New Methods for Splice Site Recognition

- Diploma Thesis -

Sören Sonnenburg
Humboldt-Universität zu Berlin

Institut für Informatik
3. Juli 2002

Supervisor

Prof. Dr. Klaus-Robert Müller
Fraunhofer Institut FIRST
Arbeitsgruppe IDA
(Intelligente Datenanalyse)

Prof. Dr. Hans-Dieter Burkhard
Humboldt-Universität zu Berlin
Institut für Informatik
Künstliche Intelligenz

Dr. Gunnar Rätsch
Australian National University
Research School for Information
Sciences and Engineering

Acknowledgements I thank **Gunnar Rätsch** for continuous support and A.Jagota, K.R. Müller and H.D. Burkhard for helpful advise. Furthermore I want to thank Hal Burgiss, Motoaki Kawanabe, Reiner Schulz, Sebastian Mika and Matthias Scholz for proof reading the draft and helpful discussions. I acknowledge the support from the IDA group at Fraunhofer FIRST, especially for the tolerance of letting me run the large number of jobs that were required to do the experiments.
This work would not have been possible without beeing supported by an award under the Merit Allocation Scheme of the National Facility of the Australian Partnerschip for Advanced Computing. The support from their staff, especially David Singleton is very much appreciated.
Last, but not least, I want to thank my small girlfriend Lydia for being the greatest.

To Lydia.

Zusammenfassung Bis heute stellt das Modellieren von Splice-Stellen in der Bioinformatik eine Herausforderung dar, da die Prozesse des Splicens noch nicht hinreichend geklärt sind. In der vorliegenden Arbeit wurden erfolgreiche *diskriminierende Lernmethoden* wie Support Vektor Maschinen (SVM) und *beschreibende Lernmethoden* wie Hidden Markov Modelle (HMM) kombiniert, um echte Splice-Stellen von falschen zu unterscheiden.

Diese bekannten Lernmethoden wurden mit neuen Kern-Funktionen wie dem TOP- und Fisher-Kern (FK), welche von generativen Modellen abgeleitet werden, kombiniert. Die Ergebnisse wurden mit denen des Locality Improved Kernel verglichen. Dabei ließ sich ein interessanter Zusammenhang mit dem Fisher-Kern herausfinden. Desweiteren erfolgte eine experimentelle Analyse von Splice-Stellen und verschiedene Lernmaschinen wurden hinsichtlich ihrer Klassifikationsqualität auf Splice-Stellen untersucht.

Abstract Modelling *splice sites* is considered a difficult task, and as of this writing, the procedure of splicing is still not well understood. We combine successful *discriminative learners* like Support Vector Machines (SVM) and *descriptive learners* like Hidden Markov Models (HMMs) to separate true splice sites from decoys.

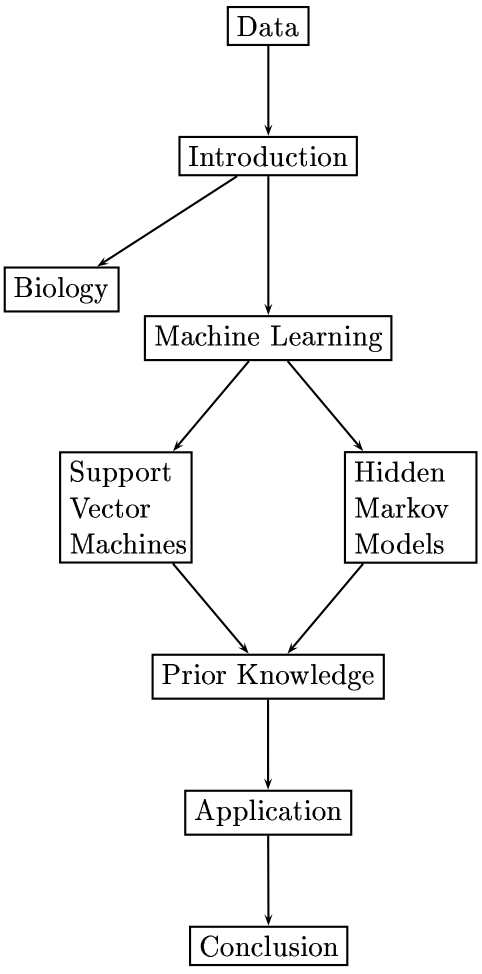
Recently developed kernel functions like the TOP- and Fisher Kernel (FK) that are derived from generative models are used to combine SVMs and HMMs. Furthermore, results for the well known Locality Improved Kernel are presented and its connection to the FK, derived from a special HMM is shown. Finally we provide an experimental analysis of splice sites and investigate the classification performance using a variety of learning machines.

Contents

1 Introduction

1.1 Overview

It was all over the press: The roundworm's DNA *C.elegans* is one of the first that has been fully sequenced! Human DNA has been sequenced! And more and more species's DNA is made publicly available. One has to ask "What can be done and what is actually done with that treasure."In principle the understanding of the "DNA construction kit" will in the near future allow people to cure genetic diseases, like cancer or trisomy 21 (Down Syndrome). However, still much work needs to be done to achieve this goal. At the moment, human DNA is not even fully sequenced. Still a number of gaps need to be filled. Researchers still do not know how many genes can be found in human DNA. The *active* structure of a large number of proteins, which are promotors of all kinds of chemical reactions in our body, are still unknown. Biologists seek to understand the underlying processes. This can be done via expensive biological experiments, or by analysing the huge available set of data. The latter forms the relatively new field of *bioinformatics.* In our work, we set for ourselves the task of modelling the process of *splicing* in eukaryotes like the roundworm *C.elegans* and later in the human by using recently developed machine learning techniques that are adapted to best suit this problem.



The graphics on the left provide an overview of this approach: The most important role plays the data. All machine learning techniques would not work without it, and even if enough data was available, they would still highly depend on carefully prepared data sets. We will describe the data sets we use in detail in Section 3.
The Introduction itself is divided into a biological part (cf. Section 1.2) and a machine learning part (cf. Section 2). We give a brief overview on genetics with the major focus on the *splicing* process. Thus readers already familiar with the biological details may want to skip this section. In the machine learning introduction, we first focus on well established methods in bioinformatics like *Hidden Markov Models* (cf. Section 2.2), and second discuss *Support Vector Machines* (cf. Section 2.3) which were introduced in 1995 and are known to produce excellent results on a variety of learning tasks.
We explain that both techniques use a very different approach in solving the standard machine learning problem of *classification.*
Since the framework of each of these machine learning techniques does not guarantee

good results per se, we explain in Section 2.4 how we ultimately incorporated prior knowledge about the splicing problem into Support Vector Machines by using special *kernels.* We compare the established *Locality Improved Kernel* [55] with the recently developed *Fisher* [22] and *TOP Kernel* [49]. The latter ones are derived from Hidden Markov Models, i.e., *prior knowledge incorporated into Hidden Markov Models and found by hmm learning is included into these kernels.*
We show that the framework of the Fisher and TOP Kernel is even more powerful than that of the Locality Improved Kernel.
Equipped with these methods, we first evaluate the performance of our methods (cf. Section 4) on a benchmark data set, where we can go beyond state of the art results. We then fine-tune Hidden Markov Models on real world splice site data samples of the roundworm *C. elegans*, again obtaining excellent results. In the end, we show preliminary results of some of our methods applied to human DNA.

1.2 Genetics

A brief history Almost 150 years ago, Mendel defined the basic nature of genes, starting a new branch in biology, *Genetics*. Back in 1944, Avery showed chemically (cf. Table 1.1) that *deoxyribonucleic acid (DNA)* is the genetic material [30]. Not much time passed between the discovery that genomes can be sequenced and the completion of sequencing the genome of the nematode *Caenorhabditis elegans* at the end of 1998, and the first draft sequence[1] of the human genome in June 2000.

1865	Genes are particulate factors
1903	Chromosomes are hereditary units
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
1945	A gene codes for a protein
1953	DNA is double helix
1958	DNA replicates semiconservatively
1961	Genetic code is triplet
1977	DNA can be sequenced
1997	Genomes can be sequenced
1998	Completion of the genome of the *Caenorhabditis Elegans*
2000	'Working Draft' of the human genome announced

Table 1.1: A brief history of genetics. This table is taken from [30] except for the last two entries.

Fundamentals 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 in whose cells nuclei and a cytoskeleton are present. 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 which 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 22 chromosome pairs, with pair 22 (according to standard enumeration) determining the sex, i.e., females have two X chromosomes, while males have one X and one Y chromosome.
Each chromosome is a pair of double-stranded sequences of DNA. A single-stranded DNA sequence consists of alternating series of pentose (sugar) and phosphate residues, and bases attached to it. Either one of the bases Adenine, Thymine, Guanine or Cytosine is linked to a sugar at position 1 (cf. Fig. 1.1). They are usually referred to by their initial letters. The name DNA is derived from the sugar-component in DNA (2-deoxyribose), which is the backbone of DNA. Metaphorically spoken, a double stranded DNA sequence looks like a twisted rope ladder. It forms a double helix, where one strand is the complement of the other, i.e., only the base pairs Adenine-Thymine and

[1]One can see the current status at `http://www.ncbi.nlm.nih.gov/genome/seq/`.
[2]To be correct, the DNA contained in chloroplasts, mitochondria and plasmids, does also belong to the genome.

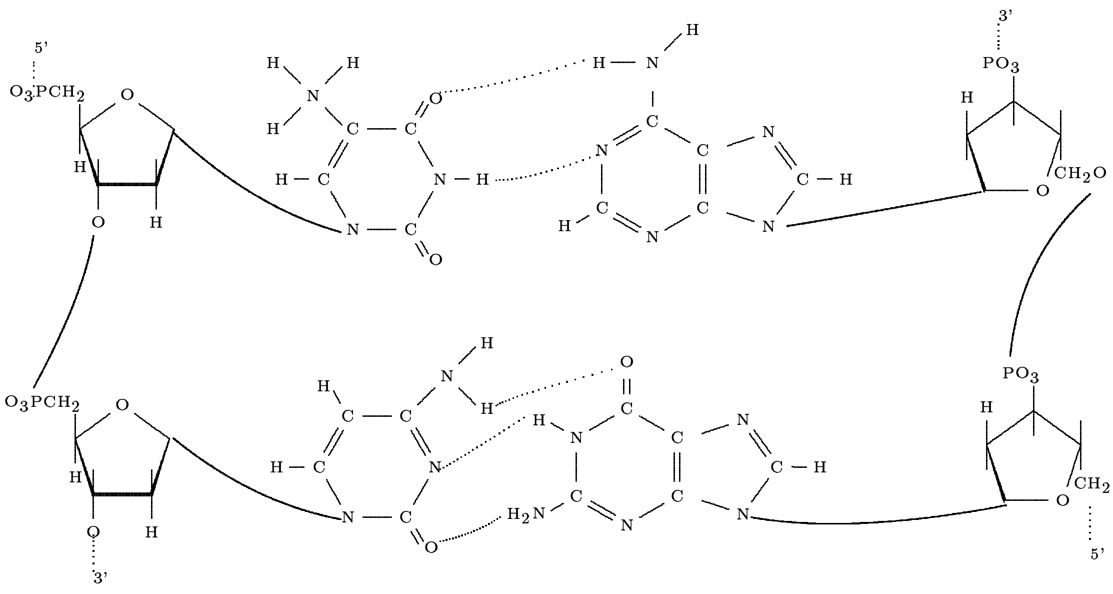


Figure 1.1: Explanation of the four bases adenine, cytosine, guanine, thymine and their pairing using hydrogen bonds. A sequence of only the two base pairs T-A (top) and C-G (bottom) is shown. On the left and right side one can see the sugar molecules, which are the backbone of DNA. The strands are antiparallel,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

Guanine-Cytosine may form the "steps" of the ladder. 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 bases are written in 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. Fig. 1.1.
Each chromosome consists of genetic and non-genetic regions, while the latter makes up most of the chromosome[3]. But where on the chromosome are the genes located and what is a gene?
A gene can be defined[4] as a region of DNA that controls a certain hereditary characteristic. It corresponds to a sequence used in the production of a specific protein.
While the question about the location of genes has not been sufficiently answered yet, it is known that a number of stages are involved in the expression of genes (i.e., the process of synthesising proteins from genes) [30] have been identified. These steps are carried out sequentially:

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

[3]The function of the non-genetic regions is mostly unknown at the time of writing.
[4]The definition of the term "gene" is still discussed, cf., `http://www.genomicglossaries.com/content/gene_def.asp`

It was discovered that genes are in an “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, the synthesis of a copy of the gene, which is encoded on only one of the two strands of the double-stranded DNA (the coding strand). 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 the enzyme

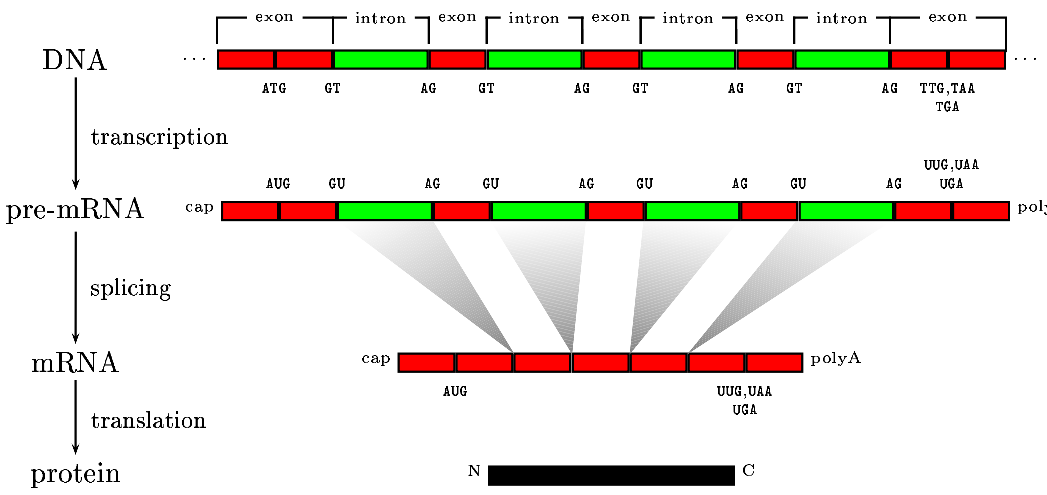


Figure 1.2: The major steps in protein synthesis. See text for details. Idea is taken from [13, 30]

‘RNA polymerase’ binds to the *promotor*, which is a special region located upstream[5] of the first base pair that is transcribed into pre-mRNA. From this *startpoint*, RNA polymerase moves *downstream* in $5' \rightarrow 3'$ direction, continuously synthesizing pre-mRNA until a terminator sequence is reached. In the ‘postprocessing’ step, the pre-mRNA is transformed into mRNA. One necessary step in the process of obtaining mature mRNA is called *Splicing*. The coding sequence of an eykaryotic gene is “interrupted” by non-coding regions called *introns*. A gene starts with an exon and is then interrupted by an *intron*, followed by another exon, intron etc. until it ends in an exon. In the splicing process, introns are removed through a lariat cf. Fig. 1.3. As a result, there are two

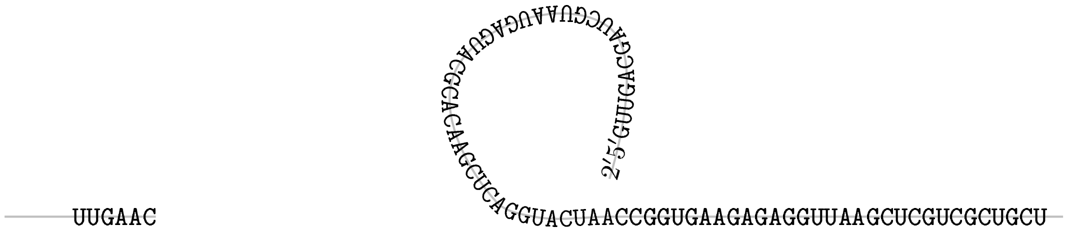

UUGAAC
2'5'GUUGACGAUCGUAAUGCAGUAGUACGGCACAGCUCAGGUACUAACCGGUGAAGAGAGGUUAAGCUCGUCGCUGCU

Figure 1.3: Illustration of the splicing process. First a cut is made at the 5’ site right before the GT. Then splicing proceeds through a lariat, in which the 5’ terminus generated at the end of the intron becomes linked by a 5’-2’ bond to a base within the intron. The target base is an A in a sequence that is called the branch site [30]. In the second stage, a cut is made at the 3’ site. The free intron is released and both exons are joined at their ends.

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

different splice sites. The exon-intron boundary, referred to as the donor site or 5' site and the intron-exon boundary, the acceptor or 3' site since it lies downstream of the acceptor site. Splice sites have strong consensus sequences, i.e., almost each position in a small window around the splice site is representative of the most frequently occurring nucleotide when many existing sequences are compared in an alignment. For example, the 5' site's consensus is

$$A_{64}G_{73}G_{100}T_{100}A_{62}A_{68}G_{84}T_{63}, \text{ while the 3' site's consensus is } C_{65}A_{100}G_{100},$$

where the subscripts denote the frequency of the symbol in percent [30]. Unfortunately the pairs `GT...AG` do occur very frequently, e.g., in human DNA (which is $\approx 3 \cdot 10^9$ base pairs in size) `GT` occurs about 1 billion times. For some crude estimate of say 10^5 genes with 10 exons each, only 0.1% of the possible splice sites are *real* splice sites. One can analogously estimate the number of occurences of `AG` which leads to a similiar result. Therefore it is not enough to look at pairs of `GT...AG`, evidently there is some intrinsic property around the splice site telling the *spliceosome*, i.e., the large biological splicing apparatus consisting of an array of proteins and ribonucleoproteins, to start the splicing mechanism. Neither is it known what this property is, nor what the details of the reaction involving RNA and the spliceosome are. While the *canonical splice sites* `GT...AG` make up the vast majority of splice sites, other possible combinations, like, e.g., `GT...TG` were discovered [10]. We will not take these *non-canonical* sites into account, which would make splice site detection even more difficult.

What is known about splicing?

- The splicing process takes place in the *nucleus.*
- An average mammalian gene has $7 - 8$ exons spread over $\approx 16kb$.
- Exons are relatively short, $100 - 200$ base pairs (bp) while introns are longer than $1kb$.
- There are no reading frames in introns.
- Splice sites are generic: they do not have a specificity for individual RNA precursors, and individual precursors do not convey specific information (such as secondary structure) that is needed for splicing.
- The apparatus for splicing is not tissue specific: RNA can usually be spliced properly by any cell, whether or not it is usually synthesised in that cell.
- Experiments show that any 5' splice site can in principle be connected to any 3' splice site, i.e., only *local* information is relevant in the splicing process.
- In *higher* eukaryotes, $18 - 40$ bp upstream of the 3' site, lies the branch site. To this site the `GU` from the 5' site connects to an `A` of the branch site.

Thus, the sequences needed for splicing are the short consensus sequences at the 5' and 3' splice sites and at the branch site. In higher eukaryotes mutations or deletions of the branch site result in a *proximate 3' site* to be taken. The branch site therefore identifies the 3' site to be used as the target for connection to the 5' site [30], but its removal does not prevent splicing.

After pre-mRNA has been spliced to form mRNA, the splicing product is transported from the nucleus to the cytoplasm. There, in the step of translation, mRNA is read as *codons*, i.e., as triplets of nucleotides. Hence, there are three different *reading frames*, 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 a gene. The translation begins almost always at the start codon AUG, called *translation initiation site* (TIS). However, only the minority of the codon AUG, which represents the amino acid methionine, really signals the translation initiation. SVMs have been successfully used to model this site [55]. When a stop codon is reached, translation is terminated and the set of amino acids, the result of the translation process, form a long polypeptide, the *protein*.

Data To gain some understanding about what the data looks like, we give some examples:
A fully sequenced genome consists of all the chromosomes found in a cell of the respective organism. Each sequenced chromosome is a sequence of the characters A,C,G,T, like that in Fig. 1.4.

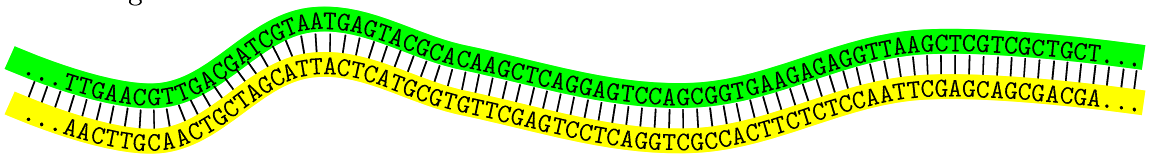

...TTGAACGTTGACGATCGTAATGAGTACGCACAAGCTCAGGAGTCCAGCGGTGAAGAGAGGTTAAGCTCGTCGCTGCT...
...AACTTGCAACTGCTAGCATTACTCATGCGTGTTCGAGTCCTCAGGTCGCCACTTCTCTCCAATTCGAGCAGCGACGA...

Figure 1.4: The two strands of DNA in an ASCII-character chain. As a result of a sequencing project, only one of these sequences is given, since Adenine is always connected to Thymine, and Guanine to Cytosine.

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.

```
AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT
CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC
TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC
TTGTTTTAATATTCAATTTTTTACAGTAAGTTGCCAATTCAATGTTCCAC
CTGTATTCAATCAATATAATTTTCAGAAACCACACATCACAATCATTGAA
TACCTAATTATGAAATTAAAATTCAGTGTGCTGATGGAAACGGAGAAGTC
```

Figure 1.5: An example that illustrates how splice site samples are constructed. Windows of fixed width are taken around the splice site, while AG is aligned to be at the same position in all samples. The left part including the AG is intronic, while the rest is exonic.

Why splice site detection is important A complete understanding of splice sites does not only help to *correctly* predict mRNA and thus proteins from DNA, but can also be of great help in localising genes. Actually several other sites, like start and stop codons, branch points, promotors and terminators of transcription and various transcription factor binding sites belonging to the class of *local sites* can help to detect genes [20]. In computational genefinding, these *signals* are often contrasted with variable length regions, like exons and introns. While the latter are recognised by methods called *content* sensors, the former can be recognised by, e.g., weight matrices, decision trees, etc., methods named *signal* sensors [20]. Three 5th-order markov chains are very common

content sensors for modelling eukaroytic introns and exons. Since signal and content sensors alone perform poorly, integrated models, such as Generalised Hidden Markov Models (GHMMs) [28] 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 promotor to the translation start is at least a certain number of bases away’, etc. can be incorporated into such models.
In the following, we focus on improving the signal sensor for the detection of splice sites. Since this detection uses only *local* information, this work is only useful when embedded in a genefinder that leaves the task of splice site detection to our proposed method.
While our splicing models were trained on the true sites, we chose to train “negative” models on decoys which were extracted from a small window around the true site (cf. Section 3). Hence, we have to make the assumption that sites the genefinder presents to our splice site detector, are either true splice sites or spots *close* to the true site. The performance guarantees as shown in Section 4 will only hold when one complies with this assumption.

2 Machine Learning

2.1 Introduction

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 [34]. In the simplest case of only two classes, one possible formalisation of this problem is to estimate a function (a classifier) $f : \mathbb{R}^D \rightarrow \{-1, +1\}$ that assigns a label $\hat{y} \in \{-1, +1\}$ to each sample $\mathbf{x} \in \mathbb{R}^D$. To estimate this function, a number of labeled training samples

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$$

are given, where the assumption is made that the pairs of training data are generated i.i.d.[6] according to an unknown probability distribution $\Pr[\mathbf{x}, y]$. One hopes that the classifier f performs well on unseen samples, 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

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

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 samples. There are several methods of estimation to accomplish this. They are called inductive principles [12]. 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 samples and used these to reason on some function f that seperates 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, which are not capable of describing the problem. This dilemma is explained best by an example. Consider[7] the task is to classify trees and non-trees. When a botanist with photographic memory has learned a number of trees, he will not realise that a newly presented tree is a tree, just because the number of leaves is different. On the other hand the botanist's lazy brother declares that if it is green it is a tree. It is therefore

[6]independent and identically distributed
[7]This example is taken from [7]

crucial to choose F appropriately.
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 $\Pr[y=+1|\boldsymbol{\theta}]$ directly. This approach is called *generative or descriptive learning*. We define $\boldsymbol{\theta} := (\boldsymbol{\theta}_+,\boldsymbol{\theta}_-,\alpha)$. Then when using the maximum likelihood (ML) inductive principle one has to choose an estimate for the parameters $\widehat{\boldsymbol{\theta}} \in \Theta'$ 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 [12]

$$\Pr[\mathbf{x}_{i_1},\ldots,\mathbf{x}_{i_{N_+}}|\widehat{\boldsymbol{\theta}}_+, y_{i_1}=+1,\ldots,y_{i_{N_+}}=+1] = \prod_{k=1}^{N_+}\Pr[\mathbf{x}_{i_k}|\widehat{\boldsymbol{\theta}}_+, y_{i_k}=+1] \text{ and}$$

$$\Pr[\mathbf{x}_{j_1},\ldots,\mathbf{x}_{j_{N_-}}|\widehat{\boldsymbol{\theta}}_-, y_{j_1}=-1,\ldots,y_{j_{N_-}}=-1] = \prod_{k=1}^{N_-}\Pr[\mathbf{x}_{j_k}|\widehat{\boldsymbol{\theta}}_-, y_{j_k}=-1],$$

are maximised independently. Here $y_{i_1},\ldots,y_{i_{N_+}}=+1$ and $y_{j_1},\ldots,y_{j_{N_-}}=-1$, and $\widehat{\boldsymbol{\theta}}$ denotes the estimate of the model parameters. The class prior can be estimated to be

$$\alpha = \Pr[y=+1|\boldsymbol{\theta}] = \frac{N_+}{N}.$$

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

$$f(\mathbf{x}) = \operatorname{sign}(\Pr[y=+1|\boldsymbol{\theta}^\star]\Pr[\mathbf{x}|y=+1,\boldsymbol{\theta}^\star] - \Pr[y=-1|\boldsymbol{\theta}^\star]\Pr[\mathbf{x}|y=-1,\boldsymbol{\theta}^\star]).$$

Furthermore one can create new examples by sampling from the class conditional distribution, and use this density function to define a metric or derive a kernel function to be used in discriminative classifiers, which creates an even more powerful decision function [22, 49].

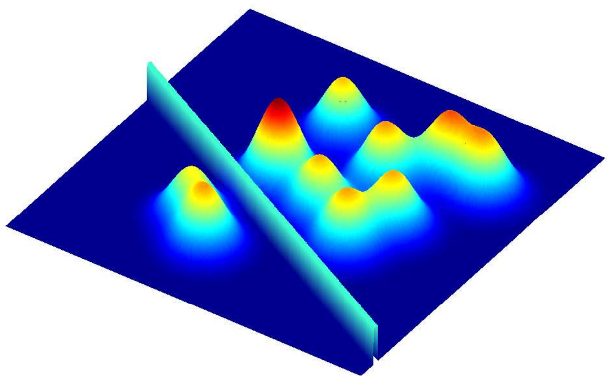


Figure 2.1: Two distributions separated by a plane. While the distribution on the left side is rather simple, the right class has a difficult structure.

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, the estimate can be very poor. Further on descriptive methods need to estimate the *whole* distribution, regardless of their complexity. See for example Fig. 2.1, 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 very complex. In such cases, discriminative learning techniques, which put special emphasise only on the samples close to the decision boundary can 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.

However it was shown that a hybrid classifier which combines the generative and the discriminative approach can, when *combined*, lead to even more powerful classifiers [22, 49].
We will now explain two representatives in more detail, the *Hidden Markov Model* as a representative of the generative model class and the *Support Vector Machine* for the discriminative models. Later on we introduce the kernel functions of Jaakola et al. [22] and Tsuda et. al. [49] which combine both methods.

2.2 Hidden Markov Models

Hidden Markov Models (HMM) are stochastic generative models that have been applied[8] successfully to, e.g., Speech Recognition [36] tasks and biological sequence analysis, such as gene finding and for identifying homologous sequences [17]. Its nature is to explain a system in a descriptive form [17].

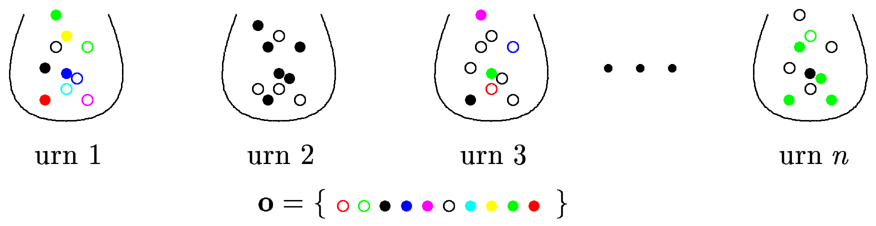


Figure 2.2: An urn and ball illustration of a HMM. Each urn corresponds to a specific hidden state of the HMM. Each urn has its own distribution of coloured balls, where the colour of a ball corresponds to some observed symbol. In an observation sequence generation process an urn is switched according to a urn-switch distribution associated with the urn. $\mathbf{o}$ is a possible observation sequence.

To give an impression of how a HMM works, we use *The Urn and Ball Model* as illustrated in Fig. 2.2. Consider a system with n urns and m different coloured balls. Within each urn there are a number of coloured balls. Now consider a genie who selects an initial urn according to some random process. From this urn a ball is chosen randomly, the colour is recorded as an observation and the ball is put back into the same urn. A new urn is chosen according to the random selection process underlying the current urn and the process of choosing a ball recording the colour as observation and selecting a new urn is repeated a finite number of times. In the end we get a finite observation sequence that consists of the colours of the balls we have chosen in this process.

For computer scientists it might be more convenient to think of 'probabilistic' Finite State Automata (see Fig. 2.3) with additional probabilities, i.e., the FSA starts in some state according to some initial state distribution, emits a symbol according to the distribution of symbols for that state, switches the state according to the probability distribution of that state and repeats this process finitely often.

Due to the fact that the state-switch and the emission of symbols is modelled as stochastic processes, we speak of a double stochastic process, where the state-switch is not directly observable - it is *hidden*.
We will now proceed with a more formal definition of a HMM.

Definition 2.1 (HMM). *A First Order Hidden Markov Model*

$$\boldsymbol{\theta}' = (\mathbf{a}, \mathbf{b}, \mathbf{p})$$

is defined by

- n, *the number of states*
- m, *the number of symbols (emissions, observations)*
- $\mathbf{z} = \{z_0, z_1, \ldots, z_{n-1}\}$, *the set of possible states of the model*

[8]See also Olivier Cappés web page `http://www-sig.enst.fr/~cappe/` on applications of HMMs.

e_j	$\Pr[e_j \mid z_0]$
e_0	b_{0,e_0}
e_1	b_{0,e_1}
$\dots$	$\dots$
e_{m-1}	$b_{0,e_{m-1}}$

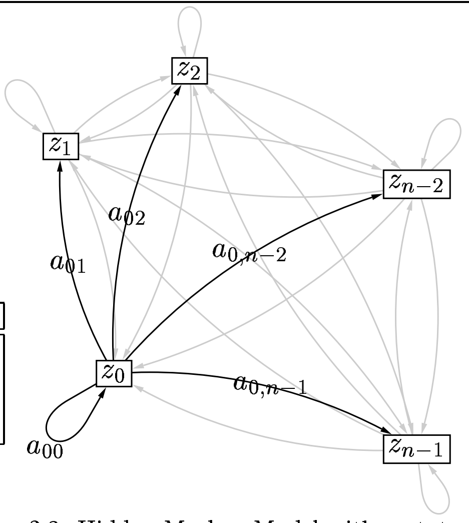


Figure 2.3: Hidden Markov Model with n states and m emissions

- $\mathbf{e} = \{e_0, e_1, \dots, e_{m-1}\}$, *the set of possible emissions (observations)*

- $$\mathbf{a} = \begin{pmatrix} a_{z_0,z_0} & a_{z_0,z_1} & \cdots & a_{z_0,z_{n-1}} \\ a_{z_1,z_0} & a_{z_1,z_1} & \cdots & a_{z_1,z_{n-1}} \\ \vdots & \vdots & \cdots & \vdots \\ a_{z_{n-1},z_0} & a_{z_{n-1},z_1} & \cdots & a_{z_{n-1},z_{n-1}} \end{pmatrix}$$

  transition matrix, i.e., a_{z_i,z_j} (in short a_{ij}) denotes the probability of a transition from state z_i to z_j.

- $$\mathbf{b} = \begin{pmatrix} b_{z_0,e_0} & b_{z_0,e_1} & \cdots & b_{z_0,e_{m-1}} \\ b_{z_1,e_0} & b_{z_1,e_1} & \cdots & b_{z_1,e_{m-1}} \\ \vdots & \vdots & \cdots & \vdots \\ b_{z_{n-1},e_0} & b_{z_{n-1},e_1} & \cdots & b_{z_{n-1},e_{m-1}} \end{pmatrix}$$

  emission matrix, i.e., b_{z_i,e_j} (in short b_{ij}) denotes the probability of emitting symbol e_j in state z_i

- $\mathbf{p} = (p_{z_0}, p_{z_1}, \dots, p_{z_{n-1}})$ *initial/start state distribution, i.e., p_{z_i} (in short p_i) denotes the probability of the HMM to start in state z_i (=the state at time 0)*

where the following conditions are satisfied:

i. ***stochasticity***[9]

$$\sum_{i=0}^{n-1} p_i = 1, \ \sum_{i=0}^{n-1} q_i = 1, \ \sum_{j=0}^{n-1} a_{ij} = 1, \ \sum_{j=0}^{m-1} b_{ij} = 1 \tag{2.1}$$

$$p_i, q_i, a_{ij}, b_{ij} \in [0,1]$$

That means, the variables are probabilities. For example the p's have to sum to 1 *since one wants the HMM to start in one of the n states with probability* 1.

[9]For those wondering what $\mathbf{q}$ stands for. It denotes the termination probability, which is introduced in Definition 2.3.

ii. ***first order property***

$$\begin{aligned}\Pr[s_{t+1} = z_j | s_t = z_i, s_{t-1} = z_k, \ldots, s_0 = z_l, \boldsymbol{\theta}'] &= \Pr[s_{t+1} = z_j | s_t = z_i, \boldsymbol{\theta}'] \\ &= a_{ij} \\ \Pr[o_t = e_j | s_t = z_i, s_{t-1} = z_k, \ldots, s_0 = z_l, \boldsymbol{\theta}'] &= \Pr[o_t = e_j | s_t = z_i, \boldsymbol{\theta}'] \\ &= b_{ij} \qquad (2.2)\end{aligned}$$

Thus the current event depends solely on the most recent past event.

iii. ***stationarity (time independence)***
a_{ij} and b_{ij} as defined here are time-independent, i.e., the HMMs parameters

$$\forall t: \; a_{ij} = \Pr[s_{t+1} = z_j | s_t = z_i], \; b_{ij} = \Pr[o_t = e_j | s_t = z_i] \qquad (2.3)$$

remain constant over time.

iv. ***independence of observations***
This is the assumption that the observation in the current state is statistically independent of the other observations in their corresponding state

$$\Pr[o_0, o_1, \ldots, o_{T-1} | s_0, s_1, \ldots, s_{T-1} \boldsymbol{\theta}'] = \prod_{i=0}^{T-1} \Pr[o_i | s_t, \boldsymbol{\theta}']. \qquad (2.4)$$

Definition 2.2 (Observation Sequence, State Sequence). *We define:*

- $\mathbf{o} = (o_0, o_1, \ldots, o_{T-1})$ *is called* observation sequence*, $o_t \in \mathbf{e}$ denotes the observation at time t*
- $\mathbf{s} = (s_0, s_1, \ldots, s_{T-1})$ *is called* state sequence*, $s_t \in \mathbf{z}$ denotes the state of the model at time t*
- *T is the length of the observation sequence*

The model starts in some state s_0 according to the start-state distribution, emits a symbol o_0 switches the state according to a_{s_0,s_1} and repeats this procedure up to s_{T-1} emitting o_{T-1}.
Since we also want to model the length of the sequence and specific end-states, we introduce the following extension to the original HMM $\boldsymbol{\theta}'$.

Definition 2.3 (HMM with end-state distribution). *We define*

$$\boldsymbol{\theta} = (\mathbf{a}, \mathbf{b}, \mathbf{p}, \mathbf{q}),$$

where $\mathbf{q} = (q_{z_0}, q_{z_1}, \ldots, q_{z_{n-1}})$ is the terminal/end-state distribution, i.e., q_{z_i} (in short q_i) denotes the probability of the HMM to terminate at time $T-1$ in state z_i, i.e., $q_i = \Pr[\boldsymbol{stop} | s_{T-1} = z_i, \boldsymbol{\theta}]$ and call it HMM with end-state distribution.

Remark 2.4. *The probability of paths shorter than $T-1$ observations is simply zero (Whenever the HMM terminated, it is unable to emit further symbols). Therefore it is sufficient to ask for the likelihood to terminate at time $T-1$ for the problems we deal with in the following sections.*

There are three problems of interest for HMMs that need to be solved for the model to make it useful in real world applications:

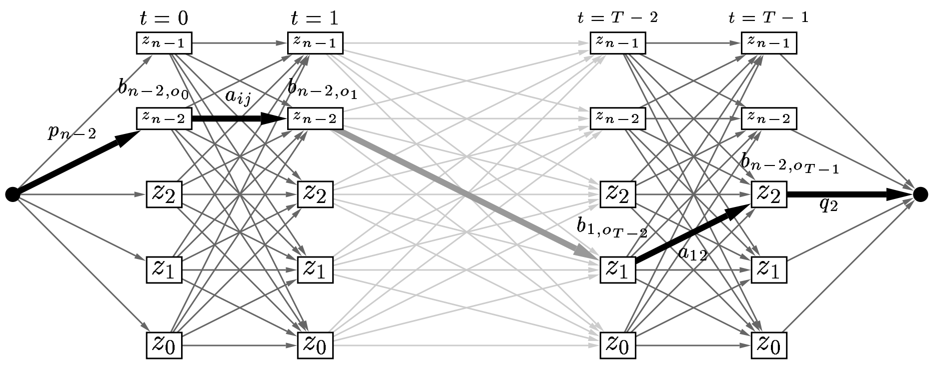


Figure 2.4: All n^T possible state sequences. The sequence $s = (z_{n-2}, z_{n-2}, \ldots, z_1, z_2)$ is shown highlighted.

Problem 1: What is the likelihood that a given HMM $\boldsymbol{\theta}$ generated an observation $\mathbf{o}$, i.e., the likelihood $\Pr[\mathbf{o}|\boldsymbol{\theta}]$ that the HMM generates the observation o_0 at time 0, o_1 at time 1 up to o_{T-1} at time $T-1$ considering all possible state sequences?

Problem 2: Given an observation and a HMM $\boldsymbol{\theta}$. Which is the most probable state sequence (path), i.e., the sequence that best describes observations?

Problem 3: We are given several observations. How do we find the HMM that best describes these observations, i.e., the model parameters that maximise $\Pr[\mathbf{o}|\boldsymbol{\theta}]$?

We are dealing with these problems in Section 2.2.1, 2.2.2 and 2.2.3.

2.2.1 Model Probability, Forward and Backward Variables

The Model Probability Consider we are given a finetuned HMM which is an estimate for a certain type of observations. When we are presented some observation $\mathbf{o}$, we might want to know whether $\mathbf{o}$ belongs to this special type of observations or not. We can use the model probability in our decision rule:

Lemma 2.5. *The probability of the model $\boldsymbol{\theta} = (\mathbf{a}, \mathbf{b}, \mathbf{p}, \mathbf{q})$, i.e., the probability that the observation sequence $\mathbf{o}$ was generated by the HMM that ends in s_{T-1}, is given by*

$$\Pr[\mathbf{o}|\boldsymbol{\theta}] = \sum_{\text{all } s_0,\ldots,s_{T-1}} p_{s_0} b_{s_0,o_0} \left(\prod_{t=0}^{T-2} a_{s_t,s_t+1} b_{s_{t+1},o_{t+1}} \right) q_{s_{T-1}}$$

Proof.

$$\begin{aligned}
\Pr[\mathbf{o}|\boldsymbol{\theta}] &= \sum_{\text{all } \mathbf{s}} \Pr[\mathbf{s}, \mathbf{o}|\boldsymbol{\theta}] \\
&= \sum_{\text{all } \mathbf{s}} \Pr[\mathbf{o}|\mathbf{s}, \boldsymbol{\theta}] \cdot \Pr[\mathbf{s}|\boldsymbol{\theta}]
\end{aligned}$$

Since

$$\begin{aligned}
\Pr[\mathbf{s}|\boldsymbol{\theta}] &= \Pr[s_{T-1}, s_{T-2}, \ldots, s_0|\boldsymbol{\theta}] = \frac{\Pr[s_{T-1}, s_{T-2}, \ldots, s_0, \boldsymbol{\theta}]}{\Pr[\boldsymbol{\theta}]} \\
&= \frac{\Pr[s_{T-1}|s_{T-2}, \ldots, s_0, \boldsymbol{\theta}] \cdot \Pr[s_{T-2}, \ldots, s_0, \boldsymbol{\theta}]}{\Pr[\boldsymbol{\theta}]} \\
&\overset{(2.2)}{=} \Pr[s_{T-1}|s_{T-2}, \boldsymbol{\theta}] \cdot \Pr[s_{T-2}, \ldots, s_0|\boldsymbol{\theta}]
\end{aligned}$$

we propose by induction

$$\Pr[\mathbf{s}|\boldsymbol{\theta}] = \Pr[s_0|\boldsymbol{\theta}] \cdot \prod_{t=0}^{T-2} \Pr[s_{t+1}|s_t, \boldsymbol{\theta}] = p_{s_0} \left(\prod_{t=0}^{T-2} a_{s_t, s_t+1} \right) q_{s_{T-1}}.$$

Considering that

$$\Pr[\mathbf{o}|\mathbf{s}, \boldsymbol{\theta}] \overset{(2.4)}{=} \prod_{t=0}^{T-1} \Pr[o_t|s_t, \ldots, s_0, \boldsymbol{\theta}] \overset{(2.2)}{=} \prod_{t=0}^{T-1} \Pr[o_t|s_t, \boldsymbol{\theta}] = \prod_{t=0}^{T-1} b_{s_t, o_t}$$

we conclude with

$$\begin{aligned}
\Pr[\mathbf{o}|\boldsymbol{\theta}] &= \sum_{\text{all } \mathbf{s}} \left(\prod_{t=0}^{T-1} b_{s_t, o_t} \right) \cdot p_{s_0} \left(\prod_{t=0}^{T-2} a_{s_t, s_t+1} \right) q_{s_{T-1}} \\
&= \sum_{\text{all } s} p_{s_0} b_{s_0, o_0} \left(\prod_{t=0}^{T-2} a_{s_t, s_t+1} b_{s_{t+1}, o_{t+1}} \right) q_{s_{T-1}}.
\end{aligned}$$

□

Forward and Backward Variables Since the forward and backward variables [17] are defined on the original HMM, we have to adjust the backward variables for our model $\boldsymbol{\theta}$ while the forward variables remain the same.

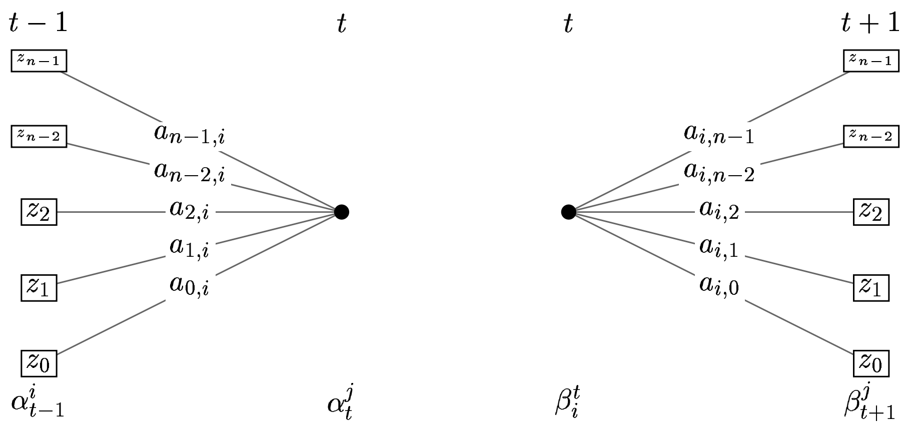


Figure 2.5: Illustration of the computation of the forward variables (left) and backward variables (right).

Definition 2.6 (Forward variables α_t^i).

$$\alpha_t^i := \Pr[o_0, o_1, \ldots, o_t, s_t = z_i|\boldsymbol{\theta}']$$

In words: α_t^i *is the probability to be at time* t *in state* z_i *and observe the sequence* $o_0, \dots, o_t$.

Definition 2.7 (Backward variables β_t^i).

$$\beta_t^i := \Pr[o_{t+1}, o_{t+2}, \dots, o_{T-1} | s_t = z_i, \boldsymbol{\theta}]$$

In words: β_t^i *is the probability to be at time* t *in state* z_i *and observe the sequence* $o_{t+1}, \dots, o_{T-1}$ *while terminating the sequence in state* s_{T-1}.

Lemma 2.8. *Using forward and backward variables the model probability can be efficiently computed using dynamic programming with Algorithms 2.1 and 2.2 with time complexity* $\mathcal{O}(n^2 \cdot T)$ *and space complexity* $\mathcal{O}(nT)$.

Proof.

Algorithm 2.1 Algorithm for calculating $\Pr[\mathbf{o}|\boldsymbol{\theta}]$ using forward variables

```
{Initialisation}
for $i = 0$ to $n - 1$ do
  $\alpha_0^i = p_i \cdot b_{o_0,i}$
end for

{Induction}
for $t = 0$ to $T - 1$ do
  for $j = 0$ to $n - 1$ do
    $\alpha_t^j = \sum_{i=0}^{n-1} \alpha_{t-1}^i \cdot a_{ij} \cdot b_{j,o_t}$
  end for
end for

{Termination}
$\Pr[\mathbf{o}|\boldsymbol{\theta}] = \sum_{i=0}^{n-1} \alpha_{T-1}^i \cdot q_i$
```

Algorithm 2.2 Algorithm for calculating $\Pr[o|\boldsymbol{\theta}]$ using backward variables

```
{Initialisation}
for $i = 0$ to $n - 1$ do
  $\beta_{T-1}^i = q_i$
end for

{Induction}
for $t = 0$ to $T - 2$ do
  for $i = 0$ to $n - 1$ do
    $\beta_t^i = \sum_{j=0}^{n-1} \beta_{t+1}^j \cdot a_{ij} \cdot b_{j,o_{t+1}}$
  end for
end for

{Termination}
$\Pr[\mathbf{o}|\boldsymbol{\theta}] = \sum_{i=0}^{n-1} \beta_0^i \cdot p_i b_{i,o_0}$
```

Since

$$\begin{aligned}
\Pr[\mathbf{o}|\boldsymbol{\theta}] &= \sum_{i=0}^{n-1} \Pr[s_0 = z_i, o_0, o_1, \ldots, o_{T-1}|\boldsymbol{\theta}] \\
&\overset{(2.2)}{=} \sum_{i=0}^{n-1} \Pr[o_0, s_0 = z_i|\boldsymbol{\theta}]\Pr[o_1, \ldots, o_{T-1}|s_0 = z_i, \boldsymbol{\theta}] \\
&= \sum_{i=0}^{N-1} p_i b_{i,o_0} \beta_0^i
\end{aligned}$$

and

$$\begin{aligned}
\Pr[\mathbf{o}|\boldsymbol{\theta}] &= \sum_{i=0}^{N-1} \Pr[o_0, o_1, \ldots, o_{T-1}, s_{T-1} = z_i, \textbf{terminate in } s_{T-1}|\boldsymbol{\theta}'] \\
&= \sum_{i=0}^{N-1} \alpha_{T-1}^i \cdot q_i
\end{aligned}$$

it is sufficient to show that the forward and backward variables are calculated properly while being in $\mathcal{O}(n^2 \cdot T)$:
Correctness of both algorithms follows from the simple fact that

$$\alpha_t^j = \sum_{i=0}^{n-1} \alpha_{t-1}^i \cdot a_{ij} b_{j,o_t} \text{ and } \beta_t^i = \sum_{j=0}^{n-1} \beta_{t+1}^j \cdot a_{ij} b_{j,o_{t+1}}$$

which becomes clear from Fig. 2.5. In both, Algorithm 2.1 and Algorithm 2.2, the first loop requires $\mathcal{O}(n)$, the second $\mathcal{O}(n^2 \cdot T)$ and last loop $\mathcal{O}(n)$ steps of computation – in total $\mathcal{O}(n^2 \cdot T)$. □

2.2.2 The Most Probable Path

We are often looking for the most probable state sequence for a given observation sequence, i.e., we are looking for $\mathbf{s}$ which maximises $\Pr[\mathbf{s}|\mathbf{o}, \boldsymbol{\theta}]$. However, since

$$\Pr[\mathbf{s}|\mathbf{o}, \boldsymbol{\theta}] = \frac{\Pr[\mathbf{s}, \mathbf{o}, \boldsymbol{\theta}]}{\Pr[\boldsymbol{\theta}]} \cdot \frac{\Pr[\boldsymbol{\theta}]}{\Pr[\mathbf{o}, \boldsymbol{\theta}]} = \frac{\Pr[\mathbf{s}, \mathbf{o}|\boldsymbol{\theta}]}{\Pr[\mathbf{o}|\boldsymbol{\theta}]} \propto \Pr[\mathbf{s}, \mathbf{o}|\boldsymbol{\theta}],$$

it is equivalent to maximising $\Pr[\mathbf{s}, \mathbf{o}|\boldsymbol{\theta}]$.

Lemma 2.9. *The probability of a state sequence* $\mathbf{s}$*, i.e., the probability that an observation* $\mathbf{o}$ *was generated by a so defined HMM while using only these particular states is given by*

$$\Pr[\mathbf{s}, \mathbf{o}|\boldsymbol{\theta}] = p_{s_0} b_{s_0,o_0} \left(\prod_{t=0}^{T-2} a_{s_t,s_{t+1}} b_{s_{t+1},o_{t+1}} \right) q_{s_{T-1}}$$

Proof. As in Lemma 2.5

$$\begin{aligned}
\Pr[\mathbf{s}, \mathbf{o}|\boldsymbol{\theta}] &= \Pr[\mathbf{o}|\mathbf{s}, \boldsymbol{\theta}] \cdot \Pr[\mathbf{s}|\boldsymbol{\theta}] \\
&= p_{s_0} b_{s_0,o_0} \left(\prod_{t=0}^{T-2} a_{s_t,s_t+1} b_{s_{t+1},o_{t+1}} \right) q_{s_{T-1}}
\end{aligned}$$

□

Algorithm 2.3 Viterbi - find most probable path

```
{Initialisation}
for $i = 0$ to $n-1$ do
  $\mu_0^i = p_i b_{i,o_0}$
  $\psi_0^i = 0$
end for

{Induction}
for $t = 1$ to $T-1$ do
  for $j = 0$ to $N-1$ do
    $\mu_t^j = \max_{0 \le i \le n-1} [\mu_{t-1}^i a_{ij}] b_{j,o_t}$
    $\psi_t^j = \operatorname{argmax}_{0 \le i \le n-1} [\mu_{t-1}^i a_{ij}]$
  end for
end for

{Termination}
$\max_{\mathbf{s}} \Pr[\mathbf{s}, \mathbf{o}|\boldsymbol{\theta}] = \max_{0 \le i \le n-1} [\mu_{T-1}^i q_i]$
$s_{T-1} = \operatorname{argmax}_{0 \le i \le n-1} [\mu_{T-1}^i q_i]$

{State sequence backtracking:}
for $t = T-2$ to $0$ do
  $s_t = \psi_{t+1}^{s_{t+1}}$
end for
```

Lemma 2.10. *The most probable path, i.e the state sequence $s_0, \dots, s_{T-1}$ with maximal probability*

$$\max_{s_0, s_1, \dots, s_{T-1}} \Pr[s_0, s_1, \dots, s_{T-1}, \mathbf{o}|\boldsymbol{\theta}]$$

can be found using the Viterbi algorithm requiring $\mathcal{O}(n^2 \cdot T)$.

Proof. We define

$$\mu_t^i = \max_{s_0, s_1, \dots, s_{t-1}} \Pr[s_0, s_1, \dots, s_t = z_i, o_0, o_1 \dots, o_t|\boldsymbol{\theta}], \tag{2.5}$$

i.e., the highest probability along a single path up to time t while observing only the first t symbols ending in state z_i.
Then Algorithm 2.3 calculates similarly to Algorithm 2.1 the most probable path. Here we exploit that $\mu_t^j = \max_i [\mu_{t-1}^i a_{ij}] \cdot b_{j,o_t}$ and use maximisations instead of sums. Unfortunately μ_t^i does not keep track of the argument that maximises Eq. (2.5). That is why we do this in an extra array ψ_t^i and retrieve the path in an additional backtracking step later. The effort for the first loop is $\mathcal{O}(n)$, recursion requires $\mathcal{O}(n^2 \cdot T)$, termination takes $\mathcal{O}(n)$ and backtracking requires $\mathcal{O}(T)$. □

2.2.3 Training a HMM

In the previous sections we assumed that $\boldsymbol{\theta}$ was given. Since we are usually given only some observations $\mathbf{O} := (\mathbf{o}_0, \mathbf{o}_1, \dots, \mathbf{o}_{N-1})^T$ we need to estimate the model parameters $\boldsymbol{\theta} = (\mathbf{a}, \mathbf{b}, \mathbf{p}, \mathbf{q})$, from the given observation "matrix"[10] $\mathbf{O}$, that "best" describe the

[10]Observations can be of different length. Thus $\mathbf{O}$ is not necessarily a matrix

observations, i.e., we want to find $\boldsymbol{\theta}$ which maximises $\Pr[\mathbf{O}|\boldsymbol{\theta}]$. Since there is no way of obtaining the optimal model parameters for the general case, we present two standard estimation-algorithms. We need more notation: By $\alpha_i^t(d)$, $\beta_i^t(d)$ we denote the forward/backward variables calculated using $\mathbf{o}_d$ whereas $b_{ij}(d) := b_{i,(\mathbf{o}_d)_j}$. The model probability is then given by

$$\Pr[\mathbf{O}|\boldsymbol{\theta}] = \prod_{d=0}^{N-1} \Pr[\mathbf{o}_d|\boldsymbol{\theta}]$$

(where we have assumed that observations sequences are independent from each other[11]) and can be easily obtained using forward and backward variables. The presented algorithms take a model[12] $\boldsymbol{\theta}$ as input and derive an estimate $\widehat{\boldsymbol{\theta}}$ from it. Note that if there are only a few observations some states might never be entered (or some symbols might never be emitted) potentially causing the model to converge in a local maximum far away from the optimum (it does overfit). To overcome this problem we introduce so called pseudo counts Π, i.e., we assume that a certain transition/emission is observed at least Π times. That is why it is called *pseudo* count. While it is in principle possible to incorporate prior knowledge using different pseudo counts per parameter, it is generally not available. As a result we restrict to one fixed pseudo value Π for all parameters, which can be fine tuned in model selection.

Baum Welch Training The Baum-Welch Algorithm [4] is an Expectation Maximisation Algorithm which has been proven to be a local optimisation technique [4]. However there might be numerous local maxima, such that convergence to a global extremum, the *critical point,* is not guaranteed.
Here $\widehat{\mathbf{p}}, \widehat{\mathbf{q}}, \widehat{\mathbf{a}}, \widehat{\mathbf{b}}$ denote the new estimates for the model parameters. The re-estimation formulas presented in Algorithm 2.4 can be derived from Baum's auxiliary function [4]

$$f(\boldsymbol{\theta}, \widehat{\boldsymbol{\theta}}) = \sum_f \Pr[f, \mathbf{O}|\boldsymbol{\theta}] \log(\Pr[f, \mathbf{O}|\widehat{\boldsymbol{\theta}}])$$

by applying the standard constrained optimisation Lagrange multiplier technique to $\widehat{\boldsymbol{\theta}}$. Note that in this formulation f is used as function and state sequence $f = \mathbf{s}$. It has been proven that

$$\max_{\widehat{\boldsymbol{\theta}}}[f(\boldsymbol{\theta}, \widehat{\boldsymbol{\theta}})] \to \Pr[\mathbf{O}, \widehat{\boldsymbol{\theta}}] \geq \Pr[\mathbf{O}, \boldsymbol{\theta}],$$

i.e., maximisation of $f(\boldsymbol{\theta}, \widehat{\boldsymbol{\theta}})$ results in increased model probability. As convergence criteria we monitor $|\Pr[\mathbf{O}|\widehat{\boldsymbol{\theta}}] - \Pr[\mathbf{O}|\boldsymbol{\theta}]|$. When it falls below a certain value or the re-estimation process takes longer than the maximum number of specified iterations the algorithm terminates.

Viterbi Training The computational costs for Algorithm 2.4 can be quite high (each iteration has an effort of $\mathcal{O}((N + M)N \cdot T)$). That is why Viterbi Training is sometimes used as an approximation. The re-estimation procedure is described in Algorithm 2.5. However this algorithm does not maximise $\Pr[\mathbf{O}|\boldsymbol{\theta}]$ but $\Pr[\mathbf{O}|\boldsymbol{\theta}, \mathbf{s}]$, where $\mathbf{s}$ is the maximum path derived in Algorithm 2.3.

[11]Observation sequences might even have different lengths T_d.
[12]The initial model is obtained by random initialisation of all model parameters.

Algorithm 2.4 Baum Welch Algorithm - estimate $\boldsymbol{\theta}$ from observations

```
Require: $\tau, \varepsilon$ { Termination conditions}
  {$\tau$ limits the number of iterations}
  {$\varepsilon$ is a limit on the improvements of $\Pr[\mathbf{O}|\boldsymbol{\theta}]$}

  {Initialisation}
  $\mathcal{I} := 0$ {Number of Iterations}
  $\widehat{\mathcal{P}} := \Pr[\mathbf{O}|\boldsymbol{\theta}]$ {Model Probability}

  repeat
    $\mathcal{P} = \widehat{\mathcal{P}}$
    for $i = 0$ to $n-1$ do
      {Estimation of $\mathbf{p}$}
      $\widehat{p_i} = \frac{1}{N + N\cdot\Pi} \cdot \left(\Pi + \sum_{d=0}^{N-1} \frac{\alpha_0^i(d)\cdot\beta_0^i(d)}{\Pr[\mathbf{o}_d|\boldsymbol{\theta}]}\right)$
      {Estimation of $\mathbf{q}$}
      $\widehat{q_i} = \frac{1}{N + N\cdot\Pi} \cdot \left(\Pi + \sum_{d=0}^{N-1} \frac{\alpha_{T_d-1}^i(d)\cdot\beta_{T_d-1}^i(d)}{\Pr[\mathbf{o}_d|\boldsymbol{\theta}]}\right)$
      {Estimation of $\mathbf{a}$}
      for $j = 0$ to $n-1$ do
        $\widehat{a_{ij}} = \frac{\Pi + \sum_{d=0}^{N-1} \frac{1}{\Pr[\mathbf{o}_d|\boldsymbol{\theta}]} \sum_{t=0}^{T_d-2} \alpha_t^i(d)\cdot a_{ij} b_{j,o_{t+1}(d)} \cdot \beta_{t+1}^j(d)}{N\cdot\Pi + \sum_{d=0}^{N-1} \frac{1}{\Pr[\mathbf{o}_d|\boldsymbol{\theta}]} \sum_{t=0}^{T_d-2} \alpha_t^i(d)\cdot\beta_t^i(d)}$
      end for
      {Estimation of $\mathbf{b}$}
      for $j = 0$ to $m-1$ do
        $\widehat{b_{ij}} = \frac{\Pi + \sum_{d=0}^{N-1} \frac{1}{\Pr[\mathbf{o}_d|\boldsymbol{\theta}]} \sum_{t=0,\ (\mathbf{o}_d)_t = e_j}^{T_d-1} \alpha_t^i(d)\cdot\beta_t^i(d)}{M\cdot\Pi + \sum_{d=0}^{N-1} \frac{1}{\Pr[\mathbf{o}_d|\boldsymbol{\theta}]} \sum_{t=0}^{T_d-1} \alpha_t^i(d)\cdot\beta_t^i(d)}$
      end for
    end for

    $\widehat{\mathcal{P}} := \Pr[\mathbf{O}|\widehat{\boldsymbol{\theta}}]$
    $\mathcal{I} := \mathcal{I} + 1$
  until $(\mathcal{I} > \tau)$ OR $\left(|\widehat{\mathcal{P}} - \mathcal{P}| < \varepsilon\right)$
```

2.2.4 Higher Order HMM

The descriptive power of first order HMMs can be limiting. For this reason one can extend the order of the underlying markov chain and define:

Definition 2.11. *A HMM is of Ω-th order when the underlying markov chain is of higher order*[13]*, i.e., transitions depend on the previous Ω states*

$$a_{(i_\Omega, i_{\Omega-1}, \ldots, i_1), j} = \Pr[s_{t+1} = z_j | s_t = z_{i_1}, s_{t-1} = z_{i_2}, \ldots, s_{t-\Omega+1} = z_{i_\Omega}].$$

[13]While not common in literature one could as well increase the order for observations, i.e., an observation depends on the previous Ω states not only the current one $b_{(i_\Omega, i_{\Omega-1}, \ldots, i_1), j} = \Pr[o_t = e_j | s_t = z_{i_1}, s_{t-1} = z_{i_2}, \ldots, s_{t-\Omega-1} = z_{i_\Omega}]$.

Algorithm 2.5 Viterbi Training- estimate $\boldsymbol{\theta}$ from observations

```
Require: $\tau, \varepsilon$ { Termination conditions}
  {$\tau$ limits the number of iterations}
  {$\varepsilon$ is a limit on the improvements of $\Pr[\mathbf{o}|\boldsymbol{\theta}]$}
  $\mathcal{I} := 0$ {Iterations}
  $\widehat{\mathcal{P}} := \overline{\Pr[\mathbf{s}, \mathbf{o}|\boldsymbol{\theta}]}$ {Average Viterbi Path Probability}
  $\boldsymbol{P} = \Pi \cdot \mathbf{1}$, $\boldsymbol{Q} = \Pi \cdot \mathbf{1}$
  $\boldsymbol{A} = \Pi \cdot \mathbf{1}_n$, $\boldsymbol{B} = \Pi \cdot \mathbf{1}_n$ {Absolute Counts}
  repeat
    $\mathcal{P} = \widehat{\mathcal{P}}$
    for $d = 0$ to $N - 1$ do
      Calculate Viterbi path $\mathbf{s}$ for $\mathbf{o}_d$
      for $i = 0$ to $n - 1$ do
        $P_i = P_i + \delta_{s_0, z_i}$ {obtain absolute counts for $P_i$}
        $Q_i = Q_i + \delta_{s_{T_d - 1}, z_i}$ {obtain absolute counts for $Q_i$}
        for $j = 0$ to $n - 1$ do
          $A_{ij} = A_{ij} + \sum_{t=0}^{T_d - 2} \delta_{s_t, z_i} \delta_{s_{t+1}, z_j}$ {obtain absolute counts for $A_{ij}$}
        end for
        for $j = 0$ to $m - 1$ do
          $B_{ij} = B_{ij} + \sum_{t=0}^{T_d - 2} \delta_{s_t, z_i} \delta_{(\mathbf{o}_d)_t, e_j}$ {obtain absolute counts for $B_{ij}$}
        end for
      end for
    end for
    for $i = 0$ to $n - 1$ do
      {Estimation of $\mathbf{p}$}
      $p_i = \frac{P_i}{N}$
      {Estimation of $\mathbf{q}$}
      $q_i := \frac{Q_i}{N}$
      {Estimation of $\mathbf{a}$}
      for $j = 0$ to $n - 1$ do
        $a_{ij} = \frac{A_{ij}}{\sum_{k=0}^{n-1} A_{ik}}$
      end for
      {Estimation of $\mathbf{b}$}
      for $j = 0$ to $m - 1$ do
        $b_{ij} = \frac{B_{ij}}{\sum_{k=0}^{m-1} B_{ik}}$
      end for
    end for

    $\widehat{\mathcal{P}} := \overline{\Pr[\mathbf{s}, \mathbf{o}|\widehat{\boldsymbol{\theta}}]}$
    $\mathcal{I} := \mathcal{I} + 1$
  until $(\mathcal{I} < \tau)$ AND $\left(|\widehat{\mathcal{P}} - \mathcal{P}| < \varepsilon\right)$
```

Lemma 2.12. *Higher order models can be mapped to first order models (cf. [17]) by mapping each of the high order states to new states z' of the order-th cross-product*

$$\mathbf{z}' \mapsto \mathbf{z} \times \mathbf{z} \times \cdots \times \mathbf{z} = \mathbf{z}^{\Omega}.$$

Proof. We define $z'_j = (z_j, z_{i_1}, z_{i_2}, \ldots, z_{i_{\Omega+1}})$ and $z'_i = (z_j, z_{i_1}, z_{i_2}, \ldots, z_{i_\Omega})$. Then the claim is easily proven by

$$\begin{aligned}\Pr[s_{t+1} = z'_j | s_t = z'_i] =& \\ &= \Pr[s_{t+1} = z_j, s_t = z_{i_1}, s_{t-1} = z_{i_2}, \dots, s_{t+\Omega+2} = z_{i_{\Omega+1}} | s_t = z_{i_1}, \\ &\qquad s_{t-1} = z_{i_2}, \dots, s_{t-\Omega+1} = z_{i_\Omega}] \\ &= \Pr[s_{t+1} = z_j | s_t = z_{i_1}, s_{t-1} = z_{i_2}, \dots, s_{t-\Omega+1} = z_{i_\Omega}],\end{aligned}$$

i.e., when the model has been in states $z_{i_1} \dots z_{i_\Omega}$ before, the joint probability of having been in the states $z_{i_1} \dots z_{i_{\Omega+1}}$ while going to state z_j is the same as the probability of switching to state z_j. One simply does not gain any information since the outcome was known before. □

While one can conveniently reuse first order HMM code, one should keep in mind that most of the algorithms have an effort of $n^2 \cdot T$, which becomes $\left(n^\Omega\right)^2 \cdot T$ making higher order HMMs intractable with this approach.
For this reason we propose to use higher order observations, i.e., we map the emissions to the cross product of emissions

$$\mathbf{e} \mapsto \underbrace{\mathbf{e} \times \mathbf{e} \times \cdots \times \mathbf{e}}_{\Omega} = \mathbf{e}^\Omega.$$

This way the number of states remains constant, but the number of observations grows exponentially. However, when the number of observations is small one can efficiently use higher order models using this "alphabet trick".

Definition 2.13 (Linear HMM). *A Linear HMM is a HMM where the underlying markov chain is linear: It consitsts of as many states as the observation sequence is long, i.e., $n = T$ while the transition parameters are fixed to*

- $p_0 = 1$,
- $\forall_{0<i<n-2} : \ a_{i,i+1} = 1$,
- $q_{n-1} = 1$

while all other transition parameters are set to zero.

Remark 2.14. *The only state sequence with non-zero probability is $z_0, z_1, \dots, z_{n-1}$, i.e., viterbi training (cf. Section 2.2.3) is equivalent to baum welch training in the case of linear HMMs. In practice, one would even skip the step of computing the most probable path (since it is known beforehand) and only count the frequencies of the symbols for each time t independently which is correct due to the assumption of independent observations (2.4). The obtained frequencies for symbol e_i and time t are then simply b_{tj}.*

Lemma 2.15. *A linear HMM $\boldsymbol{\theta} = (\mathbf{b})$ with m emissions and n states is equivalent to a Markov Model (MM) $\boldsymbol{\theta}' = (\mathbf{p}', \mathbf{a}', \mathbf{q}')$ of $m' \cdot n'$ states, where*

- $p'_i = b_{0,i}$,
- $q'_i = \delta_{i,m \cdot n-1}$,
- $\forall_{0 \le i \le n-2} : \ a'_{ij} = b_{i+1,j}$ *and*
- $a_{n-1,j} = \delta_{n-1,j}$

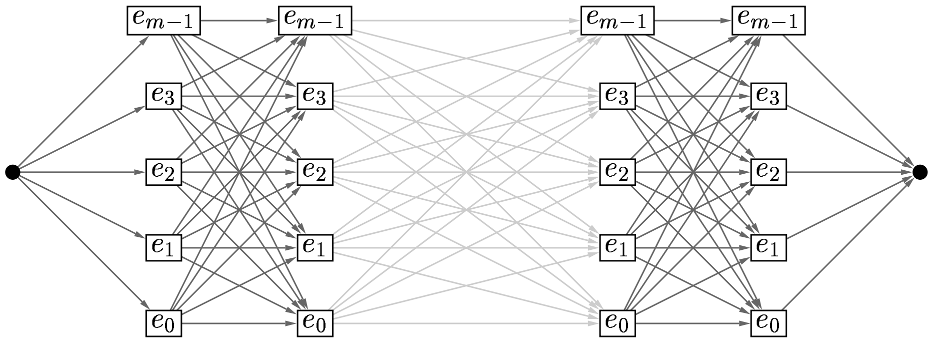


Figure 2.6: The markov chain corresponding to a linear HMM whose alphabet has been mapped to Ω-th order. For each observation we introduce a state, i.e., there are $M \cdot T$ states in the end.

as in Fig. 2.2.4. More generally, a linear first order HMM where the alphabet is mapped to be of Ω-th order is equivalent to a markov chain of Ω-th order.

Proof. Since transitions are allowed from layer to layer only, the model probability of the MM becomes

$$\Pr[\mathbf{s}'|\boldsymbol{\theta}'] = p'_{s_0} \prod_{t=\Omega-1}^{T-2} a'_{(s_{t-\Omega+1},\ldots,s_{t-1},s_t),s_{t+1}}\, q'_{s_{T-1}}$$

which is the same as the model probability of the first order HMM with "alphabet order" Ω

$$\Pr[\mathbf{o}|\boldsymbol{\theta}] = \prod_{t=\Omega-1}^{T-1} b_{s_t,(o_{t-\Omega+1},\ldots,o_{t-1},o_t)},$$

where the state sequence of the markov chain is in one to one correspondence to the observation sequence. We are therefore able to represent higher order MM using first order HMMs with higher order alphabet. However a linear HMM whose alphabet "is of order 1," is in fact a non-stationary model of zeroth order, since the observations are independent.[14] □

[14]It is often referred to as consensus or probability matrix, see e.g. [40]

2.3 Support Vector Machines

A Support Vector Machine (SVM) [6, 15, 41, 42, 51, 52] is a powerful learning machine. It is very competitive to neural networks and outperforms them on several learning tasks.[15] Suppose we are given l training samples. Each training sample consists of a pair: an input vector $\mathbf{x}_i \in \mathbb{R}^D$ and the output $y_i \in \{-1, +1\}$ which tells us about the true class belongings of the input (since we are dealing with supervised learning we know y_i for the training samples). Our learning task is to estimate a decision function $f : \mathbf{x} \mapsto \{-1, +1\}$ that predicts the label of any $\mathbf{x} \in \mathbb{R}^D$, i.e., the function separates the input $\mathbf{x}$ into two classes $+1$ and -1. What the SVM does is to construct such a decision function by using the training samples $(\mathbf{x}_i, y_i)_{i=0}^{N-1}$ of the form of a linear separating hyperplane:

$$f(\mathbf{x}) = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.6}$$

Hence, in the formulation of Section 2, one wants to find the classifier $f \in F$, where F is the set of linear classifiers, which minimises the risk functional.
So the learning task that remains is to find the normal weights $\mathbf{w}$ and the bias b.

2.3.1 The Separable Case

Here we assume that the training samples are separable by a linear hyperplane as defined in Eq. (2.6). Among the set of separating hyperplanes we choose the one that maximises the margin (for an explanation see Fig. 2.7) between the two classes such that [6]:

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 \text{ for } y_i = +1 \tag{2.7}$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 \text{ for } y_i = -1 \tag{2.8}$$

$$\Rightarrow y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq +1 \text{ for } y_i = \pm 1. \tag{2.9}$$

We denote H_1, H_2 as the hyperplanes that are parallel to the decision boundary $H = \mathbf{w} \cdot \mathbf{x} + b = 0$ and for which the condition (2.7) respectively (2.8) is sharp (such $\mathbf{x}_i$ exist for proper $\mathbf{w}, b$).

$$\begin{aligned} H_1 :\ & \mathbf{x}_i \cdot \mathbf{w} + b = +1 \\ H_2 :\ & \mathbf{x}_i \cdot \mathbf{w} + b = -1 \end{aligned}$$

The perpendicular distance between these hyperplanes is given by $\frac{2}{||\mathbf{w}||}$ and is called *margin*. It achieves a maximum for the minimal $\frac{||\mathbf{w}||}{2}$ that satisfies Eq. (2.7) and (2.8).

Why is a large margin desirable ?

Consider a decision function which achieves only a small margin between the two classes. Hence there is little scope left for perturbations on f and unseen samples $\mathbf{x}$. On the other hand when a classifier f achieves a large margin, it will be more robust in *patterns* as well as in *parameters*, i.e., slightly perturbed patterns $\mathbf{x}$ that were far away from the margin before, will still be given the same label. When one changes the parameters of f, i.e., $\mathbf{w}$ or b slightly, one would similiarly expect [45] the same label for examples far away from the decision boundary. For linear classifiers without bias, i.e., $b = 0$, it was

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

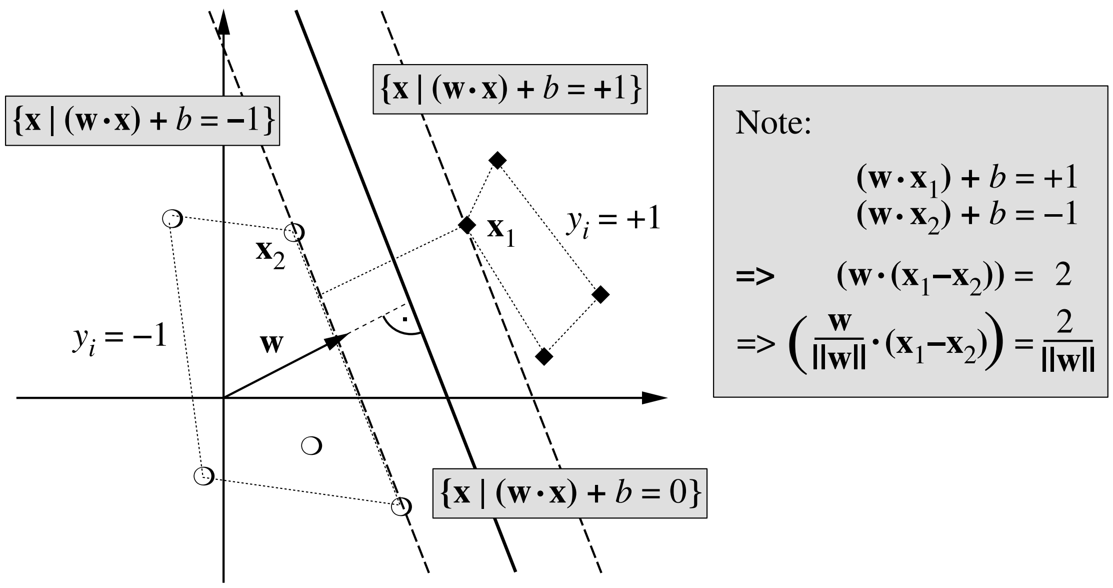


Figure 2.7: Illustration of the margin. On the left hand side you see the objects belonging to the class $y = -1$ which are separated by the linear function $\mathbf{w} \cdot \mathbf{x} + b = 0$ from the objects of the other class $y = +1$ on the right hand side. Within the dashed lines, which are corresponding to the hyperplanes H_1 and H_2, one finds the *margin* of size $\frac{||\mathbf{w}||}{2}$. The vectors lying on one of the dashed lines are called support vectors. This figure is taken from [45].

proven in [3] that the test error is in principle[16] bounded by the sum of the fraction of training samples lying within a certain margin ρ and a more complex term proportional to $\frac{R}{\rho}$, whereas the letter term decreas with the number of training samples (here R denotes the smallest radius of a sphere that contains all samples $\mathbf{x}$). Thus, the classifier that achieves maximum margin for a certain number of training samples is *guaranteed* to give the smallest test error.
It is therefore a good idea to choose the classifier with maximum margin among the separating functions.

To maximise the margin we have to minimise $\frac{1}{2}\|\mathbf{w}\|$ with respect to the constraints (2.9) which leads to the following convex quadratic optimisation problem [6]:

$$\begin{array}{ll} \text{minimise} & \frac{1}{2}||\mathbf{w}||^2 \\ \text{subject to} & y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0,\ i = 0, \ldots, N-1. \end{array}$$

To solve this we use the technique of Lagrange multipliers. Our constraints c_i are of the form $c_i \geq 0$. The rule is to introduce positive Lagrange multipliers λ_i, $(i = 0, \ldots, N-1)$ one for each constraint, multiply them with each constraint c_i and subtract them from the objective function, which gives the Lagrangian:

$$L_p \equiv \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=0}^{N-1} \lambda_i y_i(\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=0}^{N-1} \lambda_i \tag{2.10}$$

We now have to minimise L_p with respect to $\mathbf{w}$, b subject to $\frac{\partial L_p}{\partial \lambda_i} = 0$ and $\lambda_i \geq 0$. Since this optimisation problem is convex, we can equivalently solve the dual formulation L_D

[16]The exact inequality is given in [3] but we will leave aside the details and focus on the "message" of this theorem.

of the problem [7]:
Maximise L_D with respect to $\boldsymbol{\lambda}$ subject to $\frac{\partial L_D}{\partial \mathbf{w}} = 0$ and $\frac{\partial L_D}{\partial b} = 0$ and $\lambda_i \geq 0$. The solution of the so called "Wolfe's Dual" has the same solutions $\mathbf{w}$, b, $\boldsymbol{\lambda}$ like the primary minimisation problem L_p.

$$\frac{dL_D}{d\mathbf{w}} = 0 = \mathbf{w} - \sum_i \lambda_i y_i \mathbf{x}_i \qquad (2.11)$$
$$\frac{dL_D}{db} = 0 = \sum_i \lambda_i y_i$$

Plugging these conditions back into Eq. (2.10) we conclude in the dual objective:

$$\begin{aligned} L_D &= \frac{1}{2}\left\|\sum_i \lambda_i y_i \mathbf{x}_i\right\|^2 - \sum_i \lambda_i y_i \left(\mathbf{x}_i \cdot \sum_j \lambda_i y_i \mathbf{x}_i\right) + \sum_i \lambda_i \qquad (2.12) \\ &= \frac{1}{2}\sum_i \lambda_i y_i \mathbf{x}_i \cdot \sum_i \lambda_i y_i \mathbf{x}_i - \sum_i \lambda_i y_i \mathbf{x}_i \cdot \sum_j \lambda_i y_i \mathbf{x}_i + \sum_i \lambda_i \\ &= \sum_i \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j. \end{aligned}$$

The remaining task is to maximise L_D subject to $\lambda_i \geq 0$:

Karush Kuhn Tucker Conditions for the primal problem L_p The Karush Kuhn Tucker Condition [24, 27] as shown in the following, can in principle be derived for any constrained optimisation problem. They state that at a saddle point the derivatives of L_p with respect to the primal variables must vanish, i.e.,

$$\frac{\partial L_p}{\partial w_j} = w_j - \sum_{i=1}^{N} \lambda_i y_i \left(\mathbf{x}_i\right)_j = 0 \qquad j = 0, \ldots, N-1 \qquad (2.13)$$
$$\frac{\partial L_p}{\partial b} = -\sum_{i=1}^{N} \lambda_i y_i = 0 \qquad (2.14)$$

subject to the constraints

$$\begin{aligned} y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 &\geq 0 \qquad & i &= 0, \ldots, N-1 \\ \lambda_i &\geq 0 \qquad & i &= 0, \ldots, N-1. \end{aligned}$$

The Karush-Kuhn-Tucker complementary conditions state

$$\lambda_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1) = 0 \qquad i = 0, \ldots, N-1 \qquad (2.15)$$

and can be used to obtain b as will be explained in the next paragraph.

The examples corresponding to Lagrange Multipliers with $\lambda_i > 0$ are termed support vectors and lie either on one of the hyperplanes H_1 or H_2 as follows from Eq. (2.15) since for $\lambda_i > 0$ must hold $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 = 0$. Consider we have found λ_i which maximise L_D we get $\mathbf{w}$ from Eq. (2.11) (or equivalently Eq. (2.13)) and b from Eq. (2.15) for some

$\lambda_k \neq 0$. Hence the solution is given by

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i$$

$$b = \frac{1}{y_k} - \mathbf{x}_k \cdot \mathbf{w} = y_k - \mathbf{x}_k \cdot \left(\sum_{i=0}^{N-1} \lambda_i y_i \mathbf{x}_i \right)$$

and we finally get the resulting decision function

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=0}^{N-1} \lambda_i y_i \mathbf{x}_i \cdot \mathbf{x} + b \right) = \operatorname{sign}\left(\left(\sum_{i=0}^{N-1} \lambda_i y_i \mathbf{x}_i \cdot (\mathbf{x} - \mathbf{x}_k) \right) + \frac{1}{y_k} \right). \tag{2.16}$$

2.3.2 The Non-Separable Case

Since most of the problems are not separable by some hyperplane the so called softmargin was introduced [15]. For that case we define slack variables ξ_i and reformulate the problem as follows

$$\begin{aligned} \mathbf{x}_i \cdot \mathbf{w} + b &\geq +1 - \xi_i, \text{ for } y_i = 1 \\ \mathbf{x}_i \cdot \mathbf{w} + b &\leq -1 + \xi_i, \text{ for } y_i = -1 \\ \xi_i &\geq 0 \qquad \forall i. \end{aligned}$$

Note that $\left(\sum_i \xi_i\right)^k$ is an upper bound on the training error.

The new quadratic programming problem is:

Minimise $\frac{||\mathbf{w}||^2}{2} + C\left(\sum_i \xi_i\right)^k$, where C is some constant regularising term (the higher C the more get training errors punished).

We choose $k = 1$ and formulate the "Wolfe's Dual":

Maximise

$$L_D = \sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{2.17}$$

subject to $0 \leq \lambda_i \leq C$ and $\sum_i \lambda_i y_i = 0$. Note that the dual problem differs only in the additional constraints on λ_i which are now upper bounded by C.
Thus the solution is again $\mathbf{w} = \sum_{i=0}^{N-1} \lambda_i y_i \mathbf{x}_i$.

2.3.3 Kernel Functions

Up to that point all the SVM does is to construct a separating hyperplane with maximum margin *in input space.* Since a linear classifier might not be sufficient for more complex problems, we introduce a mapping $\Phi : \mathbb{R}^D \rightarrow F$ which nonlinearly[17] maps an input vector $\mathbf{x}$ into some higher dimensional feature space F [1, 6]. Since in the training process of the SVM and in the 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 introduce a kernel function

[17] F might be infinite dimensional

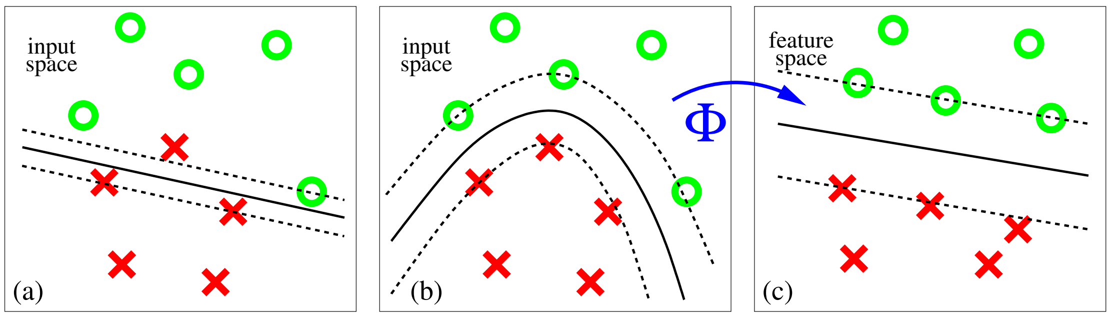


Figure 2.8: (a) a linear separation of training points is not possible without errors. (b) a nonlinear decision function is suggested. (c) the nonlinear function corresponds to a linear function in feature space. This picture is taken from [54].

$K(\mathbf{x}, \mathbf{x}') = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}'))$ that replaces the dot product, i.e., Eq. (2.17) changes to

$$L_D = \sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

and the decision function (Eq. (2.16)) becomes

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=0}^{N-1} \lambda_i y_i \left(\Phi(\mathbf{x})_i \cdot \Phi(\mathbf{x})\right) + b \right) = \operatorname{sign}\left(\sum_{i=0}^{N-1} \lambda_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \right).$$

By using a kernel function one skips the step of first calculating $\Phi(\mathbf{x})$ and instead uses the "kernel trick," i.e., one computes the result directly in input space. It was shown that all symmetric squared integrable functions that satisfy Mercer's theorem [32] are valid kernel functions, for details cf. [1, 6, 7, 15, 34, 39]. To see which functions can be kernel functions see [7, 15, 22, 41].

Example 2.16. *We give examples for kernel functions [15]:*

1. *The Polynomial kernel:*

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &:= (\mathbf{x} \cdot \mathbf{x}')^d \\ &= \left(\sum_{i=0}^{D-1} x_i x_i' \right)^d \end{aligned}$$

2. *The Gaussian RBF kernel:*

$$k(\mathbf{x}, \mathbf{x}') := \mathbf{e}^{-\left(\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{\sigma^2}\right)}$$

The choice of a good kernel function is crucial, e.g., the splice site recognition task solved with an unsuitable kernel (like the RBF kernel [38]) performs as bad as 10% (see Section 4 for details). We get much better results when using one of the kernels introduced in Section 2.4.

2.4 Improved Kernels for Splice Site Recognition

2.4.1 The Locality Improved Kernel

The Locality Improved Kernel 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 [55]. In all these experiments the parameter d_2 (see below) which is related to global information between the windows, was set to 1, since other values decreased the SVM performance, i.e., only local information was used.

Definition 2.17. *The Locality Improved Kernel [54, 55] (LIK) for two sequences* $\mathbf{x}$ *and* $\mathbf{x}'$ *is defined as*

$$K(\mathbf{x},\mathbf{x}') = \left(\sum_{t=0}^{T-1} \left(\sum_{j=-l}^{+l} w_j I_{t+j}(\mathbf{x},\mathbf{x}') \right)^{d_1} \right)^{d_2},$$

where

$$I_t(\mathbf{x},\mathbf{x}') = \begin{cases} 1, & x_t = x'_t \\ 0, & \textit{otherwise} \end{cases}$$

Comparing this to the polynomial kernel where $\mathbf{x}$ and $\mathbf{x}'$ are binary vectors[18]

$$K(\mathbf{x},\mathbf{x}') = (\mathbf{x}\cdot\mathbf{x}')^d = \left(\sum_i x_i \cdot x'_i\right)^d = \left(\sum_i I_i(\mathbf{x},\mathbf{x}')\right)^d$$

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 Fig. 2.9), which is illustrated as computing the binary dot product (.,.). Then these are taken to the power of d_1. As a 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.

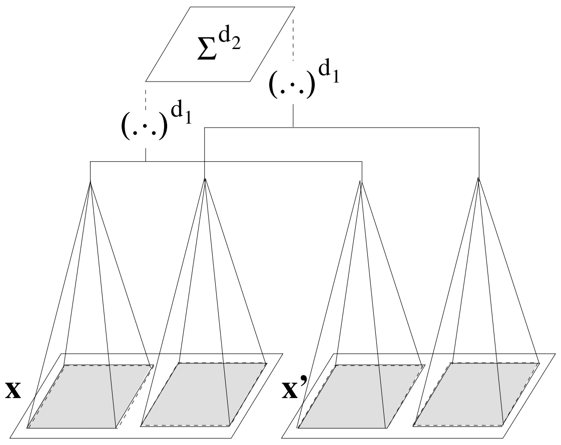


Figure 2.9: The principle of how the Locality Improved Kernel works. The sequences $\mathbf{x}$ and $\mathbf{x}'$ consist of only two symbols for simplicity. See text for explanation. This picture is based on [43].

[18]The binary representation of an observation vector $\mathbf{o}$ is the “indicator vector” $\mathbf{x}$ which contains $m \cdot T$ entries of values $x_i \in \{0,1\}$, where $(x_{t\cdot m},\ldots,x_{(t+1)\cdot m}) = (\delta_{e_0,o_t},\ldots,\delta_{e_{m-1},o_t})$.

2.4.2 The Fisher Kernel

In comparison to other kernels, the Fisher Kernel (FK) and TOP Kernel (which will be discussed later), are derived from a probabilistic model. They are guaranteed to increase performance if, and only if, the true distribution can be parametrised by the probabilistic model. Other kernels, like the LIK for example make "only" use of certain structures and are infeririour when the underlying model *is good enough.*

The Fisher Kernel [21, 22] is defined on some generative model $\Pr[\mathbf{x}|\boldsymbol{\theta}]$ (like HMMs). It is derived from the marginal distribution

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

Definition 2.18. *The Fisher kernel is defined as*

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

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* $\mathbf{Z}(\boldsymbol{\theta}) = \mathrm{E}_{\mathbf{x}}\left[\mathbf{s}_{FK}(\mathbf{x}, \boldsymbol{\theta}) \mathbf{s}_{FK}(\mathbf{x}, \boldsymbol{\theta})^\top \middle| \boldsymbol{\theta}\right]$, *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 *[43].*

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 matches), the FK can deal with sequences of arbitrary length, making it the kernel of choice in a number of applications like speech recognition and DNA analysis.
Recently [49, 50] the TOP kernel was derived and it was shown to outperform the FK on certain tasks. We will deal with it in the next subsection.

The Fisher Kernel from HMM We are now going to derive the Fisher Score from a mixture model of HMMs

$$\Pr[\mathbf{x}|\boldsymbol{\theta}] = \alpha \Pr[\mathbf{x}|\boldsymbol{\theta}, y = +1] + (1 - \alpha) \Pr[\mathbf{x}|\boldsymbol{\theta}, y = -1].$$

When no negative model $\Pr[\mathbf{x}|\boldsymbol{\theta}, y = -1]$ is available one can do without, by just setting $\Pr[\mathbf{x}|\boldsymbol{\theta}, y = -1] = c$ to some constant c. The parameter α is the prior of the class conditional distribution $\Pr[\mathbf{x}|\boldsymbol{\theta}, y = +1]$, i.e., $\alpha = \Pr[y = +1|\boldsymbol{\theta}]$ which can be tuned using a binary search in the interval $[0, 1]$. The partial derivatives for α are given by:

$$\begin{aligned}
\frac{\partial \log \Pr[\mathbf{x}|\boldsymbol{\theta}]}{\partial \alpha} &= \frac{\partial \log(\alpha \Pr[\mathbf{x}|\boldsymbol{\theta}, y = +1] + (1 - \alpha) \Pr[\mathbf{x}|\boldsymbol{\theta}, y = -1])}{\partial \alpha} \\
&= \frac{\Pr[\mathbf{x}|\boldsymbol{\theta}, y = +1] - \Pr[\mathbf{x}|\boldsymbol{\theta}, y = -1])}{\alpha \Pr[\mathbf{x}|\boldsymbol{\theta}, y = +1] + (1 - \alpha) \Pr[\mathbf{x}|\boldsymbol{\theta}, y = -1]} \\
&= \frac{1}{\Pr[\mathbf{x}|\boldsymbol{\theta}]} (\Pr[\mathbf{x}|\boldsymbol{\theta}, y = +1] - \Pr[\mathbf{x}|\boldsymbol{\theta}, y = -1])
\end{aligned}$$

Together with the partial derivatives for θ_i they make up the Fisher score:

$$\begin{aligned}
&\frac{\partial \log \Pr[\mathbf{x}|\boldsymbol{\theta}]}{\partial \theta_i} \\
&= \frac{\partial \log(\alpha \Pr[\mathbf{x}|\boldsymbol{\theta}, y=+1] + (1-\alpha)\Pr[\mathbf{x}|\boldsymbol{\theta}, y=-1])}{\partial \theta_i} \\
&= \frac{1}{\Pr[\mathbf{x}|\boldsymbol{\theta}]}\left(\alpha \frac{\partial \Pr[\mathbf{x}|y=+1,\boldsymbol{\theta}]}{\partial \theta_i} + (1-\alpha)\frac{\partial \Pr[\mathbf{x}|y=-1,\boldsymbol{\theta}]}{\partial \theta_i}\right)
\end{aligned}$$

Notice that one of the addends in the last sum is always 0 when the set of parameters in the positive model is disjunctive from the set of parameters in the negative model, which is usually the case. The model probability $\Pr[\mathbf{x}|\boldsymbol{\theta}]$ and the partial derivatives $\partial_i \Pr[\mathbf{x}|\boldsymbol{\theta}, y=\pm 1]$ are derived in Section 2.2.1 and Section A.1 respectively.

2.4.3 The TOP Kernel

Consider we have trained two generative models, one for positive samples $\Pr[\mathbf{x}|\boldsymbol{\theta}_+]$ and one for the negative class $\Pr[\mathbf{x}|\boldsymbol{\theta}_-]$.[19]
Recall the Bayes decision for the true model $\Pr[\mathbf{x}, y|\boldsymbol{\theta}^\star]$:

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

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

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

[19]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 $c \in \{+1, -1\}$.

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 2.19. *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*

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

is called TOP-Kernel *[49].*

The TOP Kernel, while applicable to the same type of problems, empirically outperforms the FK as observed in [44, 50]. Furthermore it was shown that the convergence rate when combined with an *optimal* linear classifier (which is not available in practice) is $\mathcal{O}(\frac{1}{N})$. This shows the existence of a very efficient linear boundary in the TOP feature space. However estimators of linear classifiers typically have a slower convergence rate of $\mathcal{O}(N^{1/2})$ [29], leaving room for improvement by doing, e.g., careful model selection.

The TOP Kernel from HMM Similiar to the FK, the model probability $\Pr[\mathbf{x}|\widehat{\boldsymbol{\theta}}]$ as required in the first component of the score vector $\mathbf{s}_{TOP}(\widehat{\boldsymbol{\theta}},\mathbf{x})$ and the partial derivatives of the model probability were derived in Section 2.2.1 and Section A.1 respectively.

How to use FK/TOP from HMM We summarise the steps required to use the Fisher or the TOP kernel on HMMs:

1. Train a HMM $\boldsymbol{\theta}_+$ on the positive examples
2. Train a HMM $\boldsymbol{\theta}_-$ on the negative examples
3. In the FK case, one has to tune α which is not required in the TOP kernel case, since it is implicitly obtained by the linear estimator.
4. Compute the score vectors and train linear SVM in this new feature space or compute SVM using the kernel.

Note that computing the model probability and the derivatives of the HMMs requires $\mathcal{O}(n^2 \cdot T)$, for each computation of the kernel. It is therefore inevitable to use *caches.* While we use them for forward and backward variables or the explicitly computed features our implementation also makes use of kernel caches.

2.4.4 Connection to the Locality Improved Kernel

Intuitively there is some connection between the FK and the LIK. In this section we will workout this relation more carefully.

Lemma 2.20. *For given l, d_1, d_2 there exists a* Markov Model *such that the Fisher Kernel using a diagonal matrix $\mathbf{Z} = c \cdot \mathbf{1}_D$ derived from this Markov Model is equivalent to the Locality Improved Kernel, i.e., $K_{LIK}(\mathbf{x}, \mathbf{x}') = K_{Fisher}(\mathbf{x}, \mathbf{x}')$, $\forall \mathbf{x}, \mathbf{x}'$, where c is some constant scaling factor and D is the number of parameters of the Markov Model.*

A sketch of the proof, supported by the simplest non trivial example, is shown in the appendix. Hence we can concentrate on the interpretation of this lemma. Actually when Jaakola and Haussler introduced the FK in [22], they used a constant MM in an experiment (eq. (10) in their paper), which is almost the same as using the LIK where the sum over l is dropped, i.e., only one term is taken and the remaining parameters are set to $d_1 = 1$, and $d_2 = 1$. In this case the LIK does no more than counting matches, which again is no more than the conversion of a sequence into a binary vector and then computing the dot product of these two sequences. The new idea in the LIK is to make use of local correlations which are taken into account for $l \geq 1$, $d_1 > 1$. Equivalently one could in principal consider Ω observations as *one* observation and then compute a (although high dimensional) binary vector from that. However the LIK does this job implicitly.
As a result we expect better performance for SVMs using the FK on linear and more sophisticated HMMs.

3 Data

3.1 Caenorhabditis Elegans

3.1.1 Introduction

The *Caenorhabditis elegans* [18][33] (Greek. *kaino* recent; *rhabdos* rod; Latin *elegans* elegant) is a nematode. These are unsegmented worms with elongated, rounded bodies, and are pointed at both ends, which are mostly free-living although some are parasitic, and can be both aquatic and terrestrial.
C. elegans is very small, growing to about 1mm in length.It feeds on microbes such as bacteria.

Figure 3.1: A picture of the 1mm long nematode *C. elegans* drawn to a scale of 1:140.

Judging from these facts the question arises why a large number of scientists are studying *C. elegans*, although it seems pretty insignificant to man.
Belonging to the *eucaryotes* (as humans do) this nematode shares many of the essential characteristics that are central problems of human biology. The worm undergoes a complex process of development, starting from a single fertilised egg, proceeding through morphogenesis and growth to the 959-cell adult. There a two sexes, a hermaphrodite which makes up 99.95% of the population and a male. Its DNA consists of approximately 97 million base-pairs encoding an estimated number of 17,800 distinct genes. Seemingly insignificant, it has a nervous system (about 300 cells are neurons) and a sensory system equipped with sensors for taste, smell, temperature. It is even capable of rudimentary learning and although *C. elegans* has no eyes, it might respond slightly to light exposure. In its life span of 2-3 weeks, it produces sperm and eggs, mates and reproduces.
The almost ideal compromise between complexity and tractability made the South African biologist Sydney Brenner start the *C. elegans* project back in 1963. In the experimental setup the worms were usually grown on petri plates seeded with bacteria. At the end of 1998 the whole genome of this worm has been sequenced. It was discovered that *C. elegans* is diploid, i.e., cells contain a pair of homologous chromosomes and that each cell has five pairs of non-sex chromosomes (I-V) and a pair of sex chromosomes (X) [33]. Worms with two X chromosomes develop as hermaphrodites, while those with a single X chromosome become males.
By studying the worm scientists hope to extrapolate their lessons to humans. They have already learned crucial lessons about,e.g., Alzheimer, aging [48] and cancer [16].
The task before us is the modelling the splicing process. Since the worms DNA is pretty well understood, the number of labelled examples is very high. This gives a good starting point for analysing the performance of different machine learning methods.

3.1.2 Datasets

The data was extracted from the chromosome and GFF files at `http://genome.wustl.edu/gsc/C_elegans`. This dataset is expressed sequence tag (EST) supported. ESTs are small pieces of mRNA normally of $<$ 700 nucleotides in length and obtained after

only one round[20] of sequencing [9]. 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 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).
We only consider *canonical* splice sites, i.e., splice sites that contain canonical dinucleotides `GT` and `AG` for donor and acceptor sites. These make up the vast majority of splice sites, e.g., 98.71% of the *mammalian* GenBank EST supported splice junction pairs are canonical while the fraction of canonical splice sites *might* be as high as 99.24% [9].

```
0          25                          49-50                               75           100
CT...GTAGAGC| TGTACGTACGACGTACGTCAAGCT AG GAGCGCAACGTACGTACGTCCAGT |AGAAGGT...G A
```

Figure 3.2: A window around a true splice site of length 101 base-pairs. The first 51 bases are intron, the next 50 bases, starting with `GA`, exon. The extracted positive example is shown framed.

Acceptor site Windows of -51 to +50 around the true splice site were extracted. From these windows, we extracted smaller windows of length 50 (c.f. Fig. 3.2) and removed duplicates. For the true site's windows this leads to 74, 455 positive acceptor examples.

```
0             27                          52-53                               78         100
CT...GTAGAGCTGT| ACGTACGACGTACGTCAAGCTAGG AG CGCAACGTACGTACGTCCAGTAGA |AAGGT...G A
```

Figure 3.3: Extraction of a 50 base-pair decoy window, by shifting the window 3 positions downstream to the next occurence of `AG`. All decoys are constructed such that `AG` is at positions $25 - 26$ too.

To construct negative examples we extracted all windows of length 50 that seem to be canonical splice sites, i.e contain `AG` at positions $24 - 25$ from these 101 base-pair long windows (c.f. Fig. 3.3). From these decoy windows we then removed all sequences that matched a positive example. This results in 122, 154 negative examples, where the `AG` is positioned at the same place as in the positive samples.

Donor site The construction process is the same as for the acceptor site, except that the large window is 100 bases long (instead of 101). The windows were -50 to $+50$ around the true site. The first 50 bases are exon, the next 50 starting from `GT` are intron. In all the extracted sequences `GT` is at positions $25 - 26$. This leads to 74, 505 positive and 177, 061 negative examples.

Embedding in a genefinder All our training samples are windows of length 50 centered around `AG` which were taken from the nearby positions -25 to $+26$ around the splice sites in the acceptor case. Hence the relatively good results we get in the experimental section are only valid when we are presented samples lying within that 51 basepair window. We therefore strongly depend on a genefinder, that will "filter" sites not close to the true site.

[20]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.

3.1.3 Preliminary Analysis

The performance of the models that will be shown in the experiments section heavily depends on careful model selection. However since the number of tuneable parameters (the number of states, the pseudo count, the used normalisation in FK/TOP experiments, the "optimal" window size) is too high, to be taken into account for all types of HMMs a preselection is done using the simplest type of HMMs, the linear HMMs.
We trained 207360 linear HMMs[21] on 100, 1000, 10000 and 100000 donor and acceptor site training samples averaging the results using 5 different data splits. All models were trained on the training set, the optimal parameters were selected on the validation set while finally the test set was used to get the performance of the models. We varied the order of the HMMs obtained from the positive samples as well as HMMs trained on the negative samples from 1 to 8 and used different pseudo counts (10^{-10}, 10^{-2}, 10^{-1}, 1, 5, 10, 50, 100, 1000) in all of these combinations.

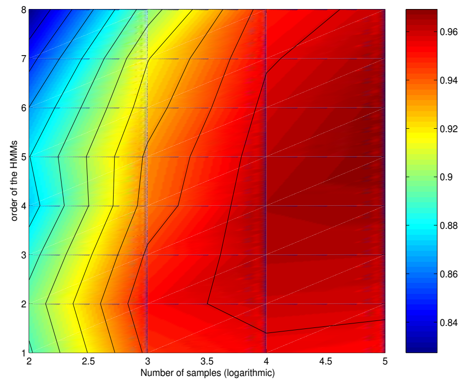


Figure 3.4: Order vs. number of training samples on the acceptor site.

In Fig. 3.4 and 3.5 both, the negative and the positive model were trained on training set sizes of 100, 1000, 10000 and 100000, which is shown logarithmically (log_{10}) on the x-axis. For each training set size the order was varied from 1-8 while both, the negative and the positive model were using the same order. This is shown on the y-axis. The colour corresponds to the classification performance achieved on the validation set using the optimal pseudocount for that particular order and training set size. In both figures one realises that the optimal choice of the order depends on the number of training examples, i.e., to get best performance one has to use HMMs of order 1 when only 100 training samples are available. As soon as the number of observations becomes larger, performance increases regardless of the order.
However higher order models perform favourably to lower order models on large training sample sizes. One clearly sees that on 100000 samples the optimal order is 5-6 for the acceptor site and around 5 on the donor site. When deciding which order to use one should follow Occams Razor and choose the lowest order that achieves good results, since the number of parameters increases exponentially.
We summarise:
• Performance increases with the number of training samples.
• High order models (order in the range of 5-6) outperform the rest when using a large number of samples.
• For a small sample size lower order models achieve better results.

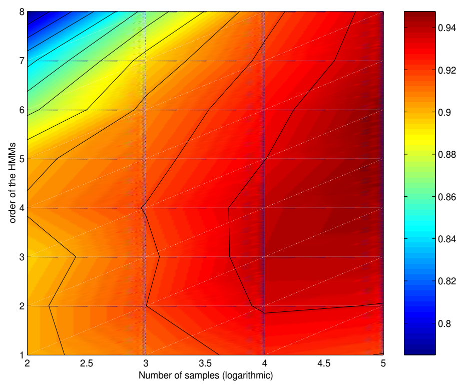


Figure 3.5: Order vs. number of training samples on the donor site.

Biologically, it is know that there are dependencies *within* codons, but we did not expect to find correlations of order $>$ 3 which means that one finds dependencies *between*

[21]Linear HMMs can be trained very efficiently. They are explained in Definition 2.13.

triplets which help to classify splice sites. There are even higher order correlations, but since the number of parameters is very high one would need even more observations to train "good" models. If one knew more about the kind of correlation, one could reduce the number of parameters drastically and thus even a small number of training examples could be sufficient.
We will now debate the question of the optimal choice of the pseudo count.

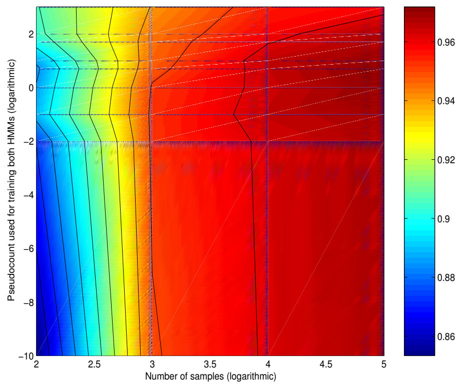


Figure 3.6: pseudo count vs. number of training samples on the acceptor site.

Similiar to the discussion of the optimal choice of the order, we show the performance of the linear HMMs for a set of the pseudo counts $\Pi \in \{10^{-10}, 10^{-2}, 10^{-1}, 1, 5, 10, 50, 100, 1000\}$. We chose to use logscale for both x and y-axis, while the former describes the pseudo count and the latter the number of training samples. The results are shown in Fig. 3.6 and 3.7. Again both, the negative and the positive model were trained using the same pseudo count and the colour corresponds to the best classification performance achieved on the validation set using the optimal order for a particular choice of the pseudo count and training set size.
Discussion: Again, the performance increases with the number of training samples. In the case of very few samples a large pseudo count is required to get reasonable results. For less than 1000 samples in the acceptor case it makes almost no difference which pseudo count to use. However when the number of samples increases best performance is achieved for pseudo count 1 and 5 on acceptor sites, while the optimal pseudo counts seems to be a bit lower on donor sites.

In conclusion:
• In the case of very few samples a high pseudo count should be used.
• When the number of samples is large, the pseudo count should be lowered
• Pseudo counts around $1 - 5$ result in good performance an a large number of samples.

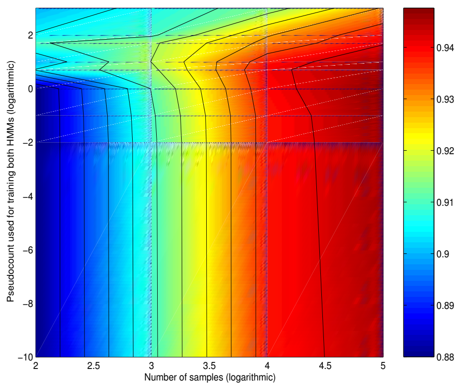


Figure 3.7: pseudo count vs. number of training samples on the donor site.

The pseudo count is associated with uncertainty of the data. We therefore observe the tendency that uncertainty (high pseudo count) decreases with a large number of samples, while in the case of very few training examples the model has to be prevented from *overfitting* by using higher pseudo counts.
While these figures can not capture the relation between the choice of pseudo count and order, we observe that high order models do require a larger pseudo count than low order HMMs. This is due to the fact that the simplicity of low order HMMs with only very few parameters (e.g., first order HMMs have only 200 parameters) already cause regularisation. On the other hand higher order models which contained up to 3276800 parameters (8th order) tend to overfit when no pseudo count is used.
However this analysis does not treat the parameters of positive and negative HMMs differently. For this reason we extend our analysis in Fig. 3.8 and 3.9. There we show

how the performance depends on the training set size, the order and pseudo count, where the positive and negative HMMs parameters were adjusted independently. The left column shows the result for the same set of pseudo count values as above while in the right column performance depending on the order is illustrated. While one can observe essentially the same tendencies "higher order is better", "low pseudo count is better" when the models were trained on a large number of samples, one also realises that the optimal order is around 5 for positive HMMs but around 7 for the negative HMMs[22] in the acceptor case. The correlations around the donor site seem to be of 5th to 6th-order for both positive and negative models (note the almost diagonal shape of the contour), and thus seem to be lower compared to the acceptor site. In the case of very few examples one can observe several bumps for donor sites and 2 in the acceptor case. The trained models are not very stable and such the model parameters vary greatly, which is a probable explanation for this strange behaviour.
As a rule of the thumb 5th order seems to be the overall best choice.

These illustrations give a good impression of how the pseudo count should be set. In the acceptor case, the optimal pseudo count for negative models is 5 independent of the number of samples. For positive models, Π varies in the range of 10^{-10} to 1. From that we can reason that the structure learned from the positive samples is important, since it favours small pseudo counts, while the negative HMM could not capture data that good. However the negative model still performs much better than a uniform one or no model. Therefore it does help having a negative model. For the acceptor one would conclude to choose pseudo count $\Pi = 5$ for the negative model, pseudo count 10^{-1}, 1, 10^{-1}, 1 for the positive and order $(2,3)$, $(2,2)$, $(3,5)$, $(5,6)$ for 100, 1000, 10000, 100000 samples, respectively, where the first number in a tupel denotes the order of the positive model, the other the order of the negative HMM.
The situation is similiar in the donor case. Here the best set of parameters is a pseudo count of 5 for positive and negative models on up to 10000 observations and then 1 on 100000 samples. Regarding the order one should select (1,2), (2,1), (3,3), (5,5).
While one would expect similiar optimal values for the pseudo count and order of other "non-linear" HMMs, one should not take for granted that these results are directly applicable to different types of HMMs. However, one will observe the same tendencies (like, e.g., "low pseudo count" is better on a large number of samples) and the optimal values for linear HMMs can be a good starting point.

[22]on 100000 samples

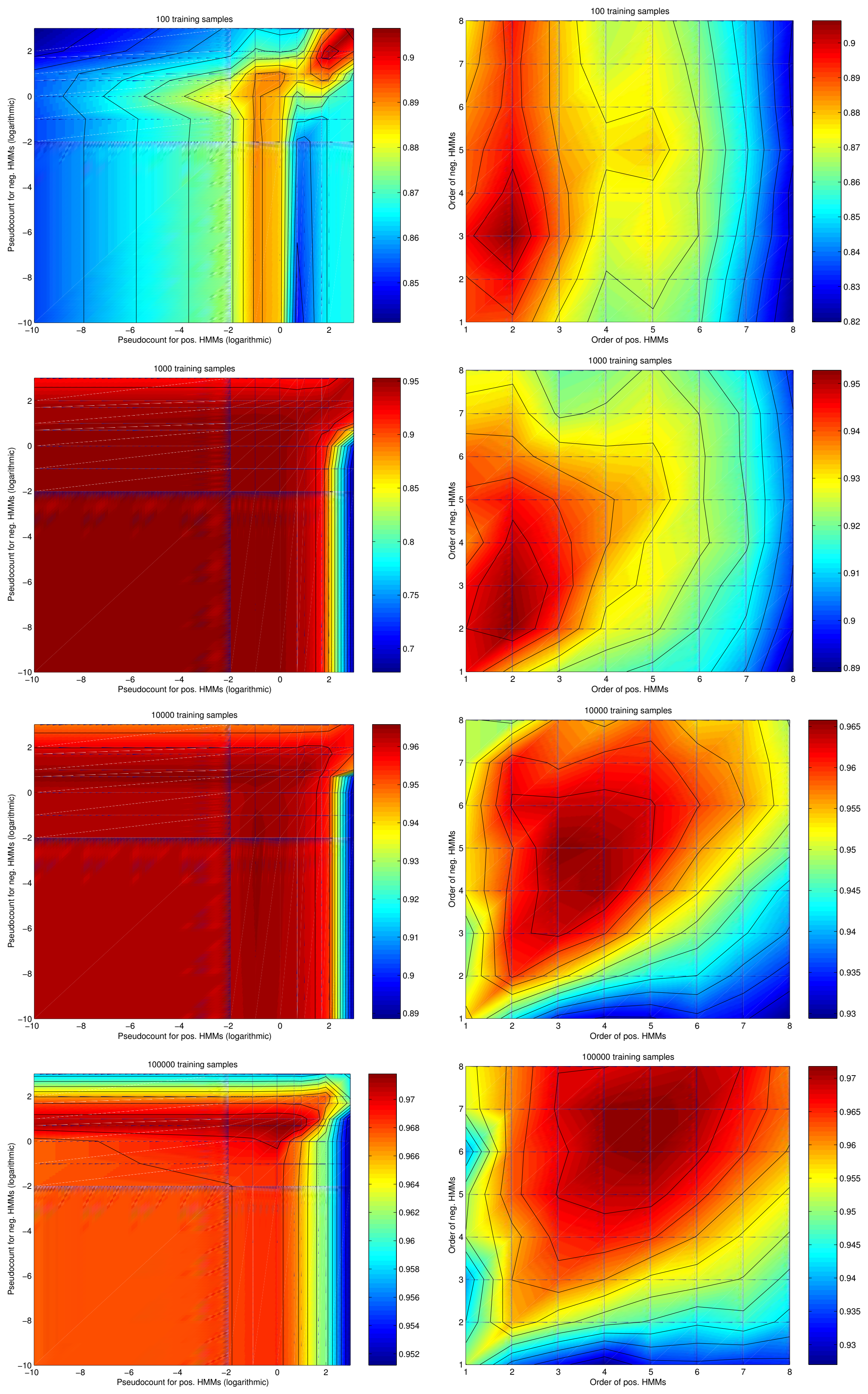


Figure 3.8: Performance comparison for linear HMMs trained on the *acceptor* site on sets of size 100, 1000, 10000 and 100000, while fixing pseudo count or order for positive (x-axis) and negative (y-axis) HMM separately.

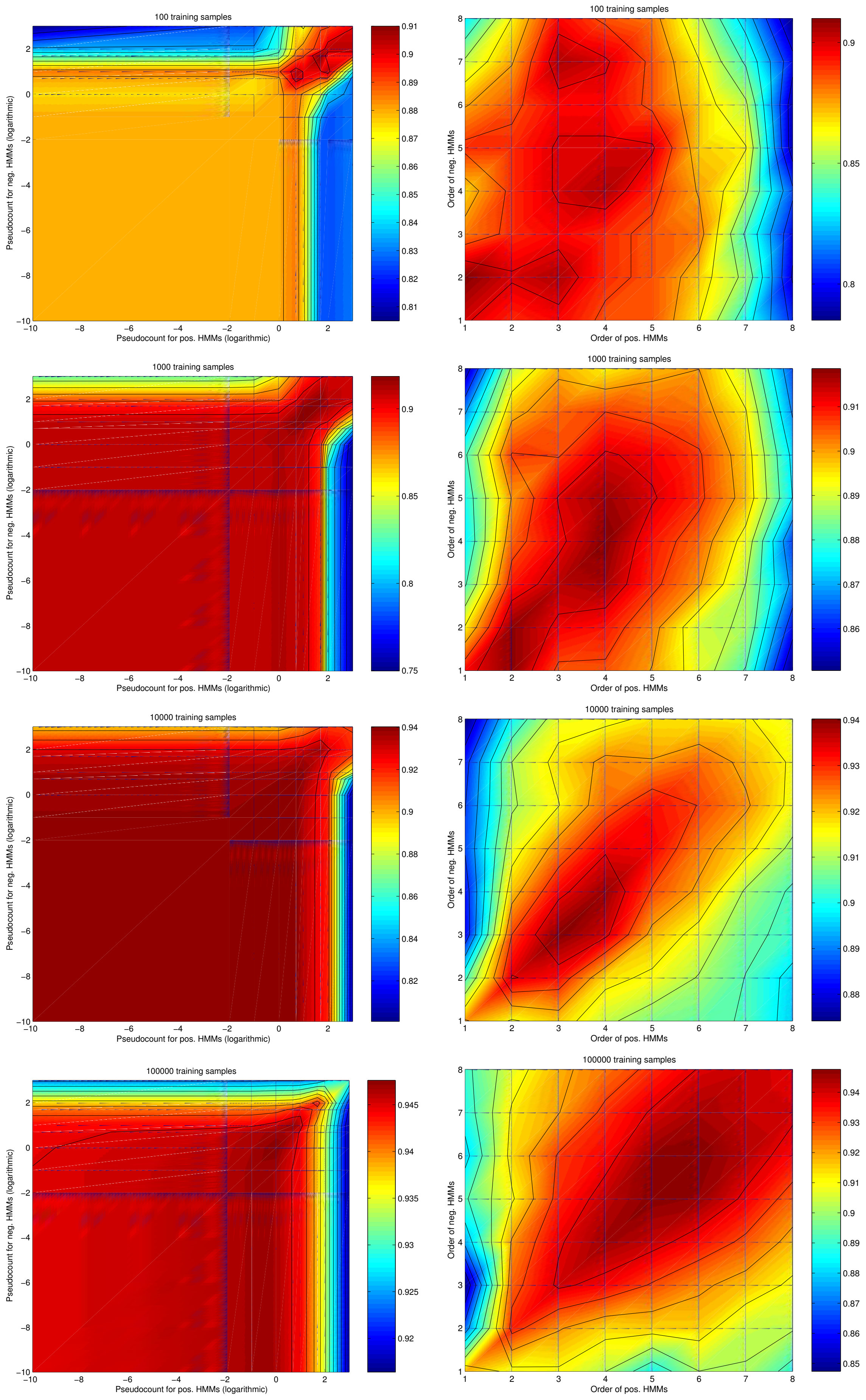


Figure 3.9: Performance comparison for linear HMMs trained on the *donor* site on sets of size 100, 1000, 10000 and 100000, while fixing pseudo count or order for positive (x-axis) and negative (y-axis) HMM seperately.

Entropy Entropy is a measure of the average uncertainty of an outcome. Thus the smaller the entropy the more certain we are of a result. We can use this information to detect regions of interest, i.e., regions where the signal appears to be non-random and thus has lower entropy.
Let X be a random variable with distribution $\Pr[X]$ for discrete emissions $e_0, \ldots, e_{m-1}$. Then the Shannon entropy is defined [17] by

$$H(X) = -\sum_{i=0}^{m-1} \Pr[e_i] \log \Pr[e_i].$$

As the probabilistic model, we use linears HMM obtained from positive training patterns and compute the entropy from this model to get some idea about the signal. Here in the DNA case the possible emissions are `A,C,G,T`. As log we used the natural algorithm.

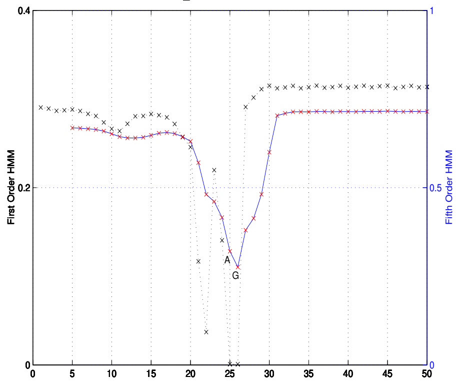


Figure 3.10: Entropy in a window around the acceptor site. The entropy obtained from first order models is shown in the dashed line while the entropy of the 5th-order linear HMMs is shown as a blue line.

To present both curves in one figure we use different scales for each curve. On the left side is the scale for first order HMMs while the one for higher order HMMs is on the right. The distribution is obtained by training linear HMMs of order 1 and order 5 on 100000 samples, using pseudo count 5 for the acceptor and 1 for the donor models. Looking at Fig. 3.10 and 3.11 we notice that the entropy is lower around the splice sites for both donor and acceptor sites, which is what we expected. Since no 5th-order information is available for the first 5 bases, the plot for 5th-order HMMs starts at position 5. In practice we fill up the observation with 4 more `A`s at the left hand side, such that the order decreases at this boundary from 5 down to 1 leading to an *artifact*. Thus, when not removing the first 5 values, one can observe a dropping down of the 5th-order HMMs curve at the beginning down to the entropy value reached by first order models.

Furthermore since higher order correlations are used, which is related to low-pass filtering, the curve appears much smoother. One can observe the entropy being generally lower in the intron (the part from beginning to `AG`; the part starting with `GT` to the end). This underscores that there is some kind of signal in the intron which is relevant in the splicing process. Only the first/last few bases (only around 5) in the exon seem to carry information about splicing. This can be explained biologically. One theory states that introns were inserted later in the evolution [30]. However only some sites might be well suited to allow insertions of introns, which might have lead to some distinguishable bases in some bases of the exon at the intron/exon and exon/intron boundaries. Another biologically justified site, *the branch site*, might have caused the entropy to be lower in the acceptor case around positions 9-12 which is only 15-18bp before the

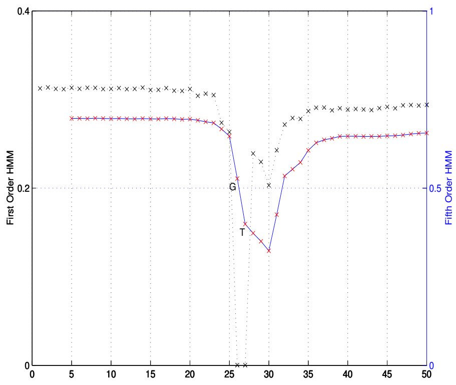


Figure 3.11: Entropy in a window around the donor site. The entropy obtained from first order models is shown in the dashed line while the entropy of the 5th-order linear HMMs is shown as a blue line.

splice site.

Relative Entropy While the entropy is "descriptive", i.e., it will tell us where to look in the area around the splice site, the relative entropy can tell us how much the positive and negative sequences differ in each positions. It is defined [17] on two distributions $\Pr_+[X]$ and $\Pr_-[X]$. as

$$H(\Pr_+||\Pr_-) = \sum_{i=0}^{m-1} \Pr_+[e_i] \log \frac{\Pr_+[e_i]}{\Pr_-[e_i]},$$

i.e., it is a weighted sum of the log-odds, also referred to the Kullback-Leibler distance. While a large relative entropy coincides with a large distance between both distributions, it is no real metric, since it is asymmetrical. We can use it to detect regions in that splice sites differ from decoys.

The relative entropy curves underscore the deductions we got from the entropy plots, since the inverted curves are similiar to the curves we observed for the entropy of both donor and acceptor sites. In the acceptor case the branch site is even more distinctive. Around the actual splice site one clearly observes a large distance between positive and negative samples. The difference is larger in the intron, but still some basepairs in the exon are discriminative. We again removed the artifact present in the beginning (first 5 values) of the entropy plot for the higher order HMM. While in the first order case the curve is almost zero outside the positions $7-12$, $17-24$, 27, in the acceptor case and $21-25$, $28-35$ it settles on a certain nonzero level for higher order HMMs. While this could give some information regarding the splicing process, it could also be justified by a certain intrinsic structure that bases have to follow. Since the relative entropy in the intron drops to the exon level for bases < 10 in the acceptor case and > 42 in the donor case, we assume the latter. So the choice of a 50bp window around the true site *appears* to be reasonable and justified by the above reasoning.

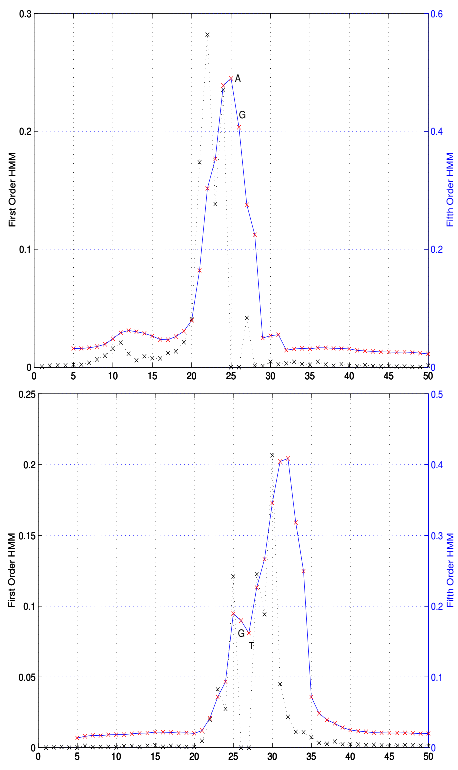


Figure 3.12: Relative entropy plots on the 50bp windows. The upper figure shows the acceptor, while the figure in the bottom shows the donor results. The entropy obtained from first order models is shown dashed, while the entropy of the 5th-order linear HMMs is shown as a blue line.

Normalisation of the Fisher and TOP Scores When dealing with SVMs which use the Fisher or TOP kernel from HMMs, it is very important to point out that one cannot instantaneously train SVMs using TOP/FK in practice since the quadratic optimisation problem that is to be solved in SVM learning does not converge due to numerical problems. Why does this happen? Consider that the model probability for the positive HMM is very small for some sequences. Then in the TOP kernel case, all scores derived from positive model parameters are divided by this quantity (cf. Section 2.4.3) and

therefore become arbitrarily large (or small for the negative model). Likewise when both the negative and the positive model consider a sequence to be very unlikely, similiarly large values occur in the FK feature scores. Since this causes the score vector for all features of certain samples to become very large (or small), one has to think of some normalisation that maintains the relationships between samples, but brings the score vectors within a certain fixed range. A standard approach is to normalise data such that it is contained within a sphere, which is done by normalising the variance to be 1 in each dimension independently. In the same step one could as well drop dimensions with almost zero variance, i.e., $\leq 10^{-6}$. As another approach one could normalise each vector $\mathbf{x}$ to be to $\|\mathbf{x}\| = 1$. When doing so, all data lies on a hyper sphere of radius 1. In Table 3.1 we demonstrate how the test error obtained on the validation set impacts on different normalisation techniques. As normalisation techniques we used no normalisation, $\|\mathbf{x}\| = 1$, variance componentwise 1 and their combinations. Note that the order of application of these normalisations methods is important.
Finally we scaled the kernel matrix by dividing with the mean of the diagonal elements

$$K'(\mathbf{x},\mathbf{x}') = \frac{K(\mathbf{x},\mathbf{x}')}{\frac{1}{N}\sum_{i=0}^{N} K(\mathbf{x}_i,\mathbf{x}_i)}$$

and computed SVMs on the first two runs using

$$C \in \{0.001, 0.1, 1, 2, 4, 5, 7, 8, 10, 100\}$$

on 100,1000 and 10000 samples.

TOP Kernel

size	no normalisation	$\|\mathbf{x}\| = 1$	variance	variance and norm	norm and variance
100	n.c.	**85.0±2.0%**	77.8±0.6%	80.5±4.1%	**84.6±2.0%**
1000	n.c.	**96.9±0.4%**	96.1±0.2%	96.0±0.6%	**96.7±0.4%**
10000	n.c.	97.9±0.1%	**98.3±0.0%**	98.2±0.0%	98.1±0.3%

Fisher Kernel

size	no normalisation	$\|\mathbf{x}\| = 1$	variance	variance and norm	norm and variance
100	n.c.	**86.2±2.1%**	80.4±2.1%	85.1±2.4%	85.2±2.6%
1000	n.c.	96.6±0.4%	**97.5±0.3%**	96.6±0.3%	96.8±0.2%
10000	n.c.	97.5±0.2%	**97.8±0.1%**	97.5±0.1%	97.6±0.2%

Table 3.1: Classification rate on the validation set using the TOP and FK method on 100,1000 and 10000 samples, with different normalisations.

The result is shown with optimal bias b, i.e., b was adjusted on the validation set to give best performance. Here the abbreviation n.c. stands for no convergence.

As one can see, performance depends on the *kernel* and *sample size* which is used. We decided to use variance only normalisation, which gives best results on large sample sizes, throughout all our experiments. One would therefore expect slightly worse results when the sample size is small.

3.2 Human Genome

3.2.1 IP-Data

IPData is a benchmark set of human splice site data from the UC Irvine machine learning repository [5]. It contains 765 acceptor sites, 767 donor sites and 1654 decoys; the latter are of low quality as they do not have a true site's consensus dint centered except by chance. The task is to classify donor and acceptor sites given a position in the middle of a predetermined window of 60 DNA letters as input. We use this data set *as it is,* to benchmark our methods. We did no further investigation on that data set, but applied our methods as they are.

3.2.2 Preprocessing the Human Genome

UCSC assembled Genome At the University of California Santa Cruz[23] a team of scientists lead by David Haussler (in particular a lot of work was done by Jim Kent to assemble the genome) in collaboration with the International Human Genome Project, released the first draft of the Human Genome in May 2000 (cf. [14]). We are using a slightly newer release of the genome from October 2000 (the CD-ROM[24] version). We used this almost 90% complete (around 75% highly accurate, remaining 25% 'draft' quality) Human Genome which can be downloaded at `http://genome-archive.cse.ucsc.edu/goldenPath/07oct2000/chromosomes/`. The human splice sites were taken from SpliceDB [10]. We matched the EST-supported human splice sites prepared back in 1999 from Burset, Seledtsov and Solovyev[25] to the newer release of the human genome.

Preprocessing In the given dataset, the window around the splice site is 82 bases long where the `AG` and `GT` is at positions 41-42 in the canonical case. We matched all sequences in the file `HumanAllSites` onto the whole genome, i.e., 28468 acceptor and 28468 donor sites. At first we generated a file containing all potential splice sites, i.e., we took each chromosome, and moved a window of size 82 bp througout the whole chromosome, writing out sequences that matched `AG` or `GT` on the $5' \rightarrow 3'$ or reversed and inverted on the $3' \rightarrow 5'$ strand. We then sorted this set of ≈ 50 GB and removed duplicates and then used the unix tool `comm` to get the sequences that did not match *exactly* (19132 acceptor samples and 19226 donor samples did not match). The ones that matched to some sequence in the set of all possible splice sites, where in a next step matched to sequences contained in the file `HumanCanonicalSites.ESTsupp+corr.acc`, such that only the EST confirmed sites which were additionally corrected by Burset et al. remained as positive samples. While this was done quickly overnight on just one computer, the unvalidated splice sites still had to be matched, requiring ≈ 14 days of computation on 32 PentiumIII-550 CPUs: The newer release of the genome is of higher accuracy, i.e., a number of non `A,C,G,T` symbols like `-,N,m,r,w,s,y,h,b,d,k`[26] have been determined to be one of these four bases. When, e.g., errors like the wrong alignment of sequences is corrected, one will observe insertions or deletions. For that reason we used a recent version of the *Basic Local Alignment Search Tool (BLAST)[2]*

[23]`http://genome.ucsc.edu/`

[24]This CD-ROM was produced at the University of California, Santa Cruz, Baskin School of Engineering, by David Haussler, Patrick Gavin, Jorge Garcia, Jim Kent, Terry Furey and Scot Kennedy. Jane Ades and Darryl Leja at the National Human Genome Research Institute, NIH, produced the label design.

[25]This data can be downloaded from `http://genomic.sanger.ac.uk/spldb/SpliceDB.html`.

[26]See `http://www.ncbi.nlm.nih.gov/BLAST/fasta.html` for the meaning of these symbols

from April 2001 to align sequences, which can deal with these issues. *BLAST uses a heuristic algorithm which seeks local as opposed to global alignments and is therefore able to detect relationships among sequences which share only isolated regions of similarity (Altschul et al., 1990) [2].*
For more details see `http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html`.
We removed the leading and trailing `N,-` from the splice sites and created a query file suitable for BLAST. The command line was:

```
blastall -P 1 -p blastn
    -d chromosome
    -i HumanAllSites.acc.minus.n.removed.query
    -o results/acc.query.chromosome
```

This query was applied to each chromosome on the missing donor and acceptor sites. When processing the BLAST output we dropped all sequences that did not match in at least 40bp and did not contain `AG,GT` at the position 40 ± 2. We had to allow these small deviations, since bases that were inserted or deleted *before* the consensus dint were shifted. We finally extracted[27] the sequences, joined them with the already validated set of 4893 acceptor and 4823 donor sites and removed duplicates to form 11908 validated 3' and 11966 5' sites. In this process we lost 3355 (22%) of the acceptor and 3297 (22%) of the donor samples from the 15263 EST confirmed and corrected splice pairs. However 2034 of the original acceptor and 2112 of the original donor sites did contain non `ACGT` characters while the sequences we got in the end are free from non `ACGT` characters. Finally another 23 3' and 27 5' sites were removed since they still did contain characters other than `ACGT`. We also repeated the extraction process with *no* threshold and allowed a window of 40 ± 3 to generate possible positive sequences. We removed all those sequences from the set of possible sequences, to get "negative" samples[28]. In a first try we tried to solve the *global* splice site prediction problem, i.e., we wanted to seperate the few positive examples from the very large number of negative samples. This number was still high, since from the $\approx$ 50 GB only $\approx$ 300 MB were removed in the last step. Experiments showed that the number of negative samples is still intractably high. All our classifiers learned the decision rule "all sequences are non-splice sites." The classifier could detect only very few real splice sites. While one can tune the classifier to allow more false positives resulting in a higher sensitivity, the number of false positives becomes incommensurately high compared to the number of correctly classified sites. For that reason, the process of splice site prediction appears to require a *local* method. One should therefore concentrate on the detection of splice sites using such local methods and later on embed these in a genefinder,

162599931	chrX
51513585	chrY
282193664	chr1
253256583	chr2
227524578	chr3
202328347	chr4
203085532	chr5
182415242	chr6
166623906	chr7
152776421	chr8
142271444	chr9
145589288	chr10
150783553	chr11
144282489	chr12
119744898	chr13
106953321	chr14
101380521	chr15
104298331	chr16
89504553	chr17
86677548	chr18
74962845	chr19
66668005	chr20
44907571	chr21
47662662	chr22
3310004818	total

Table: The size of each chromosome in bases. In total the human genome is about 3GB (Giga-Bases) in size.

[27]This is not trivial, since matches were obtained for each chromosome *and* both strands have to be taken into account.

[28]This set will definitely contain a large number of real splice sites, but still the number of real negative samples contained in this set is much higher.

which from the viewpoint of a splice site detector serves as a filter. To our knowledge no biological experiments have been done, investigating whether the spliceosome will actually cut down the number of decoys found in non-genetic regions. Such experiment would give further insight and could finally answer the question whether splice site detection can be dealt with local methods, which is what the spliceosome appears to be biologically, only.

4 Experiments

Before proceeding to the actual experiments, we give a short roadmap. At first we evaluate our methods on the IP-benchmark dataset and show that we achieve very good results. This suggests that our methods are suitable for the splice site recognition problem and we expect similiarly good results on the *C. elegans* data set which we will be dealing with in the second section. We put our main focus to the detailed discussion of our experimental results. Finally we present some preliminary results we got on human splice sites.

4.1 Benchmarking the Methods on the IP-Dataset

When one suggests a new method, one has to justify why and when this method is superior to previous techniques. While newer methods could be more appealing from the theoretical standpoint, they might not achieve better results in practice. That is why they are always evaluated on so called *benchmark* data. Such data sets are used for comparison of the methods and have usually no more scientific importance. As methods we use HMMs[29] and SVMs. For the latter we had to choose a good kernel. Thus we decided to use kernels well suited for the splice site recognition problem as introduced in Section 2.4: we use the Locality Improved, TOP and Fisher Kernel where the latter two are derived from HMMs.
For our experiments, we had to perform a careful model selection of the hyper-parameters of the HMMs and SVMs (cf. [34]). This is done separately on each of ten random splits of the data of training size 2000, test size 1186. In [37] only a single same-sized split was used and thus their results do not include error bars. As HMM architecture we used

(a) a combination of a linear model and a fully connected model for the acceptor sites (cf. Fig. 4.1, upper)

(b) a combination of two fully connected models for the donor sites (cf. Fig. 4.1, lower) and

(c) a fully connected model for modeling decoys.

These architectures can be biologically motivated: The positive donor model is a concatenation of two fully connected HMMs, which were trained seperately on the intron and exon part respectively. The positive acceptor model uses a linear HMM to model the intron. This type of HMM was already shown to perform well in Section 3.1.3. The exon is modelled by a fully connected HMM. Thus each HMM component models the statistics of *either* the intron or the exon part, while their concatenation models the whole splice site. Finally we chose to use small, fully connected HMMs as negative models due to their simplicity. These fully connected HMMs can capture relations within and between triplets.
The corresponding number of states in the components, as well as the regularization parameter of the SVMs are found by 10-fold cross validation, i.e., we split the training set into 10 parts, train 10 times using 9/10 as training data and the remaining 1/10 as validation data. The parameter combination that gave best *average* performance obtained on the validation set was used for testing. We estimated a large number of HMMs whose number of states was chosen to be $5-13$ for either component of the

[29]Only first order HMMs were used, which were trained using pseudo count $\Pi = 10^{-10}$.

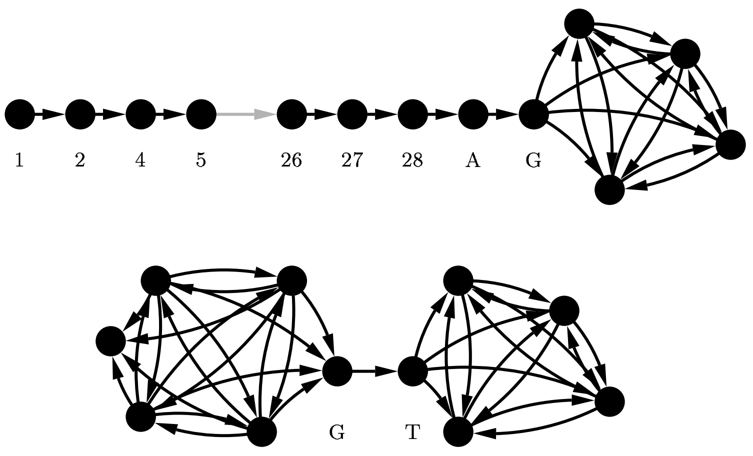


Figure 4.1: Positive acceptor (top) and donor models (bottom).

positive and the whole negative model. From the positive-negative HMM combination that gave best results, we derived the Fisher and TOP Kernel and performed model selection of the regularisation parameter C for each of those SVMs, where

$$C \in \{0.001, 0.1, 1, 2, 4, 5, 7, 8, 10, 100\}.$$

The SVMs with best average performance on the validation set were used to obtain the actual performance on the test set.

System	Neither	Donor	Acceptor	Total
LI-SVM	2.0±0.3%	0.8±0.2%	0.9±0.3%	3.7%
FK-SVM	2.1±0.4%	1.6±0.5%	1.6±0.4%	5.3%
TOP-SVM	2.2±0.4%	1.5±0.4%	1.7±0.3%	5.4%
HMM	2.6±0.5%	1.0±0.4%	2.4±0.7%	6.0%
RBF-SVM	n.d.	n.d.	n.d.	>10%
NN-BRAIN	n.d.	2.6%	4.3%	n.d.
BRAIN	4.0%	5.0%	4.0%	13.0%
KBANN	4.6%	7.6%	8.5%	20.7%
BackProp	5.3%	5.7%	10.7%	21.7%
PEBLS	6.9%	8.2%	7.6%	22.7%
Perceptron	4.0%	16.3%	17.4%	37.7%
ID3	8.8%	10.6%	14.0%	33.4%
COBWEB	11.8%	15.0%	9.5%	36.3%
Near. Neigh.	31.1%	11.7%	9.1%	51.9%

Table 4.1: Test-set errors on the IPData data set. All except the first 5 results are cited from [37], Table 6. The fifth result is cited from [38]. (n.d.=not documented)

Analogously the parameters for the Locality Improved Kernel were selected from: $d_1 \in \{2,3\}$, $d_2 \in \{1,2\}$ and $l \in \{1,2,3,4,5,6\}$, where the SVMs $C \in \{0.4, 0.8, 1, 1.5, 2, 4, 7, 10, 20\}$. This process was done on both donor *and* acceptor sites independently. We designed the models such that they can only detect canonical splice sites. Therefore all non-canonical sites immediately show up as wrongly classified. However this reduced the number of training samples, since only sites that contained the

canonical AG or GT at position $28-29$ respectively $30-31$ had to be taken into account. Note that not a small number of donor sites did contain an AG at positions $28-29$, while a number of acceptor sites contains a GT at positions $30-31$. This even exacerbates the problem unnecessarily, since one would not try to distinguish acceptor from donor sites in practice. Nevertheless this data set is of great value for comparison when not altered in any way. Thus we obey this rule.
After the above described model selection procedure, each classifier was then evaluated on the test sets and results are averaged over the 10 runs. The mean and corresponding standard deviation is given in Table 4.1, which shows the errors on each of the classes for the IPData data set. Comparing our classifiers we observe that SVMs with TOP and FK (total error 5.4% and 5.3%) cannot improve the HMM much, which already performs quite well (6.0%), except in classifying the acceptor sites. The SVM with Locality Improved Kernel does not suffer from this problem and achieves the best total error of of 3.7%. To further improve results one should invest more time in fine tuning the HMMs, since these are generally much more powerful especially when combined with the TOP or Fisher Kernel as explained in detail in Section 2.4.4.
We observe that the SVM methods outperform all other documented methods on the IP data set. These include not only the BRAIN algorithms of [37] published recently, but also other established machine learning methods such as nearest-neighbor classifiers, neural networks and decision trees. The SVM achieves test-set errors that are only half as large as the ones of the best other methods. However his was only possible by choosing a suitable kernel. If one uses plain RBF kernels, one gets worse results than the BRAIN method [34]. Furthermore this benchmark dataset gives the wrong impression that detection of acceptor sites is more difficult. In fact the results we get on *C.elegans* suggests that it is the other way around.

4.2 C. Elegans

When we started to deal with the classification of splice sites, we had very little prior knowledge on the problem. In that case one will try the simplest reasonable methods and learn more about the problem by using them. Among the "simple" methods are linear HMMs and fully connected HMMs whose results we present first. Later on, we show how we improved these methods by using specially *designed* HMMs. On the other hand we use SVMs to improve performance of our probabilistic models. Note, that all experiments were done independently on 5 different (training-, validation-, test-set) splits. The obtained results were averaged, i.e., mean and standard deviation is shown.

4.2.1 Linear HMMs

Linear HMMs are probabilistic models, which assume that the observations are pairwise independent. Therefore they can be trained easily and very efficiently. As a result we were able to do model selection on a large number of parameters as discussed in detail in Section 3.1.3. Nevertheless it is astonishing that these simple models lead to the quite good results of $2.9 \pm 0.2\%$ test error on 100000 samples on the acceptor and $5.3 \pm 0.2\%$ on the donor site (cf. Table 4.2).

4.2.2 Fully Connected HMMs

Another greedy approach, is to train fully connected HMMs on the whole set of sequences using the Baum Welch algorithm. Then the parameters to be adjusted are the number of states and the pseudocount. We were pampered by the results we got from linear HMMs and therefore expected even better ones from fully connected HMMs in our experiments. We fixed all HMMs to be first order and trained 252 positive and 252 negative HMMs on 50000 acceptor training samples using all possible combination of pseudo count

$$\Pi \in \{5000, 1000, 200, 50, 10, 5, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 0\}$$

and number of states $n \in \{20, 25, 27, 29, 30, \ldots, 39, 40, 45, 50, 60\}$. Originally we intended to train each parameter combination two times on 5 different splits of the training set. However although we spread this task over 50 CPUs (standard PCs, mostly PIII 550 or better) not even half of the experiment is finished after 6 months of computation. This is due the slow Baum Welch training algorithm[30]. Furthermore it converges slowly and thus very often required the maximum number of iterations (was set to 150). Even when the training algorithm converged (the model probability did not change more than 10^{-3} in 5 subsequent reestimation steps) the estimate was very poor. The algorithm is only guaranteed to find a *local* maximum, which was far away from the global optimum.

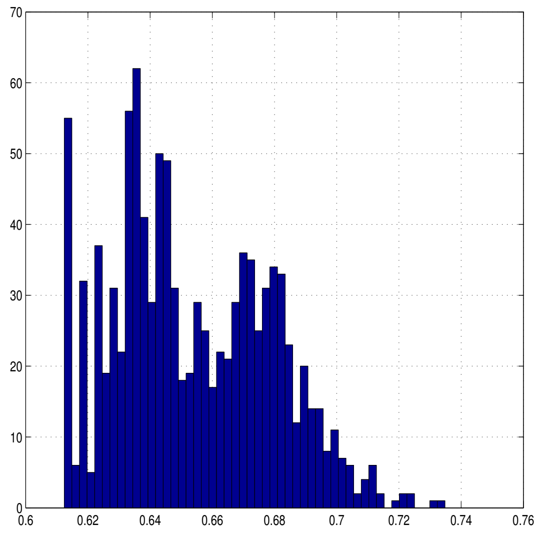


Figure 4.2: Histogram of the classification performance on the validation set.(x-axis accuracy $0 \ldots 1$, y-axis number of occurences of this value) As classifier, we used we used the 1035 positive HMMs that finished Baum Welch training combined with a single state negative HMMs. We set the threshold on the validation set, such that maximum performance was achieved. Note that the worst possible result, i.e., to consider all splice sites are decoys is achieved very often ($>$ 50 times). Since the validation set consists of 1938 positive and 3062 negative (in total 5000) samples, the worst performance that can be achieved is 61.24%

We were largely disappointed by the results (cf. Fig. 4.2) we got. Since the length of all sequences was 50 base pairs, and a 50 state model could potentially converge to a linear HMM of order 1. None of the results is even close to the performance of a linear HMM which already achieves results of $> 94\%$ accuracy, see Fig. 3.4. Considering that the Baum Welch training only optimises the model probability this seems reasonable. However comparing the performance of linear and fully connected HMMs in terms of the model probability, still *all linear HMMs outperform Baum Welch trained fully connected HMMs.*
To conclude: The model probability, basically being a constrained polynomial $f(\mathbf{x})$ of the high dimensional vector $\mathbf{x}$ has many local maxima when optimised via Baum Welch training. However since the problem of finding

$$\min_{\mathbf{x}} f(\mathbf{x})$$

is NP-Hard [26] and even the problem of computing the optimal with a certain accuracy $\epsilon > 0$ is NP-Hard [25], local optimisation methods as, e.g., Baum Welch, Newton or Gradient Decent are used.
To get a preliminary idea on how hopeless[31] the situation is, we used Baum Welch train-

[30]We tried hard to optimise the algorithm, i.e., we use forward/backward tables and "log-tables" to substitute the often occuring computation $\log(e^x + e^y)$ by a table lookup [47].
[31]This experiment requires already 1600 hours on the APAC-cluster (almost half a year of computation on a single Athlon XP 1GHz). Theoretically one would have to do this experiment for each combination of parameters.

ing to estimate 100 37-state HMMs using fixed pseudocount 10^{-10} and order 1 on an even smaller dataset[32], allowing 500 iterations, while the HMMs initial parameters were set randomly. A single state HMM was trained to be used as the negative model. We then compared the model probability of the HMMs on the *training set* (cf. Fig. 4.2.2). The best model accomplished a log-likelihood of -59.2, the worst -62.1 leading to an average and standard deviation of -61 ± 0.43. While this is an improvement compared with a completely uniform HMM (each emission probability is set to 1/4, start/end/transition probabilities are 1/37) whose model probability is -69.3. Still our specially designed HMMs using the same number of states achieve -59.0 in log-likelihood and 5.3% in classification error. Only 3 of 100 trained HMMs give comparable results in terms of model probability (5 in terms of classication performance).

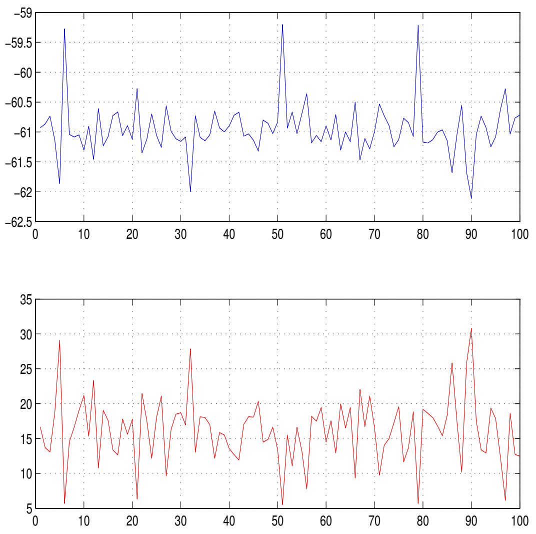


Figure 4.3: Quality of Baum Welch training. We trained 100 HMMs via Baum Welch. The upper curve illustrates the model probability for the training set, while the figure below shows the classification performance on the validation set.

Even a simple linear HMM is already at -59.3 in log-likelihood and 5.4% classification error. Therefore the use of a greedy Baum Welch training without prior knowledge seems to be a waste of time. However the comparison with a first order linear HMM is slightly unfair, since it already consists of 50 states. We tried to further improve HMM performance by setting "reasonable" initial parameters. For example we started Baum Welch training on an already trained linear HMM, treating it like a standard fully connected HMM. We altered the parameters slightly since the linear solution is already a local maximum and retrained. Performance did not improve. In another we tried to add states to the linear model, initialising the new parameters randomly. At first we added one by one and trained. Performance decreased with each state. Then we added small sets of states to linear HMMs at once which lead to similiar results.
We realised that all one can do to circumvent local maxima is to

a) enhance the optimisation technique, e.g., by implementing simulated annealing as in [17]

b) simplify the task (e.g., train a number of models on only parts of the sequences)

c) simplify the model structure (e.g., train HMMs with specific structure, like a concatenation of a linear HMM and fully connected one).

In this work we decided to focus on the latter two with which we will be dealing in the next subsection.

4.2.3 Specially Designed HMMs

Having learned our lesson on Baum Welch training, we split the *positive* data into two parts, intron and exon. In the acceptor case all bases up to `AG` formed the intron-part

[32]run 1 on 10000 positive acceptor samples

and the remaining the exon-part. On donor sites all bases before `GT` form the exon-part and the rest starting with the consensus dint is the intron. Since linear HMMs worked well the question, "what is so special about linear HMMs" arises. We think that these static models, not tolerating insertions or deletions, help to fix the consensus dint in *classification.* While this is a reasonable explanation for classification and especially since negative samples were generated by shifting the window (cf. Section 3.1.2), it does not explain why they give good performance in terms of log-likelihood. However when one does classification using a linear HMM on only the *intron part,* performance drops only *slightly.* Thus almost all of the splicing information is contained in the intron, while still some correlations in the exon help to increase the performance.
We therefore trained *linear* HMMs on the intron part and *small* fully connected HMMs on the exon part and used their concatenation as the final positive model, leading to a HMM as shown in Fig. 4.4. When doing so, one implicitly assumes that there are no dependencies between bases in the intron and bases in the exon[33].

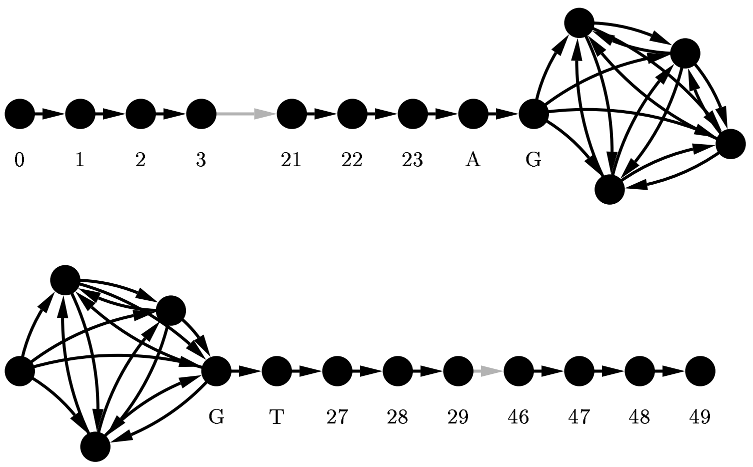


Figure 4.4: Positive acceptor (top) and positive donor models (bottom).

However this approach of splitting the data into two parts does not work on the negative samples, since these are *shifted* and one does not know either the beginning nor the end of the exon or intron. Nevertheless one can exploit the information that negative samples are constructed by moving a window of length 50 over sequences of length 100, where the splice site is almost in the middle (cf. Section 3.1.2). Therefore in all cases positive and negative samples are overlapping. We can use this to construct[34] negative HMMs, that embeds the positive HMMs utilising it to model parts of either the intron or exon. For that reason we add 25 states in the acceptor case to the left and 25 states to the right of the donor model (cf. Fig. 4.5).
To obtain the positive acceptor models we set the pseudocount and order to $\Pi = 10^{-10}$ and $\Omega = 1$ and independently trained the linear and fully connected HMMs where the latter had 5 to 14 states. Their concatentation as in Fig. 4.4 was used as the splice site model. We evaluated these HMMs on the validation set using no negative model and chose the HMM performing best to create a negative model, which was obtained by taking a linear HMM of 25 states and appending the best positive model to it. We then used Baum Welch training to estimate the emissision probabilities $b_{i',j}$ for the

[33]Having said that, we have in fact recently discovered that there are some dependencies. Thus this assumption is not quite true.
[34]This awesome idea is due G. Rätsch.

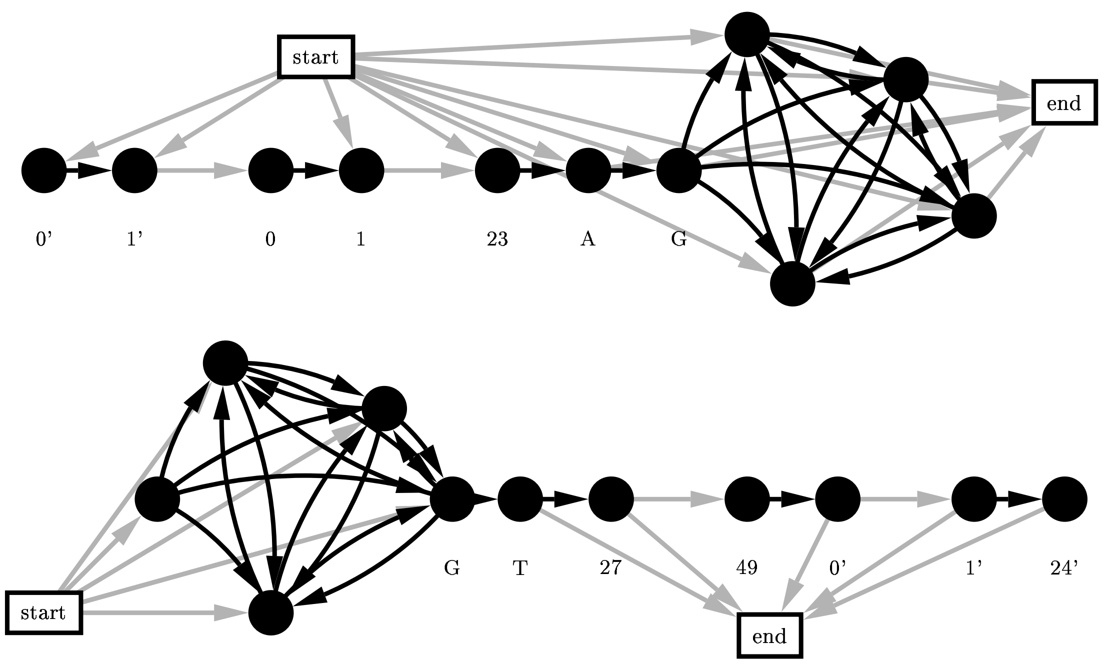


Figure 4.5: Negative acceptor (top) and negative donor models (bottom).

new states $0' \dots 24'$ and the new start and end state distribution, whereas the former start state in the positive model (state number 0) was not allowed to be a start state in the negative HMM, see Fig. 4.5. In the training process of the positive HMMs, the convergence criteria was set to 500 iterations and a maximum change of 10^{-5}. Since the negative model is more complex (about 55 states) the computation of forward and backward variables requires more time. However since only very few parameters have to be learned, we allowed only 20 iterations and a maximum change of 10^{-3}. When doing so, we even exceed all linear HMMs of higher order in the acceptor case, as one can see in Table 4.2, e.g., on 10000 samples the linear HMMs achieved 3.8% and these fine tuned HMMs 2.6% test error. In analogy we repeated this experiment on the donor sites. Here instead of inhibiting the start state of the positive model we did not allow the model to terminate in the former end state (state labeled 49). Unfortunately this technique lead to a very large test error on the donor sites ($>$ 20% on 10000 examples). One possible cause is the, in one class testing, already poorly performing positive donor model. We thus used a single state model trained on the decoys as negative model and repeated the test procedure. Performance remained on a low level, which was likely due to the still poorly performing positive HMM, which gives more than 10% test error. As a result the “positive” part contained in the “negative” sequence could not be found by the embedded positive HMM. Hence we tried further to increase the performance of the positive donor model, by setting the order to 3 and the pseudocount to 5 as is suggested by Fig. 3.9 on 10000 samples.[35] We obtained a new linear HMM on the intron part and Baum Welch trained small full connected HMMs of 1 to 17 states on the exon part. For the latter we decreased the pseudo count to 0.1 since these models have fewer parameters. We then used the concatenation of these models as in Fig. 4.5

[35]In fact one would have to include order and pseudo count in the cross validation procedure. However training higher order HMMs is computationally more expensive (due to the higher number of parameters) and thus this is beyond this work.

as the positive model. Note that the linear part is of higher order and thus one has to employ special training to not loose higher order information at the left hand boundary. In the training process of negative HMMs we set the pseudo count to the even lower value of 0.01 to allow finer adjustment of the new parameters. In testing the positive HMM we used a linear HMM of order 3 as the negative model. In many cases one could observe that HMMs with only a 1 state donor model slightly outperformed the others, but in combination with their corresponding negative model performed poorly. For that reason we decided to not preselect the best positive HMM, but obtained the best possible combination of positive and corresponding negative HMM via cross validation. In the end we get 5.3% test error on 10000 samples compared with 6.3% which is what the linear models could achieve. Interestingly the model selection result on all but one of the runs[36] chose 3 state HMMs for the exon part, which is in good biological agreement. Hence each state models the statistics of triplets (recall that the HMMs are all third order) for one of the three possible reading frames (see Section 1.2 for a biological introduction).
Up to this point, we have only dealt with *descriptive* models. We were able to tune them to perform quite well. However as we will see in the next sections, these results can still be improved by using *discriminative* techniques.

4.2.4 Locality Improved Kernel

The first candidate in our list of discriminative models uses a SVM whose kernel is the LIK. This kernel can be optimised such that one can efficiently train on large sample sizes. We used the speed tuned kernel of [55] in our experiments. Recall that the parameters of the LIK are the window width l and the degree d_1 that controls how many terms are used in this window, i.e., these parameters describe the amount of *local* information to be used. The remaining parameter d_2 controls the amount of information between windows, which is as such *global* information and was set to $d_2 \in \{1, 2\}$ for this problem, but best performance was achieved using $d_2 = 1$.
Complete model selection was performed on the local parameters l, d_1 as: $l \in \{2, 3, 4\}$ and $d_1 \in \{2, 3, 4, 5, 6, 7\}$. In each setup the regularising term of the SVM was choosen to be one of $C \in \{0.8, 1, 1.5, 3, 7, 10\}$. We tried larger windows of $l \in \{5, 6\}$. Since these did not show good performance they were excluded from the model selection. As one can see in Table 4.2 the LIK beats linear HMMs for all donor results, while performing similiar on acceptor results. We showed in Section 2.4.4, that the LIK is basically equivalent to the FK derived from *constant* linear HMMs. One can thus hope to get better results when using more clever underlying models, like the fine tuned HMMs in Fig. 4.4.

4.2.5 Fisher Kernel and TOP Kernel

We used the Fisher and TOP kernel in combination with our specially designed HMMs.[37] This "hybrid" learner, a combination of discriminative and generative HMMs does indeed outperform all stand alone models and thus we could improve the already very good results we got using HMMs and the LIK. While we observed empirical superiority of the TOP kernel in [49] both methods perform equally well on our dataset, i.e., using the FK we achieve 2.3% on the acceptor sites compared with 2.2% using TOP.

[36]dataset splits

[37]While one could derive these from linear HMMs, they require special treatment, since they are higher order, i.e., explicit computation of the TOP/Fisher feature scores is not anymore possible due to very high dimensional feature spaces (a sixth order model requires $\approx 15GB$ of memory).

Test error of our methods on *C. elegans*					
	Acceptor				
	Linear	HMM	LIK	FK	TOP
100	8.1±0.6%	10.7±2.8%	**7.6±1.0**%	13.0±5.7%	22.4±2.3%
1000	5.0±0.3%	**2.8±0.1**%	5.2±0.2%	2.8±0.2%	4.9±0.5%
10000	3.8±0.3%	2.6±0.2%	3.9±0.2%	**2.3±0.2**%	**2.2±0.2**%
100000	2.9±0.2%	2.4±0.2%	2.7±0.2%	≈**1.9**%	≈**1.9**%
	Donor				
	Linear	HMM	LIK	FK	TOP
100	14.9±1.6%	16.8±0.6%	**11.0±0.3**%	26.6±6.4%	29.7±0.5%
1000	8.9±0.3%	9.4±0.6%	**7.6±0.5**%	11.5±1.0%	19.8±1.0%
10000	6.3±0.1%	**5.3±0.5**%	6.3±0.3%	**5.2±0.4**%	5.6±0.3%
100000	5.3±0.2%	**5.0±0.4**%	**5.0±0.1**%	n.a.	n.a.

Sensitivity of our methods on *C. elegans*					
	Acceptor				
	Linear	HMM	LIK	FK	TOP
100	90.4±0.3%	90.1±2.1%	91.5±0.9%	84.6±10.5%	69.0±7.5%
1000	94.4±0.4%	96.7±0.3%	93.7±1.1%	96.8±0.2%	93.4±1.7%
10000	94.2±1.1%	96.7±0.7%	95.2±0.8%	96.8±0.6%	96.8±0.4%
100000	95.7±0.9%	97.0±0.8%	96.5±0.8%	n.a.	n.a.
	Donor				
	Linear	HMM	LIK	FK	TOP
100	87.4±8.0%	62.8±2.2%	82.9±7.2%	43.7±9.9%	00.0±0.0%
1000	83.4±2.4%	82.2±2.6%	84.7±2.2%	87.5±2.2%	80.2±3.6%
10000	85.2±0.9%	90.0±1.7%	88.5±0.9%	90.5±1.2%	90.6±1.4%
100000	90.0±1.0%	91.0±1.3%	90.3±0.9%	n.a.	n.a.

Specificity of our methods on *C. elegans*					
	Acceptor				
	Linear	HMM	LIK	FK	TOP
100	92.8±0.9%	88.7±0.4%	92.9±1.0%	88.5±3.7%	82.8±5.3%
1000	95.3±0.3%	97.5±0.2%	95.4±0.8%	97.4±0.4%	96.0±0.8%
10000	97.4±0.4%	98.0±0.3%	96.7±0.6%	98.3±0.2%	98.4±0.1%
100000	98.0±0.4%	98.0±0.3%	97.8±0.5%	n.a.	n.a.
	Donor				
	Linear	HMM	LIK	FK	TOP
100	82.8±7.0%	91.9±1.3%	91.6±3.2%	85.9±11.8%	100.0±0.0%
1000	94.4±1.1%	94.1±1.2%	95.7±1.0%	88.9±1.8%	80.2±2.7%
10000	97.3±0.4%	96.7±0.8%	96.0±0.5%	96.7±0.4%	96.0±0.6%
100000	96.7±0.3%	96.6±0.2%	97.0±0.4%	n.a.	n.a.

Table 4.2: Test errors of our 4 methods on 100-10000 examples on the acceptor and donor classification task. The shown results are averaged over 5 different runs and the standard deviation is given.

On donor sites both kernels cannot improve the result of the HMM (FK 5.2%, TOP 5.6%) (cf. Table 4.2). Since the number of dimensions is $\approx$ 5700 (acceptor) and $\approx$ 8300 (donor) and the number of support vectors (SVs) is more than twice as large on donor sites (on acceptor sites the TOP and FK SVM solutions had $\approx$ 1000 SVs while on donor sites the FK had $\approx$ 2000 and TOP $\approx$ 3000 SVs) we think, the SVM will require more samples to obtain a better solution. Furthermore we had to normalise the feature vectors as explained in Section 3.1.3 without which the SVMs optimiser ran into numerical problems (see Table 3.1 how the performance changes when different normalisation techniques are used).

4.2.6 Comparison and Discussion

In the experiments we used different machine learning techniques, like linear and fine tuned HMMs and the Locality Improved, Fisher and TOP Kernel embedded into SVMs.

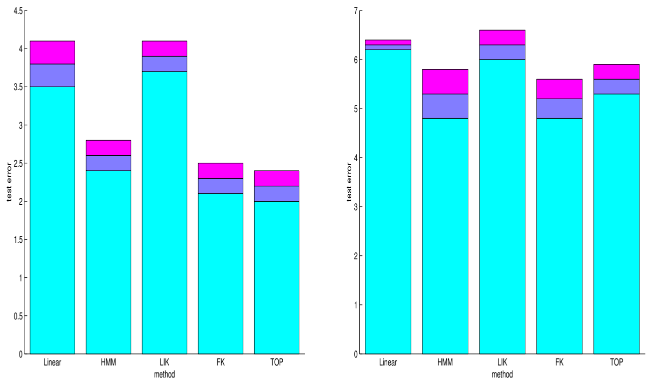


Figure 4.6: Performance of our methods on acceptor (left) and donor (right) sites on 10000 samples.

To compare these methods, the test error on acceptor and donor sites on 10000 samples is shown in Fig. 4.6. In this figure, the violet and blue bars denote the standard deviation around the mean of the test error. As one can see the task to classify acceptor sites seems to be simpler compared to the donor classification task. On the acceptor sites, the best classification results were achieved using the TOP and Fisher kernel, closely followed by the fine tuned HMM. The computationally efficient methods, as the linear models and the LIK perform equally well, but are outperformed by the other techniques. The picture is different for donor sites. There, the TOP and FK methods could not improve the HMM (for possible reasons see above), which thus gives best performance followed by the linear models and FK which again perform equally well.

While we presented the test set error in Table 4.2 we did not yet discuss sensitivity and specificity, which are stated for the above error. Sensitivity, is the ratio [8] between the number of correctly classified positives (true positives (tp)) and the number of all positives (true positives together with false negatives (fn)), i.e.,

$$sensitivity = \frac{tp}{tp + fn}.$$

Specificity on the other hand is the ratio of correctly classified negatives (true negatives (tp)) and the number of all negatives (sum of false positives (fp) and true negatives), i.e.,

$$specificity = \frac{tn}{fp + tn}.$$

While one can observe the same tendencies as for the test error, for both sensitivity and specificity, the sensitivity of TOP on small samples sizes is very low (on the donor almost zero on 100 samples). This clearly indicates that the number of samples is not

enough to obtain a reasonable estimate using SVMs.
Results in terms of specificity and sensitivity are given in [11, 13, 40]. While in these articles different splice site datasets were used and thus performance of our methods cannot be directly compared, we noticed that in all of these papers, results for linear HMMs, while named differently[38], were presented.

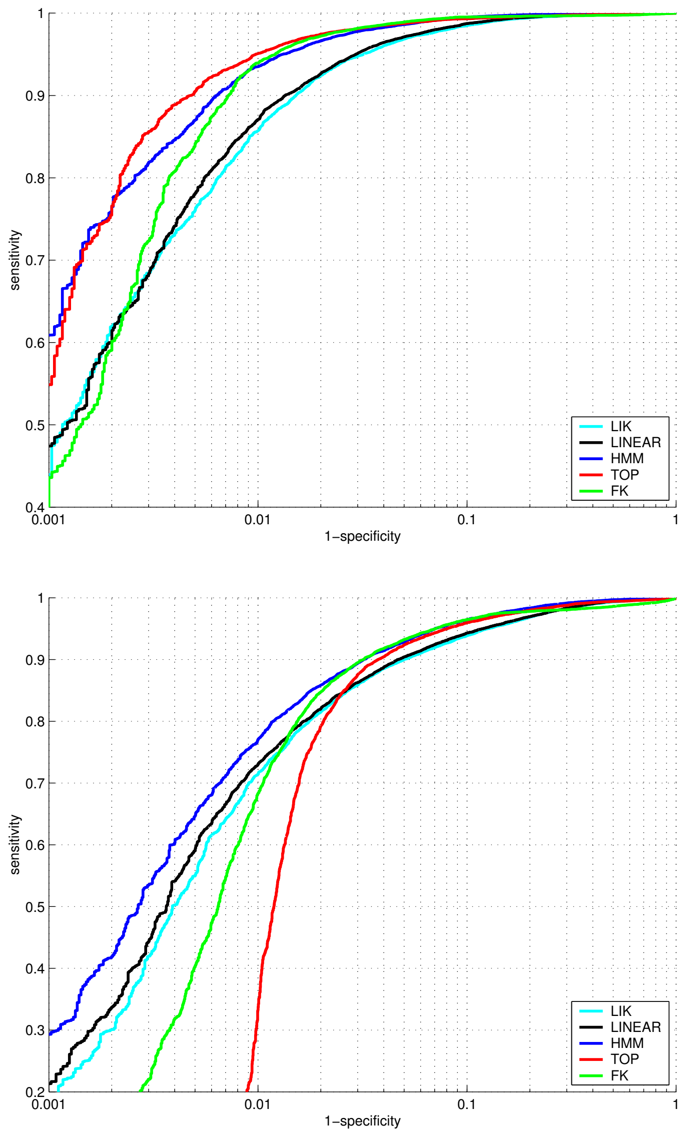


Figure 4.7: ROC Curves for comparison of our classifiers on the acceptor (top) and donor sites (bottom).

To compare the performance of our classifiers, we computed the Receiver Operating Characterstic (ROC) Curve [19], which plots 1-specificity versus sensitivity (this is the same as false positives vs. true positives on a relative scale). We already compared our methods in terms of test-set error. However since the test error largely depends on the number and ratio of positive and negative samples (e.g., when the number of negative samples is much higher than the number of positive samples a classifier can achieve a very small test set error by classifying all samples as negatives) the ROC Curve provides a more data independent performance measure. The sensitivity and specificity as shown in Table 4.2 make up only a single point on the ROC Curve, which is the point of minimal validation set test error. In the ROC one can select a certain specificity and read off the corresponding sensitivity for each classifier. Hence one can choose a certain threshold which best fits the problem (usually the costs for false positives and false negatives determine the threshold). What does the ROC tell about the performance of our methods? First of all, in tendency, the higher the sensitivity and specificity the better the classifier. While we again observe which methods outperform others, as already discussed in our comparison of the test error, some interesting properties need to be emphasised: In both curves of the LIK and the linear HMM perform almost identically. However in Table 4.2 the LIK outperforms the linear method when the number of samples becomes larger. This coincides with the observation of [35]. It furthermore

[38]In [40] they are called (Conditional) Consensus Matrices, in [13] they are referred to as Weighted Matrix- and Weighted Array Models, while they are named Independent and Chain in [11].

suggests to investigate performance of the Fisher and TOP Kernel derived from linear HMMs and even to create a stand alone kernel from linear HMMs, which can be computed as efficient as the LIK.

Another interesting property is that the TOP and FK remain below the performance level of the HMM up to a certain threshold (for the acceptor case). Thus depending on the desired specificity one has to choose the classifier which gives highest sensitivity.

4.3 Human

4.3.1 Preliminary Results on the Human Genome

For now, we did not test all of our methods on human splice sites, but only investigated the performance of linear models. As one can see, the maximum sample size is only 25000. While we could utilise 100000 samples on *C. elegans*, the number of EST confirmed sequences is still much lower on the human. However we generated the data in the same way as we did for the nematode. In this process we achieved very similiar ratios of

Performance of linear HMMs on Human Splice Sites

	Acceptor			Donor		
	Test Error	Sens.	Spec.	Test Error	Sens.	Spec.
100	12.5±1.4%	77.2±5.5%	92.0±0.6%	9.6±0.7%	83.6±2.7%	93.2±1.8%
1000	5.5±0.5%	91.6±1.1%	95.8±0.5%	5.4±0.2%	90.4±0.6%	96.3±0.4%
10000	3.7±0.2%	94.1±1.0%	97.3±4.3%	3.2±0.2%	94.5±0.8%	97.7±0.3%
25000	3.4±0.1%	94.2±6.9%	97.6±0.4%	2.7±0.1%	94.8±0.5%	98.3±0.2%

Table 4.3: Test errors of linear HMMs on 100-10000 examples on the human Acceptor and Donor sites. The shown results are averaged over 5 different runs and the standard deviation is given.

positive and negative samples. When looking at Table 4.3 one notices that in comparison to the worm the linear HMMs achieved better performance on donor samples than on the acceptor. This might indicate a certain intrinsic property of the human donor sites, which cannot be observed on *C. elegans*. However it might as well be an artifact caused by not enough or not identically distributed training samples.
Since the linear HMMs trained on human splice sites perform equally well compared with linear HMMs trained on the nematode, one would expect that our other machine learning techniques also achieve the very same good performance on human DNA.

5 Conclusion and Outlook

In our work, we presented new methods for splice site recognition. At first, we introduced the reader to genetics and machine learning (ML) (cf. Section 1.2 and 2), where we focused on two successful ML approaches: the *Hidden Markov Model* and the *Support Vector Machine* (cf. Section 2.2 and 2.3). We explained how prior knowledge can be incorporated into SVMs using kernel functions like the Locality Improved Kernel, and the Fisher and TOP kernels (see Section 2.4). While the LIK is directly applicable to SVM learning, the TOP and Fisher kernels, defined on generative models, have to be derived from the generative model that is used. We computed these kernels from HMMs (cf. Appendix A.1) and demonstrated how one can successfully combine HMMs with discriminative SVM learning using the Fisher and TOP kernels.

We proved that the already established Locality Improved Kernel is a special case of the Fisher Kernel derived from certain HMMs and thus provided new insight into the relationship between the Fisher and the LIK (cf. Appendix A.2). Moreover, we demonstrated that the FK derived from fine tuned HMMs performs favourably.
These propositions are supported by extensive experiments on splice sites. In the experimental section (see Section 4), we first compared our SVM methods to a number of standard machine learning techniques on a benchmark dataset and pointed out that our suggested methods are superior by far to previous approaches.
We underscored the importance of careful data selection and data analysis before actually applying ML methods (see Section 3). We described in detail the process of extracting splice sites and decoys to obtain training and test samples and analysed these datasets for the nematode *C.elegans* and the human species. We applied linear HMMs to these datasets and showed that one can already achieve good results using these simple models when performing cross validation over a large set of parameters. We trained fully connected HMMs using the Baum Welch algorithm on the data sets and pointed out, in a very detailed manner, why greedy application of the Baum Welch algorithm fails and described ways to overcome this problem. As one possible "workaround", we simplified the model structure by concatenating linear and fully connected HMMs and showed that these specially developed HMMs are well suited for the detection of splice sites. We incorporated the prior knowledge of these fine tuned HMMs into SVMs using the TOP and Fisher kernels and thus demonstrated that a combination of generative and discriminative learning machines can lead to even better classifiers in *practice.*

Future work will focus on enhancing the ML techniques for splice site detection. On the one hand, we will focus on other training algorithms for HMMs, e.g., we will train HMMs using other assumptions about the dependence between the observation sequences (e.g., uniform dependence cf. [53]) and simulated annealing approaches to overcome the problem of local minima. We will enhance the `genefinder` program, such that prior knowledge in the structure of HMMs can be taken into account. For example, we can force the HMM to be in a fixed state (or to give fixed output) at a certain time. This will allow training of certain parameters only and will implicitly compute the TOP and Fisher Kernel to allow higher order HMMs to be taken into account. On the other hand, we will try to improve the SVM solution by giving greater weight to "important" dimensions in the TOP and Fisher feature vectors or by using other preprocessing techniques like Principial Component Analysis (PCA). We will make use of other kernels like, e.g., String Kernels [31] and try other graphical models, like, e.g., Bayes Nets that might be better suited to incorporating prior knowledge about the splicing task.

Appendix

A Proofs

A.1 The Gradient of the Model Probability

Since the derivatives of $\Pr[\mathbf{o}|\boldsymbol{\theta}]$ for given $\mathbf{o}$ and $\boldsymbol{\theta}$ with resprect to all $\theta_i^\star$ are needed for computation of, e.g., Fisher Kernel and TOP Kernel Scores their derivation follows:

Lemma A.1. *The derivatives of the model probability* (2.1) *are given by:*

a) $\dfrac{\partial \Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial p_k} = b_{k\,o_0} \cdot \beta_0^k$

b) $\dfrac{\partial \Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial q_k} = \alpha_{T-1}^k$

c) $\dfrac{\partial \Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial a_{kl}} = \sum_{t=0}^{T-2} \alpha_t^k \cdot b_{s_l,o_{t+1}} \cdot \beta_{t+1}^l$

d) $\dfrac{\partial \Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial b_{kl}} = \sum_{t=0}^{T-1} \delta_{o_t\,z_l} \dfrac{\alpha_t^k \cdot \beta_t^k}{b_{s_k,o_t}}$

Proof. (a)

$$\begin{aligned}
\partial\frac{\Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial p_k} &= \frac{\partial}{\partial p_k}\Pr[o_o, o_1, \ldots, o_{T-1}|\boldsymbol{\theta}] \\
&\overset{(2.2)}{=} \frac{\partial}{\partial p_k}\left(\sum_{i=0}^{n-1}\Pr[o_0, s_0 = z_i|\boldsymbol{\theta}] \cdot \Pr[o_1, \ldots, o_{T-1}|\boldsymbol{\theta}, s_0 = z_i]\right) \\
&\overset{(2.4)}{=} \frac{\partial}{\partial p_k}\left(\sum_{i=0}^{n-1} p_i b_{i,o_o} \cdot \beta_0^i\right) \\
&= \left(\sum_{i=0}^{n-1}\delta_{ik} b_{i,o_o} \cdot \beta_0^i\right) = b_{k,o_o} \cdot \beta_0^k
\end{aligned}$$

□

Proof. (b)

$$\begin{aligned}
\partial\frac{\Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial q_k} &= \frac{\partial}{\partial q_k}\left(\sum_{i=0}^{n-1} Pr[o_0, \ldots, o_{T-1}, s_{T-1} = z_i|\boldsymbol{\theta}'] \cdot \Pr[\text{stop}|\boldsymbol{\theta}', s_{T-1} = z_i]\right) \\
&= \frac{\partial}{\partial q_k}\left(\sum_{i=0}^{n} \alpha_{T-1}^i \cdot q_i\right) = \left(\sum_{i=0}^{n-1}\delta_{ik}\alpha_{T-1}^i\right) = \alpha_{T-1}^k
\end{aligned}$$

□

Before proving (c) and (d), we will prepare the following lemma.

Lemma A.2 (Partial derivatives of a polynomial). *Let* $\mathbf{x} = \{x_1, x_2, \ldots, x_m\}$ *be a set of variables and* $p(\mathbf{x})$ *a polynomial defined as follows:*

$$p(\mathbf{x}) = p(x_1, \ldots, x_n) = \prod_{i=1}^{n} x_{s_i} \text{ with } s_i \in \{1, \ldots, m\}$$

Then the partial derivative of the polynomial with respect to x_k *is given by:*

$$\frac{\partial}{\partial x_k} p(\mathbf{x}) = \frac{\partial}{\partial x_k} \prod_{i=1}^{n} x_{s_i} = \sum_{i=1}^{n} \delta_{s_i,k} \prod_{j=1,j\neq i}^{n} x_{s_j}$$

Proof. We proof by induction:
$n = 1:$

$$\frac{\partial}{\partial x_k} x_{s_1} = \delta_{s_1,k} = \begin{cases} 1, & s_1 = k \\ 0, & \text{otherwise} \end{cases}$$

$n \mapsto n+1:$

$$\begin{aligned} \frac{\partial}{\partial x_k} \prod_{i=1}^{n+1} x_{s_i} &= \frac{\partial}{\partial x_k} \left(\prod_{i=1}^{n} x_{s_i} \right) x_{s_n+1} + \left(\prod_{i=1}^{n} x_{s_i} \right) \frac{\partial}{\partial x_k} x_{s_{n+1}} \\ &= \sum_{i=1}^{n} \delta_{s_i,k} \prod_{j=1,j\neq i}^{n+1} x_{s_j} + \prod_{i=1}^{n} x_{s_i} \delta_{s_{n+1},k} \\ &= \sum_{i=1}^{n+1} \delta_{s_i,k} \prod_{j=1,j\neq i}^{n} x_{s_j} \end{aligned}$$

The first equation follows directly from the application of the product rule and the second follows by induction. □

This lemma turns out to be very useful when calculating the partial derivatives with respect to a_{ij} and b_{ij} :

Proof. (c) of Lemma A.1.

$$\frac{\partial \Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial a_{kl}}$$

$$= \frac{\partial}{\partial a_{kl}} \left(\sum_{\text{all } s_0,\ldots,s_{T-1}} p_{s_0} b_{s_0,o_0} \left(\prod_{t=0}^{T-2} a_{s_t,s_{t+1}} b_{s_{t+1},o_{t+1}} \right) q_{s_{T-1}} \right) \tag{A.1}$$

$$= \sum_{\text{all } \mathbf{s}} p_{s_0} b_{s_0,o_0} q_{s_{T-1}} \prod_{t=0}^{T-2} \left(b_{s_{t+1},o_{t+1}} \right) \cdot \frac{\partial}{\partial a_{kl}} \left(\prod_{t=0}^{T-2} a_{s_t,s_{t+1}} \right) \tag{A.2}$$

$$= \sum_{\text{all } \mathbf{s}} p_{s_0} b_{s_0,o_0} q_{s_{T-1}} \prod_{t=0}^{T-2} \left(b_{s_{t+1},o_{t+1}} \right) \cdot \left(\sum_{t=0}^{T-2} \delta_{s_t,z_k} \delta_{s_{t+1},z_l} \prod_{\tau=0,\tau\neq t}^{\tau=T-2} a_{s_\tau s_{\tau+1}} \right) \tag{A.3}$$

$$= \sum_{\text{all } \mathbf{s}} p_{s_0} b_{s_0,o_0} \cdot \left(\sum_{t=0}^{T-2} \delta_{s_t,z_k} \delta_{s_{t+1},z_l} b_{s_{t+1},o_{t+1}} \prod_{\tau=0,\tau\neq t}^{\tau=T-2} b_{s_{\tau+1},o_{\tau+1}} a_{s_\tau,s_{\tau+1}} \right) q_{s_{T-1}} \tag{A.4}$$

$$= \sum_{t=0}^{T-2} \left(\sum_{\text{all } \mathbf{s}} p_{s_0} b_{s_0,o_0} \cdot \delta_{s_t,z_k} \delta_{s_{t+1},z_l} b_{s_{t+1},o_{t+1}} \prod_{\tau=0,\tau\neq t}^{\tau=T-2} \left(b_{s_{\tau+1},o_{\tau+1}} a_{s_\tau,s_{\tau+1}}\right) q_{s_{T-1}} \right) \quad (A.5)$$

$$= \sum_{t=0}^{T-2} \left(\sum_{\text{all } \mathbf{s}, s_t=z_k, s_{t+1}=z_l} p_{s_0} b_{s_0,o_0} b_{l,o_{t+1}} \prod_{\tau=0,\tau\neq t}^{\tau=T-2} \left(b_{s_{\tau+1},o_{\tau+1}} a_{s_\tau,s_{\tau+1}}\right) q_{s_{T-1}} \right) \quad (A.6)$$

$$= \sum_{t=0}^{T-2} \left(b_{lo_{t+1}} \left(\sum_{s_0,\dots,s_{t-1},s_t=z_k} p_{s_0} b_{s_0,o_0} \prod_{\tau=0}^{\tau=t-1} b_{s_{\tau+1},o_{\tau+1}} a_{s_\tau,s_{\tau+1}} \right) \cdot \right.$$

$$\left. \left(\sum_{s_{t+1}=z_l,s_{t+2},\dots,s_{T-1}} \prod_{\tau=t+1}^{\tau=T-2} b_{s_{\tau+1},o_{\tau+1}} a_{s_\tau,s_{\tau+1}} q_{s_{T-1}} \right) \right) \quad (A.7)$$

$$= \sum_{t=0}^{T-2} \left(\Pr[o_0,\dots,o_t,s_t=z_k|\boldsymbol{\theta}] \cdot b_{l,o_{t+1}} \cdot \Pr[o_{t+2},\dots,o_{T-1}|\boldsymbol{\theta},s_{t+1}=z_l] \right) \quad (A.8)$$

$$= \sum_{t=0}^{T-2} \left(\alpha_t^k \cdot b_{l,o_{t+1}} \cdot \beta_{t+1}^l \right) \quad (A.9)$$

We are summing over all $s_0,\dots,s_{T-1}$, i.e., all $\mathbf{s} \in \mathbf{z} \times \mathbf{z} \times \cdots \times \mathbf{z} = \mathbf{z}^T$. Eq. (A.1) follows from Lemma 2.5. Then we pull out the independent variables, such that we can apply Lemma A.2 and end up with Eq. (A.3). After rearranging the $b_{s_t,o_t}-$ product back into the $a_{s_\tau,s_{\tau+1}}$ product and switching the order of summation we obtain Eq. (A.5). The application of Kronecker's δ leads to Eq. (A.6) such that we can split up the summation as in Eq. (A.7). The first term in the small brackets that is summed up is the joint probability of the state sequence $s_0,\dots,s_{t-1},z_k$ and the observation sequence $o_0,\dots,o_{t-1}$ (for the second term $z_l,s_{t+2},\dots,s_{T-1}$ and $o_{t+2},\dots,o_{T-1}$ respectively). Thus the summation over the state sequences marginalises out leading to Eq. (A.8). We conclude with Eq. (A.9) by using the definition of the forward and backward variables. □

Proof. (d)

$$\partial\frac{\Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial b_{kl}}$$

$$= \sum_{\text{all } s_0,\dots,s_{T-1}} \frac{\partial}{\partial b_{kl}} \left(\prod_{t=0}^{T-1} b_{s_t,o_t} \right) \cdot \left(p_{s_0} \prod_{t=0}^{T-2} a_{s_t,s_{t+1}} q_{s_{T-1}} \right) \quad (A.10)$$

$$= \sum_{\text{all } s_0,\dots,s_{T-1}} \sum_{t=0}^{T-1} \delta_{s_t,z_k} \delta_{o_t,e_l} \left(\prod_{\tau=0,\tau\neq t}^{T-1} b_{s_\tau,o_\tau} \right) \cdot \left(p_{s_0} \prod_{t=0}^{T-2} a_{s_t,s_{t+1}} q_{s_{T-1}} \right) \quad (A.11)$$

$$= \sum_{t=0}^{T-1} \delta_{o_t,e_l} \sum_{\text{all } \mathbf{s}, s_t=z_k} b_{s_0,o_0} \left(\prod_{\tau=0,\tau\neq t-1}^{T-2} b_{s_{\tau+1},o_{\tau+1}} \right) \cdot \left(p_{s_0} \prod_{t=0}^{T-2} a_{s_t,s_{t+1}} q_{s_{T-1}} \right) \quad (A.12)$$

$$= \sum_{t=0}^{T-1} \delta_{o_t,e_l} \sum_{\text{all } \mathbf{s}, s_t=z_k} \frac{1}{b_{s_t,o_t}} \cdot \left(p_{s_0} b_{s_0,o_0} \prod_{t=0}^{T-2} a_{s_t,s_{t+1}} b_{s_{t+1},o_{t+1}} q_{s_{T-1}} \right) \quad (A.13)$$

$$= \sum_{t=0}^{T-1} \delta_{o_t,e_l} \sum_{\text{all } \mathbf{s}, s_t=z_k} \frac{\Pr[\mathbf{o},\mathbf{s}|\boldsymbol{\theta}]}{b_{z_k,e_l}} = \sum_{t=0}^{T-1} \delta_{o_t,e_l} \frac{\alpha_t^k \cdot \beta_t^k}{b_{z_k,e_l}} \quad (A.14)$$

Similar to the previous proof we rearrange the model probability into two parts: A part depending on b_{kl} and one independent of b_{kl} which gives Eq. (A.10). We then apply Lemma A.2 in Eq. (A.11), change the order of the summations, apply Kronecker's δ, pull out b_{s_0,o_0} which results in Eq. (A.12). The combination of the products leads to Eq. (A.13). The term in brackets is the path probability, that is summed over all paths in which the model is at time t in state z_k, concluding in Eq. (A.14). □

Lemma A.3. *Defining $\#x$ as the number of occurences of x in $\Pr[\mathbf{s},\mathbf{o}|\boldsymbol{\theta}]$, the derivatives of the probability of a given state sequence are given by:*

a) $$\frac{\partial \Pr[\mathbf{s},\mathbf{o}|\boldsymbol{\theta}]}{\partial p_k} = \delta_{s_0,z_k}\frac{\Pr[\mathbf{s},\mathbf{o}|\boldsymbol{\theta}]}{p_k}$$

b) $$\frac{\partial \Pr[\mathbf{s},\mathbf{o}|\boldsymbol{\theta}]}{\partial q_k} = \delta_{s_{T-1},z_k}\frac{\Pr[\mathbf{s},\mathbf{o}|\boldsymbol{\theta}]}{q_k}$$

c) $$\frac{\partial \Pr[\mathbf{s},\mathbf{o}|\boldsymbol{\theta}]}{\partial a_{kl}} = (\#a_{kl})\frac{\Pr[\mathbf{s},\mathbf{o}|\boldsymbol{\theta}]}{a_{kl}}$$

d) $$\frac{\partial \Pr[\mathbf{s},\mathbf{o}|\boldsymbol{\theta}]}{\partial b_{kl}} = (\#b_{kl})\frac{\Pr[\mathbf{s},\mathbf{o}|\boldsymbol{\theta}]}{b_{kl}}$$

Proof. Recall that $\Pr[\mathbf{s},\mathbf{o}|\boldsymbol{\theta}] = p_{s_0}b_{s_0,o_0}\left(\prod_{t=0}^{T-2} a_{s_t,s_t+1}b_{s_{t+1},o_{t+1}}\right)q_{s_{T-1}}$. Eq. (a) and Eq. (b) are correct, since the derivative is either zero when $s_0 \neq z_k$ respectively $s_{T-1} \neq z_k$ or it is the path probability divided by p_k respectively q_k. Note that $p_k, q_k \neq 0$ for paths with probability > 0. Since we calculate the derivatives of the most probable path in practice, this should be the case (otherwise, the derivatives are simply 0).
Eq. (c) and Eq. (d) follow directly from $\frac{\partial x^n \cdot p(\mathbf{y})}{\partial x} = n \cdot x^{n-1} \cdot p(\mathbf{y}) = n \cdot \frac{p(\mathbf{y})}{x}$. □

Lemma A.4. *The derivatives of the model probability of a linear HMM (cf. Definition 2.13), i.e., a HMM where the underlying chain is linear by setting $n = T$, $p_0 = 1$, $a_{i,i+1} = 1$, $a_{n-1,j} = 0$, and $q_{n-1} = 1$ are given by:*

$$\frac{\partial \Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial b_{kl}} = \delta_{o_k,e_l}\frac{\Pr[\mathbf{o}|\boldsymbol{\theta}]}{b_{kl}}$$

We use linear HMMs extensively in the experimental section, due to their simplicity and therefore very fast training methods.

A.2 Connection between Locality Improved and Fisher Kernel

Instead of presenting the whole proof, we will start with the simplest non-trivial example. Later on we briefly explain how one would extend this example to the general case.

Example A.5. *We will give an example for the simplest non-trivial case, i.e., the parameters of the LIK are set to $l = 1, d_1 = 2, d_2 = 1$. First we will derive the feature vectors of the LIK, then we introduce you to the structure of a Markov Model from that we derive the FK. We then adjust the parameters of the MM such that its FK satisfies the claim.*

Recall the definition of the Locality Improved Kernel:

$$K_{LI}(\mathbf{x}, \mathbf{x}') = \left(\sum_{t=0}^{T-1} \left(\sum_{j=-l}^{+l} w_j I_{t+j}(\mathbf{x}, \mathbf{x}') \right)^{d_1} \right)^{d_2}$$

One can see that the LIK is not well defined for the boundaries (i.e., for $t = 0$, $j = -1$ I_{0-1} is not defined). As a result it is used as

$$K_{LI}(\mathbf{x}, \mathbf{x}') = \left(\sum_{t=0+l}^{T-l-1} \left(\sum_{j=-l}^{+l} w_j I_{t+j}(\mathbf{x}, \mathbf{x}') \right)^{d_1} \right)^{d_2}$$

in experiments. Doing so we find all of the windows lying completely within the boundaries. For convenience we use $I_t := I_t(\mathbf{x}, \mathbf{x}')$ and $I_{r,s,t} = I_r I_s I_t$ synonymously:

$$\begin{aligned}
K_{LI}(\mathbf{x}, \mathbf{x}') &= \sum_{t=1}^{T-2} \left(\sum_{j=-1}^{+1} w_j I_{t+j}(\mathbf{x}, \mathbf{x}') \right)^2 \\
&= \sum_{t=1}^{T-2} (w_{-1} I_{t-1} + w_0 I_t + w_{+1} I_{t+1})^2 \\
&= \sum_{t=1}^{T-2} \Big(w_{-1}^2 I_{t-1} + w_0^2 I_t + w_{+1}^2 I_{t+1} \\
&\qquad\qquad + 2w_{-1}w_0 I_{t-1,t} + 2w_{-1}w_{+1} I_{t-1,t+1} + 2w_0 w_{+1} I_{t,t+1} \Big) \\
&= \widetilde{a_0}^2 I_0 + \widetilde{a_1}^2 I_1 + \sum_{t=2}^{T-1} \widetilde{a_t}^2 I_t + \widetilde{a_{T-2}}^2 I_{T-2} + \widetilde{a_{T-1}}^2 I_{T-1} \\
&\qquad + \widetilde{a_{01}}^2 I_{01} + \sum_{t=2}^{T-2} \widetilde{a_{t-1,t}}^2 I_{t-1,t} \\
&\qquad + \sum_{t=1}^{T-2} \widetilde{a_{t-1,t+1}}^2 I_{t-1,t+1} + \widetilde{a_{T-2,T-1}}^2 I_{T-2,T-1}
\end{aligned}$$

where $\widetilde{a_0}^2 = w_{-1}^2$, $\widetilde{a_1}^2 = (w_{-1}^2 + w_0^2)$, $\widetilde{a_t}^2 = (w_{-1}^2 + w_0^2 + w_{+1}^2)$, $\widetilde{a_{T-1}}^2 = (w_0^2 + w_{+1}^2)$, $\widetilde{a_{T-1}}^2 = w_1^2$ and $\widetilde{a_{01}}^2 = 2w_{-1}w_0$, $\widetilde{a_{t-1,t}}^2 = 2(w_{-1}w_0 + w_0 w_{+1})$, $\widetilde{a_{t-1,t+1}}^2 = 2w_{-1}w_{+1}$

and $\widetilde{a_{T-2,T-1}}^2 = 2w_0 w_{+1}$.

As we see one can isolate the matching function I_t for all times t, such that there is a constant factor in front of it, i.e., if for fixed, but arbitrary times t the observations match $x_t = x'_t$, the value that contributes to the LIK is constant *regardless of the symbols that match. To write the LIK as inner product of a score vector* $\mathbf{s}(\mathbf{x})$, *i.e., as*

$$K_{LI}(\mathbf{x},\mathbf{x}') = \mathbf{s}(\mathbf{x}) \cdot \mathbf{s}(\mathbf{x}')$$

we need to model the match function $I_t(\mathbf{x},\mathbf{x}')$ such that it can be written as dot product of two vectors. Thus we use the binary *representation of the each symbol in vector* $\mathbf{x}$, *i.e., we define*

$$\Delta_t(\mathbf{x}) = (\delta_{x_t,e_0}, \delta_{x_t,e_1}, \ldots, \delta_{x_t,e_{m-1}})^\top.$$

This way we succesfully modelled $I_t(\mathbf{x},\mathbf{x}') = \Delta_t(\mathbf{x}) \cdot \Delta_t(\mathbf{x}')$, where $\Delta_t(\mathbf{x})$ depends only on $\mathbf{x}$ and $\Delta_t(\mathbf{x}')$ only on $\mathbf{x}'$. Matchings of more than one symbol are modeled by

$$I_{r,s,t}(\mathbf{x},\mathbf{x}') = \Delta_{r,s,t}(\mathbf{x}) \cdot \Delta_{r,s,t}(\mathbf{x}'),$$

where

$$\Delta_{r,s,t}(\mathbf{x}) = (\Delta_r(\mathbf{x})^\top,\ \Delta_s(\mathbf{x})^\top,\ \Delta_t(\mathbf{x})^\top)^\top.$$

Thus we can proceed to rewrite the LIK as dot product

$$K_{LI}(\mathbf{x},\mathbf{x}') = \begin{pmatrix} \widetilde{a_0}\Delta_0(\mathbf{x}) \\ \widetilde{a_1}\Delta_1(\mathbf{x}) \\ \vdots \\ \widetilde{a_{T-1}}\Delta_{T-1}(\mathbf{x}) \\ \widetilde{a_{01}}\Delta_{01}(\mathbf{x}) \\ \widetilde{a_{12}}\Delta_{12}(\mathbf{x}) \\ \vdots \\ \widetilde{a_{T-2,T-1}}\Delta_{T-2,T-1}(\mathbf{x}) \\ \widetilde{a_{02}}\Delta_{02}(\mathbf{x}) \\ \widetilde{a_{13}}\Delta_{13}(\mathbf{x}) \\ \vdots \\ \widetilde{a_{T-3,T-1}}\Delta_{T-3,T-1}(\mathbf{x}) \end{pmatrix} \cdot \begin{pmatrix} \widetilde{a_0}\Delta_0(\mathbf{x}') \\ \widetilde{a_1}\Delta_1(\mathbf{x}') \\ \vdots \\ \widetilde{a_{T-1}}\Delta_{T-1}(\mathbf{x}') \\ \widetilde{a_{01}}\Delta_{01}(\mathbf{x}') \\ \widetilde{a_{12}}\Delta_{12}(\mathbf{x}') \\ \vdots \\ \widetilde{a_{T-2,T-1}}\Delta_{T-2,T-1}(\mathbf{x}') \\ \widetilde{a_{02}}\Delta_{02}(\mathbf{x}') \\ \widetilde{a_{13}}\Delta_{13}(\mathbf{x}') \\ \vdots \\ \widetilde{a_{T-3,T-1}}\Delta_{T-3,T-1}(\mathbf{x}') \end{pmatrix}. \tag{A.15}$$

Recall the definition of the Fisher Kernel for some model $\boldsymbol{\theta}$:

$$K_{Fisher}(\mathbf{x},\mathbf{x}') = \mathbf{s}(\mathbf{x},\boldsymbol{\theta})^\top \mathbf{Z}(\boldsymbol{\theta})^{-1}\mathbf{s}(\mathbf{x}',\boldsymbol{\theta}),$$

where $s(\mathbf{x},\boldsymbol{\theta}) = \nabla \log \Pr[\mathbf{x}|\boldsymbol{\theta}]$ is the score vector and $\mathbf{Z}(\boldsymbol{\theta})$ is the Fisher Information Matrix. In practice $\mathbf{Z}$ is often approximated as $Z_{ij}(\boldsymbol{\theta}) = \sigma_i^2\delta_{i,j}$ or even $\mathbf{Z} = \mathbf{1}_D$, where D is the number of parameters.
When using the latter approximation, one can design a Markov Model (MM) such that the corresponding Fisher Kernel is equal *to the Locality Improved Kernel. To model the match function I, we consider a MM which consists of 3 submodels of $m \cdot T$ states each, i.e., for each t there are m (the number of observations) many states, as in Fig. A.1. The first model is of order* 1, *the second of order* 2 *and the third of order* 3.[39] *Each*

[39]We will use the transition probabilities to model the match function I. That is why the order of the mixture of MMs appears to be off by one. By doing so these MM are in one to one correspondence

submodel has a fixed start and end-state. The submodels are connected via a 'super'-start state and end in some 'super'-end state, where the new parameters are constant p_1*,* p_2*,* p_3*. We name the* $(m \cdot T)^2$ *parameters of the first order markov chain,* a_{ij} *the* $(m \cdot T)^3$ *and* $(m \cdot T)^4$ *parameters of the second order and third order MM* a_{ijk} *and* a_{ijkl} *respectively, where* $0 \leq i,j,k,l \leq m \cdot T$*. We allow state switches only from 'time-slice to time-slice',i.e.,* $\forall i > j + m \;\vee\; i < j - m : a_{ij} = 0$*, where states from* $(t-1) \cdot m$ *to* $t \cdot m - 1$ *belong to time-slice* t*. In an analogous manner the parameters for the higher order MM are set, i.e,*

$$\begin{aligned}\forall_{ijkl} \quad (\quad & i > j+m \;\vee\; i < j-m \;\vee \\ & j > k+m \;\vee\; j < k-m \;\vee \\ & k > l+m \;\vee\; k < l-m) : a_{ijkl} = 0.\end{aligned}$$

The remaining parameters will be set later. Thus the number of actual parameters is $\sum_{M=1} 3m^2(T-M)$*. We define a meaning for each state, i.e., each state corresponds to some emission* e*. This way we can model an observation sequence* $\mathbf{o}$ *with the corresponding state sequence* $\mathbf{s}$*. Having a closer look at this MM we find that there are three possible ways of generating the same observation sequence (one for each* $\boldsymbol{\theta}_M$*). However the state sequence within each submodel is the same. We will therefore refer to* $\mathbf{s}$ *as the path corresponding to the observation sequence, regardless of which submodel we are dealing with. Since the models are connected only in the 'super' start and end states, only states of* one *of the submodels will be traversed.*
Calculating the FK from this MM requires the following steps:

1. *Derive the model probability*
2. *Compute the gradient with respect to the model parameters*
3. *Formulate the FK*

When doing so the model probability is

$$\begin{aligned}
&\Pr[\mathbf{o}|\boldsymbol{\theta}] \\
&= \sum_{M=1}^{3} p_M \Pr[\mathbf{s}|\boldsymbol{\theta}_M] \\
&= \sum_{M=1}^{3} p_M \Pr[s_0 = z_{i_0}, s_1 = z_{i_1}, \ldots, s_{T-1} = z_{i_T-1}|\boldsymbol{\theta}_M] \\
&= \sum_{M=1}^{3} p_M \Pr[s_{T-1} = z_{i_{T-1}}|s_{T-2} = z_{i_{T-2}}, \ldots, s_0 = z_{i_0}, \boldsymbol{\theta}_M] \Pr[s_{T-2} = z_{i_{T-2}}, \ldots, s_0 = z_{i_0}|\boldsymbol{\theta}_M] \\
&= p_1 \prod_{t=0}^{T-1} \Pr[s_{t+1} = z_{i_{t+1}}|s_t = z_{i_t}, \boldsymbol{\theta}_1] + p_2 \prod_{t=0}^{T-1} \Pr[s_{t+1} = z_{i_{t+1}}|s_t = z_{i_t}, s_{t-1} = z_{i_{t-1}}, \boldsymbol{\theta}_2] \\
&\qquad + p_3 \prod_{t=0}^{T-1} \Pr[s_{t+1} = z_{i_{t+1}}|s_t = z_{i_t}, s_{t-1} = z_{i_{t-1}}, s_{t-2} = z_{i_{t-2}}, \boldsymbol{\theta}_3],
\end{aligned}$$

where the last equality follows from the 1st, 2nd and 3rd order property.

to the linear HMMs as introduced in Definition 2.13.

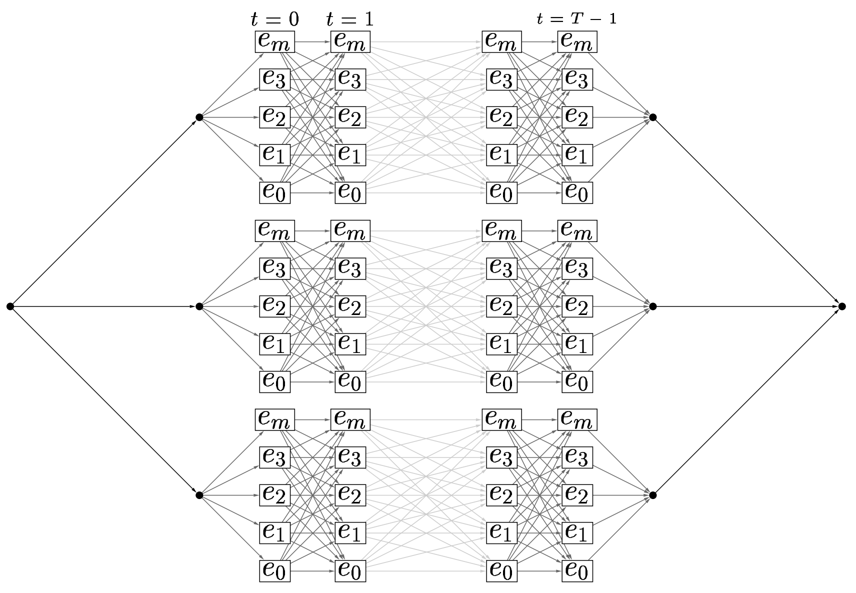


Figure A.1: The structure of the MM of which the Fisher Kernel is equivalent to the Locality Improved Kernel for some $\mathbf{Z}$. This 'super' -model consist of three submodels, one that is first order, one that is second order and one that is third order. Each submodel has $m \cdot T$ states, one for each observation at each time. We do add states ϵ_m that are connected to all the other states. These states do only loop in themselves and finally allow termination. This will satisfy $\mathbf{Z} := c\mathbf{1}_D$ (D is the number of parameters the MM) as one will see later.

Using this information, the derivatives of the log-likelihood of this linear model with respect to the parameters $a_{ij}, a_{ijk}, a_{ijkl} \neq 0$ are given by

$$\frac{\partial \log \Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial a_{ij}} = \frac{p_1}{\Pr[\mathbf{s}|\boldsymbol{\theta}]} \cdot \frac{\partial \Pr[\mathbf{s}|\boldsymbol{\theta}_1]}{\partial a_{ij}} \tag{A.16}$$

$$= \frac{p_1}{\Pr[\mathbf{o}|\boldsymbol{\theta}]} \cdot \delta_{s_t,z_i}\delta_{s_{t+1},z_j} \frac{\Pr[\mathbf{s}|\boldsymbol{\theta}_1]}{a_{ij}} \tag{A.17}$$

$$\frac{\partial \log \Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial a_{ijk}} = \frac{p_2}{\Pr[\mathbf{o}|\boldsymbol{\theta}]} \cdot \delta_{s_{t-1},z_i}\delta_{s_t,z_i}\delta_{s_{t+1},z_k} \frac{\Pr[\mathbf{s}|\boldsymbol{\theta}_2]}{a_{ijk}} \tag{A.18}$$

$$\frac{\partial \log \Pr[\mathbf{o}|\boldsymbol{\theta}]}{\partial a_{ijkl}} = \frac{p_3}{\Pr[\mathbf{o}|\boldsymbol{\theta}]} \cdot \delta_{s_{t-2},z_i}\delta_{s_{t-1},z_j}\delta_{s_t,z_k}\delta_{s_{t+1},z_l} \frac{\Pr[\mathbf{s}|\boldsymbol{\theta}_3]}{a_{ijkl}} \tag{A.19}$$

Looking at the dot product of the gradient as given in (A.17),(A.18),(A.19) for two different observation sequences $\mathbf{x}$ and $\mathbf{x}'$ one realises that we successfully modeled the match function I using Kronecker's δ. We now set the remaining parameters $a_{ij}, a_{ijk}, a_{ijkl}$ to certain values.

$$\forall_{ij} \quad (i-1)m \leq j < im : \quad a_{ij} = \frac{1}{\widetilde{a_i}}$$

$$\forall_{ijk} \quad (i-1)m \leq j < im, (j-1)m \leq k < jm : \quad a_{ijk} = \frac{1}{\widetilde{a_{i,j}}}$$

$$\forall_{ijkl} \quad (i-1)m \leq j < im, (j-1)m \leq k < jm, (k-1)m \leq l < km : \quad a_{ijkl} = \frac{1}{\widetilde{a_{i,j,k}}}$$

As a result the transition probabilities from one slice to another are constant. This

leads to fixed model probabilites $\Pr[\mathbf{o}|\boldsymbol{\theta}_M]$ and finally fixed $\Pr[\mathbf{o}|\boldsymbol{\theta}]$ independent of the choice of $\mathbf{o}$.
However the parameters do not satisfy the condition of stochasticity. We therefore would have to normalise the parameters independently in each submodel by dividing a_{ij} by $m\frac{1}{\widetilde{a}_i}$ a_{ijk} by $m\frac{1}{\widetilde{a}_{ij}}$, and a_{ijkl} by $m\frac{1}{\widetilde{a}_{ijk}}$. Unfortunately this would result in a constant model, i.e., the nonzero parameters become $a_{ij} = a_{ijk} = a_{ijkl} = \frac{1}{m}$. We can work around this limitation, by introducing an additional state ϵ_M for each submodel, which is connected to all of the other states, but is only allowed to transition to itself. Thus the model will loop in that state until it finally terminates. To this state we assign the observation @, i.e., one will find oneself in these states only for some observation that is never observed. When doing so, none of the observation $\mathbf{o}$ corresponding state sequences $\mathbf{s}$ use ϵ_M making all of the partial derivatives zero. Using this trick one can assign certain constant transition probabilities to each 'layer', i.e., we set $a_{ij} = \frac{1}{\widetilde{a}_i} + \xi_i$, $a_{ijk} = \frac{1}{\widetilde{a}_{ij}} + \iota_{i,j}$ and $a_{i,j,k,l} = \frac{1}{\widetilde{a}_{ijk}} + \zeta_{i,j,k}$ where the new variables denote the transition probability to the dead state, e.g., $\xi_i = a_{i,\epsilon_M}$ and so on.[40]
We will show that one can scale each layer of the transition probabilites in the MM independently using the transition probabilities to the dead states, while the model probability of each submodel will still be some constant, which can be shattered by p_M. Thus one can arbitrarly scale the dimensions in the FK score vector that belong to these layers of parameters.
We have to still satisfy the stochasticity constraints, i.e.,

$$1 = \sum_j a_{ij} = \sum_j \frac{1}{\widetilde{a}_i} + \xi_i$$

and so on. Since these condition need to be fulfilled for arbitrary but fixed $\widetilde{a}_i, \ldots$ we introduce a scaling parameter c, with which a_{ij} is multiplied, i.e., $a_{ij} = \frac{1}{c}(\frac{1}{\widetilde{a}_i} + \xi_i)$. To find the correct scalings for the parameters $a_{ij}, \ldots$ and the transition probabilities to the 'dead' states, one has to factor out the constant scaling parameter c, since we can only adjust the scale within each submodel using the transitions to the dead states.
Thus the following linear equations have to be fulfilled:

$$\begin{aligned}
\forall_i \ : c^{1/T} &= \sum_j \frac{1}{\widetilde{a}_i} + \xi_i = m\frac{1}{\widetilde{a}_i} + \xi_i \\
\forall_{i,j} \ : c^{1/T} &= m\frac{1}{\widetilde{\widetilde{a}}_{i,j}} + \iota_{i,j} \\
\forall_{i,j,k} \ : c^{1/T} &= m\frac{1}{\widetilde{\widetilde{\widetilde{a}}}_{i,j,k}} + \zeta_{i,j,k}
\end{aligned}$$

Here c corresponds to the normalisation constant in $\mathbf{Z} = c\mathbf{1}_D$ that occurs T times as a product in each submodel probability and $\xi_i,\ \iota_{i_j},\ \zeta_{i,j,l} \in [0,1]$ are the transition probabilities to the 'dead' states ϵ_M from the model M.
Since none of the transition probabilities to the dead states depend on each other and we can set $c^{1/T}$ to some value larger equal than any on the right hand side, e.g.,

$$\max\left(\frac{m}{\widetilde{a}_i}, \frac{m}{\widetilde{\widetilde{a}}_{ij}}, \frac{m}{\widetilde{\widetilde{\widetilde{a}}}_{ijk}}\right),$$

[40]However when one would want to avoid this 'hack', we had to change the claim to: We show that there exists some constant, i.e., observation independent diagonal matrix $\mathbf{Z}$, instead of $\mathbf{Z} = c\mathbf{1}_D$.

each of these equations can be solved easily. For example, when c is set as in the term above, $\forall_j : a_{ij} = \frac{1}{c\widehat{a_i}}$ and $a_{i,\epsilon_M} = \xi_i$ and so on.
To normalise the non-zero parameters one has to divide the parameters of submodel M by $c^{\frac{1}{T}}$. As a result we can define arbitrary scalings for each layer and conclude in

$$K_{Fisher}(\mathbf{x}, \mathbf{x}') = \nabla \log \Pr[\mathbf{x}|\boldsymbol{\theta}]\; c\mathbf{1}_D\; \nabla \log \Pr[\mathbf{x}'|\boldsymbol{\theta}] = K_{LI}(\mathbf{x}, \mathbf{x}').$$

Proof. of Lemma 2.20: Instead of presenting the whole proof, we will give a brief sketch, which is supported by the above example.

1. Derive the LIK for arbitrary parameters d_1, d_2, l

2. Separate the $I_{i_1,\dots,i_T}$ such that there are constant coefficients $\widetilde{a_j}$ in front of each $I_{i_1,\dots,i_T}$. (This is always possible since the LIK is a polynomial)

3. Construct a Markov Model (MM) similiar to the one in the example (see below). The number of submodels is the largest n in the product of the match functions $I_{i_{t-n}} \cdot \dots \cdot I_i$.

4. Derive the Fisher Kernel from this MM

5. Set the parameters independently for each submodel, such that the submodel probability is constant and the partial derivatives correspond to the $\widetilde{a}$. One can always do so since the parameters in each submodel correspond to certain $\widetilde{a}$ in front of each $I_{i_1,\dots,i_T}$ and the choice of the parameters in each submodel is independent from each other.

6. Normalise the submodel parameters using dead states independently.

□

B Notations

The set of symbols and functions used throughout this thesis:

Π	pseudocounts
$\boldsymbol{\theta}$, $\boldsymbol{\theta}'$	model parameters
$\boldsymbol{\theta}^\star$	optimal model parameters
$\widehat{\boldsymbol{\theta}}$	estimate for model parameters
n	the number of states
m	the number of emissions
$\mathbf{z}$	the set of states
$\mathbf{e}$	the set of emissions
$\mathbf{a}$	$n \times n$ matrix of transition probabilities
$\mathbf{b}$	$n \times m$ matrix of emission probabilities
$\mathbf{p}$	vector of length n containing the start state distribution
$\mathbf{q}$	vector of length n containing the end state distribution
z_i	state i
e_i	emission i
a_{ij}	transition probabilities
b_{ij}	emission probabilities
$\mathbf{s}$	state sequence
$\mathbf{o}$	observation sequence of length T
T	number of symbols in observation sequence
N	number of observation sequences (examples, dimension)
M	the index of the model
α_i^t	forward variables
β_i^t	backward variables
μ_i^t	highest probability along a single path
Ψ_i^t	state backtracking table
D	dimensionality of the space, e.g., $\mathbb{R}^D$
$\mathbf{x}, \mathbf{x}'$	input vectors of dimensionality D
y	class label
Ω	order
d	degree of, e.g., a polynomial
λ	a lagrange multiplier
$\boldsymbol{\lambda}$	the set of lagrange multipliers
L	the lagrangian
C	the regularisation parameter
ξ	a slack variable
$\mathbf{w}$	the normal
b	the bias
$\mathcal{O}$	effort
$\Pr[x]$	probability of x
$\mathrm{E}[x]$	expected value of x
sign	the signum function
δ_{ij}	Kronecker's delta
$K(.,.)$	the kernel function
$\Phi(.)$	feature mapping
$\lVert.\rVert$	two norm
$f(\mathbf{x})$	classifier or function
∂_x	the partial derivative with respect to x, i.e., $\frac{\partial}{\partial x}$

C Genefinder

To perform the many experiments presented here, I wrote (with some help of G. Rätsch) the `genefinder` program. It is based on the program developed in my student research project [47]. We subsequently extended the program in order to take advantage of various learning algorithms, such as SVMs and HMMs.
The program was written in C++ and special emphasise was put on *speed.*
While it consists of a large number of classes, it is basically structured into

(a) Graphical User Interface (GUI),

(b) GUI library,

(c) HMM,

(d) SVM,

(e) Kernel,

(f) Preprocessor and

(g) Features.

We implemented a simple text-based GUI. Also, one can use scripts with sequences of commands to control the program execution. However, since the functionality required within the GUI is encapsulated in the GUI library, one can easily port it to other GUIs. Many commands are related to HMM functions, i.e., all the algorithms introduced in Section 2.2 are implemented. Special types, as ,e.g., linear HMMs (cf. Section 2.2 as well as higher order HMMs Section 2.2.4 can be utilised.
On the other hand discriminative learners such as SVMs can be used. We embedded the implementations of [23, 46], and then generalised and speed tuned them for our task. The SVMs have a generic interface: they can make use of any kind of data since data is dealt with indirectly using kernel functions only (see Section 2.4). In addition kernels make use of a *caching* interface and can be defined on arbitrary features, for example just D-dimensional vectors $\mathbf{x} \in \mathbb{R}^D$, integers, short integers or sequences. Features on the other hand can be derived in real time from, e.g., HMMs. One can even use feature vectors directly from files (without loading the whole set of features into memory). This is done efficiently through another caching interface. Since features can contain lots of nuisance-dimensions or are badly scaled, one can assign a chain of preprocessors to each feature object. In so doing, one can preprocess features in real time or write the preprocessed features to a file. Possible preprocessors are, e.g., Principal Component Analysis (PCA), normalisation of the vectors to length 1 or the removal of dimensions with low variance. All components can be interchanged and extended easily.
While the program has already grown to about 43000 lines of source code, we plan to further extend it and make it publicly available under the GNU Public License[41] in the near future.

[41] `http://www.gnu.org/`

References

[1] 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.

[2] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

[3] P. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers, 1998.

[4] L.E. Baum. An inequality and associated maximization technique occuring in the statistical analysis of probabilistic functions of markov chains. *Inequalities*, 3:1–8, 1972.

[5] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[6] B.E. Boser, I.M. Guyon, and V.N. 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.

[7] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2):121–167, 1998.

[8] M. Burset and R. Guig. Evaluation of gene structure prediction programs. *Genomics*, 34:353–357, 1996.

[9] M. Burset, I.A. Seledtsov, and V.V. Solovyev. Analysis of canonical and non-canonical splice sites in mammalian genomes. *Nucleic Acid Research*, 28(21):4364–4375, 2000.

[10] M. Burset, I.A. Seledtsov, and V.V. Solovyev. SpliceDB: database of canonical and non-canonical mammalian splice sites. *Nucleic Acid Research*, 29(1):255–259, 2000.

[11] D. Cai, A. Delcher, B. Kao, and S. Kasif. Modelling splice sites with bayes networks. *Bioinformatics*, 16:152–158, 2000.

[12] V. Cherkassky and F. Mulier. *Learning from Data — Concepts, Theory and Methods.* John Wiley & Sons, New York, 1998.

[13] J.S. Chuang and D. Roth. Splice site prediction using a sparse network of winnows. *Technical Report UIUCDCS-R-2001-2199*, Feb 2001.

[14] The Genome International Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, Feb 2001.

[15] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273 – 297, 1995.

[16] W.B. Derry, A.P. Putzke, and J.H. Rothman. Caenorhabditis elegans p53: role in apoptosis, meiosis and stress resistance. *Science*, 294:591–595, October 2001.

[17] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

[18] M. Edgley. What is caenorhabditis elegans and why study it? An introduction for non-specialists. `http://www.biotech.missouri.edu/Dauer-World/Wormintro.html`.

[19] J. A. Hanley and B. J. McNeil. The meaning and use of the area under the receiver operating characteristic (roc) curve. *Radiology*, 143:29–36, 1982.

[20] D. Haussler. Computational genefinding. *Computational genefinding. Trends Biochem. Sci.*, 1998.

[21] T.S. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *J. Comp. Biol.*, 7:95–114, 2000.

[22] T.S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In M.S. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 487–493, 1999.

[23] T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

[24] W. Karush. Minima of functions of several variables with inequalities as side constraints. Master's thesis, Dept. of Mathematics, Univ. of Chicago, 1939.

[25] V. Kreinovich. For interval computations, if absolute-accuracy optimization is np-hard, then so is relative-accuracy optimization. 1999.

[26] V. Kreinovich, A. Lakeyev, J. Rohn, and R. Kahl. *Computational complexity and feasibility of data processing and interval computations*, volume 10. Kluwer Academic Publishers, Dordrecht and Boston and London, 1998.

[27] H.W. Kuhn and A.W. Tucker. Nonlinear programming. In *Proc. 2^{nd} Berkeley Symposium on Mathematical Statistics and Probabilistics*, pages 481–492, Berkeley, 1951. University of California Press.

[28] D. Kulp, D. Haussler, M.G. Reese, and F.H. Eeckman. A generalized hidden markov model for the recognition of human genes in DNA. *ISMB-96*, pages 134–141, 1996.

[29] L. Devroye and L. Györfi and G. Lugosi. *A Probabilistic Theory of Pattern Recognition.* Number 31 in Applications of Mathematics. Springer, New York, 1996.

[30] B. Lewin. *Genes VII.* Oxford University Press, New York, 2000.

[31] 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.

[32] 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.

[33] P. Muhlrad. Caenorhabditis elegans. 1999. `http://www.mcb.arizona.edu/wardlab/Caenowhat.html`.

[34] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 2001. in Press.

[35] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[36] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, February 1989.

[37] S. Rampone. Recognition of splice junctions on DNA sequences by BRAIN learning algorithm. *Bioinformatics*, 14(8):676–684, 1998.

[38] G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3):287–320, March 2001. also NeuroCOLT Technical Report NC-TR-1998-021.

[39] S. Saitoh. *Theory of Reproducing Kernels and its Applications.* Longman Scientific & Technical, Harlow, England, 1988.

[40] S.L. Salzberg. A method for identifying splice sites and translational start sites in eukaryotic mRNA. *Computational Applied Bioscience*, 13(4):365–376, 1997.

[41] B. Schölkopf. *Support vector learning.* Oldenbourg Verlag, Munich, 1997.

[42] B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors. *Advances in Kernel Methods – Support Vector Learning.* MIT Press, 1999.

[43] B. Schölkopf and A. J. Smola. *Learning with Kernels.* MIT Press, Cambridge, MA, 2002.

[44] 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.

[45] A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans. *Advances in Large Margin Classifiers.* MIT Press, Cambridge, MA, 2000.

[46] A.J. Smola and J. MacNicol. Scalable kernel methods. 2002. Unpublished Manuscript.

[47] S. Sonnenburg. *Hidden Markov Model for Genome Analysis.* Humbold University, 2001. Proj. Rep.

[48] J. Taub, J.F. Lau, C. Ma, J.H. Hahn, R. Hoque, J. Rothblatt, and M. Chalfie. A cytosolic catalase is needed to extend adult lifespan in *c. elegans* daf-c and clk-1 mutants. *Nature*, 399:162–166, May 1999.

[49] K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K.R. Müller. A new discriminative kernel from probabilistic models. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural information processings systems*, volume 14, 2002. In press.

[50] K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K.R. Müller. A new discriminative kernel from probabilistic models. In *Neural Computation*, 2002. In press.

[51] V.N. Vapnik. *The nature of statistical learning theory.* Springer Verlag, New York, 1995.

[52] V.N. Vapnik. *Statistical Learning Theory.* Wiley, New York, 1998.

[53] L. Xiaolin, M. Parizeau, and R. Plamondon. Training hidden markov models with multiple observations-a combinatorial method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(4):371–377, 2000.

[54] A. Zien, G. Rätsch, S. Mika, C. Lemmen B. Schölkopf, A.J. Smola, T. Lengauer, and K.-R. Müller. Engineering support vector machine kernel that recognize translation initiation sites in DNA. In *Proceedings GCB'99*, 1999.

[55] A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K.-R. Müller. Engineering Support Vector Machine Kernels That Recognize Translation Initiation Sites. *BioInformatics*, 16(9):799–807, September 2000.