ = \frac{TP}{TP+FN}$
specificity	$\text{TNR} = TN/N^- = \frac{TN}{TN+FP}$
1-sensitivity	$\text{FNR} = FN/N^+ = \frac{FN}{FN+TP}$
1-specificity	$\text{FPR} = FP/N^- = \frac{FP}{FP+TN}$
positive predictive value / precision	$\text{PPV} = TP/O^+ = \frac{TP}{TP+FP}$
false discovery rate	$\text{FDR} = FP/O^+ = \frac{FP}{FP+TP}$
negative predictive value	$\text{NPV} = TN/O^- = \frac{TN}{TN+FN}$
negative false discovery rate	$\text{FDR}_- = FN/O^- = \frac{FN}{FN+TN}$

Table 5.2: Several commonly used performance measures. N^+ denotes the number of positive examples and N^- the number of negative examples in the $N-$sample. O^+ and O^- are the number of positively and negatively predicted examples on the $N-$sample (cf. Table 5.1).

In machine learning, classically the *accuracy* (fraction of correctly classified examples on a test set) or the *test error* (fraction of wrongly classified examples on a test set) is used to compare the performance of classifiers. A variety of performance measures is displayed in Table 5.2.

Comparison of Performance Measures While the accuracy and test error measures work well on balanced datasets, i.e., datasets with a similar number of positively and negatively labelled examples, they are not very meaningful for unbalanced datasets. For example, on a 100 sample, with 95 negatively and 5 positively labelled examples the classifier $f(\mathbf{x}) = -1$ will only have 5% test error. Furthermore, the cost of a false positive and false negative may not be the same and thus one may want to adjust the bias of the classifier to reflect that uneven cost model. To address these issues the concept of the Receiver Operator Characteristic (ROC) Curve (e.g., Metz (1978), Fawcett (2003)) can be used. It is obtained by plotting the FPR against the TPR for different bias values as displayed in Figure 5.1. The more to the top left a ROC curve is the better performs its corresponding classifier. This way several classifiers can be compared in a single figure for each (FPR,TPR) pair and the classifier outperforms its opponents that

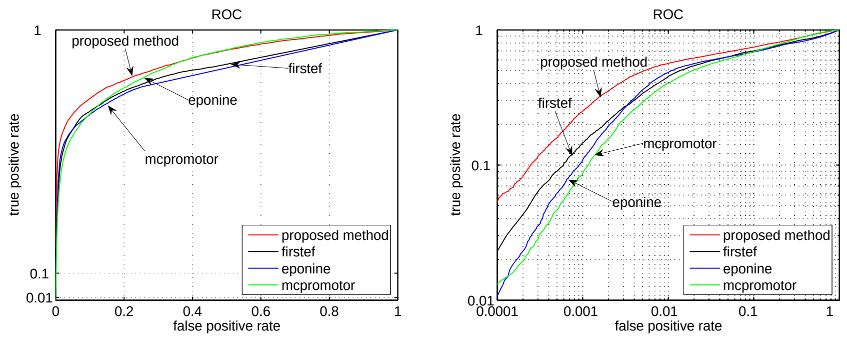


Figure 5.1: The Receiver Operator Characteristic (ROC) Curve. It is obtained using by varying the bias and recording TPR and FPR values. Note that a ROC curve is independent of class skew (class ratio) and is monotonely ascending. To put more emphasise on the region with low FPR, one may display a log-log plot as is done in the right figure to zoom into that region. Figure taken from Sonnenburg et al. (2006b).

achieves the highest TPR for a given FPR, where the FPR is chosen based on the cost model. Plotting the true positive rate against the positive predictive value (also called precision) PPV $= TP/(FP+TP)$, i.e., the fraction of correct positive predictions among all positively predicted examples, one obtains the Precision Recall Curve (PRC) (see e.g., Davis and Goadrich (2006)). As evaluating a classifier based on one of the scalar

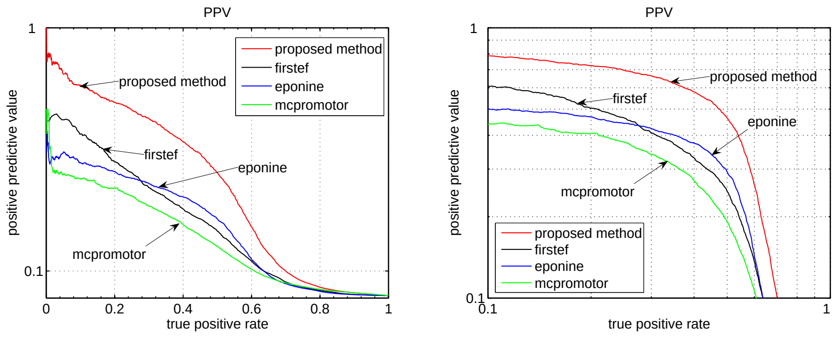


Figure 5.2: The Precision Recall Curve, is obtained by varying the bias and recording PPV and TPR. It is dependent on class skew and may not be monotone. It therefore can be more meaningful for very unbalanced datasets than the ROC curve. Figure taken from Sonnenburg et al. (2006b).

measures in Table 5.2 constitutes only a single point on the ROC or precision recall curve, one should in practice (if possible) perform a comparison based on the full curves. This way, the practitioner may choose the method, which outperforms its competitors for threshold of interest. Nevertheless, although scalar performance measures (often) hide the complexity of model selection, they are required to perform machine driven model

selection. Both curves allow for the definition of such practically relevant measures. To define a scalar performance measure that asks a classifier to perform well (i.e., achieve high true positive rate for ROC or high precision for PRC) over the whole range of biases, one may use the area under the curve. We refer to these scalar measures as *auROC* (for area under ROC) and *auPRC*, respectively. By definition, both measures are independent of class bias and are thus more robust in contrast to the scalars in Table 5.2. Note that the auROC has a statistical meaning and is in fact equivalent to the Wilcoxon test of ranks (Fawcett, 2003): If one considers the output on the $N-$sample as ranking, the area under the ROC curve is the number of swappings such that output of positive examples is greater or equal the output of negative examples divided by $N^+ \cdot N^-$. In contrast to the auROC the auPRC is *dependent* on class skew and thus may be considered more appropriate for extremely unbalanced datasets as appear in genomic sequence analysis tasks, e.g., splice or transcription start site prediction. When performing human transcription start site (TSS) prediction, we have about 6 billion loci of which, even for optimistic guesses, less than 3 million base pairs, i.e., only 0.05%, belong to true TSS. Let us consider a transcription start site finder (TSF) that correctly classifies 100% of the true TSS sites (TPR) while wrongly classifying 1% of the non-TSS loci (FPR). The area under the ROC would score at least 99% suggesting a particularly good classifier. However, if only 0.05% of the negative examples (which in absolute values is 300 million) achieve a higher score than all of the positive examples (3 million), the area under the Precision Recall Curve will be less than 1%.
For a reliably useful measure of prediction accuracy, one should thus resort to the auPRC for very unbalanced data sets. A way to make the auROC more meaningful on unbalanced datasets is to consider the auROC$_N$ - the area under the ROC curve up to the first N false-positives (or N% FPR) (McClish, 1989, Gribskov and Robinson, 1996).
To summarise, for balanced datasets it may be sufficient to use standard test error, accuracy or the auROC to compare competing classifiers in a (semi-)automatic model selection. For unbalanced data sets the balanced error, F1, auPRC and the cross-correlation coefficient may be more appropriate. Finally, for human judgement, the ROC or precision recall curve obtained on a separate held out test set (putting emphasis on the relevant region) should be used to display the final result.

5.2 Generating Training and Test Data Sets

While we postpone most of the details of the quite complex data generation procedures to the appendix (cf. Appendix B), tuning the data generation process is as crucial as the machine learning algorithms dealing with the data.[1]

Data Generation Procedure for Signal Detectors

One usually starts with a (partial) genome, available as FASTA[2] file. In addition, another database hinting at the locations of the signal is required.
For transcription start sites, this could be a database of transcription starts (dbTSS, Suzuki et al. (2002)) or Cap-analysis gene expression (CAGE) tags (Kawaji et al., 2006) that already contain the coordinates of all the signal-sites.

[1]In practice it is often very time demanding and little details turn out to be big issues later on. Therefore, running sanity checks, trying to visualise and understand every single problem about might turn out to be highly beneficial.

[2]`http://www.ncbi.nlm.nih.gov/blast/fasta.shtml`

For splice sites the picture is slightly more difficult: One first obtains full length complementary DNA (cDNA) or expressed sequence tags (EST). cDNA is the complement of fully spliced mRNA. ESTs are small pieces of mRNA normally of < 700 nucleotides in length and obtained after only one round[3] of sequencing (Burset et al., 2000). In principle, the sequences are obtained outside the nucleus from cytoplasmic extracts. Recall that the splicing process does happen in the nucleus. It is therefore almost certain that ESTs, being found outside the nucleus, were already processed, i.e., *spliced.* As a result one can detect splice sites, by aligning ESTs or cDNAs to DNA. Since the latter still contains *introns*, one can figure out the start of the intron (i.e., 5' site) and the end of the intron (3' site). Note that given enough full length cDNAs (or even ESTs employing an additional clustering step) the same procedure could be applied to obtain transcription start and other sites.

For two-class classification problems, one is required to generate positively and negatively labelled training examples. From the genomic DNA and the signal coordinates, one can now extract windows around the signal, as is displayed for acceptor splice sites in Figure 5.3.

```
0            25                        49-50                         75            100
CT...GTAGAG C| TGTACGTACGACGTACGTCAAGCT AG GAGCGCAACGTACGTACGTCCAGT |A GAAGGT...G A
```

Figure 5.3: A window around an acceptor splice site. A potentially extracted positive example is shown framed. Figure taken from Sonnenburg (2002).

In this case, windows of length 50 (c.f. Figure 5.3) were extracted. To generate final positively labelled examples, one has to specify the window size (how many nucleotides up-and downstream of the signal) and decide what one has to do with duplicates (duplicates are more likely for smaller window sizes) and incomplete data (e.g., N's).

```
0               27                        52-53                         78         100
CT...GTAGAGCTG T| ACGTACGACGTACGTCAAGCTAGG AG CGCAACGTACGTACGTCCAGTAGA |A AGGT...G A
```

Figure 5.4: Extraction of a 50 base-pair decoy window, by shifting the window 3 positions downstream to the next occurrence of AG. All decoys are constructed such that AG is at positions $25 - 26$ too. Figure taken from Sonnenburg (2002).

The construction of *useful* negatively labelled examples is more difficult: In the end one wants windows of the same size that do not contain the signal of interest at the exact same or a nearby position. The question is how one can be certain that a window is "negative"? Unfortunately, this question cannot be answered in general and (often) also not without further assumptions. For example, for acceptor splice sites one may assume to have full EST or cDNA coverage and thus constructs negative examples by extracting all windows of length 50 that seem to be canonical splice sites, i.e., contain AG at positions $24 - 25$ from a fully EST or cDNA covered region (c.f. Figure 5.4).

When dealing with splice sites, the samples are aligned such that AG appears at the same position in all samples, while each sample consists of a fixed number of bases around the site.

For transcription starts, one may assume a complete coverage of CAGE tags and sample examples from the whole genome as negatives that are not in the vicinity of true

[3]Usually the pieces of DNA are sequenced more than once to get 99.99% accuracy. Furthermore, only parts of a chromosomes can be sequenced. Thus, overlapping sequences must be aligned properly to form a fully sequenced chromosome.

```
-1 AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
+1 AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT
-1 CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC
+1 TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC
+1 TTGTTTTAATATTCAATTTTTTACAGTAAGTTGCCAATTCAATGTTCCAC
-1 CTGTATTCAATCAATATAATTTTCAGAAACCACACATCACAATCATTGAA
-1 TACCTAATTATGAAATTAAAATTCAGTGTGCTGATGGAAACGGAGAAGTC
```

Figure 5.5: An example that illustrates how splice site samples $(y_i, \mathbf{x}_i)$ are constructed. The left column shows the labelling with +1 denoting a true splice site and -1 a decoy. The right colum denotes the corresponding sequence, generated by taking windows of fixed width around the splice site (or decoy site), while `AG` is aligned to be at the same position in all samples. In case of true splice sites, the upstream part including the `AG` is intronic, while the rest is exonic.

transcription start sites. Alternatively, one could sample negative sites from the interior of a gene (assuming to have full coverage within a gene).
But even if all of these issues have been solved, one needs to deal with duplicates and overlapping sequences (between positive and negative examples and within the same class). Furthermore, to generate a fair test data set one might want to remove sequences that are too similar to the training sequences (e.g., sequences where a blat alignment has $> 30\%$ overlap). Then, what should be done about alternative sites (alternative splicing, transcription starts)? And the list of potential problems goes on like this. While constructing the data set, one has to consciously think about the assumptions used in each design decision. Finally, note that data generation is a moving target with new sequencing techniques becoming available all the time.

To perform the experiments in this thesis, we generated data sets of donor and acceptor splice sites and transcription start sites from public genomic databases. The splicing datasets were generated by extracting windows around true and decoy splice sites. This was done by aligning EST and cDNA sequences to the genome. Assuming full EST or cDNA coverage, negative examples were extracted from the interior of a gene (see Appendix B.2 for further details). For transcription start sites we again extracted windows around true and decoy sites. The true sites were obtained from dbTSS and decoy sites again from the interior of the gene (Appendix B.3 contains further details).

5.3 Accurate Splice Site Recognition

In this section, we will focus on the improvement of signal sensors for the detection of splice sites, as this sub-problem is a core element of any gene finder. A comprehensive understanding of splice sites is not only a prerequisite for splice form prediction but can also be of great value in localizing genes (Burge and Karlin, 1997, Reese et al., 1997, Salzberg et al., 1998, Delcher et al., 1999, Pertea et al., 2001).

In eukaryotic genes, splice sites mark the boundaries between exons and introns (cf. Section 1.1 for an introduction on that topic). The latter are excised from premature mRNAs in a post-processing step after transcription. Both the donor sites at the exon-intron junctions, and the acceptor sites at the intron-exon boundaries, have quite strong consensus sequences, which can, however, vary significantly from one organism to another. The vast majority of all splice sites are so-called *canonical splice sites*, which are characterised by the presence of the dimers `GT` and `AG` for donor and acceptor sites, respectively. The occurrence of the dimer is not sufficient for the splice site. Indeed, it occurs very frequently at non-splice site positions. For example, in human DNA, which is $\approx 6 \cdot 10^9$ nucleotides in size, `GT` can be found about 400 million times (summed over both strands). For some crude estimate of say $2 \cdot 10^4$ genes with 20 exons each, only 0.1% of the consensus sites are true splice sites. We therefore face two extremely unbalanced classification tasks, namely the discrimination between true donor sites and decoy positions with the consensus dimer `GT` or `GC` (the only non-canonical splice site dimer that we will consider) and the discrimination between true acceptor sites and decoy positions with the consensus dimer `AG`.

Relation to Previous Work Although present-day splice site detectors (e.g., based on Support Vector Machines, neural networks, hidden Markov models) are reported to perform at a fairly good level (Reese et al., 1997, Rampone, 1998, Cai et al., 2000, Rajapakse and Ho, 2005), several of the reported performance numbers should be interpreted with caution, for several reasons. First of all, these results are based on *small* and potentially biased data sets. Since many genomes have been fully sequenced, these results will need to be re-evaluated. Second, issues in generating negative examples (decoys) were, if recognised, often not sufficiently documented. The choice of data sets, in particular the decoys, can make a tremendous difference in the measured performance. Third, often only the single site prediction of acceptor and donor sites is considered, whereas the higher goal is to use the splice site predictor within a gene finder. It is uncertain how good the predictors perform in this setting. Keeping these in mind, we provide unbiased *genome-wide* splice site prediction, which enables further evaluation in gene finders.
We will apply Support Vector Machines (SVMs) using the string kernels described in Chapter 2 to the recognition of splice sites. In Sonnenburg et al. (2002), Sonnenburg (2002) we demonstrated that SVMs using kernels from probabilistic hidden Markov models (cf. Jaakkola and Haussler (1999), Tsuda et al. (2002a)) outperform hidden Markov models *alone*. As this approach did not scale to many training examples, we performed a comparison of different *faster* methods for splice site recognition (Rätsch and Sonnenburg, 2004), where we considered Markov models and SVMs with different kernels: the *locality improved kernel*, originally proposed for recognition of translation initiation sites (Zien et al., 2000); the *SVM-pairwise kernel*, using alignment scores (Liao and Noble, 2002); the *TOP kernel*, making use of a probabilistic model (cf. Jaakkola and Haussler (1999), Tsuda et al. (2002a)); the standard *polynomial kernel* (Vapnik, 1995); and the *weighted degree kernel* (Rätsch and Sonnenburg, 2004, Rätsch et al., 2005) – all reviewed in Chapter 2.
Other groups also reported successful SVM based splice site detectors. In Yamamura and Gotoh (2003) it was proposed to use linear SVMs on binary features computed from di-nucleotides, an approach that also outperformed previous Markov models. An even more accurate model called SpliceMachine (Degroeve et al., 2005a), not only used positional information (one- to trimers) around the splice site, but also explicitly the compositional context using tri- to hexamers. To the best of our knowledge, this approach is the current state-of-the art, outperforming previous SVM based approaches as well as GeneSplicer (Pertea et al., 2001) and GeneSplicerESE (Pertea et al., 2007). In Zhang et al. (2006) linear SVMs were used on positional features that were extracted from empirical estimates of unconditional positional probabilities. Note that this approach is similar to our TOP kernel method on zeroth-order Markov chains

(Rätsch and Sonnenburg, 2004). Recently, Baten et al. (2006a) reported improved accuracies for splice site prediction also by using SVMs. The method employed in Baten et al. (2006a) is very similar to a kernel initially proposed in Zien et al. (2000) (*Salzberg kernel*). The idea of this kernel is to use empirical estimates of conditional positional probabilities of the nucleotides around splice sites (estimated by Markov models of first order), which are then used as input for classification by an SVM.
Many other methods have been proposed for splice site recognition. For instance, multilayer neural networks with Markovian probabilities as inputs (Rajapakse and Ho, 2005). They train three Markov models on three segments of the input sequence, the upstream, signal and downstream segments. Although they outperform Baten et al. (2006a) on small datasets, the authors themselves write that the training of the neural networks is especially slow when the number of true and decoy examples are imbalanced and that they have to downsample the number of negatives for training even on small and short sequence sets. Therefore, their method does not seem suitable for large-scale genome-wide computations. Finally, Chen et al. (2005) proposed a method based on Bayesian Networks, which models statistical dependencies between nucleotide positions.
In this work we will compare a few of our previously considered methods against these approaches and show that the engineering of the kernel, the careful choice of features and a sound model selection procedure are important for obtaining accurate predictions of splice sites.

Our previous comparison in Rätsch and Sonnenburg (2004) was performed on a relatively small data set derived from the *C. elegans* genome. In addition, the data sets considered in Baten et al. (2006a) are relatively small (around 300,000 examples, whereas more than 50,000,000 examples are nowadays readily available). In this study, we therefore reevaluate the previous results on much larger data sets derived from the genomes of five model organisms, namely *Caenorhabditis elegans* ("worm"), *Arabidopsis thaliana* ("cress"), *Drosophila melanogaster* ("fly"), *Danio rerio* ("fish"), and *Homo sapiens* ("human"). Building on our recent work on large scale learning (cf.Chapter 3), we now are able to train and evaluate Support Vector Machines on such large data sets as is necessary for analyzing the whole human genome. In particular, we are able to show that increasing the number of training examples indeed helps to obtain a significantly improved performance, and thus will help to improve existing annotation (see, e.g., Rätsch et al. (2007)). We train and evaluate SVMs on newly generated data sets using nested cross-validation and provide genome-wide splice site predictions for any occurring `GT`, `GC` and `AG` dimers. We will show that the methods in some cases exhibit dramatic performance differences for the different data sets.

Organisation of the Section In the next sub-section we briefly describe the methods to approach the splice site detection problem and the details of cross-validation and model selection. All data sets in the pilot study have been obtained from the supplementary material of the corresponding papers (cf. Appendix B.2.1). The data sets for the genome-wide studies were generated according to the procedures described in Section 5.2 and in very detail Appendix B.2.2. Finally, we present the outcomes of (a) the comparison with the methods proposed in Baten et al. (2006a), Pertea et al. (2001), Degroeve et al. (2005a), Chen et al. (2005) (Section 5.3.2), (b) an assessment determining the window length that should be used for classification (Section 5.3.2) and, finally, (c) a comparison of the large scale methods on the genome-wide data sets for the five considered genomes (Section 5.3.3). After discussing our results, we also revisit the question about the interpretability of SVMs.

5.3.1 Model Selection and Evaluation

We use posterior log-odds of inhomogeneous Markov chains (cf. Section 1.2) and Support Vector Machines with the Locality Improved kernel (cf. Section 2.2), the Weighted Degree and Weighted Degree kernel *with shifts* (cf. Section 2.3) to detect the splicing signal. To be able to apply SVMs, we have to find the optimal soft margin parameter C (Schölkopf and Smola, 2002) and the kernel parameters. These are: For the LI-kernel, the degree d and window size l; for the WD kernel, the degree d; and for the WDS kernel, the degree d and the shift parameter σ (see Chapter 2 for details). For MCs, we have to determine the order d of the Markov chain and the pseudocounts for the models of positive and the negative examples (see the posterior log-odds section). To tune these parameters, we perform the cross-validation procedures described below.

NN269 and DGSplicer For the pilot study, we are learning acceptor and donor splice site detectors on the NN269 and the DGSplicer data sets originating from Reese et al. (1997) and Baten et al. (2006a) and described in detail in Appendix B.2.1. The training and model selection of our methods for each of the four tasks was done separately by partial 10-fold cross-validation on the training data. For this, the training sets for each task are divided into 10 equally-sized data splits, each containing the same number of splice sequences and the same proportion of true versus decoy sequences. For each parameter combination, we use only 3 out of the 10 folds, that is we train 3 times by using 9 out of the 10 training data splits and evaluate on the remaining training data split. Since the data is highly unbalanced, we choose the model with the highest average auPRC score on the three evaluation sets. This best model is then trained on the complete training data set. The final evaluation is done on the corresponding independent test sets (the same as in Baten et al. (2006a)). Appendix B.5 includes tables with all parameter combinations used in model selection for each task and the selected model parameters.

Worm, Fly, Cress, Fish, and Human The training and model selection of our methods for the five organisms on the acceptor and donor recognition tasks was done separately by 5-fold cross-validation. The optimal parameter was chosen by selecting the parameter combination that maximised the auPRC score. This model selection method was nested within 5-fold cross-validation for final evaluation of the performance. The reported auROC and auPRC are averaged scores over the five cross-validation splits. In Appendix B.5 the optimal parameter settings for each task are listed. All splits were based on the clusters derived from EST and cDNA alignments, such that different splits come from random draws of the genome.
In the following subsection, we discuss experimental results obtained with our methods for acceptor and donor splice site predictions for the five considered organisms.
As discussed in Section 5.1, we measure our prediction accuracy in terms of area under the Receiver Operator Characteristic Curve (auROC) (Metz, 1978, Fawcett, 2003) and area under the Precision Recall Curve (auPRC) (e.g., (Davis and Goadrich, 2006)). The auPRC is a better measure for performance, if the class distribution is very unbalanced. However, it does depend on the class priors on the test set and hence is affected by sub-sampling the decoys, as happened with the data sets used in previous studies (NN269 in Baten et al. (2006a) contains about 4 times more decoy than true sites, DGSplicer

in Baten et al. (2006a), Chen et al. (2005) about 140 times more; in contrast, in the genome scenario the ratio is one to 300 − 1000). To compare the results among the different data sets with different class sizes, we therefore also provide the auROC score that is not affected by sub-sampling.

5.3.2 Pilot Studies

Performance on the NN269 and DGSplicer Data Sets For the comparison of our SVM classifiers to the approaches proposed in Baten et al. (2006a), Chen et al. (2005), we first measure the performance of our methods on the four tasks used in Baten et al. (2006a) (cf. Appendix B.2.2). The approach in Baten et al. (2006a) is outperformed by a neural network approach proposed in Rajapakse and Ho (2005). However, we do not compare our methods to the latter method, since it already reaches its computational limits for the small datasets with only a few thousand short sequences (cf. Rajapakse and Ho (2005), page 138) and hence is not suitable for large-scale genome-wide computations. On each task we trained SVMs with the *weighted degree kernel* (WD) (Rätsch and Sonnenburg, 2004), and the *weighted degree kernel with shifts* (WDS) (Rätsch et al., 2005). On the NN269 Acceptor and Donor sets we additionally trained an SVM using the *locality improved kernel* (LIK) (Zien et al., 2000); as it gives the weakest prediction performance and is computationally most expensive we exclude this model from the following investigations. As a benchmark method, we also train higher-order Markov Chains (MCs) (e.g., Durbin et al. (1998)) of "linear" structure and predict with the posterior log-odds ratio (cf. Section 1.2). Note that Position Specific Scoring Matrices (PSSM) are recovered as the special case of zeroth-order MCs. A summary of our results showing the auROC and auPRC scores is displayed in Table 5.3. We

	MC	EBN	MC-SVM	LIK	WD	WDS
NN269						
Acceptor						
auROC	96.78	-	96.74†	98.19	98.16	98.65
auPRC	88.41	-	88.33†	92.48	92.53	94.36
Donor						
auROC	98.18	-	97.64†	98.04	98.50	98.13
auPRC	92.42	-	89.57†	92.65	92.86	92.47
DGSplicer						
Acceptor						
auROC	97.23	95.91*	95.35*	-	97.50	97.28
auPRC	30.59	-	-	-	32.08	28.58
Donor						
auROC	98.34	96.88*	95.08*	-	97.84	97.47
auPRC	41.72	-	-	-	39.72	35.59

Table 5.3: Performance evaluation (auROC and auPRC scores) of six different methods on the NN269 and DGSplicer Acceptor and Donor test sets (for further details on the data sets see Appendix B.2.2). MC denotes prediction with a Markov Chain, EBN the method proposed in Chen et al. (2005), and MC-SVM the SVM based method described in Baten et al. (2006a) (similar to Zien et al. (2000)). The remaining methods are based on SVMs using the locality improved kernel (LIK) (Zien et al., 2000), weighted degree kernel (WD) (Rätsch and Sonnenburg, 2004) and weighted degree kernel with shifts (WDS) (Rätsch et al., 2005). The values marked with an asterisk were estimated from the figures provided in Baten et al. (2006a). The values marked with † are from personal communication with the authors of Baten et al. (2006a). Table taken from Sonnenburg et al. (2007b).

first note that the simple MCs perform already fairly well in comparison to the SVM methods. Surprisingly, we find that the MC-SVM proposed in Baten et al. (2006a) performs worse than the MCs. (We have reevaluated the results in Baten et al. (2006a) with the code provided by the authors and found that the stated false positive rate of their method is wrong by a factor of 10. We have contacted the authors for clarification and they published an erratum (Baten et al., 2006b). The results for MC-SVMs given in Table 5.3 are based on the corrected performance measurement.) As anticipated, for the two acceptor recognition tasks, EBN and MCs are outperformed by all kernel models that are performing all at a similar level. However, we were intrigued to observe that for the DGSplicer Donor recognition task, the MC based predictions outperform the kernel methods. For NN269 Donor recognition, their performance is similar to the performance of the kernel methods.
There are at least two possible explanations for the strong performance of the MCs. First, the DGSplicer data set has been derived from the genome annotation, which in turn might have been obtained using a MC based gene finder. Hence, the test set may contain false predictions easier reproduced by a MC. Second, the window size for the DGSplicer Donor recognition task is very short and has been tuned in Chen et al. (2005) to maximise the performance of their method (EBN) and might be suboptimal for SVMs. We investigated these hypotheses with two experiments:

- In the first experiment, we shortened the length of the sequences in DGSplicer Acceptor from 36 to 18 (with consensus `AG` at 8,9). We retrained the MC and WD models doing a full model selection on the shortened training data. We observe that on the shortened data set the prediction performance drops drastically for both MC and WD (by 60% relative) and that, indeed, the MC outperforms the WD method (to 12.9% and 9% auPRC, respectively).

- In a second experiment, we started with a subset of our new data sets generated from the genomes of worm and human that only uses EST or cDNA confirmed splice sites (see Appendix B.2.2). In the training data, we used the same number of true and decoy donor sites as in the DGSplicer data set. For the test data, we used the original class ratios (to allow a direct comparison to following experiments; cf. Table 5.4). Training and testing sequences were shortened from 218nt in steps of 10nt down to 18nt (the same as in the DGSplicer donor data set). We then trained and tested MCs and WD-SVMs for the sets of sequences of different length. Figure 5.6 shows the resulting values for the auPRC on the test data for different sequence lengths. For short sequences, the prediction accuracies of MCs and SVMs are close for both organisms. For human donor sequences of length 18, MCs indeed outperform SVMs. With increasing sequence length, however, the auPRC of SVMs rapidly improves while it degrades for MCs. Recall that the short sequence length in the DGSplicer data was tuned through model selection for EBN, and thus the performance of the EBN method will degrade for longer sequences (Chen et al., 2005), so that we can safely infer that our methods would also outperform EBN for longer training sequences.

The results do not support our first hypothesis that the test data sets are enriched with MC predictions. However, the results confirm our second hypothesis that the poor performance of the kernel methods on the NN269 and DGSplicer donor tasks is due to the shortness of sequences. We also conclude that discriminative information between true and decoy donor sequences lies not only in the close vicinity of the splice site but also further away (see also the illustrations using POIMs in Section 5.3.4. Therefore,

	Worm		Fly		Cress		Fish		Human	
	Acceptor	Donor	Acceptor	Donor	Acceptor	Donor	Acceptor	Donor	Acceptor	Donor
Training total	1,105,886	1,744,733	1,289,427	2,484,854	1,340,260	2,033,863	3,541,087	6,017,854	6,635,123	9,262,241
Fraction positives	3.6%	2.3%	1.4%	0.7%	3.6%	2.3%	2.4%	1.5%	1.5%	1.1%
Evaluation total	371,897	588,088	425,287	820,172	448,924	680,998	3,892,454	6,638,038	10,820,985	15,201,348
Fraction positives	3.6%	2.3%	1.4%	0.7%	3.6%	2.3%	0.7%	0.4%	0.3%	0.2%
Testing total	364,967	578621	441,686	851,539	445,585	673,732	3,998,521	6,822,472	11,011,875	15,369,748
Fraction positives	3.6%	2.3%	1.4%	0.7%	3.5%	2.3%	0.7%	0.4%	0.3%	0.2%

Table 5.4: Characteristics of the genome-wide data sets containing true and decoy acceptor and donor splice sites for our five model organisms. The sequence length in all sets is 141nt, for acceptor splice sequences the consensus dimer `AG` is at position 61, for donor `GT/GC` at position 81. The negative examples in training sets of fish and human were sub-sampled by a factor of three and five, respectively. Table taken from Sonnenburg et al. (2007b).

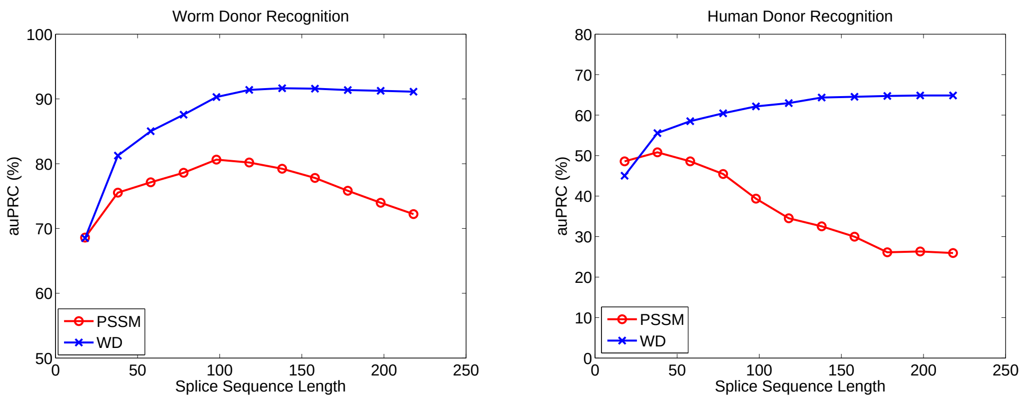


Figure 5.6: Comparison of classification performance of the weighted degree kernel based SVM classifier (WD) with the Markov chain based classifier (MC) on a subset of our *C. elegans Donor* and *Human Donor* data sets for sequences of varying length. For each length, we performed a full model selection on the training data in order to choose the best model. The performance on the test sets, measured through area under the Precision Recall Curve (auPRC), is displayed in percent. Figure taken from Sonnenburg et al. (2007b).

the careful choice of features is crucial for building accurate splice site detectors and if an appropriate window size is chosen, the WD kernel based SVM classifiers easily outperform previously proposed methods.

Comparison with SpliceMachine for Cress and Human In this section we compare SpliceMachine (Degroeve et al., 2005a) with the WD kernel based SVMs. SpliceMachine (Degroeve et al., 2005b) is the current state-of-the art splice site detector. It is based on a linear SVM and outperforms the freely available GeneSplicer (GeneSplicer, Pertea et al., 2001) by a large margin (Degroeve et al., 2005a). We therefore perform an extended comparison of our methods to SpliceMachine on subsets of the genome-wide datasets (cf. Appendix B.2.2). One fifth and one twenty-fifth of the data set was used each for training and for independent testing for cress and human, respectively. We downloaded the SpliceMachine feature extractor (Degroeve et al., 2004) to generate train and test data sets. Similar to the WD kernel, SpliceMachine makes use of positional information around the splice site. As it explicitly extracts these features, it is however limited to a low order context (small d). In addition, SpliceMachine explicitly models coding-frame specific compositional context using tri- to hexamers. Note that this compositional information is also available to a gene finding system for which we are targeting our splicing detector. Therefore, to avoid redundancy, compositional information should ideally not be used to detect the *splicing signal*. Nevertheless, for comparative evaluation of the potential of our method, we augment our WD kernel based methods with 6 spectrum kernels (cf. Section 2.1) (order 3, 4, 5, each up- and downstream of splice site) and use the same large window sizes as were found out to be optimal in Degroeve et al. (2005a). For cress acceptor $[-85, +86]$, donor $[-87, +124]$, and for human acceptor $[-105, +146]$, donor $[-107, +104]$. For the WD kernel based SVMs, we fixed the model parameters $C = 1$ and $d = 22$, and for WDS we additionally fixed the shift parameter $\sigma = 0.5$. For the SpliceMachine we performed an extensive model selection and found $C = 10^{-3}$ to be consistently optimal .We trained with $C \in \{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 5 \cdot 10^{-4}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$. Using these parameter settings we trained SVMs a) on the SpliceMachine features (SM), b) using the WD kernel (WD) c) using the WD kernel augmented by the 6 spectrum kernels (WDSP) d) using the WDS kernel (WDS) and e) using the WDS and spectrum kernels (WDSSP). Table 5.5 shows the area under the ROC and precision recall curve obtained in this com-

		SM	WD	WDSP	WDS	WDSSP	
	Acceptor	99.41	98.97	99.36	99.43	99.43	*auROC*
		91.76	84.24	90.64	92.01	92.09	*auPRC*
Cress	**Donor**	99.59	99.38	99.58	99.61	99.61	*auROC*
		93.34	88.62	93.42	93.68	93.87	*auPRC*
	Acceptor	97.72	97.34	97.71	97.73	97.82	*auROC*
		50.39	42.77	50.48	51.78	54.12	*auPRC*
Human	**Donor**	98.44	98.36	98.36	98.51	98.37	*auROC*
		53.29	46.53	54.06	53.08	54.69	*auPRC*

Table 5.5: Performance evaluation (auROC and auPRC scores) of four different methods on a subset of the genome-wide cress and human datasets. The methods compared are SpliceMachine (SM), the weighted degree kernel (WD), the weighted degree kernel complemented with six spectrum kernels (WDSP), the weighted degree kernel with shifts (WDS), and the weighted degree kernel with shifts complemented by six spectrum kernels (WDSSP). Table taken from Sonnenburg et al. (2007b).

parison. Note that SpliceMachine always outperforms the WD kernel, but is in most cases inferior to the WDS kernel. Furthermore, complementing the WD kernels with spectrum kernels (methods WDSP and WDSSP) always improves precision beyond that of SpliceMachine. As this work is targeted at producing a splicing signal detector to be used in a gene finder, we will omit compositional information in the following genome-wide evaluations. To be fair, one can note that a WDS kernel using a very large shift is able to capture compositional information, and the same holds to some extent for the WD kernel when it has seen many training examples. It is therefore impossible to draw strong conclusions on whether window size and (ab)use of compositional features will prove beneficial when the splice site predictor is used as a module in a gene finder.

Performance for Varying Data Size Figure 5.7 shows the prediction performance in terms of the auROC and auPRC of SVMs using the MC and the WD kernel on the human acceptor and donor splice data that we generated for this work (see Appendix B.2.2) for varying training set sizes. For training, we use up to 80% of all examples and the remaining examples for testing. MCs and SVMs were trained on sets of size varying between 1000 and 8.5 million examples. Here we sub-sampled the negative examples by a factor of five. We observe that the performance steadily increases when using more data for training. For SVMs, over a wide range, the auPRC increases by about 5% (absolute) when the amount of data is multiplied by a factor of 2.7. In the last step, when increasing from 3.3 million to 8.5 million examples, the gain is slightly smaller ($3.2 - 3.5\%$), indicating the start of a plateau. Similarly, MCs improve with growing training set sizes. As MCs are computationally a lot less demanding, we performed a full model selection over the model order and pseudo counts for each training set size. For the WD-SVM, the parameters were fixed to the ones found optimal in the Section 5.3.3. Nevertheless, MCs did constantly perform inferior to WD-SVMs. We may conclude that one should train using all available data to obtain the best results. If this is infeasible, then we suggest to only sub-sample the negatives examples in the training set, until training becomes computationally tractable. The class distribution in the test set, however, should never be changed unless explicitly taken into account in evaluation.

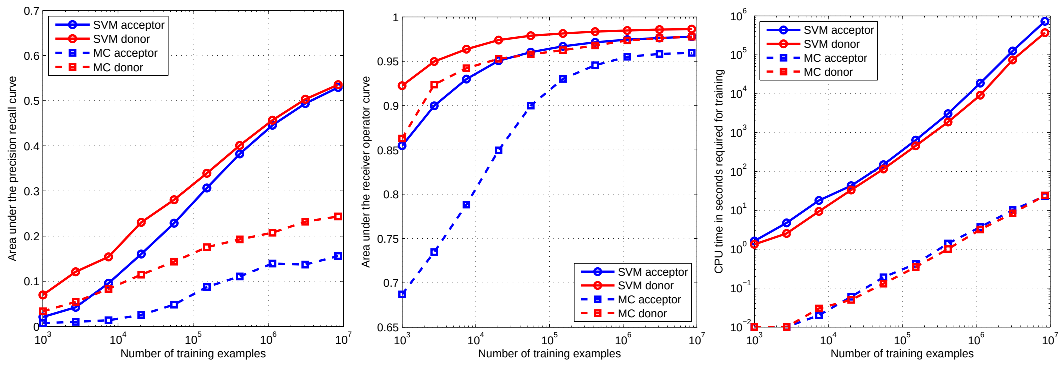


Figure 5.7: Comparison of the classification performance of the weighted degree kernel based SVM classifier (SVM) with the Markov chain based classifier (MC) for different training set sizes. The area under the Precision Recall Curve (auPRC; left) and the area under the Receiver Operator Curve (auROC; middle) are displayed in percent. On the right, the CPU time in seconds needed to train the models is shown. Figure taken from Sonnenburg et al. (2007b).

5.3.3 Results on Genome-Wide Data Sets

Based on our preliminary studies, we now proceeded to design and train the genome-wide predictors. We first generated new *genome-wide* data sets for our five model organisms: worm, fly, cress, fish, and human. As our large-scale learning methods allow us to use millions of training examples, we included all available EST information from the commonly used databases. Since the reliability of the true and decoy splice sequences is crucial for a successful training and tuning, these data sets were produced with particular care; the details can be found in Appendix B.2.2). We arrived at training data sets of considerable size containing sequences of sufficient length (see Table 5.4). For fish and human the training datasets were sub-sampled to include only 1/3 and 1/5 of the negative examples, leading to a maximal training set size of 9 million sequences for human donor sites. For a subsequent use in a gene finder system we aimed at producing unbiased predictions for *all* candidate splice sites, i.e., for all occurrences of the `GT`/`GC` and `AG` consensus dimer. For a proper model selection, and to obtain unbiased predictions on the *whole* genome we employed nested five-fold cross-validation. The results summarised in Table 5.6 are averaged values with standard deviation over the five different test partitions. Confirming our evaluations in the pilot studies, kernel methods

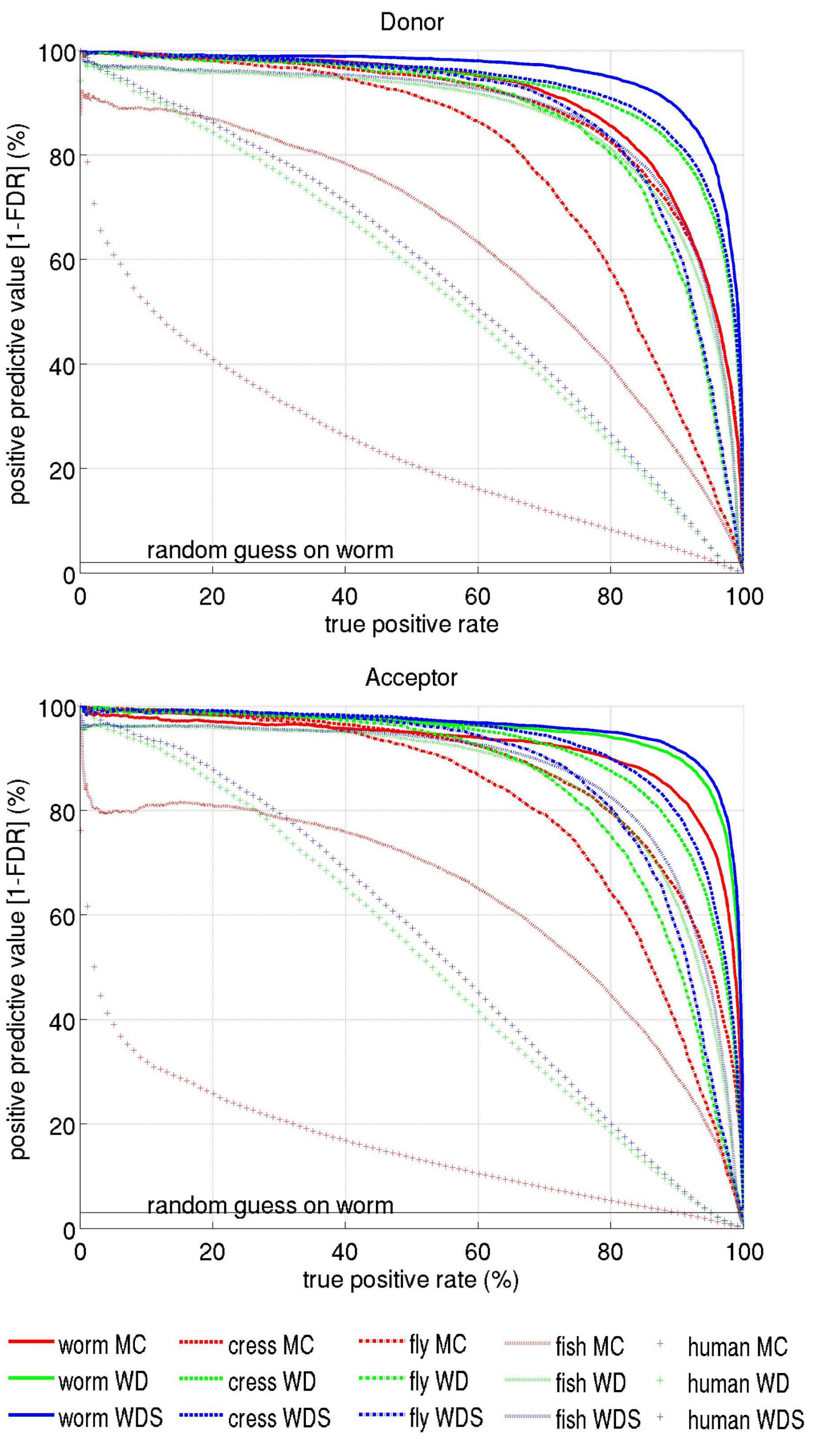


Figure 5.8: Precision Recall Curve for the three methods MC, WD, WDS estimated on the genome-wide data sets for worm, fly, cress, fish, and human in a nested cross-validation scheme. In contrast to the ROC the random guess in this plot corresponds to a horizontal line, that depends on the fraction of positive examples in the test set (e.g., 2% and 3% for worm acceptor and donor data sets, respectively). Figure taken from Sonnenburg et al. (2007b).

outperform the MC methods in all eight classification tasks. Figure 5.8 displays the precision recall curves for all five organisms comparatively, Table 5.6 the corresponding auPRC scores. For worm, fly and cress the improvement in the performance accuracy for the SVM in comparison to MC lies in a similar range of 4-10% (absolute), both for donor and for acceptor tasks. However, for fish, and especially for human the performance gain is considerable higher. For human, MCs only achieve 16% and 25% auPRC scores, whereas WDS reaches 54% and 57% for acceptor and donor recognition, respectively. The severe decrease in performance from worm to human for all classification methods in the auPRC score can partially be explained by the different fractions of positive examples observed in the test set. However, a weaker decline can also be observed in the auROC scores (also Table 5.6), which are independent of the class skew (e.g., for acceptor sites from 99.6% on worm to 96.0% on human for MC, and from 99.8% to 97.9% for WDS). The classification task on the human genome seems to be a considerably more difficult problem than the same one on the worm genome. We may speculate that this can be partially explained by a higher incidence of alternative splicing in the human genome. These sites usually exhibit weaker consensus sequences and are therefore more difficult to detect. Additionally, they often lead to mislabelled examples in the training and testing sets. Finally, it might also be due to the used protocol for aligning the sequences, which may generate more false splice sites in human than in other organisms.

5.3.4 Understanding the Learned Classifier

One of the problems with kernel methods compared to probabilistic methods, such as Position Specific Scoring Matrices (Gribskov et al., 1987) or Interpolated Markov Models (Delcher et al., 1999), is that the resulting decision function is hard to interpret and, hence, difficult to use to extract relevant biological knowledge from it (see also Kuang et al. (2004), Zhang et al. (2003, 2005)). Here, we propose to use positional oligomer importance matrices (cf. Section 4.2 and Sonnenburg et al. (2008)) based on zeroth order Markov chains as background models. We obtain POIM tables (cf. Table 5.7) and a graphical representation from which it is possible to judge where in the sequence substrings of a certain length are of importance. We plotted differential POIMs (cf. Section 4.2) corresponding to our trained models for the organisms and donor and acceptor site comparatively in Figure 5.9 and 5.10 and POIM weight mass (cf. Section 4.2) in Figure 5.11 and 5.12. Differential POIMs show the importance of substrings of a certain length for each position in the classified sequences. We can make a few interesting observations. For all organisms, the donor splicing signal is strongly concentrated around the first eight nucleotides of the intron and well conserved over the particular

	Worm		**Fly**		**Cress**		**Fish**		**Human**	
	Acc	**Don**	**Acc**	**Don**	**Acc**	**Don**	**Acc**	**Don**	**Acc**	**Don**
MC										
auROC(%)	99.62±0.03	99.55±0.02	98.78±0.10	99.12±0.05	99.12±0.03	99.44±0.02	98.98±0.03	99.19±0.05	96.03±0.09	97.78±0.05
auPRC(%)	92.09±0.28	89.98±0.20	80.27±0.76	78.47±0.63	87.43±0.28	88.23±0.34	63.59±0.72	62.91±0.57	16.20±0.22	24.98±0.30
WD										
auROC(%)	99.77±0.02	99.82±0.01	99.02±0.09	99.49±0.05	99.37±0.02	99.66±0.02	99.36±0.04	99.60±0.04	97.76±0.06	98.59±0.05
auPRC(%)	95.20±0.29	95.34±0.10	84.80±0.35	86.42±0.60	91.06±0.15	92.21±0.17	85.33±0.38	85.80±0.46	52.07±0.33	54.62±0.54
WDS										
auROC(%)	99.80±0.02	99.82±0.01	99.12±0.09	99.51±0.05	99.43±0.02	99.68±0.02	99.38±0.04	99.61±0.04	97.86±0.05	98.63±0.05
auPRC(%)	95.89±0.26	95.34±0.10	86.67±0.35	87.47±0.54	92.16±0.17	92.88±0.15	86.58±0.33	86.94±0.44	54.42±0.38	56.54±0.57

Table 5.6: Performance evaluation on the genome-wide data sets for worm, fly, cress, fish, and human. Displayed are auROC and auPRC scores for acceptor and donor recognition tasks as archived by the MC method and two support vector machine approaches, one with the weighted degree kernel (WD) and one with the weighted degree kernel with shifts (WDS). Table taken from Sonnenburg et al. (2007b).

	Donor			Acceptor			Branch Point		
	AGGTAAGT	+01	+	TTTTTCAG	-06	+	TAATAAT	-18	+
	GGTAAGTT	+02	+	ATTTTCAG	-06	+	GGGGGGG	-19	-
Worm	CAGGTAAG	+00	+	TATTTCAG	-06	+	ATAATAA	-19	+
	AGGTGAGT	+01	+	AATTTCAG	-06	+	TTAATAA	-18	+
	AAGGTAAG	+00	+	CTTTTCAG	-06	+	GCGGGGG	-19	-
	AGGTAAGT	+01	+	CTTTGCAG	-06	+	TAACTAAT	-23	+
	GGTAAGTA	+02	+	TTTTACAG	-06	+	GGGGGGGG	-22	-
Fly	GGTAAGTT	+02	+	TTTTGCAG	-06	+	CTAATAAT	-23	+
	CAGGTAAG	+00	+	TTTTTCAG	-06	+	TAATTAAT	-23	+
	AAGGTAAG	+00	+	TCTTGCAG	-06	+	GGGAGGGG	-22	-
	AGGTAAGT	+01	+	TTTTGCAG	-06	+	TAACTAAT	-26	+
	CAGGTAAG	+00	+	TTGTGCAG	-06	+	TAACTAAC	-26	+
Cress	AAGGTAAG	+00	+	TATTGCAG	-06	+	CTAACTAA	-27	+
	GGTAAGTA	+02	+	TCTTGCAG	-06	+	TTAACTAA	-27	+
	GGTAAGTT	+02	+	TGTTGCAG	-06	+	TAATTTTT	-22	+
	AGGTAAGT	+01	+	AGAAGGAG	-06	-	GGGGGGGG	-25	-
	GGTAAGTA	+02	+	AGGCAGAG	-06	-	GGGGGGAG	-24	-
Fish	CAGGTAAG	+00	+	AGAGAGAG	-06	-	GGGGGAGG	-23	-
	AAGGTAAG	+00	+	AGCAGGAG	-06	-	GGGGGCAG	-22	-
	AGGTGAGT	+01	+	AGGAGGAG	-06	-	GGGGAGGG	-22	-
	AGGTAAGT	+01	+	TTTTGCAG	-06	+	TAACTTTT	-22	+
	GGTAAGTA	+02	+	TTTTGCAG	-06	+	TAACTAAC	-27	+
Human	CAGGTAAG	+00	+	TTTTACAG	-06	+	TAACTAAT	-27	+
	AAGGTAAG	+00	+	TTTTTCAG	-06	+	TAATTTTT	-22	+
	GTAAGTAT	+03	+	TCTTGCAG	-06	+	CTAACTTT	-23	+

Table 5.7: POIM ranking tables for the donor and acceptor splice site and the branch point. Only the in absolute value top ranking 5 $k-$mers are shown as (motif, location, $\pm$). A "+" denotes a typical, a "-" denotes an atypical motif.

organisms. *For all organisms* the highest ranking motif in the POIM tables (cf. Table 5.7) is **GT**AGGTAAGT matching prior findings (Burset et al., 2000). In addition, the SVM seems to distribute its weights almost entirely around the splice site. Around the splice site, the POIM weight mass figures have a very similar shape (cf. Figure 5.11). One only finds small differences in the intron: For cress, there seems to be another signal around 10-20nt downstream of the donor: The POIM tables uncover that G rich motifs have a silencing effect and T rich motifs an enhancing effect in this region (table not shown). For worm there is another rather strong signal about 40-50nt downstream of the donor and 40-50 nt upstream of the acceptor splice sites. These two signals are related to each other, since introns in *C. elegans* are often only 50nt long one might suspect to find the corresponding acceptor and donor splice site in the POIM tables. And indeed, one recovers TTTTC**AG**A as the highest scoring 8-mer in the donor POIM table, which perfectly matches the known acceptor splicing consensus (Zhang and Blumenthal, 1996). The picture is different for the acceptor splice site (see Figure 5.10 and 5.12). While the strongest acceptor splicing signal is again located in the last bases of the intron, it is only peaked strongly for *C. elegans* but smeared over about 17bp for human. However, the highest ranking motifs in the POIM tables (cf. Table 5.7) at the acceptor splice site are very similar: Worm TTTTTC**AG**, Fly CTTTGC**AG**, Cress TTTTGC**AG**, Fish TTTTGC**AG**, Human TTTTGC**AG**.

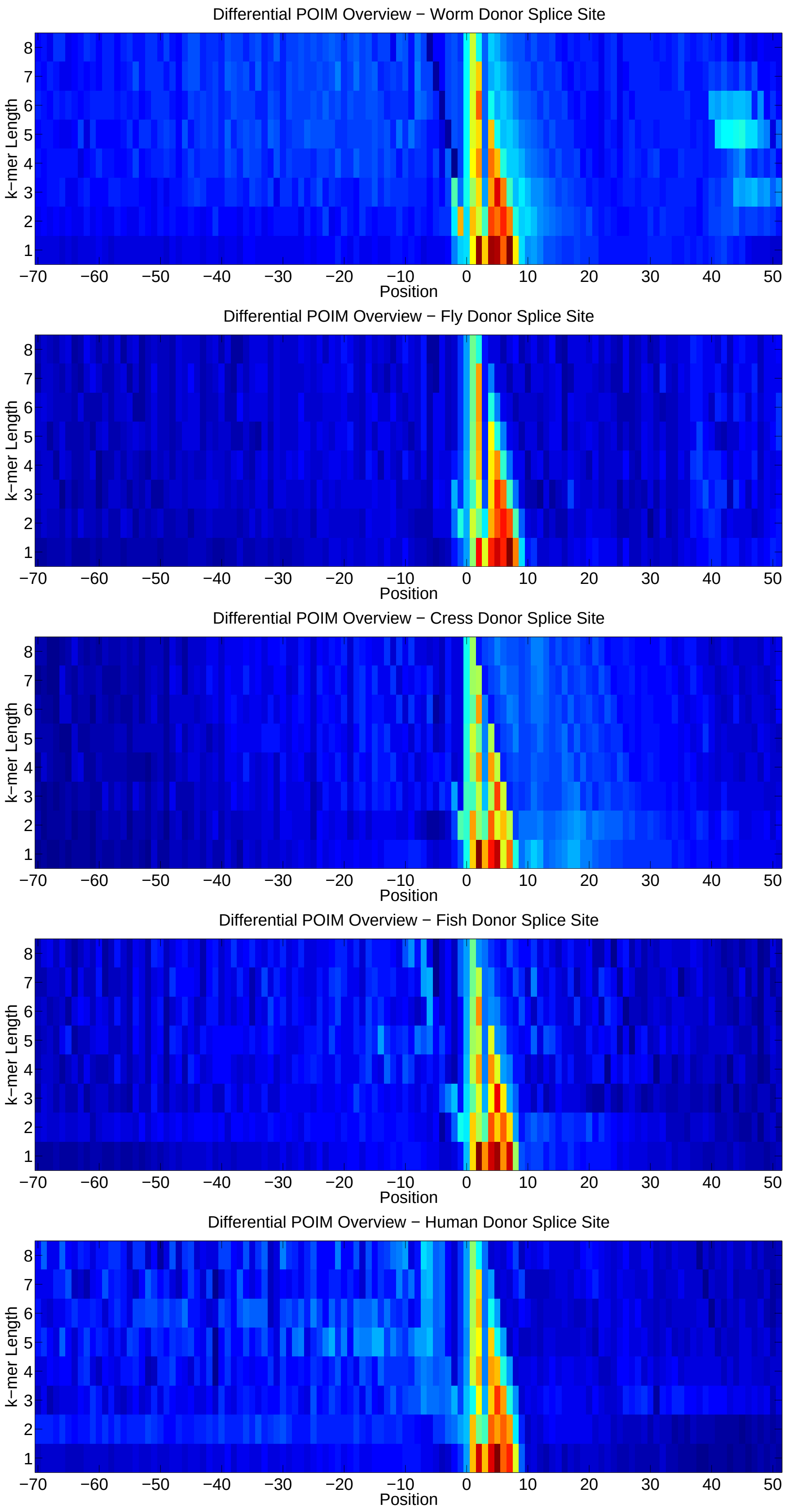


Figure 5.9: Differential POIMs comparatively for worm, fly, cress, fish, and human for the *Donor Splice Site*. They depict the importance of k-mers lengths of order up to length 8 to the decision of the trained kernel classifiers. Red values are highest contributions, blue lowest. Position 1 denotes the splice site and the start of the consensus dimer. To enhance contrast the log-transformation $\tilde{D}(p, j) = log(D(p, j) + \max_{p,j} D(p, j) - \min_{p,j} D(p, j))$ was applied.

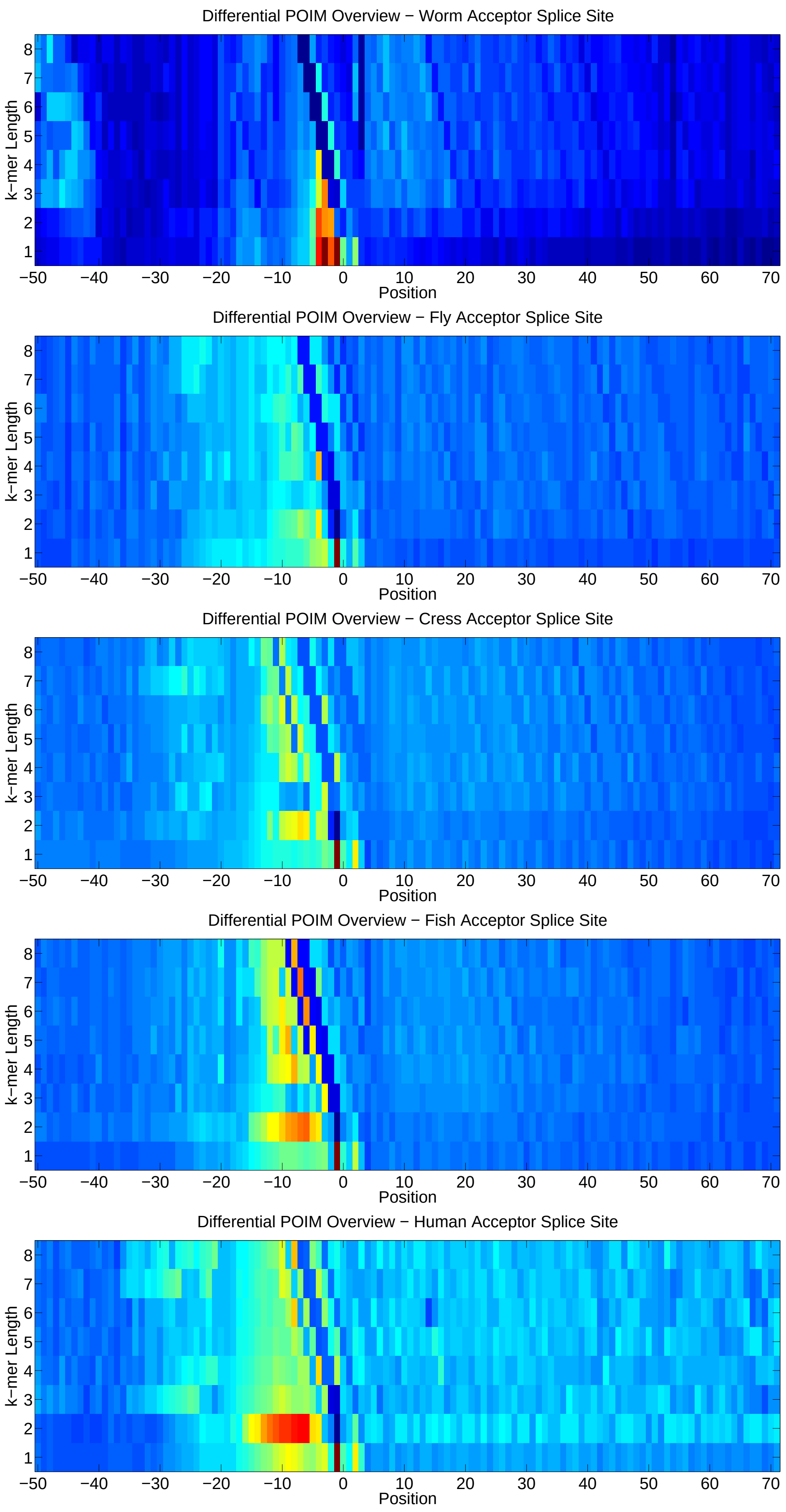


Figure 5.10: Differential POIMs comparatively for worm, fly, cress, fish, and human for the *Acceptor Splice Site*. They depict the importance of k-mers lengths of up to length 8 to the decision of the trained kernel classifiers. Red values are highest contributions, blue lowest. Position 1 denotes the splice site and the start of the consensus dimer. To enhance contrast the log-transformation $\tilde{D}(p,j) = log(D(p,j) + \max_{p,j} D(p,j) - \min_{p,j} D(p,j))$ was applied.

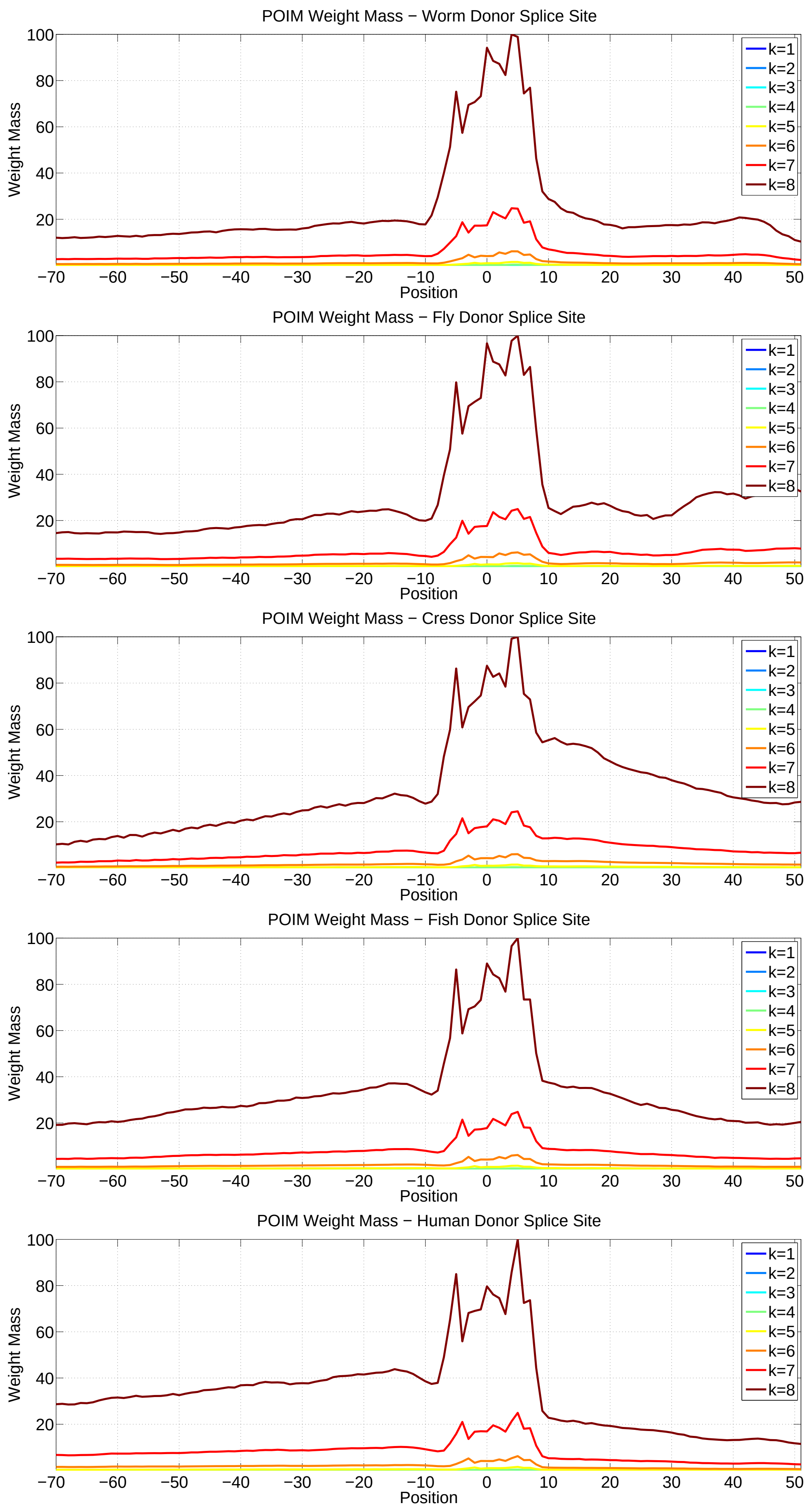


Figure 5.11: Weight Mass comparatively for worm, fly, cress, fish, and human for the *Donor Splice Site*. They depict the importance all k-mers up to order 8 to the decision of the trained kernel classifiers. Red values are highest contributions, blue lowest. Position 1 denotes the splice site and the start of the consensus dimer.

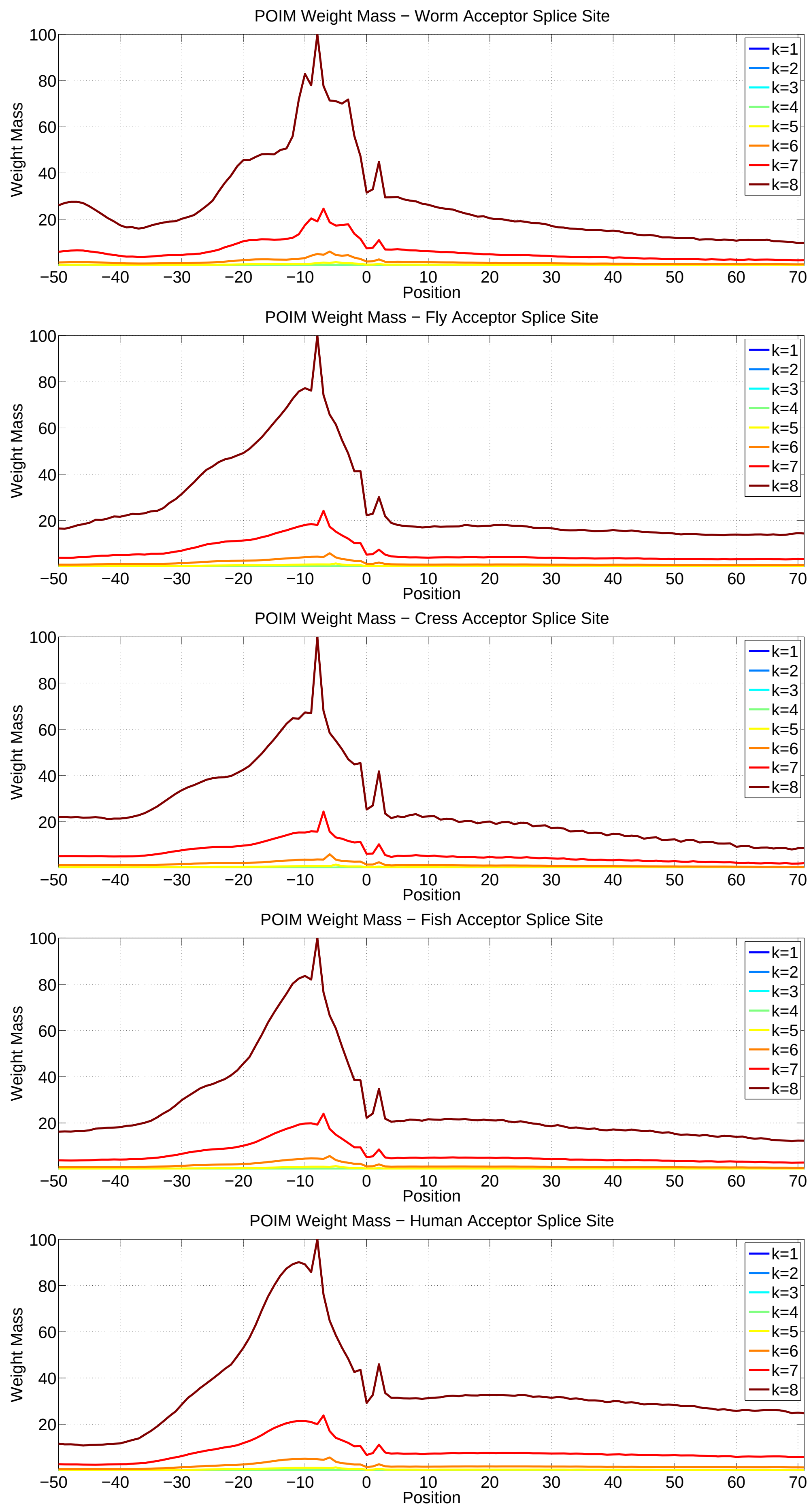


Figure 5.12: Weight Mass comparatively for worm, fly, cress, fish, and human for the *Acceptor Splice Site*. They depict the importance all k-mers up to order 8 to the decision of the trained kernel classifiers. Red values are highest contributions, blue lowest. Position 1 denotes the splice site and the start of the consensus dimer.

Additionally, we find the region 22-40nt upstream of the acceptor splice site of importance, which is very likely related to the branch point. In human it is typically located 21-34nt upstream and exhibits the consensus {C,T}T{A,C,G,T}**A**{C,T} (Gao et al., 2008), which matches the highest ranking motifs extracted from the POIM tables for human, cress and fly, which all contain TA**A**T or TA**A**C. In worms, the branch point consensus seems shorter (3-4nt) (we again find TAAT in the POIM tables) – confirming previous reports that the branch point is much weaker in worms. In addition, it is located less than 20nt upstream of the splice site. In fish we did not observe the branch point, but G rich motifs with a strong silencing effect. Finally, note that the exon sequence also carries discriminative information. The periodicity observed for instance in cress is due to the reading frame.

5.4 Accurate Recognition of Transcription Starts

Arguably the most important information about genomic DNA is the location of genes that encode proteins. For further analysis of the genes it is necessary to find their promoters and the contained binding sites of transcription factors, which are responsible for regulating the transcription of the gene. In this section, we develop new methods for finding transcription start sites (TSS) of RNA Polymerase II binding genes in genomic DNA sequences. Employing Support Vector Machines with the advanced sequence kernels presented in Chapter 2, we achieve drastically higher prediction accuracies than state-of-the-art methods.
Transcription start sites are located in the core promoter region and are usually determined by aligning complete mRNA or 5'-end EST sequences (for instance obtained by 5' RACE) against the genome. Note that protein sequences and non-5'-end ESTs are not sufficient for this task, since they typically start downstream of the TSS. For some species, including human, large scale sequencing projects of complete mRNAs have been undertaken, but many low copy number genes still evade being sequenced. To identify these genes and their promoter regions, computational TSS finding or better experimental techniques are required.
Moreover, in the vast majority of species the identification of promoters must be accomplished without the support of massive sequencing. One possibility is to exploit homology to well-characterised genes in other species. While this approach can work for common genes, for those genes specific to some species or some family of species it is likely to fail. This leaves a huge demand for accurate *ab initio* TSS prediction algorithms.

Relation to Previous Work Consequently, a fairly large number of TSS finders (TSF) has been developed. Generally TSFs exploit that the features of promoter regions and the TSS are different from features of other genomic DNA. Many different features have been used for the identification: the presence of CpG islands, specific transcription factor binding sites (TFBS), higher density of predicted TFBSs, statistical features of proximal and core promoter regions and homology with orthologous promoters (see Bajic et al. (2004), Werner (2003) for two recent reviews on mammalian promoter recognition). Methods for recognizing TSSs employed neural networks, discriminant analysis, the Relevance Vector Machine (RVM), interpolated Markov models, and other statistical methods.
In a recent large scale comparison (Bajic et al., 2004), eight TSFs have been compared. Among the most successful ones were *Eponine* (Down and Hubbard, 2002) (which trains

RVMs to recognise a TATA-box motif in a G+C rich domain), *McPromoter* (Ohler et al., 2002) (based on Neural Networks, interpolated Markov models and physical properties of promoter regions) and *FirstEF* (Davuluri et al., 2001) (based on quadratic discriminant analysis of promoters, first exons and the first donor site, using CpG islands). *DragonGSF* (Bajic and Seah, 2003) performs similarly well as the aforementioned TSFs (Bajic et al., 2004). However, it uses additional binding site information based on the TRANSFAC data base (Matys et al., 2006); thus it exploits specific information that is typically not available for unknown promoters. For this reason, and also because the program is currently not publicly available, we exclude it from our comparison.[4]
One characteristic of TSFs is that they normally rely on the combination of relatively weak features such as physical properties of the DNA or the G+C-content. In none of the above-mentioned approaches, the recognition of the actual transcription start site has been seriously considered.

Organisation of the Section This section is structured as follows: We first describe the features of the sequences and kernels that we use for learning to recognise transcription start sites. Then, we explain the learning setup and the involved model selection procedure. We discuss the experimental evaluation and the data generation for a large scale comparison of our method *ARTS* with other TFSs, and provide experimental results in Section 5.4.2. Finally we apply positional oligomer importance matrices (cf. Section 4.2) to understand the learnt classifier.

Features for TSS recognition As most other TSFs our method combines several features, thereby utilizing prior knowledge about the structure of transcription start sites. We put, however, particular care in analyzing the actual transcription start site. We have considered the following:

- The TSS is only determined up to a small number of base pairs. Further, nearby binding sites may also not be positionally fixed. To model the actual TSS site, we thus need a set of features that are approximately localised and allow for limited flexibility, which is the domain of the *Weighted Degree kernel with shifts (WDS).*
- Upstream of the TSS lies the promoter, which contains transcription factor binding sites. Comparing different promoters, it was noted that the order of TFBS can differ quite drastically. Thus, we use the *spectrum kernel* on a few hundred bps upstream of the TSS. Since it does not preserve the information where the subsequences are located, it may not be appropriate for modeling localised signal sequences such as the actual transcription start site.
- Downstream of the TSS follows the 5' UTR, and further downstream introns and coding regions. Since these sequences may significantly differ in oligo-nucleotide composition from intergenic or other regions, we use a second *spectrum kernel* for the downstream region.
- The 3D structure of the DNA near the TSS must allow the transcription factors to bind to the promoter region and the transcription to be started. To implement this insight, we apply two linear kernels to the sequence of twisting angles and stacking energies. Both properties are assigned based on dinucleotides as done by the *emboss* program *btwisted*.[5] The fourth and fifth kernel are *linear kernels* on

[4]Further, unlike *DragonGSF* all of the above TSFs could – after retraining – be applied to genomes other than human, where only a few or no TF binding sites are known.
[5]`http://www.hgmp.mrc.ac.uk/Software/EMBOSS/Apps/btwisted.html`

features derived from a sequence of DNA twisting angles and stacking energies, respectively, by smoothing with a sliding window and using only every 20th of the resulting values.

The combined kernel is simply the sum of all sub-kernels, which is equivalent to appending the feature vectors in feature space. The sub-kernels can be expected to be of different importance for the overall performance; thus, it may seem appropriate to use a weighted sum. Experiments to verify this indicated that a uniform weighting performs just as well as reducing the weights for the less important sub-kernels.[6] An explanation for this may be that the SVM is able to learn relative weights itself. The only requirement is that the (weighted) function values of the sub-kernels are on a comparable scale; otherwise, those on a low scale are effectively removed.
As before, we normalised all kernels with the exception of the linear kernels such that the vectors $\Phi(\mathbf{x})$ in feature space have unit length, by $\tilde{\mathrm{k}}(\mathbf{x}, \mathbf{x}') = \frac{\mathrm{k}(\mathbf{x},\mathbf{x}')}{\sqrt{\mathrm{k}(\mathbf{x},\mathbf{x})\,\mathrm{k}(\mathbf{x}',\mathbf{x}')}}$. This normalisation solves convergence problems of SVM optimisers and balances the importance of the kernels among each other. In total, we combine five kernels, two linear kernels, two spectrum kernels and the WDS kernel, which each have several parameters as listed in Table 5.8.

Parameter	**Set of values**	**Init. guess**	**Opt. value**	**Explanation**
TSS signal (weighted degree with shift):				
• r-start	$\{-100, -90, \dots, -10\}$	-50	-70	start of considered sequence region
• r-end	$\{+10, +20, \dots, +100\}$	+50	+70	end of considered sequence region
• order	$\{0^*, 2, \dots, 24\}$	10	24	length of substrings compared
• shift	$\{4, 8, \dots, 48\}$	20	32	positional shift (base pairs)
promoter (spectrum):				
• r-start	$\{-1000, -900, \dots, -100\} \cup \{-150\}$	-600	-600	start of considered sequence region
• r-end	$\{-200, -150, \dots, +200\}$	0	0	end of considered sequence region
• order	$\{0^*, 1, \dots, 6\}$	3	4	length of substrings considered
1^{st} **exon (spectrum):**				
• r-start	$\{-100, -50, \dots, +300\}$	+100	0	start of considered sequence region
• r-end	$\{+100, +200, \dots, +1000\}$	+600	+900	end of considered sequence region
• order	$\{0^*, 1, \dots, 6\}$	3	4	length of substrings considered
angles (linear):				
• r-start	$\{-1000, -900, \dots, -200\}$	-600	-600	start of considered sequence region
• r-end	$\{-600, -500, \dots, +200\}$	-100	-100	end of considered sequence region
• smoothing	$\{0^*, 10, \dots, 100\}$	50	70	width of smoothing window
energies (linear):				
• r-start	$\{-1000, -900, \dots, -200\}$	-600	-	start of considered sequence region
• r-end	$\{-600, -500, \dots, +200\}$	-100	-	end of considered sequence region
• smoothing	$\{0^*, 10, \dots, 100\}$	50	0*	width of smoothing window
SVM:				
• C	$\{2^{-2.5}, 2^{-2}, \dots, 2^{+2.5}\}$	2^0	2^1	regularisation constant

Table 5.8: Parameters of the combined kernels and the SVM for TSS recognition. The ranges are specified according to our prior knowledge or intuition. A parameter value of 0 marked with * means that the sub-kernel is excluded from the combined kernel. Table taken from Sonnenburg et al. (2006b).

[6]Using multiple kernel learning to learn the kernel weights resulted in a sparse solution with the TSS kernel getting almost weight 1.

5.4.1 Model Selection and Evaluation

As seen before (Table 5.8), there are many (in fact, 17) parameters that need to be set to reasonable values for our approach to work well. We treat this as a model selection problem: each parameter setting corresponds to a set of assumptions, i.e., a model, on what distinguishes the surroundings of TSS from other genomic loci. We want to select the closest approximation (within the framework defined by the kernel function) to reality, which can be identified by having the best predictive power. Thus, we train the SVM with different parameter settings and assess the resulting prediction performance on a separate validation set.

While model selection is often done by trying all points on a regular grid in the space of parameters, this is computationally infeasible for more than a few parameters. Therefore, we resort to iterated independent axis-parallel searches. First, we specify a start point in parameter space based on prior knowledge and intuition. Then, in each round candidate points are generated by changing any single parameter to any value from a pre-defined small set; this is done for every parameter independently. Finally, the new parameter setting is assembled by choosing for each parameter the value that performed best while leaving the other parameter values unchanged.

We choose the model that yields the highest auROC on the validation set. It achieves 93.99% auROC and 78.20% auPRC (99.93% and 99.91% on the training data, respectively). The selected parameter settings are shown in the second but last column of Table 5.8.

Efficiency Considerations Our model is complex in that it consists of several sophisticated kernels applied to rather long stretches of DNA. Furthermore, we have to train it on as many examples as possible to attain a high prediction accuracy.[7] Even with highly optimised general purpose SVM packages like *LibSVM* or *SVM*^{light}, training and tuning our model with tens of thousands of points is intractable. However, implementing the `linadd` optimisations for the here considered linear and string kernels (cf. Chapter 3) training on such large datasets becomes feasible.

In addition, we need to compute predictions for every position in the human genome ($\approx 7 \cdot 10^9$). For kernel methods that generate several thousands of support vectors, each of which is of length one thousand, this would mean more than 10^{16} floating point operations. This is too much even for modern computer cluster systems. Luckily we can again use the `linadd` trick (cf. Chapter 3) to efficiently compute the SVM prediction $f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i \, \mathrm{k}(\mathbf{x}_i, \mathbf{x})$ for new sequences. While this leads to dramatic speedups, it still requires $\approx$ 350h of computing time to do predictions on the entire human genome.

Importance of the Kernels In addition to optimizing the parameter settings of all sub-kernels, we investigate whether and how much each sub-kernel contributes to the overall classification. To do so, we remove each sub-kernel and retrain the remaining model (with all other parameters kept fixed at the selected values). The accuracies obtained on the validation set are shown in Table 5.9. Removing the WDS kernel, which models the signal at $[-70, +70]$ around the TSS, decreases the performance of the classifier most, although it still performs rather well (auROC > 90%). The 1^{st} exon kernel, which models the 4-mer nucleotide frequency in the range $[0, +900]$ downstream

[7]For instance, on a splice site recognition task we were able to perpetually improve the prediction accuracy when increasing the amount of training data – over a wide range of training set sizes Figure 5.7.

is of second most importance in our kernel ensemble. Removing the linear kernels, which take into account the binding energies and the twisting of the DNA, has almost no effect on the result.

Subkernel	area under ROC	area under PRC
w/o TSS signal	**90.75%**	**70.72%**
w/o promoter	93.33%	74.94%
w/o 1^{st} exon	92.76%	74.94%
w/o angles	93.99%	78.26%
complete	93.99%	78.20%

Table 5.9: Results obtained by removing sub-kernels. The energies kernel is already turned off by the model selection. Table taken from Sonnenburg et al. (2006b).

A different view on the contribution of the individual kernels can be obtained by retraining single-kernel SVMs. The respective results are displayed in Table 5.10. Again the WDS kernel contributes most, followed by the two spectrum kernels modeling the first exon and the promoter. The DNA twistedness angle-measure performs even worse than at random, probably because SVM's regularisation parameter C was not properly tuned for the single kernel case.

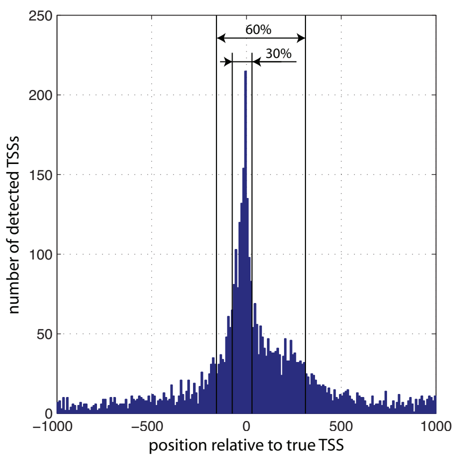
Subkernel	area under ROC	area under PRC
TSS signal	**91.42%**	**69.38%**
promoter	86.55%	55.33%
1^{st} exon	88.54%	64.29%
angles	45.31%	7.86%

Table 5.10: Results obtained when only a single specific sub-kernel is used. The actual TSS signal discriminates strongest, but also the 1^{st} exon carries much discriminative information. Table taken from Sonnenburg et al. (2006b).

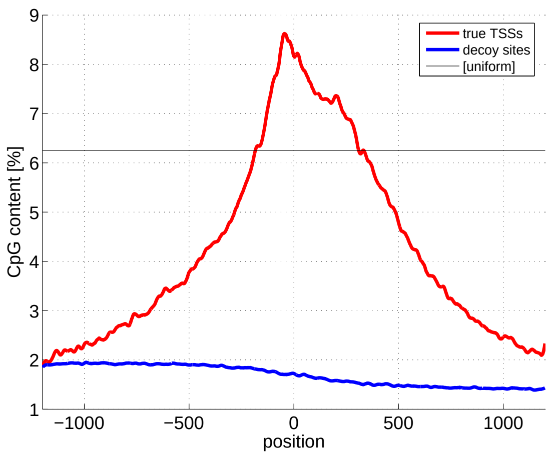
For illustration we analyze in Figure 5.13 how the TSS signal predictions are localised relative to the true transcription start sites. We consider a window of ± 1000 around a true TSS and record the location of the maximal TSS signal prediction (TSS signal kernel only). Figure 5.13 displays a histogram of the recorded positions on our validation set. We observe an expected strong concentration near the true TSS. We also observe that the distribution is skewed – a possible explanation for this is offered by Figure 5.13: the predictor might be mislead by the distribution of CpG islands, which is skewed in a similar manner. In conclusion, it seems to be the WDS kernel that models the region around the TSS best. The relatively large shift of 32 found by the model selection suggests the existence of motifs located around the TSS at highly variable positions. Neither *Eponine* nor *FirstEF* model this regions explicitly. Thus, the WDS kernel seems to be one of the reasons for *ARTS*' superior accuracy.

5.4.2 Genome-Wide Evaluation on the Human Genome

We compare the performance of *ARTS*, our proposed TSS finding method, to that of *McPromoter* (Ohler et al., 2002), *Eponine* (Down and Hubbard, 2002) and *FirstEF* (Davuluri et al., 2001), which are among the best previously reported methods (Bajic et al., 2004). The evaluation protocol highly affects a TSF comparison; we thus give a detailed explanation (Section 5.4.2) of the criteria we use.



(a) Localisation of *ARTS*' TSS signal predictions.



(b) Average positional frequency of CpG dinucleotides around true TSS and in decoys.

Figure 5.13: (a) Localisation of *ARTS*' TSS signal predictions: Shown is a histogram over the location with maximal score in a window ±1000 around true TSSs. In 60% of the cases the predicted TSSs is within $[-150, +270]$bp of the true TSS (30% within $[-70, +40]$) (b) Average positional frequency of CpG dinucleotides around true TSS and in decoy sequences (smoothed by convolution with a triangle of 39 bps length). Figure taken from Sonnenburg et al. (2006b).

Setup As POL II binds to a rather vague region, there seems to be no single true TSS location, but rather regions of roughly $[-20, +20]$ bp constituting potential TSSs. For that reason, one has to use evaluation criteria different from the ones used in standard two-class-classification. Bajic *et al.* (Bajic et al., 2004) suggest to cluster predicted TSS locations that have at most 1000bp distance to the neighboring locations. As evaluation criterion for each gene, they score a true positive if a prediction is located within ±2000bp of the true TSS (otherwise, a false negative is counted); false positives and true negatives are counted from the TSS position +2001 to the end of the gene. However, each TSF is tuned to obtain a maximum true positive rate at a *different* false positive rate. Hence, this criterion suffers from the fact that it remains unclear how to compare results when the sensitivity and positive predictive value are both different (cf. Table 2 in Bajic et al. (2004)).

To alleviate this problem and allow for direct comparison via Receiver Operator Characteristic and Precision Recall Curves (ROC and PRC), we propose a different evaluation criterion. We compute whole genome point-wise predictions, which are then converted into *non-overlapping* fixed length chunks (e.g., of size 50 or 500). Within each chunk the maximum TSF output is taken. One can think of this chunking[8] process as a “lens”, allowing us to look at the genome at a lower resolution. Obviously, for a chunk size of 1 this is the same as a point-wise TSS prediction. As “lenses”, we use chunk sizes of 50 and 500. A chunk is labelled as +1 if it falls within the range ±20bp of an annotated TSS; chunks downstream of this range until the end of the gene are labelled −1.

Note that according to the above scheme some TSS will label two chunks as positive. This, however, does not constitute a problem, as it is a rare event if the chunk size is large. Furthermore, it is not unlikely that a TSF predicts both two chunks as positive,

[8]Not to be confused with the “chunking” (decomposition) algorithms used for SVM training.

as the maximum of the scores within each chunk is taken. We also considered an alternative criterion, in which only the chunk in which the maximum TSFs output is larger is labelled as positive, whereas the other chunk is removed from the evaluation. As a downside, this introduces a labeling that is dependent on the TSF output (i.e., there is no ground truth labeling over all TSFs), and leads to only small variations (auROC/auPPV increased/decreased by $\leq 3.5\%$ for chunk size 50 and $\leq 1\%$ for *all* TSFs for chunk size 500). Chunks obtain negative labels if they were not positively labelled and fall within the range gene start+20bp to gene end and are excluded from evaluation otherwise.

This way, the labeling of the genome stays the same for all TSFs. Taking into account *all* TSSs in dbTSSv5 we obtain labelings for chunk size 50 (500) with 28,366 (16,892) positives and 16,593,892 (1,658,483) negatives where TSS fall into two chunks in 15,223 (1,499) cases, covering in total $829,694,600$bp ($\approx 12\%$) of the human genome.[9] In summary, the chunking allows for a controlled amount of positional deviations in the predictions. Unlike the clustering of predictions, it does not complicate the evaluation or hamper the comparability of TSF.

TSF Performance evaluation As the performance evaluation via ROC/PRC curve needs (genome-wide) real valued outputs for each TSF, we set the TSF's thresholds to the lowest acceptable values. *Eponine* is run with the options `-threshold 0.5`. As *McPromoter* provides the outputs for every tenth base pair we can use the unfiltered raw values directly. *FirstEF* does not provide a single score as output, but probability scores for the promoter, exon, and donor. By default, a prediction is made if each probability equals or is larger than a pre-defined threshold (promoter: 0.4, exon: 0.5, donor 0.4), which yields just a single point in the ROC and PRC space. We therefore set all thresholds to 0.001 and later use the product of the scores as a single output.[10] Next we chunk the output, as described above in Section 5.4.2, and perform evaluation on all genes whose TSS was newly annotated in dbTSSv5 (cf. Section B.3). Tables 5.11 and 5.12 display the results for the performance measures area under the ROC and PRC curve for *ARTS*, *FirstEF*, *McPromoter*, and *Eponine*. Table 5.11 shows results for chunk size 50 and Table 5.12 for chunk size 500, corresponding to different levels of positional accuracy or resolution. In both cases our proposed TSS finder, *ARTS*, clearly outperforms the other methods in terms of both auROC and auPRC. This is also seen in Figure 5.14, which supplies detailed information on the true positive rates (top) and the positive predictive values (bottom) for a range of relevant true positive rates.
An interesting observation is that, judging by the auROC, *McPromoter* constitutes the (second) best performing TSF, while, on the other hand, it performs worst in the auPRC evaluation. An explanation can be found when looking at the ROC and PRC in Figure 5.14 where the left column displays ROC/PRC for chunk size 50 and the right for chunk size 500. Analyzing the ROC figures, we observe that *McPromoter* outperforms *FirstEF* and *Eponine* for false positive rates starting around 10% – a region contributing most to the auROC (note that both axes are on log scale). All of the three aforementioned promoter detectors perform similarly well. At a reasonable false positive level of 0.1% the TSFs perform as follows (chunk size 50): *ARTS* 34.7%, *Eponine* 17.9%, *FirstEF*

[9]Here we used the dbTSS field *Position(s) of 5'end(s) of NM_(or known transcript)* as the field *Selected representative TSS* is often empty.

[10]As a validation we run *FirstEF* with the default settings and a variety of other thresholds. The obtained TPR/FPR and TPR/PPV values fit the curves produced using the single score extremely well (cf. Figure 5.14 below).

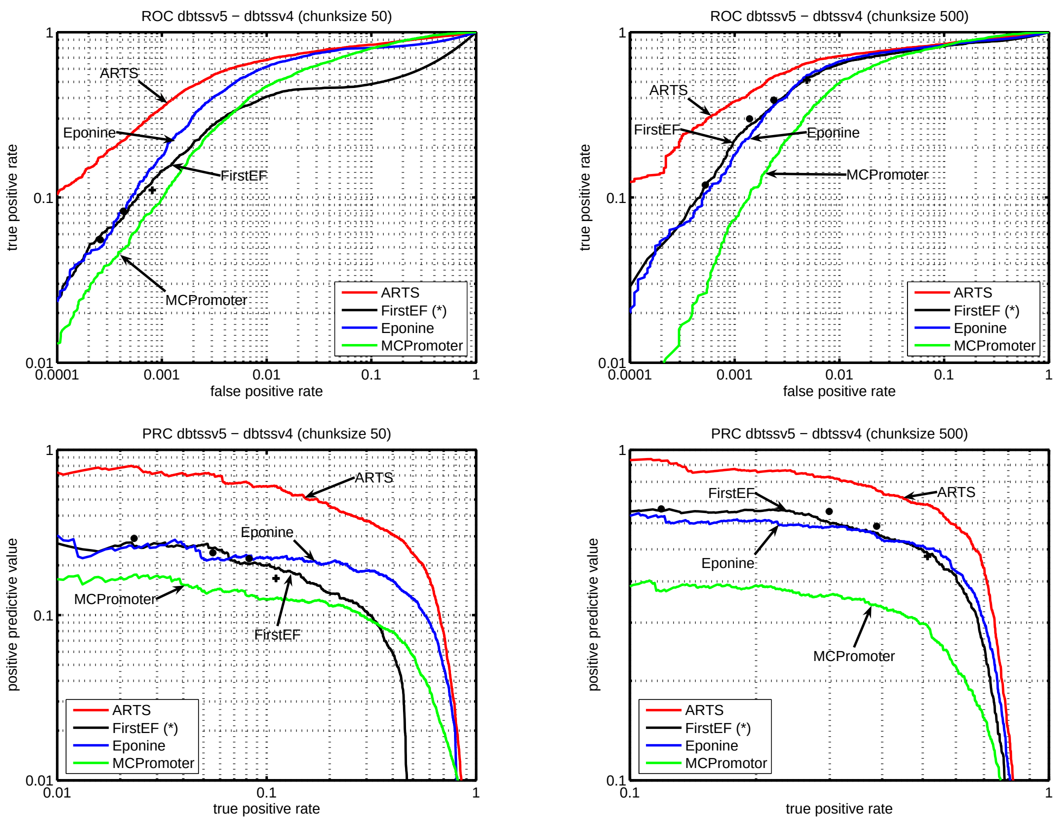


Figure 5.14: Performance evaluation of the *ARTS*, *FirstEF*, *Eponine* and *McPromoter* TSS predictors. Evaluation was done on all genes whose TSS was newly annotated in dbTSSv5 (i.e., genes whose TSS was not already in dbTSSv4). Receiver Operator Characteristic and Precision Recall Curves on decreased output resolution were computed (taking the maximum output within non-overlapping chunks of size 50 (left column) and 500 (right column for more details see text). Windows were marked positive if a known TSS lies in a range of ±20bp and negative otherwise. Please note that the 'bumps' in the upper right corner in the *FirstEF*/*Eponine* plots for low window sizes are artifacts, caused by the method not giving predictions for every position. However, the interesting area is in the left (lower false positive rate). Figure taken from Sonnenburg et al. (2006b).

dbTSSv5-dbTSSv4 evaluation on chunk size 50		
TSF	area under ROC	area under PRC
Eponine	88.48%	11.79%
McPromoter	**92.55%**	6.32%
FirstEF	71.29%	6.54%
ARTS	**92.77%**	**26.18**

Table 5.11: Evaluation of the Transcriptions Start Finder at a chunk size resolution of 50 on dbTSSv5 excluding dbTSSv4 using the area under the Receiver Operator Characteristic Curve and the area under the Recall Precision Curve (larger values are better). For details see text. Table taken from Sonnenburg et al. (2006b).

14.5% and *McPromoter* 9.8%. The ROC curves for both chunk sizes are very similar, as the ROC curves are independent of class ratios between the negative and the positive

dbTSSv5-dbTSSv4 evaluation on chunk size 500		
TSF	area under ROC	area under PRC
Eponine	91.51%	40.80%
McPromoter	**93.59%**	24.23%
FirstEF	90.25%	40.89%
ARTS	**93.44%**	**57.19%**

Table 5.12: Evaluation of the Transcriptions Start Finder at a chunk size resolution of 500 on dbTSSv5 excluding dbTSSv4 using the area under the Receiver Operator Characteristic Curve and the area under the Recall Precision Curve (larger values are better). For details see text. Table taken from Sonnenburg et al. (2006b).

class.[11] On the other hand, class ratios affect the PRC quite significantly: For instance, at a true positive rate of 50%, *ARTS* achieves a PPV of 23.5% for chunk size 50 and 68.3% for chunk size 500. *ARTS*' ROC and PRC, however, constantly remains well above its competitors.

5.4.3 Comparison with EP3 and ProSOM

Recently, two new promoter prediction programs called Easy Promoter Prediction Program (EP3, Abeel et al. (2008a)) and ProSOM (Abeel et al., 2008b) where developed. Both approaches are based on rather simple structural profiles that score all dinucleotides in a window according to prior knowledge (e.g., dinucleotide scores obtained from the inverted base-stacking structural profile values). In EP3, a promoter is predicted if the window score is above a certain threshold and no promoter is predicted otherwise. In their evaluation, the authors showed a surprisingly good performance of EP3 compared to *ARTS*, even though the spectrum kernel components in *ARTS* that are used for the promoter and first exon are closely related to the profile model in EP3: Using a spectrum kernel of order 2 the SVM learns a di-nucleotide weighting, which is applied to a window and thresholded. We therefore re-evaluated EP3 on our validation data set, which lead to very different results (cf. Table 5.13). We notified the authors of EP3 about a flaw in their evaluation scheme in February 2008.[12] However, it should be noted that the evaluation scheme differs from the one used in this section, as whole genome coverage of CAGE tags (Kawaji et al., 2006) was assumed and thus negative examples were obtained genome-wide (not only from the interior of the gene). As this sounds desirable, a re-evaluation of *ARTS* trained on CAGE data is called for. Nevertheless, it is surprising that a simple dinucleotide based model (with pre-specified dinucleotide scores) obtains a high accuracy, especially since EP3 works well on a wide range of organisms.

A to EP3 related approach is ProSOM (Abeel et al., 2008b). It is based on learning self organizing maps from base stacking energy structural profiles. In contrast to EP3

[11]They vary very slightly as for smaller chunk sizes TSS more often fall into two chunks.

[12]It turns out that the F1-measure used in the comparison requires a careful tuning of the bias of the competing methods, which was at least for *ARTS not tuned*. Personal communication with the authors: Using the evaluation criteria from Abeel et al. (2008a), *ARTS* obtains a F1 measure of 50% (not as reported 12%) when tuning the threshold, window size and removing `N` rich regions from the evaluation. EP3 achieves 46%. Furthermore the evaluation was done also on genomic regions that contain a large number of `N`'s. In addition, even though *ARTS* provides *point predictions* only predictions using a 500 bp resolution were used.

Method	auROC	auPRC
ARTS promoter kernel	86.55%	55.33%
ARTS first exon kernel	88.51%	64.29%
ARTS full model	93.99%	78.20%
EP3 window size 50	84.51%	48.06%
EP3 window size 100	86.15%	52.00%
EP3 window size 200	87.39%	54.18%
EP3 window size 400	87.25%	53.19%
EP3 window size 600	86.57%	50.69%
EP3 window size 800	85.60%	47.35%
EP3 window size 1000	84.57%	43.92%

Table 5.13: Comparison of EP3 with *ARTS* on the human validation set. The performance of the full *ARTS* model, as well as only the performance of the single kernel SVMs based on the promoter and first exon spectrum kernels are shown. For EP3 we depict the performance for different window sizes around the TSS.

the structural profiles contain positional information and thus are capable of measuring trends (e.g., low CpG frequency before and high CpG frequency after the TSS). This is a promising new feature and worth integrating into future versions of *ARTS*. Abeel et al. (2008b) report a performance of ProSOM on human CAGE data of F1=48% and *ARTS* F1=50% at a 500 bp resolution. On Ensembl ProSOM achieves F1=45% and *ARTS* F1=56% at 500 bp resolution (Abeel et al., 2008b). The authors also report that *ARTS* performance drops below that of ProSOM, *Eponine* and EP3 when evaluated at 50 bp resolution (Abeel et al., 2008b). However, in Table 5.11 we observe that *ARTS* by far outperforms *Eponine* on chunk size resolution 50. This contradiction may be explained by a more clean evaluation dataset and different evaluation criteria used in Abeel et al. (2008b). In addition, *ARTS* prediction are readily available for download only at a 50 bp resolution in the form of custom tracks. Thus no point predictions of *ARTS* were used in the evaluation.

5.4.4 Understanding the Learned Classifier

Understanding the Model: Motif Weighting As the spectrum kernels (which do not use position information at all) strengthen the classifier, they seem to indeed pick up TF binding site motifs in the upstream (promoter) region, and 4-mer compositional bias (potentially of coding regions) in the downstream region. To verify this, we investigate the weights that the trained single-kernel SVMs (from Table 5.10) assign to the 256 possible 4-mers. Table 5.14 shows the 4-mers with the largest impact in these two spectrum kernel SVMs. Here positively weighted 4-mers favor a TSS prediction, whereas negatively weighted 4-mers disfavor it.

It seems that both kernels have some nucleotide preferences: The promoter kernel prefers 4-mers rich of As and disfavors Gs. The 1^{st}-exon kernel prefers Gs and Cs and dislikes As (as only a single A appears in the top ten motifs). Furthermore, it seems to be mostly looking for CpG-islands, as 6 of the top ten ranked 4-mers contain a CG. Thus, the SVM has learnt something that is often explicitly built into TSFs (e.g., *Eponine* and *FirstEF* both look for elevated downstream CpG content). By training our more general model, we retain its freedom to pick up other (weaker) yet unknown patterns, that might help to distinguish true TSS from decoys.

Table 5.14: Weights of the ten 4-mers with highest absolute weights, for the promoter and the 1^{st}-exon kernel. The numbers in parentheses show the relation to the sum of all weights.

rank	promoter kernel			1^{st}-exon kernel		
1	CGGG	-3.65	(1.37%)	GCGT	3.92	(2.29%)
2	AAAC	3.58	(1.34%)	GTCT	3.75	(2.19%)
3	AACT	3.23	(1.21%)	CGCT	3.46	(2.02%)
4	CGAC	3.19	(1.20%)	GCGC	3.18	(1.86%)
5	AAAA	3.03	(1.14%)	GCCT	3.00	(1.75%)
6	CGGT	-3.01	(1.13%)	CGGG	2.66	(1.55%)
7	CCTA	-3.00	(1.13%)	CGTA	2.61	(1.52%)
8	AAAG	2.95	(1.11%)	CCGT	2.52	(1.47%)
9	CGAG	-2.77	(1.04%)	TCCT	2.24	(1.31%)
10	TTCG	2.73	(1.02%)	GGTC	2.23	(1.30%)

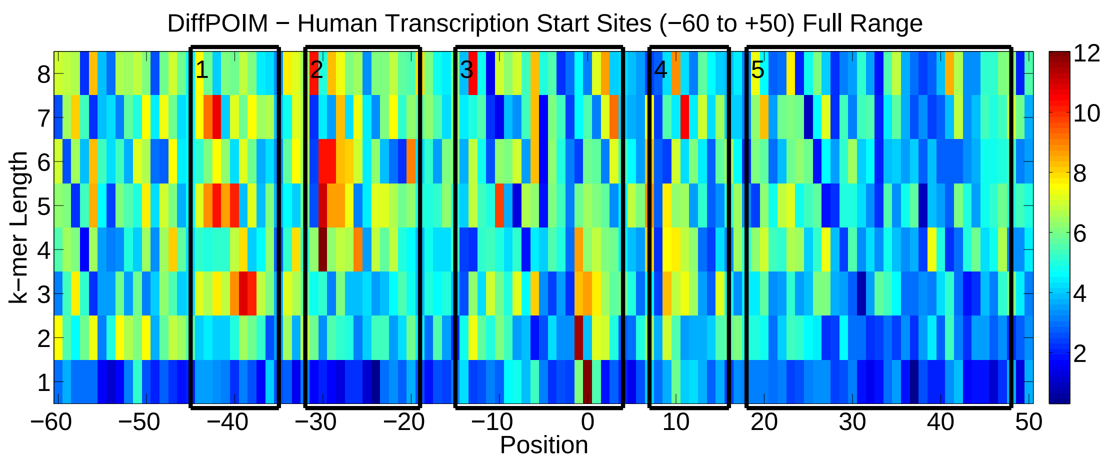
Understanding the Model: Promoter Regulatory Elements from POIMs We finally apply our new ranking and visualisation techniques to the search for promoter regulatory elements. To this end we use the obtained support vectors in conjunction with the WDS kernel with the optimal parameters (cf. Table 5.8) to obtain POIMs (see Section 4.2). The POIM analysis is displayed in Figure 5.15 and 5.16. In Figure 5.15 we have highlighted five regions of interest. A region far upstream (region 1; -45nt to -35nt), another upstream region (region 2; -32 to -19nt), a central region around the TSS (region 3; -15 to +4nt), a downstream region (region 4; +7 to +16) and a far downstream region (region 5; +18 to +48). Inspection of POIM tables in these regions we retrieve the following known motifs:

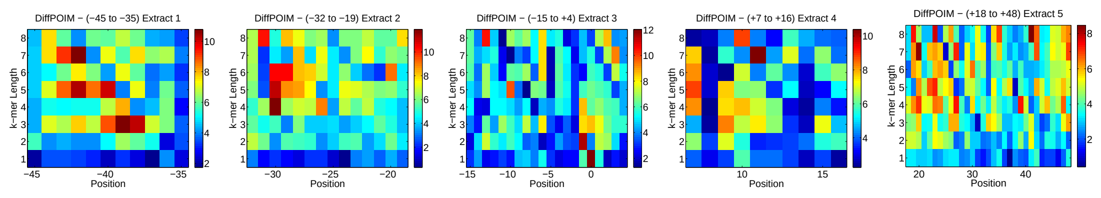
Upstream (region 1; -45nt to -35nt): we observe many high scoring `GC` rich motifs (cf. also the di-nucleotide POIMs in Figure 5.16), which might correspond to a CpG island. However, more likely is that they are part of the BRE element (Jin et al., 2006), which is usually located -37nt to -32nt upstream (Jin et al., 2006). While the 8-mer `GCGCGCC` (rank 9) matches the BRE consensus $\frac{C}{G}\frac{C}{G}\frac{A}{G}\texttt{CGCC}$ well, it is observed slightly further upstream at position -42.

Upstream (region 2; -32 to -19nt): the TATA box's core motif `TATAAA` (see e.g., Jin et al. (2006)) is found -32 to -26 upstream, with peak score at -30; its variants such as `TATAA, TATATA, TATAAG, ATAAAA, TATAAT, TATA` (and motifs overlapping with parts of `TATA`). This perfectly matches prior findings in which the `TATA` motif was found between -31 to -26 and with consensus $\texttt{TATA}\frac{T}{A}\texttt{AA}\frac{A}{G}$

Central location (region 3; -15 to +4nt) one would expect to find the Initiatior (Inr) motif which has the weak consensus $\frac{C}{T}\frac{C}{T}\texttt{A}\{A,C,G,T\}\frac{T}{A}\frac{C}{T}\frac{C}{T}$ (Jin et al., 2006). While some of the highest scoring 4-mers match that consensus (e.g. `TCAT`; rank 2 or `CCAT`; rank 5) we cannot draw any strong conclusions. In addition, we observe motifs that contain the start codon `ATG` (rank 1,2,9), which might indicate non-clean training data. That artefact could be caused by training sequences having a zero length UTR. It should be noted that the SVM puts most of its weight around this location (see Figure 5.16.

Downstream (region 4; +7 to +16 and (region 5; +18 to +48): many CG rich motifs indicative of the CpG islands downstream of the TSS all score high and peak at +7nt and +20nt. The di-nucleotide POIMs (cf. Figure 5.16) underline the `CG`





GCGCGCG	-42	+
CGCGCGG	-41	+
CGCGCGC	-43	+
CCGCGCG	-42	+
CGGCGCG	-42	+
GCGCGGC	-40	+
GGCGCGG	-41	+
CGCGCGT	-41	+
GCGCGCC	-42	+
GGGCGCG	-42	+

TATAAA	-30	+
CTATAA	-31	+
TATATA	-30	+
ATATAA	-31	+
TATAAG	-30	+
TTATAA	-31	+
ATAAAA	-29	+
CTATAT	-31	+
ATAAAG	-29	+
CGGCGG	-24	+

CATG	-01	+
TCAT	-02	+
ATGA	+00	+
TCCC	+00	-
CCAT	-02	+
CCCC	+00	-
GCAT	-02	+
CATT	-01	+
ATGC	+00	+
GTCA	-03	+

CGGCGGC	+07	+
GCGGCCG	+09	+
CGACGGC	+07	+
CAGCGGC	+07	+
GGCGGCG	+08	+
GGCGGCC	+08	+
GCGGCTG	+09	+
GCGGCGC	+10	+
TGGCGGC	+07	+
GCGGCGG	+09	+

CGGCGGC	+20	+
GCGGCGG	+19	+
GCTGCTC	+44	+
GCGGCAG	+22	+
CTGCTGC	+45	+
TGCTGCT	+46	+
CTGCTCG	+45	+
GGCGGCG	+21	+
CTGCTCC	+45	+
TGCTGCG	+46	+

Figure 5.15: **(top)** POIM visualisation for human promoter sites, as obtained by *ARTS*. **(center)** POIM visualisation zoomed in to the five marked regions. Note that the color scaling varies among figures. **(bottom)** Top ten motifs extracted from the POIM tables for five marked regions. The three-tuples show (motif, location, score). A positive score denotes an enhancer or characteristic motif. A negative score denotes a silencer or atypical motif.

enrichment in this region. At the same time CCCCCCC at position +15, TATATAT (and variants of the TATA motif) at +18 and AAAAAAA at +38 are very atypical motifs downstream of the TSS. Further known but vague motifs downstream of the TSS could not be recovered: The MTE element located +18 to +27; consensus $C\frac{C}{G}A\frac{A}{G}C\frac{C}{G}\frac{C}{G}AACG\frac{C}{G}$ and the Downstream Promoter Element (DPE) located +28 to +32 with consensus $\frac{A}{G}\texttt{G}\frac{A}{T}\frac{C}{T}\{A,C,G\}$ (Jin et al., 2006) could not be recovered, as they are hidden under the very dominant CpG islands. Only further cluster-based analysis might identify their importance.

5.5 Summary and Guidelines

In the first part we have evaluated several approaches for the recognition of splice sites in worm, fly, cress, fish, and human. In a first step we compared MCs, a Bayesian method (EBN) and SVM based methods using several kernels on existing data sets generated from the human genome. We considered the kernel used in Baten et al. (2006a) based

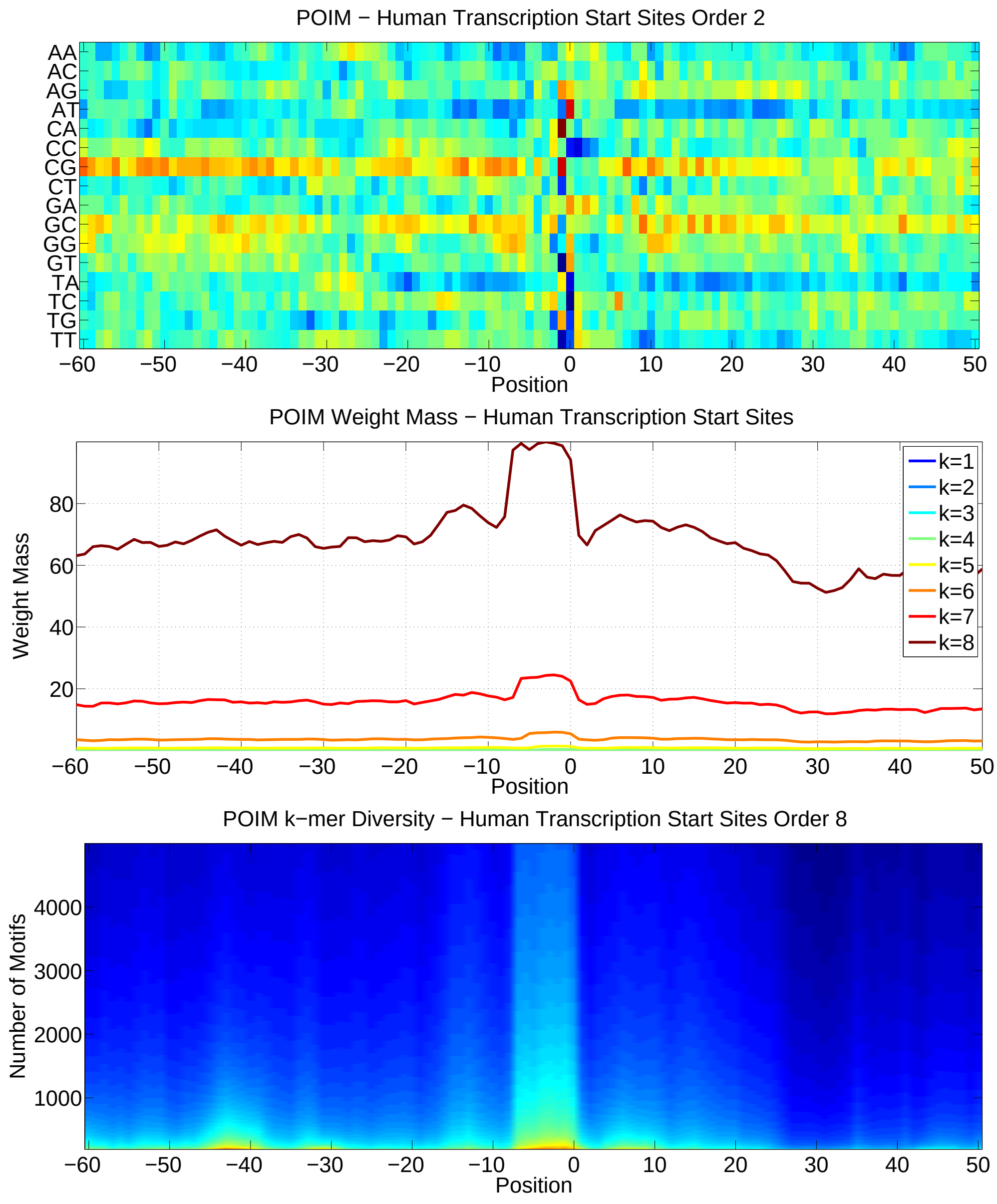


Figure 5.16: POIM visualisation (cf. Section 4.2) for human promoter sites, as obtained by *ARTS*. **(top)** POIMs of order two, showing the importance of di-nucleotides. **(center)** POIM weight mass, illustrates to which regions the SVM assigns its weights. **(bottom)** POIM diversity for order eight shows the amount of motifs (and their score) the SVM is using in detecting transcription start sites.

on MCs, the locality improved kernel (Zien et al., 2000) and two variants of the weighted degree kernel (Rätsch and Sonnenburg, 2004, Rätsch et al., 2005). We found that these existing data sets have limitations in that the sequences used for training and evaluation turn out to be too short for optimal discrimination performance. For SVMs, we showed that they are able to exploit – albeit presumably weak – features as far as 80nt away from the splice sites. In a comparison to SpliceMachine, we were able to show that our approach performs favorably when complemented with compositional information. Using the protocol proposed in Rätsch et al. (2007), we generated new data sets for the five organisms. These data sets contain sufficiently long sequences and for human as many as 9 million training examples. Based on our previous work on large scale kernel

learning (cf. Chapter 3), we were able to train SVM classifiers also on these rather big data sets. Moreover, we illustrated that the large amount of training data is indeed beneficial for significantly improving the SVM prediction performance, while MCs do not significantly improve when using much more training examples. We therefore encourage using as many examples for training as feasible to obtain the best generalisation results. For worm, fly and cress we were able to improve the performance by 4%-10% (absolute) compared to MCs. The biggest difference between the methods is observed for the most difficult task: acceptor and donor recognition on human DNA. The MCs reach only 16% and 25% auPRC, while SVMs achieve 54% and 57%, respectively. The drastic differences between organisms in the prediction performance scores can be understood as a consequence of the smaller fraction of positive examples and a higher incidence of alternative splicing in the human genome compared to the other genomes. For further comparative studies, we provide and discuss $k-$mer scoring matrices elucidating the features that are important for discrimination.
To facilitate the use of our classifiers for other studies, we provide whole genome predictions for the five organisms. Additionally, we offer an open-source stand-alone prediction tool allowing, for instance, the integration in other gene finder systems. The predictions, data sets and the stand-alone prediction tool are available for download on the supplementary website `http://www.fml.mpg.de/raetsch/projects/splice`.
In the second part we have developed a novel and *accurate* transcription start finder, called *ARTS*, for the human genome. It is based on Support Vector Machines that previously were computationally too expensive to solve this task. It has therefore been an important part of our work to develop more efficient SVM training and evaluation algorithms using sophisticated string kernels. In a carefully designed experimental study, we compared *ARTS* to other state-of-the-art transcription start finders that are used to annotate TSSs in the human genome. We show that *ARTS by far* outperforms all other methods: it achieves true positive rates that are twice as large as those of established methods. In the future, we plan to train and evaluate the *ARTS* system on other genomes and make the system and its predictions publicly available. It would be interesting to see how much of *ARTS*' higher accuracy can be translated into improved *ab initio* gene predictions.
In contrast to several previous approaches that use prior knowlege to obtain good generalisation performance (e.g., Saeys et al. (2007)), we follow an orthogonal approach: Using SVMs with sophisticated sequence kernels and large amounts of training data (as is often available for many sequence detection problems), we require only relatively little prior knowlege about the sequence detection problem. When motifs (most of the time) occur at specific positions away from the actual signal, we use the Weighted Degree kernel (cf. Section 2.3). If the motif is not that strongly localised but positionally slightly more variant we use the Weighted Degree kernel *with shifts* (cf. Section 2.3). When motifs appear without any particular localisation, i.e., there is no positional information that could be used, we resort to the spectrum kernel (cf. Section 2.1). Depending on the signal detection problem, it might be worth to use a combination of these kernels to model (weaker) sub-signals, which in the end may lead to higher prediction accuracies. Another advantage of this approach is that based on our work on positional oligomer importance matrices (cf. Section 4.2.2), we are able to derive biological insights that may later be used as additional prior information to improve the accuracy of the detectors.
In the following we summarise how many well known signal and content detection problems could be addressed using the string kernel framework, leading to highly accurate detectors (in many cases current state-of-the-art).

5.5.1 Signal Sensors

Signal sensors distinguish between true signal sites and any other possible candidate site. As potentially any location in the genome could be a true signal site, this problem may be extremely unbalanced. The usual setup is to extract windows around the true signal sites and windows around decoy sites (cf. Section 5.2) on which we train a string kernel SVM.

Transcription Start Site Recognition

One of the key elements to recognizing genic regions in the genome is to identify the promoter site. The method for finding transcription start sites (TSS) of RNA-polymerase II binding genes described in Section 5.4 and Sonnenburg et al. (2006b) mainly use a Weighted Degree kernel *with shifts* around the site and a spectrum kernel modelling the upstream and a spectrum kernel modelling the downstream regions.

Splice Site Recognition The classification tasks for the splice site sensors consist in discriminating true splice sites from decoy sites, which also exhibit the consensus dimers `AG` or `GT`. Using an SVM with a plain weighted degree or weighted degree kernel *with shifts* as in Section 5.3 and Sonnenburg et al. (2007b) leads to state-of-the-art accuracies for Donor and Acceptor splice sites.

Alternative Splicing In contrast to prior belief, the application of the splicing machinery does not always result in a unique transcript. Instead, alternative splice sites are taken leading to multiple forms of alternative splicing, like exon skipping, alternative donor splice site, alternative acceptor splice site and intron retention. Rätsch et al. (2005), Ong and Rätsch (2008) use SVMs to perform *ab-initio* detection of these major alternative splice forms at high accuracies. To this end, the Weighted Degree kernel *with shifts* is applied around the flanking splice sites and complemented with further information such as the exon and intron length distribution, reading frame and stop codon information.

Trans-acceptor Splice Sites Trans-acceptor splicing is an event that adds a splice leader sequence to an independently transcribed pre-mRNA. While unknown or fairly insignificant in most organisms, this process occurs frequently in nematodes. According to Graber et al. (2007) 70% of pre-mRNAs in *C. elegans* are affected by trans-splicing. To obtain good predictions of gene starts in nematodes, it is crucial to accurately recognise trans-splicing events. As the trained acceptor sensor also discriminates true acceptor sites from decoy sites, one needs to train a sensor that distinguishes cis-splicing from trans-splicing. Schweikert et al. (2008) use a vanilla Weighted Degree kernel and obtains high accuracies.

Translation Initiation and Termination Sites Translation converts the nucleotide sequence of mRNA into the sequence of amino acids comprising a protein. Not the whole length of mRNA is translated, but only the coding region (CDS), defined by the start (`ATG`) and stop ({`TAA,TAG,TGA`}) codons that exhibit additional sequence motifs that are identified by their associated sensors. Schweikert et al. (2008) use the Weighted Degree kernel *with shifts* and obtain high accuracies.

Polyadenylation Signal Recognition Polyadenylation of pre-mRNA is the process by which the 3' ends of eukaryotic organisms are formed. It consists of the addition of a stretch of around 50-250 Adonises at the cleavage site. Relative to the cleavage site, there is an upstream element consisting of a highly conserved AATAAA hexamer, called the polyadenylation signal (PAS), and a downstream element often described as a poorly conserved GT- or T-rich sequence. For the polyA sensor, Philips et al. (2008) combine a weighted degree kernel with shifts growing linearly to both ends of the sequence (shift 0 on the consensus signal in the center of the sequence) with four spectrum kernels, each using a quarter of the sequence.

Cleavage Site Recognition The cleavage site is located approximately 10-25 base pairs downstream of the polyadenylation signal. Around 10-30 nucleotides downstream of the cleavage site a U-rich motif is located (Chen et al., 1995). As the cleavage and polyadenylation signal site are close, prediction of the polyadenylation signal and the cleavage site is tightly coupled. Here a Weighted Degree kernel *with shifts* around the cleavage site is used (Schweikert et al., 2008).

5.5.2 Content Sensors

In contrast to signal sensors, content sensors are applied to variable length segments and are designed to recognise the typical sequence composition of the individual segments. They are used to distinguish segment types. In gene finding multiple such segment types occur, like intergenic, inter-cistronic, UTR, coding exon and coding intron. In Schweikert et al. (2008), a content sensor for each of the aforementioned segment types is learnt, by casting the problems as binary, one-against-the-rest, classification problems (as similarly done in Rätsch et al. (2007)). To avoid the influence of different length distributions of the segments on the discrimination, the negative examples are chosen such that their length distribution equals that of the positive sequences. Weighted Spectrum kernels counting the k-mers only once are used in all of these classifiers.

General Guidelines

In summary this recipe worked for us to obtain high precision sequence classifiers for many problems (see also Ben-Hur et al. (2008) for a tutorial):

1. Data preparation — the most time consuming part: Collect as much high-quality *labelled* data as possible and be aware of how you generate decoys (cf. Section 5.2). Note that SVMs are relatively robust to a few outliers (even mislabelled examples): They may compensate for outliers with an increased amount of training data.

2. Learn about the properties of the signal detection problem (very localised position dependent features, semi-position dependent, position independent) and choose a Weighted Degree kernel, a Weighted Degree kernel *with shifts* or spectrum kernel or even a combination of these kernels.[13]

3. Decide about performance measure (auROC, auPRC,...): As signal detection problems are usually very unbalanced it is suggested to use the auPRC.

[13]In case a number of positionally variable motifs are known a-priori, one may use sequence information around the “best” match of the motif on the sequence and the distance of the match to a reference site to increase performance (Schultheiss et al., 2008).

4. Perform model selection (C, Kernels, Kernel parameters, window sizes around the signal and regions kernels are applied to): When you have few (positive) data points use cross-validation. In case you have *several thousand* of examples a fixed-split split into training, validation and test set is sufficient.
5. Evaluate method on unseen test data using your performance measure.

6 Learning to Predict Gene Structures

In the first part of this chapter (Section 6.1), we use the improved splicing signal and exon/intron content sensors together with the Hidden Semi-Markov Support Vector Machine framework from Rätsch and Sonnenburg (2007), Rätsch et al. (2007) to accurately detect splice forms. The second part (Section 6.2) gives an outlook on the steps involved to extend this system into a full-fledged gene finding system.
This chapter is largely based on Rätsch and Sonnenburg (2007), Rätsch et al. (2007), Schweikert et al. (2008).

6.1 Splice Form Prediction

The problem of gene structure prediction is to segment nucleotide sequences (so-called pre-mRNA sequences generated by transcription; Figure 6.1 revisits the protein synthesis process as already explained in more detail in Section 1.1) into exons and introns. The exon-intron and intron-exon boundaries (cf. Figure 6.1) are defined by sequence motifs almost always containing the letters `GT` and `AG`, respectively. However, these dimers appear very frequently and one needs sophisticated methods to recognise true splice sites as the ones presented in Section 5.3. While such accurate splice site detectors seem sufficient to segment a gene, they are limited to "point predictions" ignoring any context, like average length of a segment or the ordering of segments. To incorporate further knowledge and to learn additional features of the segmentation problem at hand, one requires structured output learning methods. Currently, mostly HMM-based

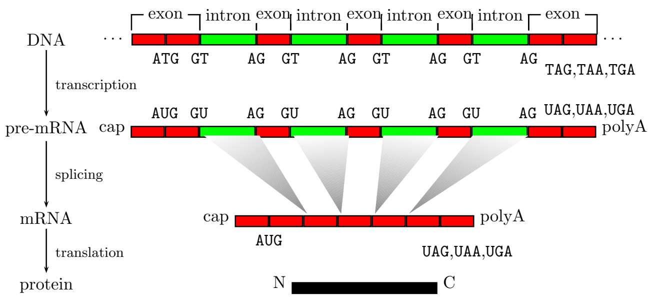


Figure 6.1: The major steps in protein synthesis (Figure is based on Lewin (2000)). A transcript of a gene starts with an exon and may then be interrupted by an *intron*, followed by another exon, intron and so on until it ends in an exon. In this section, we learn the unknown formal mapping from the pre-mRNA to the mRNA.

methods such as *Genscan* (Burge and Karlin, 1997), Snap (Korf, 2004) or *ExonHunter* (Brejova et al., 2005) have been applied to this problem and also to the more difficult problem of gene finding. In this section, we show that our newly developed method *mSplicer* (Rätsch and Sonnenburg, 2007, Rätsch et al., 2007) is applicable to this task and achieves very competitive results.
The grammar for basic splice form prediction is very simple. A gene starts with an exon which is interrupted by an intron followed by another exon, and so on, finally ending with an exon. Figure 6.2 illustrates the "grammar" that we use for gene structure prediction. We only require four different labels: start, exon-end, exon-start and end.

Biologically it makes sense to distinguish between first, internal, last and single exons, as their typical lengths are quite different. Each of these exon types correspond to one transition in the model. States two and three recognise the two types of splice sites and the transition between these states defines an intron. In this model (cf. Figure

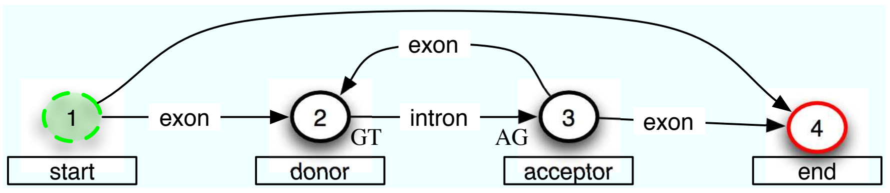


Figure 6.2: An elementary state model for unspliced mRNA: The start is either directly followed by the end or by an arbitrary number of donor-acceptor splice site pairs. Figure taken from Rätsch and Sonnenburg (2007).

6.2), states correspond to segment boundaries to be detected by the splicing signal sensors and transitions $a \rightarrow b$ denote segments to be detected by the content sensors. State 1 and 4 (translation start and end) are assumed to be given a-priori. Note that the *mSplicer* model employing reading frame information is significantly more complex. The reader is referred to Rätsch et al. (2007) for further details.

6.1.1 Learning the Signal and Content Sensors

Learning the Splice Site Signal Detectors From the training sequences (Set 1, cf. Appendix B.4), we extracted sequences of confirmed splice sites (intron start and end). For intron start sites we used a window of $[-80, +60]$ around the site. For intron end sites we used $[-60, +80]$. From the training sequences we also extracted non-splice sites, which are within an exon or intron of the sequence and have `AG` or `GT` consensus. We train Weighted Degree kernel based signal detectors like in Section 5.3 for acceptor and donor splice sites. The result are the two discriminative functions $\bar{F}_2$ and $\bar{F}_3$. All model parameters (including the window size) have been tuned on the validation set (Set 2). SVM training for *C. elegans* resulted in 79,000 and 61,233 support vectors for detecting intron start and end sites, respectively. The area under the Receiver Operator Characteristic Curve for the resulting classifiers on the test set are 99.74% (intron start) and 99.62% (intron end).

Learning the Exon and Intron Content Sensors To obtain the exon content sensor, we derived a set of exons from the training set. As negative examples we used subsequences of intronic sequences sampled such that both sets of strings have roughly the same length distribution. We trained SVMs using a variant of the *Spectrum kernel* (Zhang et al., 2003) of degree $d = 3, 4, 5, 6$. We proceeded analogously for the intron content sensor. The model parameters have been obtained by tuning them on the validation set.

6.1.2 Integration

The idea is to learn a function that assigns a score to a splice form such that the true splice form is ranked highest while all other splice forms have a significantly lower score. The function depends on parameters that are determined during training of the algorithm. In our case it is defined in terms of several functions determining the contributions of the content sensors ($f_{E,d}$ and $f_{I,d}$), the splice site predictors (S_{AG} and S_{GT}) and the lengths of introns and exons (S_{L_I}, S_{L_E}, $S_{L_{E,s}}$, $S_{L_{E,f}}$ and $S_{L_{E,l}}$).

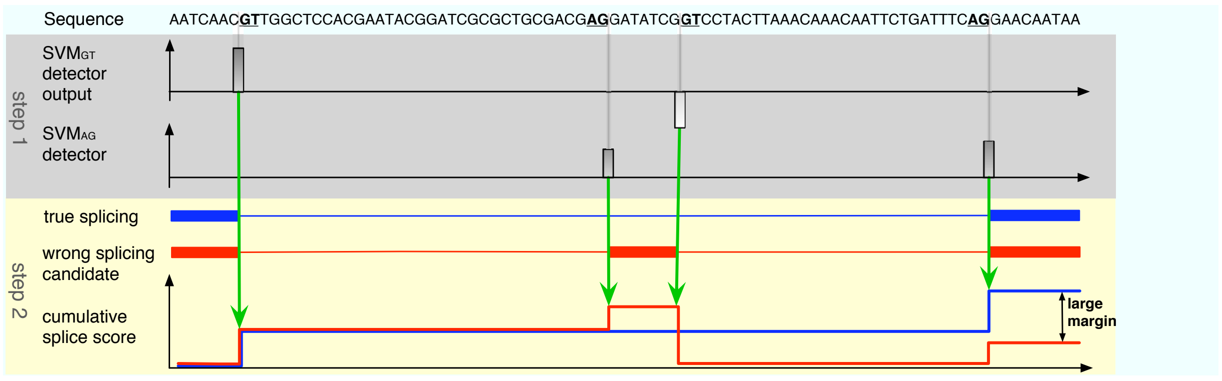


Figure 6.3: Given the start of the first and the end of the last exon, our system (*mSplicer*) first scans the sequence using SVM detectors trained to recognise donor (SVM_{GT}) and acceptor splice sites (SVM_{AG}). The detectors assign a score to each candidate site, shown below the sequence. In combination with additional information, including outputs of SVMs recognizing exon/intron content, and scores for exon/intron lengths (not shown), these splice site scores contribute to the cumulative score for a putative splicing isoform. The bottom graph (step 2) illustrates the computation of the cumulative scores for two splicing isoforms, where the score at end of the sequence is the final score of the isoform. The contributions of the individual detector outputs, lengths of segments as well as properties of the segments to the score are adjusted during training. They are optimised such that the *margin* between the true splicing isoform (shown in blue) and all other (wrong) isoforms (one of them is shown in red) is maximised. Prediction on new sequences works by selecting the splicing isoform with the maximum cumulative score. This can be implemented using dynamic programming related to decoding generalised HMMs (Kulp et al., 1996), which also allows one to enforce certain constrains on the isoform (e.g., an open reading frame). Figure taken from Rätsch et al. (2007).

We assume that the start of the first exon p_s and the end of last exon p_e are given. Then, a splice form for a sequence s is given by a sequence of donor-acceptor pairs $(p_i^{\text{GT}}, p_i^{\text{AG}})$. The *cumulative splice score* $S(s, p_s, p_e, \{p_i^{\text{GT}}, p_i^{\text{AG}}\}_{i=1}^n)$ for a sequence s was computed as follows:

- If there is only a single exon, i.e., $n = 0$, then

$$S(s, p_s, p_e, \{\}) = S_E(s_{[p_s,p_e]}) + S_{L_{E,s}}(p_e - p_s),$$

  where $s_{[a,b]}$ is the subsequence of s between positions a and b, $S_E(s) := \sum_{d=3}^{6} f_{E,d}(\text{SVM}_{E,d}(s))$ is the score for the exon content and $S_{L_E,s}(l)$ is the score for the length l of a single exon, whereby $\text{SVM}_{E,d}(s)$ is the output of the exon content sensor using a kernel of degree d as described above.

- Otherwise, we used the following function:

$$\begin{aligned}
S(s, p_s, p_e, \{p_i^{\text{GT}}, p_i^{\text{AG}}\}_{i=1}^n) &:= S_{L_{E,f}}(p_1^{\text{GT}} - p_s) + S_E(s_{[p_s,p_1^{\text{GT}}]}) \\
&+ S_{L_{E,l}}(p_e - p_n^{\text{AG}}) + S_E(s_{[p_n^{\text{AG}},p_e]}) \\
&+ \sum_{i=1}^{n} \left[S_{L_I}(p_i^{\text{AG}} - p_i^{\text{GT}}) + S_I(s_{[p_i^{\text{GT}},p_i^{\text{AG}}]}) + S_{\text{AG}}(p_i^{\text{AG}}) + S_{\text{GT}}(p_i^{\text{GT}}) \right] \\
&+ \sum_{i=1}^{n-1} \left[S_E(s_{[p_i^{\text{AG}},p_{i+1}^{\text{GT}}]}) + S_{L_E}(p_i^{\text{AG}} - p_{i+1}^{\text{GT}}) \right],
\end{aligned}$$

  where $S_I(s) := \sum_{d=3}^{6} f_{I,d}(\text{SVM}_{I,d}(s))$ is the intron content score using the SVM intron content output $\text{SVM}_{I,d}(s)$ using a kernel of degree d, $S_{\text{AG}}(p) :=$

$f_{\mathrm{AG}}(\mathrm{SVM}_{\mathrm{AG}}(p))$ and $S_{\mathrm{GT}}(p) := f_{\mathrm{GT}}(\mathrm{SVM}_{\mathrm{GT}}(p))$ are the scores for acceptor and donor splice sites, respectively, using the $\mathrm{SVM}_{\mathrm{AG}}$ and $\mathrm{SVM}_{\mathrm{GT}}$ output for the putative splice sites at position p. Moreover, $S_{L_{E,f}}(l)$, $S_{L_{E,l}}(l)$, $S_{L_E}(l)$ and $S_{L_I}(l)$ are the length scores for first exons, last exons, internal exons and introns, respectively, of length l.

The above model has 15 functions as parameters. We model them as piecewise-linear functions with $P = 30$ support points at $\frac{1}{P-1}$ quantiles as observed in the training set. For S_{AG}, S_{GT}, $S_{E,d}$ and $S_{I,d}$ $(d = 3, \ldots, 6)$ we require that they are monotonically increasing, since a larger SVM output should lead to a larger score.
To determine the parameters of the model, we propose to solve the following optimisation problem (Rätsch and Sonnenburg, 2007) that uses a set of N training sequences $s_1, \ldots, s_N$ with start points $p_{s,1}, \ldots, p_{s,N}$, end points $p_{e,1}, \ldots, p_{e,N}$ and true splicing isoforms $\sigma_1, \ldots, \sigma_N$:

$$\begin{aligned} \text{minimise} \quad & \sum_{i=1}^{N} \xi_i + C\mathbf{P}(\theta) \\ \text{subject to} \quad & S(s_i, p_{s,i}, p_{e,i}, \sigma_i) - S(s_i, p_{s,i}, p_{e,i}, \tilde{\sigma}_i) \geq 1 - \xi_i, \end{aligned} \tag{6.1}$$

for all $i = 1, \ldots, N$ and all possible splicing isoforms $\tilde{\sigma}_i$ for sequence s_i, where

$$\theta = [\theta_{\mathrm{AG}}, \theta_{\mathrm{GT}}, \theta_{E,3}, \ldots, \theta_{E,6}, \theta_{I,3}, \ldots, \theta_{I,6}, \theta_{L_E}, \theta_{L_{E,f}}, \theta_{L_{E,l}}, \theta_{L_{E,s}}, \theta_{L_I}]$$

is the parameter vector parameterizing all 15 functions (the 30 function values at the support points) and $\mathbf{P}$ is a regulariser. The parameter C is the regularisation parameter. The regulariser is defined as follows:

$$\begin{aligned} \mathbf{P}(\theta) := & \sum_{i=1}^{P-1} |\theta_{L_{E,s},i} - \theta_{L_{E,s},i+1}| + \sum_{i=1}^{P-1} |\theta_{L_{E,f},i} - \theta_{L_{E,f},i+1}| \\ & + \sum_{i=1}^{P-1} |\theta_{L_{E,l},i} - \theta_{L_{E,l},i+1}| + \sum_{i=1}^{P-1} |\theta_{L_E,i} - \theta_{L_E,i+1}| \\ & + \sum_{i=1}^{P-1} |\theta_{L_I,i} - \theta_{L_I,i+1}| + (\theta_{\mathrm{AG},P} - \theta_{\mathrm{AG},1}) \\ & + (\theta_{\mathrm{GT},P} - \theta_{\mathrm{GT},1}) + \sum_{d=3}^{6} (\theta_{E,d,P} - \theta_{E,d,1}) + \sum_{d=3}^{6} (\theta_{I,d,P} - \theta_{I,d,1}) \end{aligned}$$

with the intuition that the piecewise linear functions should have small absolute differences (reducing to the difference from start to end for monotonic functions).
Based on the ideas presented in Altun et al. (2003), we solve the optimisation problem (6.1) using sequences from the cDNA sequences in the training set (these sequence were not used for training the signal and content sensors). For the model selection for parameters C and P, we use an independent validation set of cDNA sequences.

Outline of an Optimization Algorithm The number of constraints in (6.1) can be very large, which may constitute challenges for efficiently solving problem (6.1). Fortunately, only a few of the constraints usually are active and working set methods can be applied in order to solve the problem for a larger number of examples. The idea is to start with small sets of negative (i.e. false) labellings $\tilde{\sigma}_i$ for every example. One solves (6.1) for the smaller problem and then identifies labellings $\sigma \in \tilde{\Sigma}$ (with $\tilde{\Sigma}$ being the set of all false labellings) that maximally violate constraints, i.e.

$$\sigma = \operatorname*{argmax}_{\sigma \in \tilde{\Sigma}} S(s_i, p_{s,i}, p_{e,i}, \sigma_i) - S(s_i, p_{s,i}, p_{e,i}, \sigma) \quad (6.2)$$

where θ is the intermediate solution of the restricted problem. The new constraint generated by the negative labelling is then added to the optimization problem. The method described above is also known as column generation method or cutting-plane algorithm and can be shown to converge to the optimal solution θ^* (Tsochantaridis et al., 2005). In our case, training of step 2 takes about 2h on a standard PC employing an off-the-shelf optimizer (ILOG CPLEX, 1994) for solving the resulting linear programs.

Decoding of Splice Forms

To produce the "best" splice form prediction $\hat{\sigma}$ based on the splice form scoring function $S(s, p_s, p_e, \sigma)$, one has to maximise S with respect to the splice form σ, i.e.,

$$\hat{\sigma}(s, p_s, p_e) = \operatorname*{argmax}_{\sigma \in \Sigma(s, p_s, p_e)} S(s, p_s, p_e, \sigma).$$

To *efficiently* determine the splicing isoform $\hat{\sigma}$ with maximal score, Viterbi decoding (Viterbi, 1967) is used. We assume that the sequence s, the starting position p_s and end positions p_e are given. The prediction $\hat{\sigma}$ has to satisfy certain rules, in particular that introns are terminated with the `GT/GC` and `AG` splice sites dimers and that they are not overlapping. Additionally, we require that introns are at least 30nt and exons at least 2nt long and restrict the maximal intron and exon length to 22,000 (the longest known intron in *C. elegans*). If one uses open reading frame information, one additionally has to make sure that the spliced sequence does not contain stop codons.

6.1.3 Experimental Evaluation

To estimate the out-of-sample accuracy, we apply our method to the independent test dataset (cf. Appendix B.4). We compare our proposed method *mSplicer* to *ExonHunter*[1] on 1177 test sequences. We greatly outperform the *ExonHunter* method: our method obtains almost 1/3 of the test error of *ExonHunter* (cf. Table 6.1) on the (CI) dataset.

[1]The method was trained by their authors on the same training data.

Data	*CI* set				
Method	error rate	exon Sn	exon Sp	exon nt Sn	exon nt Sp
mSplicer OM	4.8%	98.9%	99.2%	99.2%	99.9%
ExonHunter	9.8%	97.9%	96.6%	99.4%	98.1%
SNAP	17.4%	95.0%	93.3%	99.0 %	98.9%
	UCI set				
mSplicer SM	13.1%	96.7%	96.8%	98.9%	97.2%
ExonHunter	36.8%	89.1%	88.4%	98.2%	97.4%

Table 6.1: Shown are the splice form error rates (1-accuracy), exon sensitivities, exon specificities, exon nucleotide sensitivities, exon nucleotide specificities of *mSplicer* (with (*OM*) and without (*SM*) using ORF information) as well as *ExonHunter* and *SNAP* on two different problems: mRNA including (UCI) and excluding (CI) untranslated region.

Simplifying the problem by only considering sequences between the start and stop codons allows us to also include *SNAP* in the comparison. In this setup additional biological information about the so-called “open reading frame” is used: As there was only a version of *SNAP* available that uses this information, we incorporated this extra knowledge also in our model (marked *mSplicer* OM) and also used another version of *Exonhunter* that also exploits that information, in order to allow a fair comparison. The results are shown in Table 6.1. On dataset (UCI), the best competing method achieves an error rate of 9.8%, which is more than twice the error rate of our method.

Retrospective Evaluation of the Wormbase Annotation Comparing the splice form predictions of our method with the annotation on completely unconfirmed genes,[2] we find disagreements in 62.5% (*SM*) or 50.0% (*OM*) of such genes, respectively. Assuming that on this set, our method performs as well as reported above, one could conclude that the annotation is rather inaccurate on yet *unconfirmed genes*.
To validate this assumption Rätsch et al. (2007) performed a retrospective analysis, where the performance of *mSplicer* was investigated on newly annotated (confirmed) genes: The predictions of *mSplicer* largely agreed with the annotation on the (new) confirmed genes. However, a significant disagreement between the new annotation and our predictions on unconfirmed genes was observed, which strongly suggests inaccuracies in the current annotation. A further wet lab confirmation on 20 unconfirmed genes randomly chosen from those where the *mSplicer* predictions differed significantly from the annotation was performed. The predictions of *mSplicer* without ORF information were completely correct in 15 out of the 20 cases (error rate 25%), while the annotation was never exactly matching all new splice sites. A custom track of *mSplicer* is now available at Wormbase.

6.2 Summary and Outlook: Gene Finding

We have successfully applied our method on large scale gene structure prediction appearing in computational biology, where our method obtains less than a half of the error rate of the best competing HMM-based method. We are currently working on incorporating our prediction in the public genome annotations. To extend the mSplicer (cf. Section 6.1 and Rätsch et al. (2007)) gene structure predictor to become a full-fledged gene-finding system, the already complex *mSplicer* model has to undergo complex changes to integrate further signals. Recall that mSplicer currently assumes a given transcription start and end site and only predicts the spliceform, i.e., the segmentation of the gene into exons and introns. To this end it is using a two layer architecture: In the first layer signal sensors, that predict the acceptor and donor splice site and content sensors that distinguish exons from introns are learnt using the string kernels discussed in Chapter 2 and the `linadd` based learning algorithm for string kernel based SVMs from Chapter 3. Even though these “simple” two-class detectors achieve state-of-the-art results solving their respective sequence detection problem, they taken alone perform poorly when predicting spliceforms. The second layer combines the first layer predictions, normalizing and properly weighting their scores against each other. Here semi-hidden Markov support vector machines (Rätsch and Sonnenburg, 2007) are used and a state model describing the valid transitions, like in Figure 6.2, is defined. In this model, states denote signal and transitions content sensors, which are normalised via piecewise linear functions whose parameters are learnt in semi HM-SVM training. This system can be

[2]A gene is called *confirmed* if it is covered by cDNA and EST sequences and *unconfirmed* otherwise.

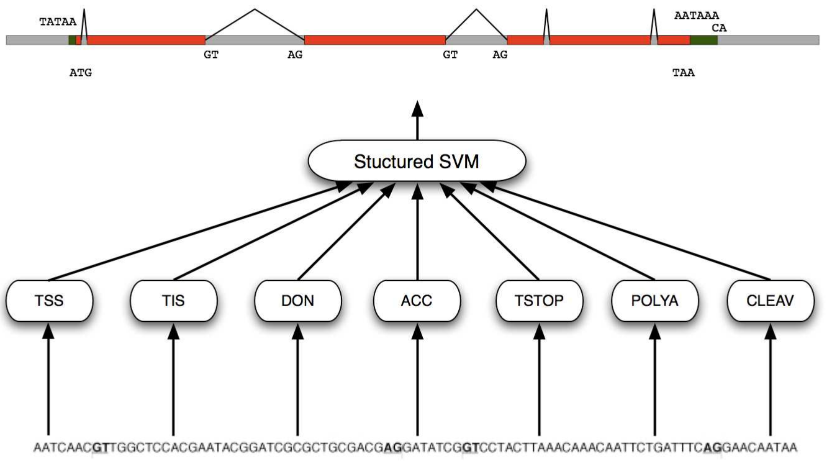

TATAA
ATG
GT
AG
GT
AG
AATAAA
CA
TAA
Stuctured SVM
TSS
TIS
DON
ACC
TSTOP
POLYA
CLEAV
AATCAACGTTGGCTCCACGAATACGGATCGCGCTGCGACGAGGATATCGGTCCTACTTAAACAAACAATTCTGATTTCAGGAACAATAA

Figure 6.4: The design of the mGene (Schweikert et al., 2008) gene finding system. Figure taken from Schweikert et al. (2008).

upscaled into a full gene finder by introducing further signal and content sensors, that not only recognise the donor and acceptor splicing signal but also others like the transcription start and stop site, the translation initiation site, the polyadenylation signal and the cleavage site, as well as further content sensors that distinguish exon content in the untranslated regions from the ones in the coding regions etc. Figure 6.4 displays the extended model of the mGene gene finder (Schweikert et al., 2008). This gene finding system is indeed an extension of the previously described mSplicer and is currently being developed at the Friedrich Miescher Laboratory in Tübingen by members of Gunnar Rätsch's group (mainly by Gabriele Schweikert). The first version of this gene finder took part in the nGASP genome annotation challenge, and even this early version was one of the top performing (if not the best) on the worm *C. elegans* (Schweikert et al., 2008). The state model of mGene is a lot more complex and the increased amount of signal detectors requires that model selection and training are done in an automated way. In this model, mistakes may occur not only at the splice junctions but also at the other signal sites, like the transcription start site. Since the cost of errors may vary (e.g., a wrongly predicted transcription start is more costly than a wrongly predicted splice site) one needs ways to treat errors differently. This can be resolved by introducing a new loss function in Equation (6.1). Other extensions include additional states to model operons (i.e., a transcriptional unit coding for multiple proteins) and a single state to model the large intergenic regions.

7 Summary and Outlook

With the development of novel sequencing technologies the way has been paved for cost efficient high-throughput whole genome sequencing. Handling the increasing wealth of data requires efficient and accurate computational methods for sequence analysis. These methods are required to solve one of the most important problems in computational biology — the localisation of genes on DNA.
In this work, we developed novel machine learning methods for genomic sequence analysis. First, we introduced the reader to a branch of bioinformatics, the field of sequence analysis and signal detection (cf. Section 1.1), and to the learning of classifiers from DNA with a focus on Support Vector Machines (SVMs, Section 1.2).

The aim was to improve detection of various genomic signals using SVMs. Based on prior knowledge and previous work (Sonnenburg, 2002), we developed sophisticated and computationally efficient string kernels that are applicable to a wide range of signal detection problems. These kernels are the TOP kernel (in Section 2.4.2), the Weighted Degree kernel and the Weighted Degree kernel *with shifts* (cf. Section 2.3). In Section 2.4.3, we related the TOP and Fisher kernels (Jaakkola and Haussler, 1999) with the WD and Spectrum kernels. As it turns out, the latter are special cases of the TOP/FK for particular probabilistic models.

Until now, string kernel based SVMs were not applicable to *genomic scale* signal detection problems. We developed fast string kernels that require only linear computational effort in the length of the input sequence and are suitable for a wide range of sequence classification tasks. However, SVM training and testing was too computationally demanding for large sample sizes. We significantly improved on this situation by deriving large scale learning methods that enable training of string kernel based SVMs on up to 10 million sequences and their application to 6 billion examples within a reasonable time (cf. Chapter 3).

Even though modern kernel methods with complex, oligomer-based sequence kernels are very powerful for biological sequence classification, until now no satisfactory means were available to aid the understanding of their complex decision surfaces. We have developed Positional Oligomer Importance Matrices and Multiple Kernel Learning algorithms that allow us to determine the most discriminating motifs and thus, provide some insight about the learnt representation and the application at hand (cf. Chapter 4). Multiple Kernel Learning is general in the sense that it works with arbitrary kernels and is well suited to fuse data from different sources. POIMs were specifically developed to understand string kernel based SVMs and offer a leap of quality on that domain.

Equipped with this tool set, we can now accurately detect various genomic signals, like the transcription start of a gene (Section 5.4), splice sites (Section 5.3) and many other signals (Section 5.5). Specifically, we obtained state-of-the-art detectors that perform at less than *half* the error rate of established methods.

From the beginning, the ultimate goal of learning these isolated signal detectors, was to combine them to predict genes. As a first step, we addressed the problem of splice form prediction. Using a computationally efficient two-layer approach, the predictions from the signal sensors in Chapter 5 are integrated in a structure learning layer, solving a label sequence learning problem (cf. Section 6.1). Our method *mSplicer*, obtains error rates that are again less than half of the error rate of the best competing HMM-based method for the problem of splice form prediction in *C. elegans*.
All of the core algorithms presented in this doctoral thesis are implemented in the SHOGUN machine learning toolbox (cf. Section C) and are freely available from `http://www.shogun-toolbox.org`.

We concluded with an overview of how the *mSplicer* (cf. Section 6.1 and Rätsch et al. (2007)) splice form predictor has been developed into the complete gene finding system *mGene* (Schweikert et al., 2008). To be computationally efficient, *mGene* currently uses a two-layer approach. Therefore, *mGene* may not be capable of capturing certain important inter-dependencies (e.g., exclusive OR relations). However, *mGene* outperformed most of its competitors in a recent gene prediction challenge on nematodes. As a result, it is now included in Wormbase (Wormbase) and aids annotation of the *C. elegans* genome and that of other nematodes. While our work on large scale learning has paved the way towards a complete gene finding system based on highly accurate string kernel SVMs, the framework is currently only tractable for organisms with relatively small genomes. Recently, training *linear* SVMs has been sped up significantly (Chang et al., 2008, Franc and Sonnenburg, 2008). We expect that string kernel based SVMs and label sequence learning techniques will benefit from these advances. Being able to efficiently learn the label sequence with *mGene* using a single layer approach could significantly increase its expressive power. Thus, future research directed at large scale label sequence learning methods is called for to ultimately scale the system to large genomes, like the one of our species.

A Derivations

A.1 Derivation of the MKL Dual for Generic Loss Functions

In this section we derive the dual of Equation (4.9). In analogy to Bach et al. (2004) we treat the problem as a second order cone program. This derivation also appeared in (Sonnenburg et al., 2006a).

We start from the MKL primal problem Equation (4.9):

$$\begin{array}{rl}
\min & \frac{1}{2}\left(\sum_{k=1}^{K}\|\mathbf{w}_k\|\right)^2 + \sum_{i=1}^{N} L(f(\mathbf{x}_i), y_i) \\
\text{w.r.t.} & \mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_K) \in \mathbb{R}^{D_1} \times \cdots \times \mathbb{R}^{D_K} \\
\text{s.t.} & f(\mathbf{x}_i) = \sum_{k=1}^{K} \langle \Phi_k(\mathbf{x}_i), \mathbf{w}_k \rangle + b, \;\; \forall i = 1, \ldots, N
\end{array}$$

Introducing $u \in \mathbb{R}$ allows us to move $\sum_{k=1}^{K}\|\mathbf{w}_k\|$ into the constraints and leads to the following equivalent problem

$$\begin{array}{rl}
\min & \frac{1}{2}u^2 + \sum_{i=1}^{N} L(f(\mathbf{x}_i), y_i) \\
\text{w.r.t.} & u \in \mathbb{R}, \; (\mathbf{w}_1, \ldots, \mathbf{w}_K) \in \mathbb{R}^{D_1} \times \cdots \times \mathbb{R}^{D_K} \\
\text{s.t.} & f(\mathbf{x}_i) = \sum_{k=1}^{K} \langle \Phi_k(\mathbf{x}_i), \mathbf{w}_k \rangle + b, \; \forall i = 1, \ldots, N \\
 & \sum_{k=1}^{K} \|\mathbf{w}_k\| \le u
\end{array}$$

Using $t_k \in \mathbb{R}, \; k = 1, \ldots, K$, it can be equivalently transformed into

$$\begin{array}{rl}
\min & \frac{1}{2}u^2 + \sum_{i=1}^{N} L(f(\mathbf{x}_i), y_i) \\
\text{w.r.t.} & u \in \mathbb{R}, \; t_k \in \mathbb{R}, \mathbf{w}_k \in \mathbb{R}^{D_k}, \; \forall k = 1, \ldots, K \\
\text{s.t.} & f(\mathbf{x}_i) = \sum_{k=1}^{K} \langle \Phi_k(\mathbf{x}_i), \mathbf{w}_k \rangle + b, \; \forall i = 1, \ldots, N \\
 & \|\mathbf{w}_k\| \le t_k, \;\; \sum_{k=1}^{K} t_k \le u.
\end{array}$$

Recall that the second-order cone of dimensionality D is defined as

$$\mathcal{K}_D = \{(\mathbf{x}, c) \in \mathbb{R}^D \times \mathbb{R}, \; \|\mathbf{x}\|_2 \le c\}.$$

We can thus reformulate the original MKL primal problem (Equation (4.9)) using the following *equivalent* second-order cone program, as the norm constraint on $\mathbf{w}_k$ is implicitly taken care of:

A.1.1 Conic Primal

$$\begin{array}{rl}
\min & \frac{1}{2}u^2 + \sum_{i=1}^{N} L(f(\mathbf{x}_i), y_i) \\
\text{w.r.t.} & u \in \mathbb{R},\ t_k \in \mathbb{R}, (\mathbf{w}_k, t_k) \in \mathcal{K}_{D_k},\ \forall k = 1, \dots, K \\
\text{s.t.} & f(\mathbf{x}_i) = \sum_{k=1}^{K} \langle \Phi_k(\mathbf{x}_i), \mathbf{w}_k \rangle + b,\ \forall i = 1, \dots, N \\
& \sum_{k=1}^{K} t_k \le u
\end{array}$$

We are now going to derive the conic dual following the recipe of Boyd and Vandenberghe (2004) (see p. 266). First, we derive the conic Lagrangian and then using the infimum w.r.t. the primal variables to obtain the conic dual. We therefore introduce Lagrange multipliers $\boldsymbol{\alpha} \in \mathbb{R}^K$, $\gamma \in \mathbb{R}$, $\gamma \ge 0$ and $(\boldsymbol{\lambda}_k, \mu_k) \in \mathcal{K}^*{}_D$ living on the self dual cone $\mathcal{K}^*{}_D = \mathcal{K}_D$. Then, the conic Lagrangian is given as

$$\begin{aligned}
\mathcal{L}(\mathbf{w}, b, \boldsymbol{t}, u, \boldsymbol{\alpha}, \gamma, \boldsymbol{\lambda}, \boldsymbol{\mu}) \quad = \quad & \frac{1}{2}u^2 + \sum_{i=1}^{N} L(f(\mathbf{x}_i), y_i) - \sum_{i=1}^{N} \alpha_i f(\mathbf{x}_i) + \\
+ \sum_{i=1}^{N} \alpha_i \sum_{k=1}^{K} \left(\langle \Phi_k(\mathbf{x}_i), \mathbf{w}_k \rangle + b \right) + \gamma & \left(\sum_{k=1}^{K} t_k - u \right) - \sum_{k=1}^{K} \left(\langle \boldsymbol{\lambda}_k, \mathbf{w}_k \rangle + \mu_k t_k \right).
\end{aligned}$$

To obtain the dual, the derivatives of the Lagrangian w.r.t. the primal variables, $\mathbf{w}, b, \boldsymbol{t}, u$ have to vanish, which leads to the following constraints

$$\begin{aligned}
\partial_{\mathbf{w}_k} \mathcal{L} &= \sum_{i=1}^{N} \alpha_i \Phi_k(\mathbf{x}_i) - \boldsymbol{\lambda}_k \ \Rightarrow\ \boldsymbol{\lambda}_k = \sum_{i=1}^{N} \alpha_i \Phi_k(\mathbf{x}_i) \\
\partial_b \mathcal{L} &= \sum_{i=1}^{N} \alpha_i \ \Rightarrow\ \sum_{i=1}^{N} \alpha_i = 0 \\
\partial_{t_k} \mathcal{L} &= \gamma - \mu_k \ \Rightarrow\ \gamma = \mu_k \\
\partial_u \mathcal{L} &= u - \gamma \Rightarrow \gamma = u \\
\partial_{f(\mathbf{x}_i)} \mathcal{L} &= L'(f(\mathbf{x}_i), y_i) - \alpha_i \Rightarrow f(\mathbf{x}_i) = L'^{-1}(\alpha_i, y_i).
\end{aligned}$$

In the equation, L' is the derivative of the loss function w.r.t. $f(x)$ and L'^{-1} is the inverse of L' (w.r.t. $f(x)$) for which to exist L is required to be strictly convex and differentiable. We now plug in what we have obtained above, which makes $\boldsymbol{\lambda}_k$, μ_k and all of the primal variables vanish. Thus, the dual function is

$$\begin{aligned} D(\boldsymbol{\alpha},\gamma) &= -\frac{1}{2}\gamma^2 + \sum_{i=1}^{N} L(L'^{-1}(\alpha_i,y_i),y_i) - \sum_{i=1}^{N}\alpha_i L'^{-1}(\alpha_i,y_i) + \\ &\quad + \sum_{i=1}^{N}\alpha_i \sum_{k=1}^{K}\langle\Phi_k(\mathbf{x}_i),\mathbf{w}_k\rangle - \sum_{k=1}^{K}\sum_{i=1}^{N}\alpha_i\langle\Phi_k(\mathbf{x}_i),\mathbf{w}_k\rangle \\ &= -\frac{1}{2}\gamma^2 + \sum_{i=1}^{N} L(L'^{-1}(\alpha_i,y_i),y_i) - \sum_{i=1}^{N}\alpha_i L'^{-1}(\alpha_i,y_i). \end{aligned}$$

As constraints remain $\gamma \geq 0$, due to the bias $\sum_{i=1}^{N}\alpha_i = 0$ and the second-order cone constraints

$$\|\boldsymbol{\lambda}_k\| = \left\|\sum_{i=1}^{N}\alpha_i\Phi_k(\mathbf{x}_i)\right\|_2 \leq \gamma, \quad \forall k = 1,\ldots,K.$$

This leads to:

$$\begin{aligned} \max &\quad -\frac{1}{2}\gamma^2 + \sum_{i=1}^{N} L(L'^{-1}(\alpha_i,y_i),y_i) - \sum_{i=1}^{N}\alpha_i L'^{-1}(\alpha_i,y_i) \\ \text{w.r.t.} &\quad \gamma\in\mathbb{R},\ \boldsymbol{\alpha}\in R^N \\ \text{s.t.} &\quad \gamma \geq 0,\ \sum_{i=1}^{N}\alpha_i = 0 \\ &\quad \left\|\sum_{i=1}^{N}\alpha_i\Phi_k(\mathbf{x}_i)\right\|_2 \leq \gamma,\ \forall k=1,\ldots,K \end{aligned}$$

Squaring the latter constraint, multiplying by $\frac{1}{2}$, relabeling $\frac{1}{2}\gamma^2 \mapsto \gamma$ and dropping the $\gamma \geq 0$ constraint as it is fulfilled implicitly, we obtain the MKL dual for arbitrary strictly convex loss functions.

A.1.2 Conic Dual

$$\begin{aligned} \min &\quad \underbrace{\gamma - \sum_{i=1}^{N} L(L'^{-1}(\alpha_i,y_i),y_i) + \sum_{i=1}^{N}\alpha_i L'^{-1}(\alpha_i,y_i)}_{:=T} \\ \text{w.r.t.} &\quad \gamma\in\mathbb{R},\ \boldsymbol{\alpha}\in R^N \\ \text{s.t.} &\quad \sum_{i=1}^{N}\alpha_i = 0 \\ &\quad \frac{1}{2}\left\|\sum_{i=1}^{N}\alpha_i\Phi_k(\mathbf{x}_i)\right\|_2^2 \leq \gamma,\ \forall k=1,\ldots,K. \end{aligned}$$

Finally, adding the second term in the objective (T) to the constraint on γ and relabeling $\gamma+T \mapsto \gamma$ leads to the reformulated dual Equation (4.10), the starting point from which one can derive the SILP formulation in analogy to the classification case.

A.1.3 Loss functions

Quadratic Loss For the quadratic loss case $L(x, y) = C(x - y)^2$ we obtain as the derivative $L'(x, y) = 2C(x - y) =: z$ and $L'^{-1}(z, y) = \frac{1}{2C}z + y$ for the inverse of the derivative. Recall the definition of

$$S_k(\boldsymbol{\alpha}) = -\sum_{i=1}^{N} L(L'^{-1}(\alpha_i, y_i), y_i) + \sum_{i=1}^{N} \alpha_i L'^{-1}(\alpha_i, y_i) + \frac{1}{2}\left\|\sum_{i=1}^{N} \alpha_i \Phi_k(\mathbf{x}_i)\right\|_2^2.$$

Plugging in L, L'^{-1} leads to

$$\begin{aligned} S_k(\boldsymbol{\alpha}) &= -\sum_{i=1}^{N}(\frac{1}{2C}\alpha_i + y_i - y_i)^2 + \sum_{i=1}^{N} \alpha_i(\frac{1}{2C}\alpha_i + y_i) + \frac{1}{2}\left\|\sum_{i=1}^{N} \alpha_i \Phi_k(\mathbf{x}_i)\right\|_2^2 \\ &= \frac{1}{4C}\sum_{i=1}^{N}\alpha_i^2 + \sum_{i=1}^{N}\alpha_i y_i + \frac{1}{2}\left\|\sum_{i=1}^{N} \alpha_i \Phi_k(\mathbf{x}_i)\right\|_2^2. \end{aligned}$$

Logistic Loss Very similar to the Hinge loss the derivation for the logistic loss $L(x, y) = \log(1 + e^{-xy})$ will be given for completeness.

$$L'(x, y) = \frac{-ye^{-xy}}{1 + e^{-xy}} = -\frac{ye^{(1-xy)}}{1 + e^{(1-xy)}} =: z.$$

The inverse function for $y \neq 0$ and $y + z \neq 0$ is given by

$$L'^{-1}(z, y) = -\frac{1}{y}\log\left(-\frac{z}{y + z}\right)$$

and finally we obtain

$$S_k(\boldsymbol{\alpha}) = \sum_{i=1}^{N}\log\left(1 - \frac{\alpha_i}{y_i + \alpha_i}\right) - \sum_{i=1}^{N}\frac{\alpha_i}{y_i}\log\left(-\frac{\alpha_i}{y_i + \alpha_i}\right) + \frac{1}{2}\left\|\sum_{i=1}^{N} \alpha_i \Phi_k(\mathbf{x}_i)\right\|_2^2.$$

Smooth Hinge Loss Using the Hinge Loss $L(x, y) = \frac{C}{\sigma}\log(1 + e^{\sigma(1-xy)})$ with $\sigma > 0$, $y \in \mathbb{R}$ fixed, $x \in \mathbb{R}$ one obtains as derivative

$$L'(x, y) = \frac{-\sigma C y e^{\sigma(1-xy)}}{\sigma(1 + e^{\sigma(1-xy)})} = -\frac{Cye^{\sigma(1-xy)}}{1 + e^{\sigma(1-xy)}} =: z.$$

Note that with y fixed, z is bounded: $0 \leq \text{abs}(z) \leq \text{abs}(Cy)$ and $\text{sign}(y) = -\text{sign}(z)$ and therefore $-\frac{z}{Cy+z} > 0$ for $Cy + z \neq 0$. The inverse function is derived as

$$\begin{aligned}
z + ze^{\sigma(1-xy)} &= -Cye^{\sigma(1-xy)} \\
(Cy+z)e^{\sigma(1-xy)} &= -z \\
e^{\sigma(1-xy)} &= -\frac{z}{Cy+z} \\
\sigma(1-xy) &= \log(-\frac{z}{Cy+z}) \\
1 - xy &= \frac{1}{\sigma}\log(-\frac{z}{Cy+z}) \\
x &= \frac{1}{y}(1 - \frac{1}{\sigma}\log(-\frac{z}{Cy+z})),\ y \neq 0 \\
L'^{-1}(z,y) &= \frac{1}{y}(1 - \frac{1}{\sigma}\log(-\frac{z}{Cy+z}))
\end{aligned}$$

Define $C_1 = \frac{1}{2}\left\|\sum_{i=1}^N \alpha_i \Phi_k(\mathbf{x}_i)\right\|_2^2$ and $C_2 = \sum_{i=1}^N \alpha_i \frac{1}{y_i}\left(1 - \frac{1}{\sigma}\log(-\frac{\alpha_i}{Cy_i+\alpha_i})\right)$
Using these ingredients it follows for $S_k(\boldsymbol{\alpha})$

$$\begin{aligned}
S_k(\boldsymbol{\alpha}) &= -\sum_{i=1}^N L\left(\frac{1}{y_i}\left(1 - \frac{1}{\sigma}\log(-\frac{\alpha_i}{Cy_i+\alpha_i})\right), y_i\right) + C_2 + C_1 \\
&= -\sum_{i=1}^N \frac{1}{\sigma}\log\left(1 + e^{\sigma\left(1-\left(\frac{y_i}{y_i}\left(1-\frac{1}{\sigma}\log(-\frac{\alpha_i}{Cy_i+\alpha_i})\right)\right)\right)}\right) + C_2 + C_1 \\
&= -\sum_{i=1}^N \frac{1}{\sigma}\log\left(1 - \frac{\alpha_i}{Cy_i+\alpha_i}\right) + \sum_{i=1}^N \frac{\alpha_i}{y_i}\left(1 - \frac{1}{\sigma}\log(-\frac{\alpha_i}{Cy_i+\alpha_i})\right) + C_1.
\end{aligned}$$

A.2 Computation of Positional Oligomer Importances

In this section we derive an efficient recursive algorithm to compute Positional Oligomer Importance Matrices (POIMs, cf. Section 4.2). Recall the definition of a POIM from Equation (4.16)

$$Q(\mathbf{z}, j) := \mathbb{E}[\, s(\mathbf{X}) \mid \mathbf{X}[j] = \mathbf{z}\,] - \mathbb{E}[\, s(\mathbf{X})\,] \ . \tag{A.1}$$

Despite its conceptual simplicity, the computation of POIMs is demanding, as it involves computing an expectation over $|\Sigma|^{l_{\mathbf{x}}}$ values (for each of the $|\Sigma|^k$ k-mers $\mathbf{z}$, and each of the $l_{\mathbf{x}} - k + 1$ positions j).

This section is structured as follows: First, we make three observations that ultimately will help to reduce the computational burden (Appendix A.2.1). We continue with the derivation of an efficient recursive algorithm to compute POIMs (Appendix A.2.2) for the general case of using Markov chains of order d as the background probability distribution. The major result is Theorem A.5 (and its proof). The section concludes with the application of this theorem to special cases of the background distribution, namely the zeroth-order Markov distribution and the uniform distribution (Appendix A.2.3). This derivation is largely based on our technical report (Zien et al., 2007).

A.2.1 Observations

Observation A.1 (Independent PO Terms Vanish). *One of the reasons why we use the subtractive normalization w.r.t. the unconditionally expected score in (Equation* (A.1)*) is about computational efficiency: this normalization makes the problem tractable, especially when features are mostly independent. To see why independent features are of computational advantage, recall that computation of a single (possibly conditional) expectation would require summing over all features, i.e., over* $\mathcal{I} = \bigcup_{k=1}^{K} \left(\Sigma^k \times \{1, \ldots, l_{\mathbf{x}} - k + 1\}\right)$ *(cf. Section 4.2.1)*

$$\begin{aligned}
\mathbb{E}\,[\,s(\mathbf{X})\,] &= \mathbb{E}\left[\sum_{(\mathbf{y},i)\in\mathcal{I}} w_{(\mathbf{y},i)}\mathbb{I}\left\{\mathbf{X}\,[i] = \mathbf{y}\right\} + b\right] \\
&= \sum_{(\mathbf{y},i)\in\mathcal{I}} w_{(\mathbf{y},i)}\mathbb{E}\big[\mathbb{I}\left\{\mathbf{X}\,[i] = \mathbf{y}\right\}\big] + b \\
&= \sum_{(\mathbf{y},i)\in\mathcal{I}} w_{(\mathbf{y},i)} \Pr\left[\,\mathbf{X}\,[i] = \mathbf{y}\,\right] + b \ ,
\end{aligned}$$

and respectively

$$\begin{aligned}
\mathbb{E}\big[\,s(\mathbf{X})\,\big|\,\mathbf{X}\,[j] = \mathbf{z}\,\big] &= \mathbb{E}\left[\sum_{(\mathbf{y},i)\in\mathcal{I}} w_{(\mathbf{y},i)}\mathbb{I}\left\{\mathbf{X}\,[i] = \mathbf{y}\right\} + b\,\middle|\,\mathbf{X}\,[j] = \mathbf{z}\right] \\
&= \sum_{(\mathbf{y},i)\in\mathcal{I}} w_{(\mathbf{y},i)}\mathbb{E}\big[\,\mathbb{I}\left\{\mathbf{X}\,[i] = \mathbf{y}\right\}\,\big|\,\mathbf{X}\,[j] = \mathbf{z}\,\big] + b \\
&= \sum_{(\mathbf{y},i)\in\mathcal{I}} w_{(\mathbf{y},i)} \Pr\big[\,\mathbf{X}\,[i] = \mathbf{y}\,\big|\,\mathbf{X}\,[j] = \mathbf{z}\,\big] + b \ .
\end{aligned}$$

By subtraction however, all features which are independent of $(\mathbf{z}, j)$ *vanish from the difference because in this case the conditional probabilities are equal to the unconditional probabilities. That is,*

$$\Pr\big[\,\mathbf{X}\,[i] = \mathbf{y}\,\big|\,\mathbf{X}\,[j] = \mathbf{z}\,\big] \quad = \quad \Pr\big[\mathbf{X}\,[i] = \mathbf{y}\big]$$

whenever $\mathbf{X}\,[i] = \mathbf{y}$ *and* $\mathbf{X}\,[j] = \mathbf{z}$ *are statistically independent events, which we denote by* $(\mathbf{y}, i) \perp (\mathbf{z}, j)$*. Otherwise, i.e., if they are (possibly) dependent, we write* $(\mathbf{y}, i) \not\perp (\mathbf{z}, j)$*. Thus,*

$$\begin{aligned}
Q(\mathbf{z}, j) &= \mathbb{E}\big[\,s(\mathbf{X})\,\big|\,\mathbf{X}\,[j] = \mathbf{z}\,\big] - \mathbb{E}\,[\,s(\mathbf{X})\,] \\
&= \sum_{(\mathbf{y},i)\in\mathcal{I}} w_{(\mathbf{y},i)}\left[\Pr\left[\,\mathbf{X}\,[i] = \mathbf{y}\ |\,\mathbf{X}\,[j] = \mathbf{z}\,\right] - \Pr\left[\,\mathbf{X}\,[i] = \mathbf{y}\,\right]\right] \\
&= \sum_{(\mathbf{y},i)\in\mathcal{I},\,(\mathbf{y},i)\not\perp(\mathbf{z},j)} w_{(\mathbf{y},i)}\left[\Pr\left[\,\mathbf{X}\,[i] = \mathbf{y}\ |\,\mathbf{X}\,[j] = \mathbf{z}\,\right] - \Pr\left[\,\mathbf{X}\,[i] = \mathbf{y}\,\right]\right] \qquad \text{(A.2)}
\end{aligned}$$

Note that, under the Markov model of order d, two POs $(\mathbf{y}, i)$ *and* $(\mathbf{z}, j)$ *are always independent if they are separated by at least d positions. This means that either* $i + |\mathbf{y}| - 1 < j - d$ *or* $j + |\mathbf{z}| - 1 < i - d$*. Otherwise they are dependent in all but degenerate cases.*

Observation A.2 (Incompatible Conditional PO Terms Vanish). *Another property of PO pairs is that of compatibility. We say that* $(\mathbf{y}, i)$ *and* $(\mathbf{z}, j)$ *are compatible, denoted* *by* $(\mathbf{y}, i) \sim (\mathbf{z}, j)$*, if they agree on any shared positions they might have. For example,* $(TATA, 30)$*, and* $(AAA, 32)$ *are incompatible, since they share positions* $\{32, 33\}$ *but disagree on position* 32*, whereas* $(TATA, 30)$ *and* $(TACCA, 32)$ *are compatible. If* $(\mathbf{y}, i)$ *and* $(\mathbf{z}, j)$ *are incompatible, then it holds that* $\Pr[\mathbf{X}[i] = \mathbf{y} \mid \mathbf{X}[j] = \mathbf{z}] = 0$*. Thus, the sum (Equation* (A.2)*) can be simplified to range over less summands,*

$$Q(\mathbf{z}, j) = \sum_{(\mathbf{y},i) \in \mathcal{I}(\mathbf{z},j)} w_{(\mathbf{y},i)} \Pr\left[\mathbf{X}[i] = \mathbf{y} \,\middle|\, \mathbf{X}[j] = \mathbf{z}\right] \tag{A.3}$$

$$- \sum_{(\mathbf{y},i) \in \mathcal{I}, (\mathbf{y},i) \not\perp (\mathbf{z},j)} w_{(\mathbf{y},i)} \Pr[\mathbf{X}[i] = \mathbf{y}] \ , \tag{A.4}$$

where we denote by $\mathcal{I}(\mathbf{z}, j) := \{ (\mathbf{y}, i) \in \mathcal{I} \mid (\mathbf{y}, i) \not\perp (\mathbf{z}, j) \text{ and } (\mathbf{y}, i) \sim (\mathbf{z}, j) \}$ *the set of POs that are dependent on and compatible with* $(\mathbf{z}, j)$*.*

Observation A.3 (PO Importances are Weighted Sums of Conditional Terms)**.** *As a last observation we show that once we precomputed conditional sums (Equation* (A.3)*), we can use them to easily compute the positional oligomers importances.*
Denote the two sum terms from (Equation (A.3)*) and (Equation* (A.4)*) as*

$$u(\mathbf{z}, j) := \sum_{(\mathbf{y},i) \in \mathcal{I}(\mathbf{z},j)} w_{(\mathbf{y},i)} \Pr[\mathbf{X}[i] = \mathbf{y} \mid \mathbf{X}[j] = \mathbf{z}] \tag{A.5}$$

$$v(\mathbf{z}, j) := \sum_{(\mathbf{y},i) \not\perp (\mathbf{z},j)} w_{(\mathbf{y},i)} \Pr[\mathbf{X}[i] = \mathbf{y}] \ . \tag{A.6}$$

Thus, $Q(\mathbf{z}, j) = u(\mathbf{z}, j) - v(\mathbf{z}, j)$*. To compute* $v(\mathbf{z}, j)$*, we show that we only have to use terms of the type* $u(\mathbf{z}', j)$*, with* $|\mathbf{z}'| = |\mathbf{z}|$*. First note, for Markov chains of any order* d*, for any* $\mathbf{z}' \in \Sigma^{|\mathbf{z}|}$*, the set equality* $\{ (\mathbf{y}, i) \in \mathcal{I} \mid (\mathbf{y}, i) \not\perp (\mathbf{z}, j) \} = \{ (\mathbf{y}, i) \in \mathcal{I} \mid (\mathbf{y}, i) \not\perp (\mathbf{z}', j) \}$*. With this,*

$$\begin{aligned}
v(\mathbf{z}, j) &= \sum_{(\mathbf{y},i) \not\perp (\mathbf{z},j)} w_{(\mathbf{y},i)} \Pr[\mathbf{X}[i] = \mathbf{y}] \\
&= \sum_{(\mathbf{y},i) \not\perp (\mathbf{z},j)} w_{(\mathbf{y},i)} \sum_{\mathbf{z}' \in \Sigma^{|\mathbf{z}|}} \Pr\left[\mathbf{X}[i] = \mathbf{y} \wedge \mathbf{X}[j] = \mathbf{z}'\right] \\
&= \sum_{(\mathbf{y},i) \not\perp (\mathbf{z},j)} w_{(\mathbf{y},i)} \sum_{\mathbf{z}' \in \Sigma^{|\mathbf{z}|}} \Pr\left[\mathbf{X}[i] = \mathbf{y} \,\middle|\, \mathbf{X}[j] = \mathbf{z}'\right] \Pr\left[\mathbf{X}[j] = \mathbf{z}'\right] \\
&= \sum_{(\mathbf{y},i) \not\perp (\mathbf{z},j)} \sum_{\mathbf{z}' \in \Sigma^{|\mathbf{z}|}} \Pr\left[\mathbf{X}[j] = \mathbf{z}'\right] w_{(\mathbf{y},i)} \Pr\left[\mathbf{X}[i] = \mathbf{y} \,\middle|\, \mathbf{X}[j] = \mathbf{z}'\right] \\
&= \sum_{\mathbf{z}' \in \Sigma^{|\mathbf{z}|}} \Pr\left[\mathbf{X}[j] = \mathbf{z}'\right] \sum_{(\mathbf{y},i) \not\perp (\mathbf{z},j)} w_{(\mathbf{y},i)} \Pr\left[\mathbf{X}[i] = \mathbf{y} \,\middle|\, \mathbf{X}[j] = \mathbf{z}'\right] \\
&= \sum_{\mathbf{z}' \in \Sigma^{|\mathbf{z}|}} \Pr\left[\mathbf{X}[j] = \mathbf{z}'\right] \sum_{(\mathbf{y},i) \not\perp (\mathbf{z}',j)} w_{(\mathbf{y},i)} \Pr\left[\mathbf{X}[i] = \mathbf{y} \,\middle|\, \mathbf{X}[j] = \mathbf{z}'\right] \\
&= \sum_{\mathbf{z}' \in \Sigma^{|\mathbf{z}|}} \Pr\left[\mathbf{X}[j] = \mathbf{z}'\right] \sum_{(\mathbf{y},i) \in \mathcal{I}(\mathbf{z}',j)} w_{(\mathbf{y},i)} \Pr\left[\mathbf{X}[i] = \mathbf{y} \,\middle|\, \mathbf{X}[j] = \mathbf{z}'\right] \\
&= \sum_{\mathbf{z}' \in \Sigma^{|\mathbf{z}|}} \Pr\left[\mathbf{X}[j] = \mathbf{z}'\right] u(\mathbf{z}', j) \ .
\end{aligned}$$

Finally, we arrive at the following formula for the POIM computation, which shows that positional oligomers importances can be easily computed from the table of all conditional *terms,*

$$Q(\mathbf{z},j) \;=\; u(\mathbf{z},j) - \sum_{\mathbf{z}'\in\Sigma^{|\mathbf{z}|}} \Pr\left[\,\mathbf{X}\,[j] = \mathbf{z}'\,\right] u(\mathbf{z}',j)\ . \qquad \text{(A.7)}$$

A.2.2 Efficient Recursive Computation of POIMs

In the following we derive efficient recursions to compute $Q(\mathbf{z},j)$ for the general case of Markov chains of order d. From (Equation (A.7)) we see that we can compute $Q(\mathbf{z},j)$ by summing over the index set $\mathcal{I}(\mathbf{z},j)$ *only* (instead of $\mathcal{I}$). However, even the reduced set $\mathcal{I}(\mathbf{z},j)$ of relevant POs is too large to allow for efficient naive summation over it for each PO $(\mathbf{z},j)$. We thus develop an efficient recursive algorithm which will allow for use of efficient tree structures. The crucial idea is to treat the POs in $\mathcal{I}(\mathbf{z},j)$ separately according to their relative position to $(\mathbf{z},j)$. To do so for the general Markov chain of order d, we subdivide the set $\mathcal{I}(\mathbf{z},j)$ into substrings, superstrings, left neighbours of $(\mathbf{z},j)$ with gaps of at most $d-1$, and right neighbours of $(\mathbf{z},j)$ with gaps of at most $d-1$. We reproduce the formal definition from Section 4.2.2 of these terms here (cf. Figure 4.7 for an illustration).

Definition A.4.

$$\begin{array}{llcr}
\textit{(substrings)} & \mathcal{I}^{\vee}(\mathbf{z},j) & := & \{\,(\mathbf{y},i)\in\mathcal{I}(\mathbf{z},j)\mid i\ge j \textit{ and } |\mathbf{y}|+i\le|\mathbf{z}|+j\,\} \\
\textit{(superstrings)} & \mathcal{I}^{\wedge}(\mathbf{z},j) & := & \{\,(\mathbf{y},i)\in\mathcal{I}(\mathbf{z},j)\mid i\le j \textit{ and } |\mathbf{y}|+i\ge|\mathbf{z}|+j\,\} \\
\textit{(left p. o.)} & \mathcal{I}^{<}(\mathbf{z},j) & := & \{\,(\mathbf{y},i)\in\mathcal{I}(\mathbf{z},j)\mid i< j \textit{ and } |\mathbf{y}|+i<|\mathbf{z}|+j \textit{ and } |\mathbf{y}|+i-1\ge j\,\} \\
\textit{(right p. o.)} & \mathcal{I}^{>}(\mathbf{z},j) & := & \{\,(\mathbf{y},i)\in\mathcal{I}(\mathbf{z},j)\mid i> j \textit{ and } |\mathbf{y}|+i>|\mathbf{z}|+j \textit{ and } |\mathbf{z}|+j-1\ge i\,\}
\end{array}$$

With these sets we can decompose $u(\mathbf{z},j)$ defined in (Equation (A.5)) as follows (note that $(\mathbf{z},j)$ is element of both $\mathcal{I}^{\vee}(\mathbf{z},j)$ and $\mathcal{I}^{\wedge}(\mathbf{z},j)$, and that $w_{(\mathbf{z},j)}$ is thus counted twice):

$$u(\mathbf{z},j) \;=\; u^{\vee}(\mathbf{z},j)+u^{\wedge}(\mathbf{z},j)+u^{<}(\mathbf{z},j)+u^{>}(\mathbf{z},j)-w_{(\mathbf{z},j)}\ ,$$

where, for each $\star\in\{\vee,\wedge,>,<\}$:

$$u^{\star}(\mathbf{z},j) \;:=\; \sum_{(\mathbf{y},i)\in\mathcal{I}^{\star}(\mathbf{z},j)} \Pr\left[\,\mathbf{X}\,[i]=\mathbf{y}\ \middle|\ \mathbf{X}\,[j]=\mathbf{z}\,\right] w_{(\mathbf{y},i)}\ .$$

We further define the auxilliary conditional sums of left and right extensions of $\mathbf{z}\in\Sigma^p$:

$$L(\mathbf{z},j) \;:=\; \sum_{k=0}^{K-p}\sum_{\boldsymbol{t}\in\Sigma^k} \underbrace{\Pr\left[\,\mathbf{X}\,[j-k]=\boldsymbol{t}\mathbf{z}\,\middle|\,\mathbf{X}\,[j]=\mathbf{z}\,\right]}_{=\Pr\left[\,\mathbf{X}[j-k]=\boldsymbol{t}\,\middle|\,\mathbf{X}[j]=\mathbf{z}\,\right]} w_{(\boldsymbol{t}\mathbf{z},j-k)}$$

$$R(\mathbf{z},j) \;:=\; \sum_{k=0}^{K-p}\sum_{\boldsymbol{t}\in\Sigma^k} \underbrace{\Pr\left[\,\mathbf{X}\,[j]=\mathbf{z}\boldsymbol{t}\,\middle|\,\mathbf{X}\,[j]=\mathbf{z}\,\right]}_{=\Pr\left[\,\mathbf{X}[j+p]=\boldsymbol{t}\,\middle|\,\mathbf{X}[j]=\mathbf{z}\,\right]} w_{(\mathbf{z}\boldsymbol{t},j)}\ .$$

As the following theorem shows, $u^{\vee}(\mathbf{z},j)$, $u^{\wedge}(\mathbf{z},j)$, $L(\mathbf{z},j)$ and $R(\mathbf{z},j)$ can be computed recursively, in a way which will enable us to use efficient tree data structures for their computation. Additionally, $u^{>}(\mathbf{z},j)$ and $u^{>}(\mathbf{z},j)$ can be computed from the tables of values of $L(\mathbf{z},j)$ and $R(\mathbf{z},j)$. This theorem, together with Equation (A.5) and (A.7), enables us to compute PO importance matrices efficiently.

Theorem A.5 (Computing POIMs for Markov chains of order d)**.** *Let the sequences* $\mathbf{X}$ *be Markov chains of order d. For* $0\le|\mathbf{z}|\le K-2$ *and* $\sigma,\tau\in\Sigma$, *we then have the* *following upward recursion for substrings:*

$$\begin{aligned} u^\vee(\sigma, j) &= w_{(\sigma,j)} \\ u^\vee(\sigma\mathbf{z}\tau, j) &= w_{(\sigma\mathbf{z}\tau,j)} + u^\vee(\sigma\mathbf{z}, j) + u^\vee(\mathbf{z}\tau, j+1) - u^\vee(\mathbf{z}, j+1) \ . \end{aligned}$$

Further, for $2 \le p \le K$, $p := |\mathbf{z}|$, we have the following downward recursions for superstrings and neighbour sequences:

$$\begin{aligned} u^\wedge(\mathbf{z}, j) &= w_{(\mathbf{z},j)} - \frac{1}{\Pr[\mathbf{X}[j] = \mathbf{z}]} \sum_{(\sigma,\tau)\in\Sigma^2} \Pr[\mathbf{X}[j+1] = \sigma\mathbf{z}\tau]\, u^\wedge(\sigma\mathbf{z}\tau, j-1) \\ &\quad + \frac{1}{\Pr[\mathbf{X}[j] = \mathbf{z}]} \left[\sum_{\sigma\in\Sigma} \Pr[\mathbf{X}[j] = \sigma\mathbf{z}]\, u^\wedge(\sigma\mathbf{z}, j-1) + \sum_{\tau\in\Sigma} \Pr[\mathbf{X}[j+1] = \mathbf{z}\tau]\, u^\wedge(\mathbf{z}\tau, j) \right] \\ u^<(\mathbf{z}, j) &= \sum_{\boldsymbol{t}\in\Sigma^d} \sum_{\sigma\in\Sigma} \frac{\Pr[\mathbf{X}[j-d-1] = \sigma\boldsymbol{t}]}{\Pr[\mathbf{X}[j-d] = \boldsymbol{t}]} \sum_{l=1}^{\min\{p+d,K\}-1} L\left(\sigma((\boldsymbol{t}\mathbf{z})[1]^l), j-d-1\right) \qquad \text{(A.8)} \\ u^>(\mathbf{z}, j) &= \sum_{\boldsymbol{t}\in\Sigma^d} \sum_{\tau\in\Sigma} \frac{\Pr[\mathbf{X}[j+p] = \boldsymbol{t}\tau]}{\Pr[\mathbf{X}[j+p] = \boldsymbol{t}]} \sum_{l=1}^{\min\{p+d,K\}-1} R\left((\mathbf{z}\boldsymbol{t})[p+d-l+1]^l\tau, j+p+d-l\right) \ , \qquad \text{(A.9)} \end{aligned}$$

with the recursion anchored at too long sequences by $u^\wedge(\mathbf{z}, j) = 0$, $u^<(\mathbf{z}, j) = 0$, and $u^>(\mathbf{z}, j) = 0$ for $p = |\mathbf{z}| \le K$. Here L and R are computed recursively for $p \le K$ by

$$\begin{aligned} L(\mathbf{z}, j) &= w_{(\mathbf{z},j)} + \sum_{\sigma\in\Sigma} \frac{\Pr[\mathbf{X}[j-1] = \sigma\mathbf{z}]}{\Pr[\mathbf{X}[j] = \mathbf{z}]} L(\sigma\mathbf{z}, j-1) \qquad \text{(A.10)} \\ R(\mathbf{z}, j) &= w_{(\mathbf{z},j)} + \sum_{\tau\in\Sigma} \frac{\Pr[\mathbf{X}[j] = \mathbf{z}\tau]}{\Pr[\mathbf{X}[j] = \mathbf{z}]} R(\mathbf{z}\tau, j) \ , \qquad \text{(A.11)} \end{aligned}$$

with $L(\mathbf{z}, j) = R(\mathbf{z}, j) = 0$ otherwise (i.e., for $p > K$).

Proof. We show how to compute the values of the functions $u^\vee$, $u^\wedge$, $u^<$, and $u^>$ recursively. Recall first that for general Markov chains of order d

$$\begin{aligned} \Pr[\mathbf{X}[i] = \mathbf{y}] &= \prod_{l=1}^{d} \Pr\left[\mathbf{X}[i+l-1] = y[l] \,\middle|\, \mathbf{X}[i]^{l-1} = y[1]^{l-1} \right] \\ &\quad \times \prod_{l=d+1}^{|\mathbf{y}|} \Pr\left[\mathbf{X}[i+l-1] = y[l] \,\middle|\, \mathbf{X}[i+l-d-1]^d = y[l-d]^d \right] \ . \end{aligned}$$

Substrings Note that for any substring $(\mathbf{y}, i) \in \mathcal{I}^\vee(\mathbf{z}, j)$, it holds that $\Pr[\mathbf{X}[i] = \mathbf{y} \mid \mathbf{X}[j] = \mathbf{z}] = 1$. Therefore, $u^\vee(\mathbf{z}, j) = \sum_{(\mathbf{y},i)\in\mathcal{I}^\vee(\mathbf{z},j)} w_{(\mathbf{y},i)}$.
For $|\mathbf{z}| \ge 2$, let $\sigma, \tau \in \Sigma$ such that $\mathbf{z} = \sigma\boldsymbol{t}\tau$. Then the following set relations are easy to see:

$$\mathcal{I}^\vee(\mathbf{z}, j) = \{(\mathbf{z}, j)\} \cup \mathcal{I}^\vee(\sigma\boldsymbol{t}, j) \cup \mathcal{I}^\vee(\boldsymbol{t}\tau, j+1)$$

$$\begin{aligned} \mathcal{I}^\vee(\sigma\boldsymbol{t}, j) \cap \mathcal{I}^\vee(\boldsymbol{t}\tau, j+1) &= \mathcal{I}^\vee(\boldsymbol{t}, j+1) \\ \mathcal{I}^\vee(\sigma\boldsymbol{t}, j) \cap \{(\mathbf{z}, j)\} &= \emptyset \\ \mathcal{I}^\vee(\boldsymbol{t}\tau, j+1) \cap \{(\mathbf{z}, j)\} &= \emptyset \ . \end{aligned}$$

Thus, for $|\mathbf{z}| \geq 2$, we can set up the following recursion:

$$\begin{aligned} u^{\vee}(\mathbf{z}, j) &= \sum_{(\mathbf{y},i)\in\mathcal{I}^{\vee}(\mathbf{z},j)} w_{(\mathbf{y},i)} \\ &= w_{(\mathbf{z},j)} + \sum_{(\mathbf{y},i)\in\mathcal{I}^{\vee}(\sigma\boldsymbol{t},j)} w_{(\mathbf{y},i)} + \sum_{(\mathbf{y},i)\in\mathcal{I}^{\vee}(\boldsymbol{t}\tau,j+1)} w_{(\mathbf{y},i)} - \sum_{(\mathbf{y},i)\in\mathcal{I}^{\vee}(\boldsymbol{t},j+1)} w_{(\mathbf{y},i)} \\ &= w_{(\mathbf{z},j)} + u^{\vee}(\sigma\boldsymbol{t}, j) + u^{\vee}(\boldsymbol{t}\tau, j+1) - u^{\vee}(\boldsymbol{t}, j+1) \ . \end{aligned}$$

Superstrings For superstrings $(\mathbf{y}, i) \in \mathcal{I}^{\wedge}(\mathbf{z}, j)$, it is easy to see that

$$\Pr[\mathbf{X}[i] = \mathbf{y} \mid \mathbf{X}[j] = \mathbf{z}] \ = \ \frac{\Pr[\mathbf{X}[i] = \mathbf{y} \wedge \mathbf{X}[j] = \mathbf{z}]}{\Pr[\mathbf{X}[j] = \mathbf{z}]} = \frac{\Pr[\mathbf{X}[i] = \mathbf{y}]}{\Pr[\mathbf{X}[j] = \mathbf{z}]} \ .$$

Note that

$$\mathcal{I}^{\wedge}(\mathbf{z}, j) \ = \ \{(\mathbf{z}, j)\} \cup \left[\bigcup_{\sigma\in\Sigma} \mathcal{I}^{\wedge}(\sigma\mathbf{z}, j-1)\right] \cup \left[\bigcup_{\sigma\in\Sigma} \mathcal{I}^{\wedge}(\mathbf{z}\sigma, j)\right]$$

$$\begin{aligned} \mathcal{I}^{\wedge}(\sigma\mathbf{z}, j-1) \cap \mathcal{I}^{\wedge}(\mathbf{z}\tau, j) &= \mathcal{I}^{\wedge}(\sigma\mathbf{z}\tau, j-1), && \forall \sigma, \tau \in \Sigma \\ \mathcal{I}^{\wedge}(\sigma\mathbf{z}, j-1) \cap \{(\mathbf{z}, j)\} &= \emptyset, && \forall \sigma \in \Sigma \\ \mathcal{I}^{\wedge}(\mathbf{z}\tau, j) \cap \{(\mathbf{z}, j)\} &= \emptyset, && \forall \tau \in \Sigma \\ \mathcal{I}^{\wedge}(\sigma\mathbf{z}\tau, j-1) \cap \mathcal{I}^{\wedge}(\sigma'\mathbf{z}\tau', j-1) &= \emptyset, && \forall (\sigma, \tau) \neq (\sigma', \tau') \\ \mathcal{I}^{\wedge}(\sigma\mathbf{z}, j-1) \cap \mathcal{I}^{\wedge}(\sigma'\mathbf{z}, j-1) &= \emptyset, && \forall \sigma \neq \sigma' \\ \mathcal{I}^{\wedge}(\mathbf{z}\tau, j) \cap \mathcal{I}^{\wedge}(\mathbf{z}\tau', j) &= \emptyset, && \forall \tau \neq \tau' \ . \end{aligned}$$

Thus

$$\begin{aligned} \left[\bigcup_{\sigma\in\Sigma} \mathcal{I}^{\wedge}(\sigma\mathbf{z}, j-1)\right] \cap \left[\bigcup_{\sigma\in\Sigma} \mathcal{I}^{\wedge}(\mathbf{z}\sigma, j)\right] &= \bigcup_{(\sigma,\tau)\in\Sigma^2} \left(\mathcal{I}^{\wedge}(\sigma\mathbf{z}, j-1) \cap \mathcal{I}^{\wedge}(\mathbf{z}\tau, j)\right) \\ &= \bigcup_{(\sigma,\tau)\in\Sigma^2} \left(\mathcal{I}^{\wedge}(\sigma\mathbf{z}\tau, j-1)\right) \end{aligned}$$

and therefore

$$\begin{aligned} \sum_{(\mathbf{y},i)\in\mathcal{I}^{\wedge}(\mathbf{z},j)} \cdots = \sum_{(\mathbf{y},i)=(\mathbf{z},j)} \cdots + \sum_{(\mathbf{y},i)\in\left[\bigcup_{\sigma\in\Sigma}\mathcal{I}^{\wedge}(\sigma\mathbf{z},j-1)\right]} \cdots \\ + \sum_{(\mathbf{y},i)\in\left[\bigcup_{\sigma\in\Sigma}\mathcal{I}^{\wedge}(\mathbf{z}\sigma,j)\right]} \cdots - \sum_{(\mathbf{y},i)\in\bigcup_{(\sigma,\tau)\in\Sigma^2}(\mathcal{I}^{\wedge}(\sigma\mathbf{z}\tau,j-1))} \cdots \\ = \sum_{(\mathbf{y},i)=(\mathbf{z},j)} \cdots + \sum_{\sigma\in\Sigma} \sum_{(\mathbf{y},i)\in\mathcal{I}^{\wedge}(\sigma\mathbf{z},j-1)} \cdots + \sum_{\tau\in\Sigma} \sum_{(\mathbf{y},i)\in\mathcal{I}^{\wedge}(\mathbf{z}\tau,j)} \cdots - \sum_{(\sigma,\tau)\in\Sigma^2} \sum_{(\mathbf{y},i)\in\mathcal{I}^{\wedge}(\sigma\mathbf{z}\tau,j-1)} \cdots \end{aligned}$$

It follows that

$$
\begin{aligned}
u^{\wedge}(\mathbf{z},j) &= \frac{1}{\Pr[\mathbf{X}[j]=\mathbf{z}]} \sum_{(\mathbf{y},i)\in\mathcal{I}^{\wedge}(\mathbf{z},j)} \Pr[\mathbf{X}[i]=\mathbf{y}]\, w_{(\mathbf{y},i)} \\
&= \frac{1}{\Pr[\mathbf{X}[j]=\mathbf{z}]} \Bigg[\Pr[\mathbf{X}[j]=\mathbf{z}]\, w_{(\mathbf{z},j)} + \sum_{\sigma\in\Sigma} \sum_{(\mathbf{y},i)\in\mathcal{I}^{\wedge}(\sigma\mathbf{z},j-1)} \Pr[\mathbf{X}[i]=\mathbf{y}]\, w_{(\mathbf{y},i)} \\
&\qquad + \sum_{\tau\in\Sigma} \sum_{(\mathbf{y},i)\in\mathcal{I}^{\wedge}(\mathbf{z}\tau,j-1)} \Pr[\mathbf{X}[i]=\mathbf{y}]\, w_{(\mathbf{y},i)} \\
&\qquad - \sum_{(\sigma,\tau)\in\Sigma^2} \sum_{(\mathbf{y},i)\in\mathcal{I}^{\wedge}(\sigma\mathbf{z}\tau,j+1)} \Pr[\mathbf{X}[i]=\mathbf{y}]\, w_{(\mathbf{y},i)} \Bigg]
\end{aligned}
$$

By considering the definition of $u^{\wedge}$ and correcting for the conditional probabilities, we can set up the following recursion:

$$
\begin{aligned}
u^{\wedge}(\mathbf{z},j) &= w_{(\mathbf{z},j)} - \frac{1}{\Pr[\mathbf{X}[j]=\mathbf{z}]} \sum_{(\sigma,\tau)\in\Sigma^2} \Pr[\mathbf{X}[j-1]=\sigma\mathbf{z}\tau]\, u^{\wedge}(\sigma\mathbf{z}\tau,j-1) \\
&\quad + \frac{1}{\Pr[\mathbf{X}[j]=\mathbf{z}]} \sum_{\sigma\in\Sigma} \big[\Pr[\mathbf{X}[j-1]=\sigma\mathbf{z}]\, u^{\wedge}(\sigma\mathbf{z},j-1) + \Pr[\mathbf{X}[j]=\mathbf{z}\sigma]\, u^{\wedge}(\mathbf{z}\sigma,j) \big] .
\end{aligned}
$$

Left and right neighbours Recall the definition of $\mathcal{L}(\mathbf{z},j)$ and $\mathcal{R}(\mathbf{z},j)$ as unions of sets containing all strings with the same suffix respectively prefix $\mathbf{z}$ of length $p := |\mathbf{z}|$:

$$
\mathcal{L}(\mathbf{z},j) := \bigcup_{l=0}^{K-p} \left\{ (\boldsymbol{t}\mathbf{z}, j-l) \,\middle|\, \boldsymbol{t}\in\Sigma^l \right\} \;,\quad \mathcal{R}(\mathbf{z},j) := \bigcup_{l=0}^{K-p} \left\{ (\mathbf{z}\boldsymbol{t}, j) \,\middle|\, \boldsymbol{t}\in\Sigma^l \right\} \;,
$$

Generalizing the zeroth-order case in (Equation (A.12)) and (Equation (A.13)) to $d \geq 0$, they allow us to further decompose the sets of left and right neighbours as unions of *disjoint* sets with fixed prefixes and suffixes:

$$
\begin{aligned}
\mathcal{I}^{<}(\mathbf{z},j) &= \bigcup_{\boldsymbol{t}\in\Sigma^d} \bigcup_{\sigma\in\Sigma} \bigcup_{l=1}^{p+d-1} \mathcal{L}\Big(\sigma((\boldsymbol{t}\mathbf{z})[1]^l), j-d-1\Big) \\
\mathcal{I}^{>}(\mathbf{z},j) &= \bigcup_{\boldsymbol{t}\in\Sigma^d} \bigcup_{\tau\in\Sigma} \bigcup_{l=1}^{p+d-1} \mathcal{R}\Big((\mathbf{z}\boldsymbol{t})[p+d-l+1]^l\tau, j\Big) .
\end{aligned}
$$

Thus we can write

$$
\begin{aligned}
u^{<}(\mathbf{z},j) &= \sum_{\boldsymbol{t}\in\Sigma^d} \sum_{\sigma\in\Sigma} \sum_{l=1}^{p+d-1} \sum_{(\mathbf{y},i)\in\mathcal{L}\left(\sigma((\boldsymbol{t}\mathbf{z})[1]^l),j-d-1\right)} \Pr[\mathbf{X}[i]=\mathbf{y} \mid \mathbf{X}[j]=\mathbf{z}]\, w_{(\mathbf{y},i)} \\
u^{>}(\mathbf{z},j) &= \sum_{\boldsymbol{t}\in\Sigma^d} \sum_{\tau\in\Sigma} \sum_{l=1}^{p+d-1} \sum_{(\mathbf{y},i)\in\mathcal{R}\left((\mathbf{z}\boldsymbol{t})[p+d-l+1]^l\tau,j\right)} \Pr[\mathbf{X}[i]=\mathbf{y} \mid \mathbf{X}[j]=\mathbf{z}]\, w_{(\mathbf{y},i)} .
\end{aligned}
$$

By considering the definition of L and R and correcting for the conditional probabilities we obtain Equation (A.8) and (A.9). □

A.2.3 Applications

Using Theorem A.5 we can now derive efficient recursions for zeroth-order Markov and uniform background distributions.

Recursive computation of $Q(\mathbf{z}, j)$ for the zeroth-order Markov distribution For the special case of the zeroth-order Markov distribution, i.e., $d = 0$, the equations in Theorem A.5 simplify considerably. First, note that dependence of two POs $(\mathbf{y}, i)$ and $(\mathbf{z}, j)$ is easy to check: in all but degenerate cases, dependence is equivalent with being overlapping in $\mathbf{X}$. Thus, instead of considering neighbours with distances up to $d - 1$, we can now focus on overlaps.
The second simplification is that the zeroth-order Markov model decouples the positions completely, i.e., we have an independent single-symbol distribution at each position. Thus the probabilities in the theorem can be computed simply by

$$\Pr[\mathbf{X}[i] = \mathbf{y}] = \prod_{l=1}^{|\mathbf{y}|} \Pr[\mathbf{X}[i + l - 1] = \mathbf{y}[l]] \ .$$

To aid in understanding the recursions, we additionally define the following unions of sets containing all strings with the same suffix respectively prefix $\mathbf{z}$, i.e., complete overlaps of the positional p-mer $\mathbf{z}$:

$$\mathcal{L}(\mathbf{z}, j) := \bigcup_{l=0}^{K-p} \left\{ (\boldsymbol{t}\mathbf{z}, j - l) \,\middle|\, \boldsymbol{t} \in \Sigma^l \right\} \ , \quad \mathcal{R}(\mathbf{z}, j) := \bigcup_{l=0}^{K-p} \left\{ (\mathbf{z}\boldsymbol{t}, j) \,\middle|\, \boldsymbol{t} \in \Sigma^l \right\} \ ,$$

Now we can decompose left and right partial overlaps as unions of sets with fixed prefixes and suffixes:

$$\mathcal{I}^{<}(\mathbf{z}, j) = \bigcup_{\sigma\in\Sigma} \bigcup_{l=1}^{p-1} \mathcal{L}\left(\sigma(\mathbf{z}[1]^l), j - 1\right) \tag{A.12}$$

$$\mathcal{I}^{>}(\mathbf{z}, j) = \bigcup_{\tau\in\Sigma} \bigcup_{l=1}^{p-1} \mathcal{R}\left(\mathbf{z}[p - l + 1]^l \tau, j\right) \ . \tag{A.13}$$

The structure of these equations is mimiked by the recursions in the following theorem.

Corollary A.6 (Markov Chains of order 0)**.** *Let the sequences* $\mathbf{X}$ *be Markov chains of order* 0. *For* $2 \le p \le K$, $p := |\mathbf{z}|$, *we then have the following downward recursions for superstrings and partial overlaps:*

$$\begin{aligned}
u^{\wedge}(\mathbf{z}, j) &= w_{(\mathbf{z},j)} - \sum_{(\sigma,\tau)\in\Sigma^2} \Pr[\mathbf{X}[j-1] = \sigma] \Pr[\mathbf{X}[j+p] = \tau]\, u^{\wedge}(\sigma\mathbf{z}\tau, j-1) \\
&\quad + \sum_{\sigma\in\Sigma} \Pr[\mathbf{X}[j-1] = \sigma]\, u^{\wedge}(\sigma\mathbf{z}, j-1) + \sum_{\tau\in\Sigma} \Pr[\mathbf{X}[j+p] = \tau]\, u^{\wedge}(\mathbf{z}\tau, j) \\
u^{<}(\mathbf{z}, j) &= \sum_{\sigma\in\Sigma} \Pr[\mathbf{X}[j-1] = \sigma] \sum_{l=1}^{\min\{p,K\}-1} L\left(\sigma(\mathbf{z}[1]^l), j-1\right) \\
u^{>}(\mathbf{z}, j) &= \sum_{\tau\in\Sigma} \Pr[\mathbf{X}[j+p] = \tau] \sum_{l=1}^{\min\{p,K\}-1} R\left(\mathbf{z}[p-l+1]^l\tau, j+p-l\right) \ ,
\end{aligned}$$

where L and R are computed recursively by

$$\begin{aligned} L(\mathbf{z},j) &= w_{(\mathbf{z},j)} + \sum_{\sigma\in\Sigma} \Pr\left[\,\mathbf{X}\left[j-1\right]=\sigma\,\right] L(\sigma\mathbf{z},j-1) \\ R(\mathbf{z},j) &= w_{(\mathbf{z},j)} + \sum_{\tau\in\Sigma} \Pr\left[\,\mathbf{X}\left[j+p\right]=\tau\,\right] R(\mathbf{z}\tau,j) \ . \end{aligned}$$

Recursive computation of $Q(\mathbf{z},j)$ for the uniform distribution We will also state the following corollary for the simplest possible case, for the uniform distribution over $\mathbf{X}$ (with length length $|\mathbf{X}| = l_{\mathbf{x}}$), for which

$$\Pr\left[\,\mathbf{X}=\mathbf{x}\,\right] = |\Sigma|^{-l_{\mathbf{x}}} \ .$$

It is easy to see that this is equivalent to the assumption that at each position in the sequence, each element of the alphabet Σ is equally likely with probability $1/|\Sigma|$. It also implies that single characters at each position are independent of characters at all other positions, and $\Pr\left[\,\mathbf{X}\left[j\right]=\mathbf{z}\,\right] = |\Sigma|^{-p}$, where $p = |\mathbf{z}|$. This makes the computation of PO importances much easier.

Corollary A.7 (Uniform distribution)**.** *For the uniform distribution, with the notations from above, the PO importances $Q(\mathbf{z},j) = u(\mathbf{z},j) - v(\mathbf{z},j)$ can be computed from*

$$Q(\mathbf{z},j) = u(\mathbf{z},j) - \frac{1}{|\Sigma|^p} \sum_{\mathbf{z}'\in\Sigma^p} u(\mathbf{z}',j)$$

by the partial terms

$$\begin{aligned} u^{\wedge}(\mathbf{z},j) &= w_{(\mathbf{z},j)} - \frac{1}{|\Sigma|^2} \sum_{(\sigma,\tau)\in\Sigma^2} u^{\wedge}(\sigma\mathbf{z}\tau,j-1) + \frac{1}{|\Sigma|} \sum_{\sigma\in\Sigma} u^{\wedge}(\sigma\mathbf{z},j-1) + \frac{1}{|\Sigma|} \sum_{\tau\in\Sigma} u^{\wedge}(\mathbf{z}\tau,j) \\ u^{<}(\mathbf{z},j) &= \frac{1}{|\Sigma|} \sum_{\sigma\in\Sigma} \sum_{l=1}^{p-1} L(\sigma\mathbf{z}[1]^l,j-1) \\ u^{>}(\mathbf{z},j) &= \frac{1}{|\Sigma|} \sum_{\tau\in\Sigma} \sum_{l=1}^{p-1} R(\mathbf{z}[p-l+1]^l\tau,j+p-l) \ , \end{aligned}$$

and L and R are computed recursively by

$$\begin{aligned} L(\mathbf{z},j) &= w_{(\gamma,j)} + \frac{1}{|\Sigma|} \sum_{\sigma\in\Sigma} L(\sigma\mathbf{z},j-1) \\ R(\mathbf{z},j) &= w_{(\gamma,j)} + \frac{1}{|\Sigma|} \sum_{\tau\in\Sigma} R(\mathbf{z}\tau,j) \ . \end{aligned}$$

B Data Generation and Model Parameters

B.1 Toy Sequence Data

We generated $11,000$ sequences of length 50, where the symbols of the alphabet $\{A, C, G, T\}$ follow a uniform distribution. We chose $1,000$ of these sequences to be positive examples and hid two motifs of length seven: at position 10 and 30 the motifs `GATTACA` and `AGTAGTG`, respectively. The remaining $10,000$ examples were used as negatives. Thus, the ratio between examples of class $+1$ and class -1 is $\approx 9\%$. In the positive examples, we then randomly replaced $s \in \{0, 2, 4, 5\}$ symbols in each motif. Leading to four different data sets, which where randomly permuted and split such that the first $1,000$ examples became training and the remaining $10,000$ validation examples.

B.2 Splice Site Sequences

C. elegans

We collected all known *C. elegans* ESTs from Wormbase (Harris et al., 2004) (release WS118; 236,868 sequences), dbEST (Boguski and Tolstoshev, 1993) (as of February 22, 2004; 231,096 sequences) and UniGene (Wheeler et al, 2003) (as of October 15, 2003; 91,480 sequences). Using *blat* (Kent, 2002) we aligned them against the genomic DNA (release WS118). We refined the alignment by correcting typical sequencing errors, for instance by removing minor insertions and deletions. If an intron did not exhibit the `GT/AG` or `GC/AG` dimers at the 5' and 3' ends, respectively, then we tried to achieve this by shifting the boundaries up to 2 nucleotides. For each sequence, we determined the longest open reading frame (ORF) and only used the part of each sequence within the ORF. In a next step we merged agreeing alignments, leading to 135,239 unique EST-based sequences. We repeated the above with all known cDNAs from Wormbase (release WS118; 4,848 sequences) and UniGene (as of October 15, 2003; 1,231 sequences), which lead to 4,979 unique sequences. We removed all EST matches fully contained in the cDNA matches, leaving 109,693 EST-based sequences.
We clustered the sequences to obtain independent training, validation and test sets. In the beginning, each of the above EST and cDNA sequences were in a separate cluster. We iteratively joined clusters, if any two sequences from distinct clusters a) match to the genome at most 100nt apart (this includes many forms of alternative splicing) or b) have more than 20% sequence overlap (at 90% identity, determined by using *blat*). We obtained 17,763 clusters with a total of 114,672 sequences. There are 3,857 clusters that contain at least one cDNA. Finally, we removed all clusters that showed alternative splicing.
Since the resulting data set is still too large, we only used sequences from randomly chosen 20% of clusters with cDNA and 30% of clusters without cDNA to generate true acceptor splice site sequences (15,507 of them). Each sequence is 398nt long and has the `AG` dimer at position 200. Negative examples were generated from any occurring `AG` within the ORF of the sequence (246,914 of them were found). We used a random subset of $60,000$ examples for testing, $100,000$ examples for parameter tuning and up

	NN269		DGSplicer	
	Acceptor	**Donor**	**Acceptor**	**Donor**
Sequence length	90	15	36	18
Consensus positions	`AG` at 69	`GT` at 8	`AG` at 26	`GT` at 10
Train total	5788	5256	322156	228268
Fraction positives	19.3%	21.2%	0.6%	0.8%
Test total	1087	990	80539	57067
Fraction positives	19.4%	21.0%	0.6%	0.8%

Table B.1: Characteristics of the NN269 and DGSplicer data sets containing true and decoy acceptor and donor splice sites derived from the human genome.

to 100, 000 examples for training (unless stated otherwise).

B.2.1 NN269 and DGSplicer data sets

For the pilot study, we use the NN269 and the DGSplicer data sets originating from Reese et al. (1997) and Baten et al. (2006a), respectively. The data originates from FruitFly (a) and the training and test splits can be downloaded from Degroeve et al. (2005b). The data sets only include sequences with the canonical splice site dimers `AG` and `GT`. We use the same split for training and test sets as used in Baten et al. (2006a). A description of the properties of the data set is given in Table B.1.

B.2.2 Worm, fly, cress, fish, and human

We collected all known ESTs from dbEST (Boguski and Tolstoshev, 1993) (as of February 28, 2007; 346,064 sequences for worm, 514,613 sequences for fly, 1,276,130 sequences for cress, 1,168,572 sequences for fish, and 7,915,689 sequences for human). We additionally used EST and cDNA sequences available from wormbase (Wormbase) for worm, (file `confirmed_genes.WS170`) FruitFly (b) for fly, (files `na_EST.dros` and `na_dbEST.same.dmel`) A.thaliana for cress, (files `cDNA_flanking_050524.txt` and `cDNA_full_reading_050524.txt`) Ensemble for fish, (file `Danio_rerio.ZFISH6.43.cdna.known.fa`) and Mammalian Gene Collection for fish and human (file `dr_mgc_mrna.fasta` for fish and `hs_mgc_mrna.fasta` for human). Using *blat* (Kent, 2002) we aligned ESTs and cDNA sequences against the genomic DNA (releases WS170, dm5, ath1, zv6, and hg18, respectively). If the sequence could not be unambiguously matched, we only considered the best hit. The alignment was used to confirm exons and introns. We refined the alignment by correcting typical sequencing errors, for instance by removing minor insertions and deletions. If an intron did not exhibit the consensus `GT/AG` or `GC/AG` at the 5' and 3' ends, we tried to achieve this by shifting the boundaries up to two base pairs (bp). If this still did not lead to the consensus, then we split the sequence into two parts and considered each subsequence separately. Then, we merged alignments if they did not disagree and if they shared at least one complete exon or intron.
In a next step, we clustered the alignments: In the beginning, each of the above EST and cDNA alignments were in a separate cluster. We iteratively joined clusters, if any two sequences from distinct clusters match to the same genomic location (this includes many forms of alternative splicing).
From the clustered alignments we obtained a compact splicing graph representation (Ong and Rätsch, 2008), which can be easily used to generate a list of positions of true

acceptor and donor splice sites. Within the boundaries of the alignments (we cut out 10nt at both ends of the alignments to exclude potentially undetected splice sites), we identified all positions exhibiting the `AG`, `GT` or `GC` dimer and which were not in the list of confirmed splice sites. The lists of true and decoy splice site positions were used to extract the disjoint training, validation and test sets consisting of sequences in a window around these positions. Additionally, we divided the whole genome into regions, which are disjoint contiguous sequences containing at least two complete genes; if an adjacent gene is less than 250 base pairs away, we merge the adjacent genes into the region. Genes in the same region are also assigned to the same cross-validation split. The splitting was implemented by defining a linkage graph over the regions and by using single linkage clustering. The splits were defined by randomly assigning clusters of regions to the split.

B.3 Transcription Start Sites

Both for training our transcription start site finder and for assessing its accuracy we need known transcription start sites (TSS) as well as known non-TSS.

Datasets for Training and Model-Evaluation

To generate TSS data for training, we use dbTSS (Suzuki et al., 2002) version 4 ("dbTSSv4"), which is based on the human genome sequence and annotation version 16 ("hg16"). It contains transcription start sites of 12763 RefSeq genes (Pruitt et al., 2005). First, we extract RefSeq identifiers from dbTSSv4 and then obtain the corresponding mRNA sequences using NCBI nucleotide batch retrieval.[1] Next, we align these mRNAs to the hg16 genome using BLAT (Kent, 2002).[2] From dbTSS we extracted putative TSS positions (Field: Position of TSS), which we compared with the best alignment of the mRNA. We discard all positions that do not pass all of the following checks: 1. Chromosome and strand of the TSS position and of the best BLAT hit match. 2. The TSS position is within 100 base pairs from the gene start as found by the BLAT alignment. 3. No already processed putative TSS is within 100bp of the current one. This procedure leaves us with 8508 genes, each annotated with gene start and end. To generate positive training data, we extract windows of size $[-1000, +1000]$ around the putative TSS.
To discriminatively train a classifier, one also needs to generate "negative" data. However there is no single natural way of doing this: since there are further yet unknown TSS hidden in the rest of the genome, it is dangerous to sample negative points randomly from it. We choose to proceed in a similar way to Bajic et al. (2004) by extracting "negative" points (again, windows of size $[-1000, +1000]$) from the interior of the gene. More precisely, we draw 10 negatives at random from locations between 100bp downstream of the TSS and the end of the gene.[3] We finally obtain 8508 positive and 85042 negative examples, of which we will use 50% for training a TSS classifier and 50% for validating it. The final evaluation will be done on a differently generated test dataset (see below).

[1]`http://ncbi.nih.gov/entrez/batchentrez.cgi?db=Nucleotide`
[2]We used the options `-tileSize=16 -minScore=100 -minMatch=4 -minIdentity=98 -t=dna -q=rna`.
[3]If a gene is too short, fewer or even no negative examples are extracted from that particular gene.

Genome-Wide Human Test Dataset

To allow for a fair comparison of promoter detectors, one needs to create a proper test set such that no promoter detector has seen the examples in training. We decide to take all “new” genes from dbTSSv5 (Yamashita et al., 2006) (which is based on hg17) for which a representative TSS was identified (i.e., the field “The selected representative TSS” is not empty). From dbTSSv5, we remove all genes that already appear in dbTSSv4 according to the RefSeq NM identifier. To take care of cases in which IDs changed over time or are not unique, we also remove all genes from dbTSSv5 for which mRNAs overlap by more than 30%. This leads to a total of 1,024 TSS to be used in a comparative evaluation. The comparison is done on this test set using chunk sizes 50 and 500 as resolutions (cf. Section 5.4.2), which results in 1,588 (943) positives and 1,087,664 (108,783) negatives. In 816 (67) cases, the TSS fall into two chunks.

B.4 Generating Genestructure and Sequence Data for Caenorhabditis elegans

EST Sequences and cDNA Sequences

EST Sequences

We collected all known *C. elegans* ESTs from Wormbase (Harris et al., 2004) (release WS120; 236,893 sequences) and dbEST (Boguski and Tolstoshev, 1993) (as of February 22, 2004; 231,096 sequences). Using *blat* (Kent, 2002) we aligned them against the genomic DNA (release WS120). The alignment was used to confirm exons and introns. We refined the alignment by correcting typical sequencing errors, for instance by removing minor insertions and deletions. If an intron did not exhibit the consensus `GT/AG` or `GC/AG` at the 5’ and 3’ ends, then we tried to achieve this by shifting the boundaries up to 2 base pairs (bp). If this still did not lead to the consensus, then we split the sequence into two parts and considered each subsequence separately. In a next step we merged alignments, if they did not disagree and shared at least one complete exon or intron. This lead to a set of 124,442 unique EST-based sequences.

cDNA Sequences

We repeated the above procedure with all known cDNAs from Wormbase (release WS120; 4,855 sequences). These sequences only contain the coding part of the mRNA. We use their ends as annotation for start and stop codons.

Clustering and Splitting

Clustering

We clustered the sequences to obtain independent training, validation and test sets. In the beginning, each of the above EST and cDNA sequences were in a separate cluster. We iteratively joined clusters, if any two sequences from distinct clusters match to the same genomic location (this includes many forms of alternative splicing). We obtained 21,086 clusters, while 4072 clusters contained at least one cDNA.

Splitting into Training, Validation and Test Sets

For the *training set* we chose 40% of the clusters containing at least one cDNA (1536) and all clusters not containing a cDNA (17215). For the *validation set*, we used 20% of clusters with cDNA (775). The remaining 40% of clusters with at least one cDNA (1,560) was filtered to remove confirmed alternative splice forms. This left 1,177 cDNA sequences for *testing* with an average of 4.8 exons per gene and 2,313bp from the 5' to the 3' end.

B.5 Splicing Model Selection Parameters

In this section the parameter values explored in model selection for the NN269, DGSplicer and *genome-wide* splice site detectors are listed. Table B.2 shows the parameter values for NN269 and DGSplicer acceptor and donor tasks and Table B.3 the optimal parameter setting. Table B.4 displays the optimal parameter settings obtained in model selection of the *genome-wide* donor and acceptor splice site predictors from Section 5.3. Further information like genome-wide predictions, custom tracks and a stand-a-lone tool are available from `http://www.fml.tuebingen.mpg.de/raetsch/projects/splice/`.

Name	MC	LIK
N269	$d \in \{3,4,5,6,7,8\}$	$C \in \{0.25,0.5,0.75,1,2,5,10,20\}$
Acceptor	$\pi_+ \in \{0.001,0.1,1,8,10,100,1000,10^4,10^5\}$	$l \in \{1,2,3,4,5,6\}$
	$\pi_- \in \{0.001,0.1,1,8,10,100,150,1000,10^5,10^6,10^7\}$	$d \in \{1,2,3,4,5\}$
NN269	$d \in \{3,4,5,6,7,8\}$	$C \in \{0.25,0.5,0.75,1,2,5,10,20\}$
Donor	$\pi_+ \in \{0.001,0.1,1,8,10,100,1000,10^4,10^5\}$	$l \in \{1,2,3,4,5,6\}$
	$\pi_- \in \{0.001,0.1,1,8,10,100,150,1000,10^5,10^6,10^7\}$	$d \in \{1,2,3,4,5\}$
DGSplicer	$d \in \{3,4,5,6,7,8\}$	-
Acceptor	$\pi_+ \in \{0.001,0.1,1,8,10,100,1000,10^4,10^5\}$	
	$\pi_- \in \{0.001,0.1,1,8,10,100,150,1000,10^5,10^6,10^7\}$	-
DGSplicer	$d \in \{3,4,5,6,7,8\}$	-
Donor	$\pi_+ \in \{0.001,0.1,1,8,10,100,1000,10^4,10^5\}$	
	$\pi_- \in \{0.001,0.1,1,8,10,100,150,1000,10^5,10^6,10^7\}$	

Name	WD	WDS1	WDS2
NN269	$C \in \{0.1,1,2,5,10\}$	$C \in \{0.1,1,2,5,10\}$	$C \in \{2,5,10\}$
Acceptor	$d \in \{5,10,15,18,20,25,28,30\}$	$d \in \{5,10,15,18,20,25,28,30\}$	$d \in \{20,25,28\}$
		$S \in \{5,10\}$	$S_v \in \{4,8,16\}$
NN269	$C \in \{0.1,1,2,5,10\}$	$C \in \{0.1,1,2,5,10\}$	$C \in \{0.1,1,2\}$
Donor	$d \in \{5,10,15,18,20,25,28,30\}$	$d \in \{5,10,15,18,20,25,28,30\}$	$d \in \{18,20,22\}$
		$S \in \{5,10\}$	$S_v \in \{0.5,0.7,1.4,2.8,7,14\}$
DGSplicer	$C \in \{0.1,1,2,5,10\}$	$C \in \{0.1,1,2,5,10\}$	$C \in \{0.1,0.5,1,2,5,10\}$
Acceptor	$d \in \{5,10,15,18,20,25,28,30\}$	$d \in \{5,10,15,18,20,25,28,30\}$	$d \in \{18,20,2230\}$
		$S \in \{5,10\}$	$S_v \in \{0.9,1.5,3,6,9\}$
DGSplicer	$C \in \{0.1,1,2,5,10\}$	$C \in \{0.1,1,2,5,10\}$	$C \in \{0.5,1,2\}$
Donor	$d \in \{5,10,15,18,20,25,28,30\}$	$d \in \{5,10,15,18,20,25,28,30\}$	$d \in \{18,20,22\}$
		$S \in \{5,10\}$	$S_v \in \{0.9,1.8,3.6,9,18\}$

Table B.2: Parameter values explored in model selection for NN269 and DGSplicer acceptor and donor tasks, for the Markov Chain (MC), Locality Improved Kernel (LIK) and Weighted Degree (WD) kernels (cf. Section 1.2 and 2.3). $\pi_\pm$ denotes the pseudo count of the Markov chain (cf. Section 1.2), S and S_v denote the shift parameter of the Weighted Degree kernel *with shifts* (WDS1 and WDS2), with $S(l) = S$ in one experiment and $S(l) = S_v|p-l|$ in the other experiment. Here p is the position of the splice site: the further away a motif is from the splice site, the less precisely it needs to be localised in order to contribute to the kernel value.

Name	PSSM	LIK	WD	WDS1	WDS2
NN269 Acceptor	$d = 3$ $\pi_+ = 0.1$ $\pi_- = 8$	$C = 0.5$ $l = 2$ $d = 3$	$C = 2.0$ $d = 25$	$C = 5.0$ $d = 25$ $S = 10$	$C = 2.0$ $d = 20$ $S_v = 4.0$
NN269 Donor	$d = 3$ $\pi_+ = 10$ $\pi_- = 10$	$C = 10.0$ $l = 6$ $d = 3$	$C = 0.1$ $d = 5$	$C = 1.0$ $d = 20$ $S = 5$	$C = 0.5$ $d = 22$ $S_v = 2.8$
DGSplicer Acceptor	$d = 3$ $\pi_+ = 8$ $\pi_- = 1000$	-	$C = 2.0$ $d = 30$	$C = 2.0$ $d = 20$ $S = 10$	$C = 2.0$ $d = 22$ $S_v = 1.5$
DGSplicer Donor	$d = 3$ $\pi_+ = 10$ $\pi_- = 1000$	-	$C = 2.0$ $d = 30$	$C = 2.0$ $d = 20$ $S = 5$	$C = 0.5$ $d = 22$ $S_v = 3.6$

Table B.3: Final model parameters for NN269 and DGSplicer acceptor and donor tasks. These parameters were determined through partial 10-fold crossvalidation on the training sets. See Table B.2 for an explanation of the parameters.

	window	C	d	S_v	π_+	π_-	type	method
H. sapiens	$199 + [-60, +80]$	3	22	0	-	-	acceptor	WD-SVM
	$199 + [-60, +80]$	3	22	0.3	-	-	acceptor	WDS-SVM
	$199 + [-25, +25]$	-	3	-	10	1000	acceptor	MCs
	$199 + [-80, +60]$	3	22	0	-	-	donor	WD-SVM
	$199 + [-80, +60]$	3	22	0.3	-	-	donor	WDS-SVM
	$199 + [-17, +18]$	-	3	-	0.01	1000	donor	MCs
A. thaliana	$199 + [-60, +80]$	3	22	0	-	-	acceptor	WD-SVM
	$199 + [-60, +80]$	3	22	0.5	-	-	acceptor	WDS-SVM
	$199 + [-80, +80]$	-	4	-	10	1	acceptor	MCs
	$199 + [-80, +60]$	3	26	0	-	-	donor	WD-SVM
	$199 + [-80, +60]$	3	22	0.5	-	-	donor	WDS-SVM
	$199 + [-80, +80]$	-	4	-	10	10	donor	MCs
C. elegans	$199 + [-60, +80]$	3	22	0	-	-	acceptor	WD-SVM
	$199 + [-60, +80]$	3	22	0.3	-	-	acceptor	WDS-SVM
	$199 + [-25, +25]$	-	3	-	10	1000	acceptor	MCs
	$199 + [-80, +60]$	3	22	0	-	-	donor	WD-SVM
	$199 + [-80, +60]$	3	22	0.3	-	-	donor	WDS-SVM
	$199 + [-17, +18]$	-	3	-	0.01	1000	donor	MCs
D. rerio	$199 + [-60, +80]$	3	22	0	-	-	acceptor	WD-SVM
	$199 + [-60, +80]$	3	22	0.3	-	-	acceptor	WDS-SVM
	$199 + [-60, +60]$	-	3	-	0	1000	acceptor	MCs
	$199 + [-80, +60]$	3	22	0	-	-	donor	WD-SVM
	$199 + [-80, +60]$	3	22	0.3	-	-	donor	WDS-SVM
	$199 + [-60, +60]$	-	3	-	0	1000	donor	MCs

Table B.4: Optimal parameter settings for the in Section 5.3 considered *genome-wide* splice site detectors. Weighted Degree kernel based SVM (WD-SVM), Weighted Degree kernel *with shifts* (WDS-SVM), Markov Chain (MC) for the donor and acceptor splice site.

C Shogun

SHOGUN is a machine learning toolbox with focus on kernel methods, and especially on Support Vector Machines (SVMs) and contains implementations of all the algorithms presented in this doctoral thesis. It provides a generic SVM object interfacing to several different SVM implementations, among them the state of the art SVM-light (Joachims, 1999), LibSVM (Chang and Lin, 2001), GPDT (Zanni et al., 2006), SVMLin (V. Sindhwani, 2007) and OCAS (Franc and Sonnenburg, 2008). Each of the SVMs can be combined with a variety of kernels. The toolbox not only provides efficient implementations of the most common kernels, like the Linear, Polynomial, Gaussian and Sigmoid Kernel but also comes with several recent string kernels such as the Locality Improved (Zien et al., 2000), Fisher (Jaakkola and Haussler, 1999), TOP (Tsuda et al., 2002a), Spectrum (Leslie et al., 2002), Weighted Degree Kernel (with shifts) (Rätsch and Sonnenburg, 2004, Rätsch et al., 2005). For the latter, the efficient `linadd` (Sonnenburg et al., 2007a) optimisations are implemented. A detailed description of the string kernels and the `linadd` algorithm are also given Chapter 2 and 3. SHOGUN also offers the freedom of working with custom pre-computed kernels. One of its key features is the ”combined kernel” that can be constructed by a weighted linear combination of a multiple sub-kernels, each of which not necessarily working on the same domain. An optimal sub-kernel weighting can be learnt using Multiple Kernel Learning (Sonnenburg et al., 2006a) and also Section 4.1. Currently SVM two-and multiclass classification and regression problems can be handled. However, SHOGUN also implements many linear methods like Linear Discriminant Analysis (LDA), Linear Programming Machine (LPM), (Kernel) Perceptrons and features algorithms to train hidden markov models. The input feature-objects can be dense, sparse or strings and of Integer (8,16,32 or 64bit wide signed or unsigned) or Floatingpoint (32 or 64bit) and can be converted into different feature types. Chains of ”preprocessors” (e.g., subtracting the mean) can be attached to each feature object allowing for on-the-fly pre-processing. In addition, the performance measures mentioned in Section 5.1 are implemented in SHOGUN.
It provides user interfaces of two types: Static and modular interfaces to various scripting languages (stand-a-lone command-line, python, octave, R, matlab).
SHOGUN is available from http://www.shogun-toolbox.org and licensed under the GNU General Public License version 3 or later.
All classes and functions are documented and come with numerous examples. Code quality is ensured by a test suite running the algorithms for each interface on different inputs detecting breakage.

C.1 Modular, Extendible Object Oriented Design

SHOGUN is implemented in an object oriented way using C++ as programming language. All objects inherit from `CSGObject`, which provides means for garbage collection via reference counting and also I/O and versioning information of the object. Many classes make heavy use of *templates*, which is one of the reasons why shogun supports many different data types without code duplication. As the source code and user

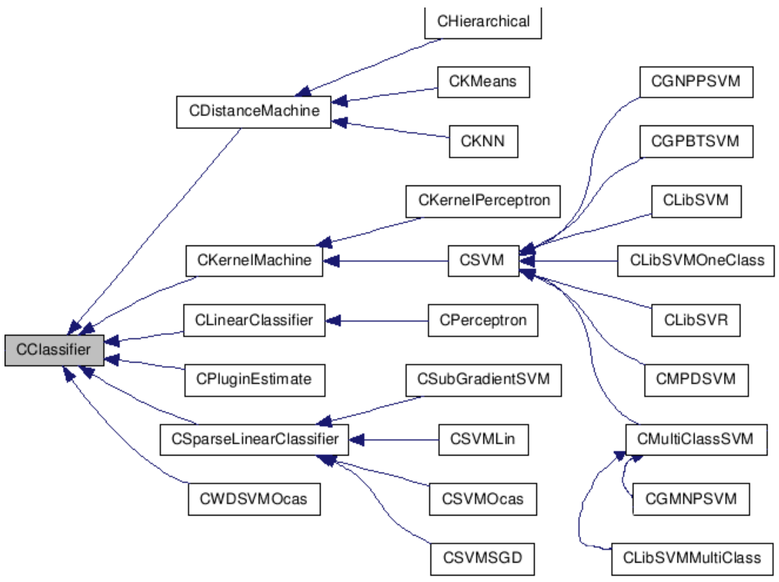


Figure C.1: SHOGUNs internal object oriented class design, explained using the classifier class hierarchy (autogenerated using doxygen)

documentation is automatically generated using doxygen and written in-place in the header files, it also drastically reduces the amount of work needed (following the DRY (don't repeat yourself) principle) to maintain the documentation. As and example to SHOGUNs object oriented design, consider the class `CClassifier`: From this class, e.g., `CKernelMachine` is derived and provides basic functions to apply a trained kernel classifier (computing $f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \, \mathrm{k}(\mathbf{x}, \mathbf{x}_i) + b$). The same holds for `CDistanceMachine` and `CLinearClassifier` (which provides $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$).

C.2 Interfaces to Scripting Languages and Applications

The core of SHOGUN consists of several interfaces like, e.g., `CClassifier`, `CFeatures` etc. For each of the interfaces, a class hierarchy similar to the one in Figure C.1, providing means to re-use code whenever possible, is found. Built around SHOGUN's core are

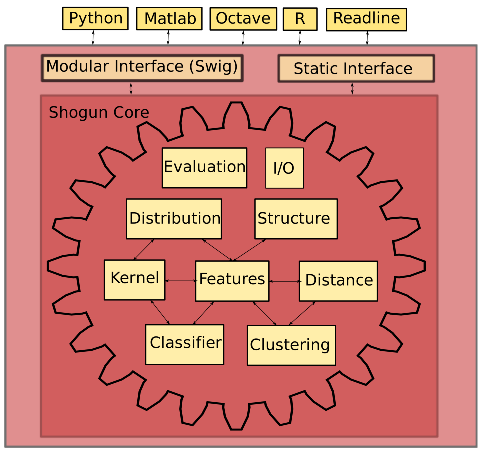


Figure C.2: Object Oriented Class Design

two types of interfaces: A modular interface that makes use of the SWIG (Simplified Wrapper and Interface Generator, c.f. `http://www.swig.org`) and a static interface. Thanks to the use of SWIG, the modular interface provides the exact same objects in a modular object oriented way that are available from C++ to other languages, such as R, python and octave. Using so-called `typemaps` it is conveniently possible to provide type mappings from the datatype used in the interface to shogun. For example, a function `void set_features(double* features, int n, int m)` can be directly called from within octave using a matrix as an argument `set_features(randn(3,4))`, the size of the matrix is then mapped to the variables n and m and the pointer to the block of the matrix, stored in fortran order, is then handed over to the shogun core. This also makes it easy to provide interfaces to not just a single language, as only the `typemaps` have to be rewritten for the particular target language.
The static interface was historically first and can only deal with a single object of a kind. It is therefore limited to, e.g., training single classifiers, as only one classifier object is kept in memory (this limitation does not exist in the modular interfaces). Similar to SWIG, the interface is implemented abstractly and independent of the target language through a class `CSGInterface`. This class provides abstract functions to deliver data to or obtain data from a particular interface (like e.g., python). This interface still has its benefits as it does not have a build dependency on `swig`, the syntax is the same for all interfaces and offers a command-line interface for interactive use.
For example, to train a support vector machine classifier using an RBF-kernel of width `width` one would issue (the same for *Matlab*, *Octave*, *Python* and *R*):

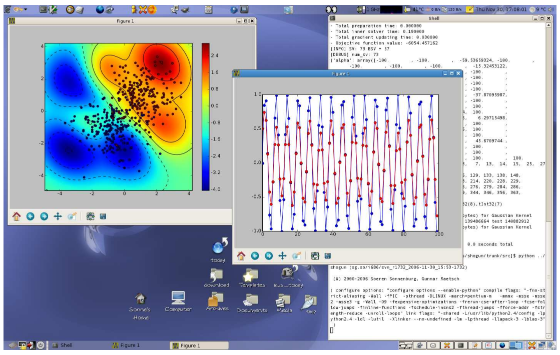
```
sg('set_labels', train_label);
sg('set_features', 'TRAIN', train_input);
sg('set_kernel', 'GAUSSIAN', 'REAL', cache_size, width);
sg('init_kernel', 'TRAIN');
sg('new_classifier', 'LIBSVM');
sg('train_classifier');
sg('set_features', 'TEST', test_input);
sg('init_kernel', 'TEST');
prediction = sg('classify');
```

For the modular interfaces (currently available for *Python*, *Octave* and *R* this code would look like this:

```
y=Labels(train_label);
x=RealFeatures(train_input);
kernel=GaussianKernel(x, x, width);
svm=LibSVM(C, kernel, y);
svm.train();
x_test=RealFeatures(test_input);
kernel.init(x, x_test);
prediction = svm.classify().get_labels();
```

Platforms SHOGUN runs on POSIX platforms, Figure C.3 shows an SVM-classification and SV-regression for the python interface (using the plotting functions from matplotlib (c.f. `http://matplotlib.sf.net`).

Applications Due to its flexible design, SHOGUN has been used in a variety of publications and applications, that are large scale genomic signal detection, interpretability



```
- Total preparation time: 0.000000
- Total inner solver time: 0.190000
- Total gradient updating time: 0.030000
- Objective function value: -6054.457162
[INFO] SV: 73 BSV = 57
[DEBUG] num_sv: 73
{'alpha': array([-100.        , -100.        ,  -59.53659324, -100.        ,
       -100.        , -100.        , -100.        ,  -15.32453122,
                                                  , -100.        ,
                                                  , -100.        ,
                                                  , -100.        ,
                                                  , -100.        ,
                                                  ,  -37.87095987,
                                                  , -100.        ,
                                                  ,  100.        ,
                                                 4,  100.        ,
                                                 5,    6.29715498,
                                                  ,  100.        ,
                                                 5,  100.        ,
                                                  ,  100.        ,
                                                  ,   45.6709744 ,
                                                  ,  100.        ,
                                                  ,  100.        ,
                                                  ,  100.        ,   100.
                                                 3,    7,   13,   14,   15,   25,   27

                                                 5, 129, 133, 138, 148,
                                                 3, 214, 220, 228, 229,
                                                 5, 276, 279, 284, 286,
                                                 9, 344, 346, 356, 363,

                                                 32(8),tInt32(7)

                                                 bytes) for Gaussian Kernel
                                                  139486664 test 140882912
                                                 bytes) for Gaussian Kernel

                                                  0.0 seconds total

                                                 s/shogun/trunk/src]$ python ../
shogun (sg.so/i686/svn_r1732_2006-11-30_15:53-1732)

 (W) 2000-2006 Soeren Sonnenburg, Gunnar Raetsch

( configure options: "configure options --enable-python" compile flags: "-fno-st
rict-aliasing -Wall -fPIC  -pthread -DLINUX -march=pentium-m   -mmmx -msse -msse
2 -msse3 -g -Wall -O9 -fexpensive-optimizations -frerun-cse-after-loop -fcse-fol
low-jumps -finline-functions -fschedule-insns2 -fthread-jumps -fforce-addr -fstr
ength-reduce -unroll-loops" link flags: "-shared -L/usr/lib/python2.4/config -lp
ython2.4 -ldl -lutil  -Xlinker --no-undefined -lm -lpthread -llapack-3 -lblas-3"
 )
```

Figure C.3: *Python* Interface running on Debian GNU Linux

of kernel machines and label sequence learning. The following enumeration summarises the key applications:

1. Large Scale Genomic Signal Detection
    - Splice sites (training on 10 *million* examples, testing on 500 *million* examples (whole genome)) (Sonnenburg et al., 2007b)
    - Promoter and PolyA detection (linear combination of 5 string-kernels), evaluation on 6 billion examples (Sonnenburg et al., 2006b)
    - Alternative splicing (combination of kernels) (Rätsch et al., 2005)
2. Interpretability
    - Multiple kernel learning (MKL) (Sonnenburg et al., 2006a)
    - Positional Oligomer Importance Matrices (POIM) (Sonnenburg et al., 2008)
3. Label Sequence Learning
    - *mSplicer* (Rätsch et al., 2007) (predicting the exon, intron segmentation of genes)
    - *PALMA* (Schulze et al., 2007) (mRNA to Genome Alignments using Large Margin Algorithms
    - *QPALMA* (De Bona et al., 2008) (Optimal Spliced Alignments of Short Sequence Reads)
    - *mGene* (Schweikert et al., 2008) (accurate gene finder based on signal sensors and extension of mSplicer)

A community around SHOGUN is continuously developing with a number of projects relying on it, see e.g., `http://mloss.org/software/tags/shogun` and a mailing-list with about 50 subscribed users (October 2008). We express our hope that it will be of high impact in the future, especially in the field of bioinformatics.

D Notation

The set of symbols and functions used throughout this Phd thesis:

N	number of examples
D	dimensionality of the space, e.g., $\mathbb{R}^D$
$\mathbf{x}, \mathbf{x}'$	input vectors of dimensionality D or strings
$l_{\mathbf{x}}, l_{\mathbf{x}'}$	length of string $\mathbf{x}$ and $\mathbf{x}'$
y	class label
d, K	degree of, e.g., a polynomial; order
$\boldsymbol{\alpha}$	the set of lagrange multipliers
$\boldsymbol{\beta}$	the set of sub-kernel weights
$\boldsymbol{\theta}$	the set of model parameters
$\mathcal{L}(\dots)$	the lagrangian
$L(.,.)$	a loss function
C	the regularisation parameter
ξ	a slack variable
$\mathbf{w}$	the normal
b	the bias
$\Pr[\mathbf{X}]$	probability of $\mathbf{X}$
$\mathbb{E}[\mathbf{X}]$	expected value of $\mathbf{X}$
$\Pr[\mathbf{X}\vert\mathbf{Y}]$	conditional probability of $\mathbf{X}$ given $\mathbf{Y}$
$\mathbb{E}[\mathbf{X}\vert\mathbf{Y}]$	conditional expectation of $\mathbf{X}$ given $\mathbf{Y}$
sign	the signum function
$I(.)$	Kronecker's delta
$\Phi(.)$	kernel feature mapping
$\mathrm{k}(.,.)$	the kernel function
$\Vert\cdot\Vert_p$	ℓ_p norm
$f(\mathbf{x})$, $g(\mathbf{x})$	classifier or function
∂_x	the partial derivative with respect to x, i.e., $\frac{\partial}{\partial x}$
θ, γ	auxilary scalar
$\mathbf{1}$	vector of ones or matrix with ones on diagonal (rest zero)
$\mathbf{x} \cdot \mathbf{x}'$	dot product between vectors $\mathbf{x}$ and $\mathbf{x}'$
Σ	an alphabet, e.g., the DNA alphabet $\Sigma = \{A, C, G, T\}$
$\mathbb{I}(\cdot)$	indicator function; evaluates to 1 when its argument is *true*, 0 otherwise.
$\mathcal{O}(.)$	big O notation, used to analyse (computational) effort
$\#\cdot$	shortcut for number of
Q	size of working set
$Q(.,.)$	Positional Oligomer Importance Matrix (POIM)
$k-$mer, $k-$gram	a string of length k
$\boldsymbol{u}$	a string of length d
$(\mathbf{x}, i), \mathbf{x}[i]^k$	Positional Oligomer (PO) - a substring of $\mathbf{x}$ that starts at position i and has length k

References

T. Abeel, Y. Saeys, E. Bonnet, P. Rouzé, and Y. V. de Peer. Generic eukaryotic core promoter prediction using structural features of DNA. *Genome Research*, 18:310—323, 2008a.

T. Abeel, Y. Saeys, P. Rouzé, and Y. V. de Peer. ProSOM: core promoter prediction based on unsupervised clustering of DNA physical profiles. *Bioinformatics*, 24(13): i24–31, July 2008b.

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.

Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines. In T. Fawcett and N. Mishra, editors, *Proc. 20th International Conference on Machine Learning*, pages 3–10. AAAI Press, 2003.

A.thaliana. Riken cress sequence, 2007. `http://rarge.psc.riken.jp/archives/rafl/sequence/`.

O. T. Avery, C. M. MacLeod, and M. McCarty. Studies on the Chemical Nature of the Substance Inducing Transformation of Pneumococcal Types: Induction of Transformation by a Desoxyribonucleic Acid Fraction Isolated from Pneumococcus Type III. *J. Exp. Med.*, 79(2):137–158, 1944.

F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In C. E. Brodley, editor, *Twenty-first international conference on Machine learning*, pages 41–48. ACM, 2004.

V. Bajic and S. Seah. Dragon gene start finder: an advanced system for finding approximate locations of the start of gene transcriptional units. *Nucleic Acids Research*, 31: 3560–3563, 2003.

V. Bajic, M. Brent, R. Brown, A. Frankish, J. Harrow, U. Ohler, V. Solovyev, and S. Tan. Performance assessment of promoter predictions on encode regions in the egasp experiment. *Genome Biology*, 7(Suppl. 1):1–13, Aug. 2006.

V. B. Bajic, S. L. Tan, Y. Suzuki, and S. Sugano. Promoter prediction analysis on the whole human genome. *Nat Biotechnol*, 22(11):1467–73, Nov. 2004.

P. 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. MIT Press, Cambridge, MA, 1998.

A. Baten, B. Chang, S. Halgamuge, and J. Li. Splice site identification using probabilistic parameters and svm classification. *BMC Bioinformatics 7(Suppl. 5):S15*, 2006a.

A. Baten, B. Chang, S. Halgamuge, and J. Li. Correction notes to bmc bioinformatics 2006, 7(suppl 5):s15, 2006b. `http://www.mame.mu.oz.au/bioinformatics/splicesite`.

A. Ben-Hur, C. S. Ong, S. Sonnenburg, B. Schölkopf, and G. Rätsch. Support vector machines and kernels for computational biology. *PLoS Comput Biology*, 4(10): e1000173, Oct 2008.

K. P. Bennett, M. Momma, and M. J. Embrechts. MARK: a boosting algorithm for heterogeneous kernel models. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 24–31. ACM, 2002.

A. Bernal, K. Crammer, A. Hatzigeorgiou, and F. Pereira. Global discriminative learning for higher-accuracy computational gene prediction. *PLoS Computational Biology*, 3(3):e54, 2007.

J. Bi, T. Zhang, and K. P. Bennett. Column-generation boosting methods for mixture of kernels. In W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel, editors, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 521–526. ACM, 2004.

H. D. Block. The perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34:123–135, 1962. Reprinted in "Neurocomputing" by Anderson and Rosenfeld.

M. Boguski and T. L. C. Tolstoshev. dbEST–database for "expressed sequence tags". *Nature Genetics*, 4(4):332–3, 1993.

B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, 1992. ACM Press.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.

M. Braun, J. Buhmann, and K.-R. Müller. On relevant dimensions in kernel feature spaces. *JMLR*, 9:1875–1908, Aug 2008.

B. Brejova, D. Brown, M. Li, and T. Vinar. ExonHunter: a comprehensive approach to gene finding. *Bioinformatics*, 21(Suppl 1):i57–i65, 2005.

C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268:78–94, 1997.

C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2):121–167, 1998.

M. Burset, I. Seledtsov, and V. Solovyev. Analysis of canonical and non-canonical splice sites in mammalian genomes. *Nucleic Acid Research*, 28(21):4364–4375, 2000.

D. Cai, A. Delcher, B. Kao, and S. Kasif. Modeling splice sites with Bayes networks. *Bioinformatics*, 16(2):152–158, 2000.

C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. Coordinate descent method for large-scale l2-loss linear support vector machines. *Journal of Machine Learning Research*, 9: 1369–1398, Jul 2008.

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(1–3):131–159, 2002.

F. Chen, C. C. MacDonald, and J. Wilusz. Cleavage site determinants in the mammalian polyadenylation signal. *Nucleic Acids Res*, 23:2614–2620, July 1995.

T.-M. Chen, C.-C. Lu, and W.-H. Li. Prediction of splice sites with dependency graphs and their expanded bayesian networks. *Bioinformatics*, 21(4):471–482, 2005.

V. Cherkassky and F. Mulier. *Learning from Data — Concepts, Theory and Methods*. John Wiley & Sons, New York, 1998.

J. Chuang and D. Roth. Splice site prediction using a sparse network of winnows. *Technical Report UIUCDCS-R-2001-2199*, Feb 2001.

R. Clark, G. Schweikert, C. Toomajian, S. Ossowski, G. Zeller, P. Shinn, N. Warthmann, T. Hu, G. Fu, D. Hinds, H. Chen, K. Frazer, D. Huson, B. Schölkopf, M. Nordborg, G. Rätsch, J. Ecker, and D. Weigel. Common sequence polymorphisms shaping genetic diversity in Arabidopsis thaliana. *Science*, 317(5836), July 2007.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

C. Cortes, P. Haffner, and M. Mohri. Rational kernels: Theory and algorithms. *Journal of Machine Learning Research*, 5:1035–1062, 2004.

F. Crick, L. Barnett, S. Brenner, and R. Watts-Tobin. General nature of the genetic code for proteins. *Nature*, 192:1227–1232, Dec 1961.

R. Dahm. Friedrich miescher and the discovery of DNA. *Human Genetics*, 278:274–88, Feb 2005.

M. Damashek. Gauging similarity with *n*-grams: Language-independent categorization of text. *Science*, 267(5199):843–848, 1995.

J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. Technical report #1551, University of Wisconsin Madison, January 2006.

R. V. Davuluri, I. Grosse, and M. Q. Zhang. Computational identification of promoters and first exons in the human genome. *Nat Genet*, 29(4):412–417, Dec. 2001.

F. De Bona, S. Ossowski, K. Schneeberger, and G. Rätsch. Optimal spliced alignments of short sequence reads. *Bioinformatics/Proc. ECCB'08*, 2008.

S. Degroeve, Y. Saeys, B. D. Baets, P. Rouzé, and Y. V. de Peer. SpliceMachine feature extractor, 2004. `http://bioinformatics.psb.ugent.be/supplementary_data/`

svgro/splicemachine/downloads/splice_machine_sept_2004.zip.

S. Degroeve, Y. Saeys, B. D. Baets, P. Rouzé, and Y. V. de Peer. Splicemachine: predicting splice sites from high-dimensional local context representations. *Bioinformatics*, 21(8):1332–8, 2005a.

S. Degroeve, Y. Saeys, B. D. Baets, P. Rouzé, and Y. V. de Peer. SpliceMachine, 2005b. http://bioinformatics.psb.ugent.be/webtools/splicemachine/.

A. Delcher, D. Harmon, S. Kasif, O. White, and S. Salzberg. Improved microbial gene identification with GLIMMER. *Nucleic Acids Research*, 27(23):4636–4641, 1999.

T. Down and T. Hubbard. Computational detection and location of transcription start sites in mammalian genomic DNA. *Genome Res*, 12:458–461, 2002.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

Ensemble. Ensemble, 2007. http://www.ensemble.org.

T. Fawcett. Roc graphs: Notes and practical considerations for data mining researchers. Technical report hpl-2003-4, HP Laboratories, Palo Alto, CA, USA, January 2003.

W. Fiers, R. Contreras, F. Duerinck, G. H. G, D. Iserentant, J. Merregaert, W. M. Jou, F. Molemans, A. Raeymaekers, A. V. den Berghe, G. Volckaert, and M. Ysebaert. Complete nucleotide sequence of bacteriophage ms2 rna: primary and secondary structure of the replicase gene. *Nature*, 260:500–5007, Apr 1976.

V. Franc and S. Sonnenburg. OCAS optimized cutting plane algorithm for support vector machines. In *Proceedings of the 25nd International Machine Learning Conference*, pages 320–327. ACM Press, 2008.

E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

FruitFly. Fruit fly genome sequence, 2007a. http://www.fruitfly.org/sequence/human-datasets.html.

FruitFly. Fruit fly expression sequence tags, 2007b. http://www.fruitfly.org/EST.

K. Gao, A. Masuda, T. Matsuura, and K. Ohno. Human branch point consensus sequence is yUnAy. *Nucl. Acids Res.*, page gkn073, 2008.

M. Gardiner-Garden and M. Frommer. Cpg islands in vertebrate genomes. *Journal of Molecular Biology*, 196:261–282, July 1987.

GeneSplicer. Genesplicer, 2001. http://www.cbcb.umd.edu/software/GeneSplicer/.

A. Goffeau, B. Barrell, H. Bussey, R. Davis, B. Dujon, H. Feldmann, F. Galibert, J. Hoheisel, C. Jacq, M. Johnston, E. Louis, H. Mewes, Y. Murakami, P. Philippsen, H. Tettelin, and S. Oliver. Life with 6000 Genes. *Science*, 274(5287):546–567, 1996.

P. Goujon. *From Biotechnology to Genomes: The Meaning of the Double Helix*. World Scientific, 2001.

J. Graber, J. Salisbury, L. Hutchins, and T. Blumenthal. C. elegans sequences that control trans-splicing and operon pre-mRNA processing. *RNA*, 13, Sept. 2007.

A. Graf, F. Wichmann, H. H. Bülthoff, and B. Schölkopf. Classification of faces in man and machine. *Neural Computation*, 18:143–165, 2006.

Y. Grandvalet and S. Canu. Adaptive scaling for feature selection in SVMs. In S. T. S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 553–560, Cambridge, MA, 2003. MIT Press.

S. G. Gregory et al. The DNA sequence and biological annotation of human chromosome i. *Nature*, 441:315–321, May 2006.

M. Gribskov and N. L. Robinson. Use of receiver operating characteristic (roc) analysis to evaluate sequence matching. *Comput.Chem.*, 20:25–33, 1996.

M. Gribskov, A. McLachlan, and D. Eisenberg. Profile analysis: Detection of distantly related proteins. *Proceedings of the National Academy of Sciences*, 84:4355–4358, 1987.

R. Guigo, P. Flicek, J. Abril, A. Reymond, J. Lagarde, F. Denoeud, S. Antonarakis, M. Ashburner, V. Bajic, E. Birney, R. Castelo, E. Eyras, C. Ucla, T. Gingeras, J. Harrow, T. Hubbard, S. Lewis, and M. Reese. Egasp: the human encode genome annotation assessment project. *Genome Biology*, 7(Suppl 1):S2, 2006.

D. Gusfield. *Algorithms on strings, trees, and sequences.* Cambridge University Press, 1997.

T. Harris et al. Wormbase: a multi-species resource for nematode biology and genomics. *Nucl. Acids Res.*, 32, 2004. Database issue:D411-7.

D. Haussler. Computational genefinding. *Computational genefinding. Trends Biochem. Sci.*, 1998.

D. Haussler. Convolutional kernels on discrete structures. Technical Report UCSC-CRL-99 - 10, Computer Science Department, UC Santa Cruz, 1999.

E. C. Hayden. Company claims to have sequenced man's genome cheaply. *Nature*, Feb 2008. URL `http://www.nature.com/news/2008/080213/full/451759c.html`.

R. Hettich and K. O. Kortanek. Semi-infinite programming: Theory, methods and applications. *SIAM Review*, 3:380–429, 1993.

D. Hinds, L. Stuve, G. Nilsen, E. Halperin, E. Eskin, D. Ballinger, K. Frazer, and D. Cox. Whole-genome patterns of common DNA variation in three human populations. *Science*, 307(5712):1072–1079, 2005.

Illumina. Illumina Completes Sequencing of African Trio, May 2008. `http://investor.illumina.com/phoenix.zhtml?c=121127&p=irol-newsArticle&ID=1140169`.

ILOG CPLEX. *Using the CPLEX Callable Library.* CPLEX Optimization Incorporated, Incline Village, Nevada, 1994.

T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 487–493, 1999.

T. S. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *J. Comp. Biol.*, 7:95–114, 2000.

V. X. Jin, G. A. Singer, F. J. Agosto-Perez, S. Liyanarachchi, and R. V. Davuluri. Genome-wide analysis of core promoter elements from conserved human and mouse orthologous pairs. *BMC Bioinformatics*, 7(1):114, March 2006.

T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In C. Nédellec and C. Rouveirol, editors, *ECML '98: Proceedings of the 10th European Conference on Machine Learning*, Lecture Notes in Computer Science, pages 137–142, Berlin / Heidelberg, 1998. Springer-Verlag.

T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, USA, 1999. MIT Press.

W. Jou, G. Haegeman, M. Ysebaert, and W. Fiers. Nucleotide sequence of the gene coding for the bacteriophage ms2 coat protein. *Nature*, 237:82–88, May 1972.

P. P. Kapranov, A. Willingham, and T. Gingeras. Genome-wide transcription and the implications for genomic organization. *Nature Reviews Genetics*, 8:413–423, 2007.

H. Kawaji, T. Kasukawa, S. Fukuda, S. Katayama, C. Kai, J. Kawai, P. Carninci, and Y. Hayashizaki. CAGE Basic/Analysis Databases: the CAGE resource for comprehensive promoter analysis. *Nucl. Acids Res.*, 34(Suppl. 1):D632–636, 2006.

W. J. Kent. BLAT–the BLAST-like alignment tool. *Genome Res*, 12(4):656–664, Apr. 2002.

D. E. Knuth. *The art of computer programming*, volume 3. Addison-Wesley, 1973.

I. Korf. Gene finding in novel genomes. *BMC Bioinformatics*, 5:59, 2004.

A. Kornberg. *For the Love of Enzymes: The Odyssey of a Biochemist.* by Harvard University Press, 1991.

R. Kuang, E. Ie, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C. Leslie. Profile-based string kernels for remote homology detection and motif extraction. In *Computational Systems Bioinformatics Conference 2004*, pages 146–154, 2004.

D. Kulp, D. Haussler, M. Reese, and F. Eeckman. A generalized hidden markov model for the recognition of human genes in DNA. *ISMB 1996*, pages 134–141, 1996.

G. Lanckriet, T. D. Bie, N. Cristianini, M. Jordan, and W. Noble. A statistical framework for genomic data fusion. *Bioinformatics*, 20:2626–2635, 2004.

E. Lehmann. *Testing Statistical Hypotheses.* Springer, New York, second edition edition, 1997.

C. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 5:1435–1455, 2004.

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. Lauderdale, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575, Kaua'i, Hawaii, 2002.

C. Leslie, E. Eskin, J. Weston, and W. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4), 2003a.

C. Leslie, R. Kuang, and E. Eskin. Inexact matching string kernels for protein classification. In *Kernel Methods in Computational Biology*, MIT Press series on Computational Molecular Biology, pages 95–112. MIT Press, 2003b.

S. Levy, G. Sutton, P. Ng, L. Feuk, and A. Halpern. The diploid genome sequence of an individual human. *PLoS Biology*, 5:e254, 2007.

B. Lewin. *Genes VII.* Oxford University Press, New York, 2000.

L. Liao and W. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *Proceedings of the Sixth Annual International Conference on Research in Computational Molecular Biology*, pages 225–232, April 2002.

K. Liolios, K. Mavromatis, N. Tavernarakis, and N. C. Kyrpides. The Genomes On Line Database (GOLD) in 2007: status of genomic and metagenomic projects and their associated metadata. *Nucl. Acids Res.*, page gkm884, 2007.

H. Liu, H. Han, J. Li, and L. Wong. An *in-silico* method for prediction of polyadenylation signals in human sequences. In S. M. M. Gribskov, M. Kanehisa and T. Takagi, editors, *Genome Informatics 14*, pages 84–93. Universal Academic Press, Tokyo, 2003.

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.

Mammalian Gene Collection. Mammalian gene collection, 2007. `http://mgc.nci.nih.gov`.

K. Marusina. The next generation of DNA sequencing genetic engineering, 2006. URL `http://www.genengnews.com/articles/chitem.aspx?aid=1946`.

V. Matys, O. Kel-Margoulis, E. Fricke, I. Liebich, S. Land, A. Barre-Dirrie, I. Reuter, D. Chekmenev, M. Krull, K. Hornischer, N. Voss, P. Stegmaier, B. Lewicki-Potapov, H. Saxel, A. Kel, and E. Wingender. TRANSFAC and its module TRANSCompel: Transcriptional gene regulation in eukaryotes. *Nucleic Acids Res.*, 34:D108–110, 2006.

D. K. McClish. Analyzing a portion of the roc curve. *Med. Decision Making*, 9:190–195, 1989.

P. Meinicke, M. Tech, B. Morgenstern, and R. Merkl. Oligo kernels for datamining on biological sequences: A case study on prokaryotic translation initiation sites. *BMC Bioinformatics*, 5(1):169, 2004.

R. Meir and G. Rätsch. An introduction to boosting and leveraging. In S. Mendelson and A. Smola, editors, *Proc. of the first Machine Learning Summer School in Canberra*, LNCS, pages 119–184. Springer, 2003.

J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, A 209:415–446, 1909.

G. E. Merrihew, C. Davis, B. Ewing, G. Williams, L. Kall, B. E. Frewen, W. S. Noble, P. Green, J. H. Thomas, and M. J. MacCoss. Use of shotgun proteomics for the identification, confirmation, and correction of C. elegans gene annotations. *Genome Res.*, page gr.077644.108, 2008.

M. Meselson and F. Stahl. The replication of DNA in escherichia coli. *Proc Natl Acad Sci*, 44:671–682, Jul 1958.

C. Metz. Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, VIII(4), October 1978.

A. Mood, F. Graybill, and D. Boes. *Introduction to the Theory of Statistics*. McGraw-Hill, third edition edition, 1974.

Mouse Genome Sequencing Consortium. Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420:520–62, Dec 2002.

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.

K. Mullis. Polymerase chain reaction (pcr), 2004. URL `http://www.karymullis.com/pcr.html`.

M. Nirenberg. Historical review: Deciphering the genetic code — a personal account. *Trends in Biochemical Sciences*, 29, Jan 2004.

A. B. J. Novikoff. On convergence proofs on perceptrons. In *In Proceedings of the Symposium on the Mathematical Theory of Automata*, volume XII, pages 615–622, 1962.

U. Ohler, G. C. Liao, H. Niemann, and G. M. Rubin. Computational analysis of core promoters in the drosophila genome. *Genome Biology*, 3(12):research0087.1–0087.12, 2002.

C. S. Ong and G. Rätsch. Genome wide analysis of novel alternative splicing events. *In preparation*, 2008.

C. S. Ong, A. J. Smola, and R. C. Williamson. Hyperkernels. In S. T. S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, volume 15, pages 478–485, Cambridge, MA, 2003. MIT Press.

H. Pearson. Genetics: what is a gene? *Nature*, 441:398–401, 2006.

A. Pedersen and H. Nielsen. Neural Network Prediction of Translation Initiation Sites in Eukaryotes: Perspectives for EST and Genome analysis. In *ISMB'97*, volume 5, pages 226–233, 1997.

M. Pertea, X. Lin, and S. Salzberg. Genesplicer: a new computational method for splice site prediction. *Nucleic Acids Research*, 29(5):1185–1190, 2001.

M. Pertea, S. Mount, and S. Salzberg. A computational survey of candidate exonic splicing enhancer motifs in the model plant arabidopsis thaliana. *BMC Bioinformatics*, 8 (1):159, 2007.

P. Philips, K. Tsuda, J. Behr, and G. Rätsch. Accurate recognition of polyadenylation sites. In preparation, 2008.

J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, USA, 1999. MIT Press.

K. D. Pruitt, T. Tatusova, and D. R. Maglott. NCBI Reference Sequence (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucl. Acids Res.*, 33(S1):D501–504, 2005. doi: 10.1093/nar/gki025. URL `http://nar.oxfordjournals.org/cgi/content/abstract/33/suppl_1/D501`.

L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, Feb. 1989.

J. Rajapakse and L. Ho. Markov encoding for detecting signals in genomic sequences. *IEEE ACM Transactions on Computational Biology and Bioinformatics*, 2(2):131–142, 2005.

S. Rampone. Recognition of splice junctions on DNA sequences by BRAIN learning algorithm. *Bioinformatics*, 14(8):676–684, 1998.

G. Rätsch. *Robust Boosting via Convex Optimization.* PhD thesis, University of Potsdam, Potsdam, Germany, 2001.

G. Rätsch and S. Sonnenburg. *Accurate Splice Site Prediction for Caenorhabditis Elegans*, pages 277–298. MIT Press series on Computational Molecular Biology. MIT Press, 2004.

G. Rätsch and S. Sonnenburg. Large scale hidden semi-markov svms. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1161–1168. MIT Press, Cambridge, MA, 2007.

G. Rätsch and M. Warmuth. Efficient margin maximization with boosting. *Journal of Machine Learning Research*, 6:2131–2152, 2005.

G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3):287–320, Mar. 2001. also NeuroCOLT Technical Report NC-TR-1998-021.

G. Rätsch, A. Demiriz, and K. Bennett. Sparse regression ensembles in infinite and finite hypothesis spaces. *Machine Learning*, 48(1–3):193–221, 2002. Special Issue on New Methods for Model Selection and Model Combination. Also NeuroCOLT2 Technical Report NC-TR-2000-085.

G. Rätsch, S. Sonnenburg, and B. Schölkopf. RASE: Recognition of alternatively spliced exons in C. elegans. *Bioinformatics*, 21:i369–i377, 2005.

G. Rätsch, S. Sonnenburg, and C. Schäfer. Learning interpretable svms for biological sequence classification. *BMC Bioinformatics*, 7((Suppl 1)):S9, Mar. 2006.

G. Rätsch, S. Sonnenburg, J. Srinivasan, H. Witte, K.-R. Müller, R. Sommer, and B. Schölkopf. Improving the c. elegans genome annotation using machine learning. *PLoS Computational Biology*, 3(2):e20, 2007.

M. Reese, F. H. Eeckman, D. Kulp, and D. Haussler. Improved splice site detection in Genie. *Journal of Computational Biology*, 4:311–323, 1997.

K. Rieck, P. Laskov, and K.-R. Müller. Efficient algorithms for similarity measures over sequential data: A look beyond kernels. In *Pattern Recognition, 28th DAGM Symposium*, LNCS, pages 374–383. Springer-Verlag, Sept. 2006.

K. Rieck, P. Laskov, and S. Sonnenburg. Computation of similarity measures for sequential data using generalized suffix trees. In *Advances in Neural Information Processing Systems 19*, pages 1177–1184, Cambridge, MA, 2007. MIT Press.

M. Ronaghi, S. Karamohamed, B. Pettersson, M. Uhlén, and P. Nyrén. Real-time DNA sequencing using detection of pyrophosphate release. *Analytical Biochemistry*, 242 (1):84–89, November 1996.

Y. Saeys, T. Abeel, S. Degroeve, and Y. Van de Peer. Translation initiation site prediction on a genomic scale: beauty in simplicity. *Bioinformatics*, 23(13):i418–423, 2007.

S. Saitoh. *Theory of Reproducing Kernels and its Applications.* Longman Scientific & Technical, Harlow, England, 1988.

G. Salton. Mathematics and information retrieval. *Journal of Documentation*, 35(1): 1–29, 1979.

S. Salzberg, A. Delcher, K. Fasman, and J. Henderson. A decision tree system for finding genes in DNA. *Journal of Computational Biology*, 5(4):667–680, 1998.

F. Sanger and A. Coulson. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. *J Mol Biol*, 94(3):441–448, May 1975.

F. Sanger, S. Nicklen, and A. Coulson. DNA sequencing with chain-terminating inhibitors. *Proc Natl Acad Sci*, 74:5463–5470, Dec 1977.

B. Schölkopf. *Support vector learning.* Oldenbourg Verlag, Munich, 1997.

B. Schölkopf and A. J. Smola. *Learning with Kernels.* MIT Press, Cambridge, MA, 2002.

B. Schölkopf, C. Burges, and A. Smola, editors. *Advances in Kernel Methods – Support Vector Learning.* MIT Press, 1999.

B. Schölkopf, K. Tsuda, and J.-P. Vert. *Kernel Methods in Computational Biology.* MIT Press, Cambridge, MA, 2004.

S. J. Schultheiss, W. Busch, J. U. Lohmann, O. Kohlbacher, and G. Rätsch. KIRMES: Kernel-based identification of regulatory modules in euchromatic sequences. In *Proceedings of the German Conference on Bioinformatics.* GI, Springer Verlag, 2008. URL `http://www.fml.tuebingen.mpg.de/raetsch/projects/kirmes`.

U. Schulze, B. Hepp, C. S. Ong, and G. Ratsch. PALMA: mRNA to genome alignments using large margin algorithms. *Bioinformatics*, 23(15):1892–1900, 2007.

G. Schweikert, G. Zeller, A. Zien, J. Behr, C.-S. Ong, P. Philips, A. Bohlen, S. Sonnenburg, and G. Rätsch. mGene: A novel discriminative gene finding system. In preparation, 2008.

H. O. Smith, C. A. H. , C. Pfannkoch, and J. C. Venter. Generating a synthetic genome by whole genome assembly: ΦX174 bacteriophage from synthetic oligonucleotides. *Proceedings of the National Academy of Sciences of the United States of America*, 100(26):15440–15445, 2003.

N. Smith and M. Gales. Speech recognition using SVMs. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans. *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 2000.

S. Sonnenburg. New methods for splice site recognition. Master's thesis, Humboldt University, 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 *Proceedings of the International Conference on Artifical Neural Networks.*, pages 329–336, 2002. Copyright by Springer.

S. Sonnenburg, G. Rätsch, and C. Schäfer. Learning interpretable SVMs for biological sequence classification. In S. Miyano, J. P. Mesirov, S. Kasif, S. Istrail, P. A. Pevzner, and M. Waterman, editors, *Research in Computational Molecular Biology, 9th Annual International Conference, RECOMB 2005*, volume 3500, pages 389–407. Springer-Verlag Berlin Heidelberg, 2005a.

S. Sonnenburg, G. Rätsch, and B. Schölkopf. Large scale genomic sequence SVM classifiers. In L. D. Raedt and S. Wrobel, editors, *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 849–856, New York, NY, USA, 2005b. ACM Press.

S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large Scale Multiple Kernel Learning. *Journal of Machine Learning Research*, 7:1531–1565, July 2006a.

S. Sonnenburg, A. Zien, and G. Rätsch. ARTS: Accurate Recognition of Transcription Starts in Human. *Bioinformatics*, 22(14):e472–480, 2006b.

S. Sonnenburg, G. Rätsch, and K. Rieck. Large scale learning with string kernels. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, pages 73–103. MIT Press, 2007a.

S. Sonnenburg, G. Schweikert, P. Philips, J. Behr, and G. Rätsch. Accurate Splice Site Prediction. *BMC Bioinformatics, Special Issue from NIPS workshop on New Problems and Methods in Computational Biology Whistler, Canada, 18 December 2006*, 8:(Suppl. 10):S7, December 2007b.

S. Sonnenburg, A. Zien, P. Philips, and G. Rätsch. POIMs: positional oligomer importance matrices — understanding support vector machine based signal detectors. *Bioinformatics*, 2008. (received the Best Student Paper Award at ISMB0́8).

L. Stein, D. Blasiar, A. Coghlan, T. Fiedler, S. McKay, and P. Flicek. ngasp gene prediction challenge, Mar. 2007. `http://www.wormbase.org/wiki/index.php/Gene_Prediction`.

C. Y. Suen. N-gram statistics for natural language understanding and text processing. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 1(2):164–172, Apr. 1979.

Y. Suzuki, R. Yamashita, K. Nakai, and S. Sugano. Dbtss: Database of human transcriptional start sites and full-length cDNAs. *Nucleic Acids Res*, 30(1):328–331, Jan. 2002.

C.-H. Teo and S. V. N. Vishwanathan. Fast and space efficient string kernels using suffix arrays. In *Proc. 23rd ICMP*, pages 939–936. ACM Press, 2006.

The C. *elegans* Sequencing Consortium. Genome sequence of the Nematode Caenorhabditis elegans. a platform for investigating biology. *Science*, 282:2012–2018, 1998.

The International HapMap Consortium. A haplotype map of the human genome. *Nature*, 437:1290–1320, 2005.

The International Human Genome Mapping Consortium. A physical map of the human genome. *Nature*, 409:934–941, Feb 2001.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, sep 2005.

K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K. Müller. A new discriminative kernel from probabilistic models. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural information processings systems*, volume 14, pages 977–984, Cambridge, MA, 2002a. MIT Press.

K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K. Müller. A new discriminative kernel from probabilistic models. *Neural Computation*, 14:2397–2414, 2002b.

K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 18:268S–275S, 2002c.

E. Ukkonen. Online construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

B. Üstün, W. J. Melssen, and L. M. Buydens. Visualisation and interpretation of support vector regression models. *Anal Chim Acta*, 595(1-2):299–309, July 2007.

S. K. V. Sindhwani. Newton methods for fast solution of semi-supervised linear svms. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, pages 155–174. MIT Press, 2007.

V. Vapnik. *The nature of statistical learning theory.* Springer Verlag, New York, 1995.

V. Vapnik. *Statistical Learning Theory.* Wiley, New York, 1998.

J.-P. Vert, H. Saigo, and T. Akutsu. Local alignment kernels for biological sequences. In K. T. B. Schoelkopf and J.-P. Vert, editors, *Kernel Methods in Computational Biology.* MIT Press, 2004.

S. Vishwanathan and A. J. Smola. Fast kernels for string and tree matching. In *Proc. NIPS '2002*, pages 569–576, 2003.

S. Vishwanathan and A. J. Smola. *Kernels and Bioinformatics*, chapter Fast Kernels for String and Tree Matching, pages 113–130. MIT Press, 2004.

A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Informat. Theory*, IT-13:260–269, Apr 1967.

M. Warmuth, J. Liao, and G. Rätsch. Totally corrective boosting algorithms that maximize the margin. In *ICML '06: Proceedings of the 23nd international conference on Machine learning*, pages 1001–1008. ACM Press, 2006.

C. Watkins. Dynamic alignment kernels. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 39–50, Cambridge, MA, 2000. MIT Press.

J. Watson and F. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171:737–738, Apr 1953.

T. Werner. The state of the art of mammalian promoter recognition. *Brief Bioinform*, 4(1):22–30, Mar. 2003.

D. Wheeler, M. Srinivasan, M. Egholm, Y. Shen, L. Chen, A. McGuire, W. He, Y. Chen, V. Makhijani, G. Roth, X. Gomes, K. Tartaro, F. Niazi, C. Turcotte, G. Irzyk, J. Lupski, C. Chinault, X. Song, Y. Liu, Y. Yuan, L. Nazareth, X. Qin, D. Muzny, M. Margulies, G. Weinstock, R. Gibbs, and J. Rothberg. The complete genome of an individual by massively parallel DNA sequencing. *Nature*, 452:872–876, Apr 2008.

D. Wheeler et al. Database resources of the national center for biotechnology. *Nucl. Acids Res*, 31:38–33, 2003.

Wikipedia. 454 life sciences — wikipedia, the free encyclopedia, 2008a. URL `http://en.wikipedia.org/wiki/454_Life_Sciences`. [Online; accessed 24-September-2008].

Wikipedia. DNA sequencing — wikipedia, the free encyclopedia, 2008b. URL `http://en.wikipedia.org/wiki/DNA_sequencing`. [Online; accessed 24-September-2008].

Wikipedia. List of sequenced eukaryotic genomes — Wikipedia, the free encyclopedia, 2007. URL `http://en.wikipedia.org/wiki/List_of_sequenced_eukaryotic_genomes`. [Online; accessed 02-Feb-2007].

Wormbase. Wormbase, 2008. `http://www.wormbase.org`.

M. Yamamura and O. Gotoh. Detection of the splicing sites with kernel method approaches dealing with nucleotide doublets. *Genome Informatics*, 14:426–427, 2003.

R. Yamashita, Y. Suzuki, H. Wakaguri, K. Tsuritani, K. Nakai, and S. Sugano. Dbtss: Database of human transcription start sites, progress report 2006. *Nucleic Acids Res*, 34:D86–89, Jan. 2006. Database issue.

L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training. *Journal of Machine Learning Research*, 7:1467–1492, July 2006.

H. Zhang and T. Blumenthal. Functional analysis of an intron 3' splice site in Caenorhabditis elegans. *RNA*, 2(4):380–388, 1996.

X. Zhang, C. Leslie, and L. Chasin. Dichotomous splicing signals in exon flanks. *Genome Research*, 15(6):768–79, Jun 2005.

X. H.-F. Zhang, K. A. Heller, I. Hefter, C. S. Leslie, and L. A. Chasin. Sequence information for the splicing of human pre-mRNA identified by support vector machine classification. *Genome Research*, 13(12):637–50, 2003.

Y. Zhang, C.-H. Chu, Y. Chen, H. Zha, and X. Ji. Splice site prediction using support vector machines with a bayes kernel. *Expert Systems with Applications*, 30:73–81, Jan 2006.

J. Zhao, L. Hyman, and C. Moore. Formation of mRNA 3' ends in eukaryotes: Mechanism, regulation, and interrelationships with other steps in mRNA synthesis. *Microbiology and Molecular Biology Reviews*, 63:405–445, June 1999.

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, Sept. 2000.

A. Zien, P. Philips, and S. Sonnenburg. Computing Positional Oligomer Importance Matrices (POIMs). Research Report; Electronic Publication 2, Fraunhofer FIRST, Dec. 2007.