

Gene prediction using a combination of GHMM and SVM

SEMESTER THESIS

at

HOCHSCHULE RAPPERSWIL
ABTEILUNG ELEKTOTECHNIK

Autors:

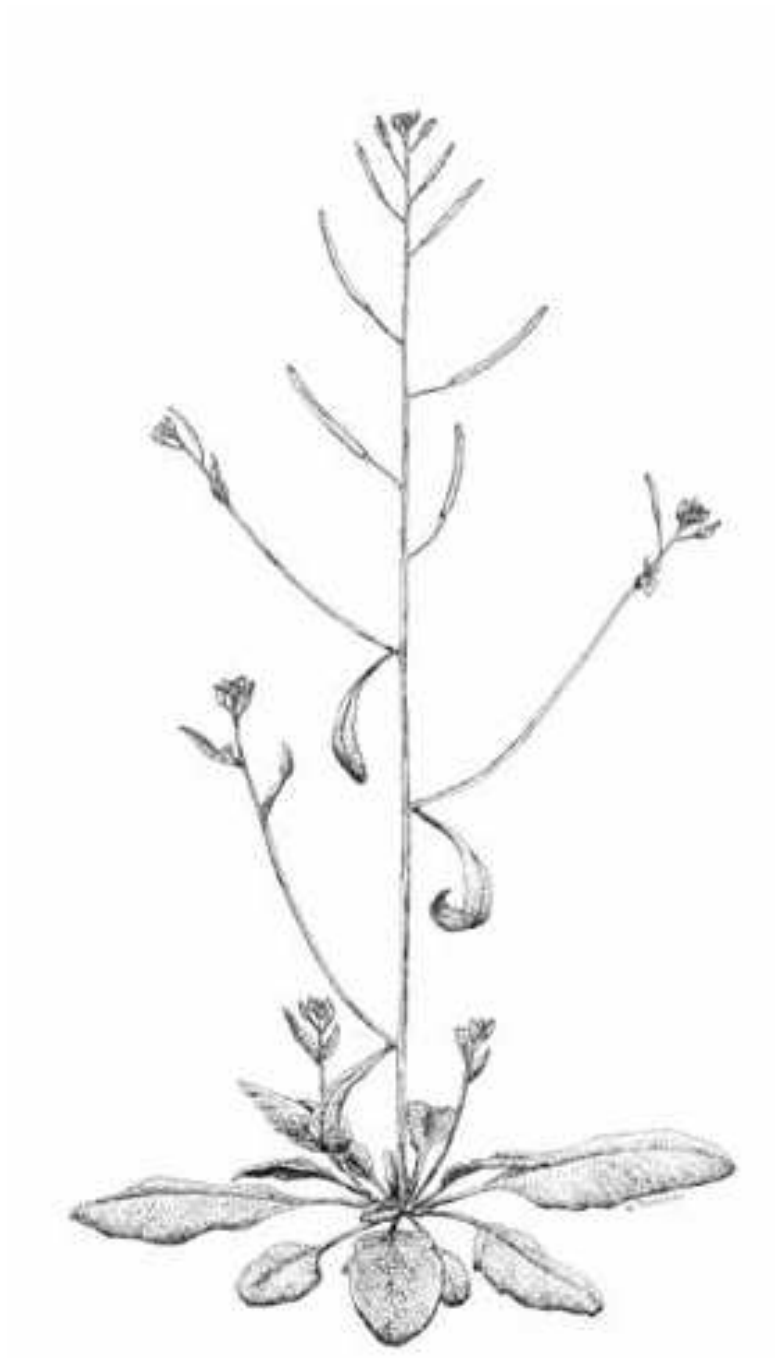
Philipp Beck

Pascal Frei

Supervisor:

Prof. Dr. Guido Schuster

Rapperswil, July 6, 2006



*Arabidopsis Thaliana*¹

¹ Source:http://naturalsystems.uchicago.edu/naturalsystems/index_files/image004.jpg

Abstract

The progress of biologist during the last years has required the help of computerised methods. These days, one of the main issues is to identify genes in a DNA. A large amount of codon indices have been developed to make it possible to differentiate between so called coding and non coding regions.

The aim of this thesis is to present a combination of a *Support Vectors Machines* and a *Generalized Hidden Markov Model* to predict genes in eukarotic DNAs.

Since the SVM has shown a good ability to detect patterns, it is used to find the approximate positions of coding regions. The input vectors for the SVM are generated by different known codon indices as Z-Curve analysis, Fourier analysis and others. It is shown that the accuracy of the SVM is more precise than the accuracy of the individual codon indices. Therefore the output of the SVM is used as weighting factor for the emissions and signal scores of the GHMM. The GHMM uses four different signal sensors and six different content sensors to predict the exact position of the genes in the DNA.

Extensive measures of a entire chromosome have been executed to prove the accuracy of this combination. The obtained results are comparable with available gene prediction programs, however these are showing identical problems which is suffering from high false positive rates.

Contents

1	Introduction	1
2	Genetic Fundamentals	3
2.1	History of the DNA	3
2.2	Structure of the DNA	3
2.3	From the DNA to the genes	4
2.3.1	Intergenic region	4
2.3.2	Translation Initiation Sites (TIS)	4
2.3.3	Exon	4
2.3.4	Intron	5
2.3.5	Translation stop site TSS	5
2.4	Protein machine	5
2.4.1	Transcription	5
2.4.2	Splicing	6
2.4.3	Translation	6
3	Genetic statistics	9
3.1	Introduction	9
3.2	Statistic for the GHMM	9
3.2.1	Amino acid distribution	9
3.2.2	Length distribution	10
3.3	Promoters	10
4	GHMM	13
4.1	Indtroduction	13
4.2	From Markov Model to Generalized Hidden Markov Model	13
4.2.1	Markov Model	13
4.2.2	Hidden Markov Model	16
4.2.3	GHMM	19
4.3	The implemented GHMM	20
4.3.1	Signal states	22
4.3.2	Emission states	23
4.3.3	Length probability	24
4.3.4	Transition probability	24
4.3.5	The phase problem	24
4.3.6	Eclipsing	26
4.3.7	Back tracking	28
4.3.8	GHMM step by step	28

5	Support Vector Machine	35
5.1	Introduction	35
5.2	Support Vector Classification	35
5.2.1	The maximal margin	36
5.3	The Kernel Trick	40
5.4	The Support Vectors	41
5.5	Decision Function	41
5.5.1	Linear classifier	41
5.5.2	Nonlinear classifier	42
5.6	Parameter Evaluation	42
5.6.1	Grid-Search	42
5.6.2	Cross-Validation	43
5.7	Multiclass SVM	43
5.7.1	One against one	43
5.7.2	One against all	43
5.7.3	Error Correction Output Code	44
5.8	Probabilistic Output	44
5.9	Implementation	44
6	Application of the SVM on genetic problems	45
6.1	Introduction	45
6.2	Translation Initiation Sites (TIS)	45
6.2.1	Edit Kernel	45
7	Identifying Coding Regions	49
7.1	Introduction	49
7.2	Fourier Analysis	49
7.3	Detrended Fluctuation Analysis	51
7.4	The Z-Curve	54
7.5	Hexacount Analysis	57
7.6	Identifying coding regions with help of a SVM	58
8	Combination of SVM and GHMM	61
8.1	Introduction	61
8.2	Changes in the GHMM	61
9	Results	63
9.1	Translation Initiation Sites (TIS)	63
9.1.1	Validating of the implementation	63
9.1.2	TIS recognition in DNA	63
9.2	Identifying coding regions with help of a SVM	64
9.2.1	Training of the SVM	64
9.2.2	Results	65
9.3	GHMM	68
9.3.1	Training of the GHMM	68

9.3.2	Results	68
9.3.3	Speed up with signal eclipsing	70
9.4	Combination of GHMM and SVM	71
9.4.1	Training of the SVM for GHMM	71
9.4.2	Parameter evaluation	71
9.4.3	Results	72
10	Conclusion	75
10.1	Review	75
10.2	Outlook	75
A	Measures of prediction accuracy	77
A.1	Mathematical basics	77
A.2	Nucleotide Level	78
A.3	Exon Level	78
A.4	Protein Level	79
B	Notation	81
C	Abbreviations	83
D	Matlab files	85
D.1	Gene prediction with a SVM	85
D.2	Own SVM implementation	86
D.3	GHMM prediction	87
D.4	GHMM penalty estimation	88
E	Substitution tables	91
	Bibliography	93

List of Figures

1.1	Exponential growth of identified nucleotides.	1
2.1	Structural view of the DNA	4
2.2	Processes from the DNA to the proteins	5
2.3	Alternative Splicing: Some exons are skipped to build a protein	6
3.1	Length distribution according to [8]	10
3.2	Promoter region in front of a start codon	11
4.1	Fax compression	14
4.2	Markov Model for fax compression	14
4.3	Markov model of genetic data	15
4.4	A N-state urn-and-ball model	17
4.5	A simple HMM for genetic data	18
4.6	DA Model from 167000 to 177000	20
4.7	Implemented GHMM	21
4.8	Signals	23
4.9	Fifth order markov model	24
4.10	How to add the scores and the switch over the phase	25
4.11	An example of a forward strand with signals	26
4.12	An example of a reverse strand with signals	26
4.13	Eclipsing examples	27
5.1	Concept of a hyperplane as a linear classifier	35
5.2	Margin around the hyperplane	36
5.3	Slack variables defining a soft margin	38
5.4	Mapping of the input space X to the feature space F	40
7.1	Fouriertransformation of a DNA region a) noncoding , b) coding	50
7.2	Sliding Window	50
7.3	Fourier prediction	51
7.4	DNA walk	52
7.5	DFA on a window with length $N = 512$ and segment length $l = 64$	53
7.6	Detrended Fluctuation Analysis of a DNA region a) noncoding , b) coding	54
7.7	Period-3 fractals deviations of the three phases	54
7.8	MAXFD codon index	55
7.9	YZ-Score for a sequence of 10000 nucleotides.	57
7.10	Hexacount analysis	58

9.1	Building TIS and pseudo TISs of a DNA	64
9.2	SVM prediction high accuracy	67
9.3	GHMM prediction from 1383350 to 1390700	68
9.4	GHMM prediction from 16921500 to 16923000	69
9.5	GHMM prediction from 1500000 to 1560000	70
9.6	GHMM prediction with SVM support from 1383350 to 1390700	72
9.7	GHMM prediction with SVM support from 16921500 to 16923000	73
9.8	GHMM prediction with SVM support from 1500000 to 1560000	74
A.1	Measures of prediction accuracy	77

List of Tables

2.1	Translation from codons to amino acids	7
2.2	Signaling codons	7
3.1	Nucleotide probabilities of arabidopsis thaliana	10
4.1	Transition probability matrix A of the genetic model	15
4.2	Specialised transition matrices A_{coding} and $A_{nonCoding}$ of the genetic model .	15
4.3	models and length of each signal	22
4.4	Transition probabilities	24
4.5	Distances where the model eclipse the signals	27
5.1	Error Correction Output matrix for digit recognition. Source: [29]	44
6.1	Example of the Levenshtein algorithm	47
7.1	Binary mapping	49
9.1	Validation of the implemented edit kernel.	63
9.2	Obtained results on TIS prediction on a DNA sequence	64
9.3	Used Windowsizes	65
9.4	Occupied Accuracy of the 6 fold cross validation	65
9.5	Measures over the whole chromosome IV, SVM	66
9.6	Measures over the whole chromosome IV, only Z-Curve	66
9.7	Results of the GHMMs	69
9.8	Results of the GHMMs for a long sequence	71
9.9	Speedup with eclipsing	71
9.10	Penalty values	72
9.11	Results of the GHMM with SVM support	73
9.12	Results of the GHMM with SVM support over a long sequence	74
E.1	The ASCM250 Matrix	91
E.2	The PAM1 Matrix. Entries are scaled with 10000	92
E.3	The SCM250 Matrix	92

Chapter 1

Introduction

During the last 15 years biologists have made huge progress with extracting the DNA of different organisms. The amount of extracted DNA sequences has even been growing exponential in the last recent years. Figure 1.1 shows the gene bank size of the European Bioinformatics Institute. This gene bank is open to scientist who want to publish their DNA sequences. At the moment there are DNA sequences with a total length of 141'772'680'164 nucleotides¹. A nucleotide is the logical unit a DNA is made of.

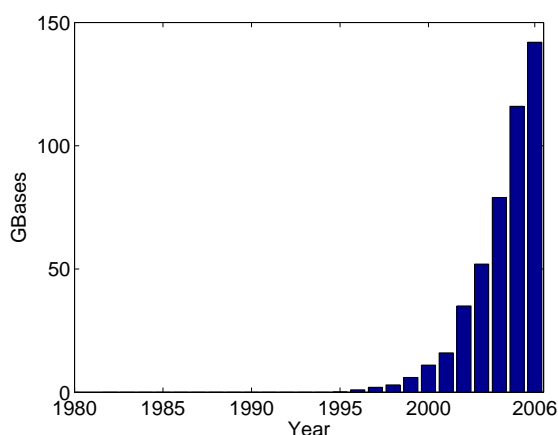


Figure 1.1: Exponential growth of identified nucleotides.

Source:European Bioinformatics Institute

The enormous amount of this data has created a new scientific field, called Bioinformatic. Bioinformatic, as its name implies, deals with applying computer science methods on biological problems. It is cut into different subareas, whereas the area concerning this thesis is “Gene prediction”.

The main part of a DNA sequence does not contain any relevant information, but a small part does contain the genes. The field of “Gene Prediction” deals with the problem of finding this genes, which still is not solved satisfyingly.

The goal of this thesis is to present a combination of a General Hidden Markov Model, well known gene identifying methods and a Support Vector Machine. The Support Vector Machine is used to predict, with help of the known identifying methods, where in a DNA

¹27. June 2006

sequence a gene most likely could be. Then, this information is used by the General Hidden Markov Model to identify the exactly position of the genes.

Chapter 2

Genetic Fundamentals

2.1 History of the DNA

Every organism is made of billions and trillions of cells. There exist two different types of cells, namely the procaryotic and the eukaryotic cell. The main difference between this two cell types are the size of the cell and the location of the genome. The size of the procaryotic cell is about a tenth of the size of the eukaryotic cell. In contrast to the procaryotic cell, the genome in the eukaryotic cell is located in the nucleus. In this paper we only consider the eukaryotic cell, which the human, animals and plants are part of.

In 1869 Friedrich Miescher discovered that the nucleus is different to the rest of the cell. With experiments he showed the similarity of the nucleus to acids. Miescher also wrote in his paper that if there is any information about heredity in a cell, it must be stored in the nucleus.

After the dead of Miescher his discovery became of interest again in the forties of the last century because an evidence showed the appearance of the genetic information in the nucleus.

In 1953 James Watson and Francis Crick could explain the structure of the DNA and its function. Their work is deemed to be one of the most important discovery in the biological field and was prized with an Nobel prize in 1962.

2.2 Structure of the DNA

The DNA(Deoxyribonucleic acid) is formed of two anti parallel strands, which are complementary and wrapped around an imaginary axis in form of a double helix structure. These two strands consist of the sugar deoxyribose and a phosphate, referred to as backbone, and a base. The combination of these three forms the structural unit of the DNA, the so called nucleotide. There exist four types of nucleotides with the abbreviation A, C, G and T, standing for adenine, cytosine, guanine and thymine. Since each nucleotides forms hydrogen bonds readily to only one other, namely A with T and C with G, each of the two strands is dependent of the other. Figure 2.1 shows a planar structural view of the DNA. Interestingly, although the two strands are dependent, each one contains separate genetic information. This makes it necessary to differ between these two strands. This is done by the so called direction. A strand begins always with a deoxyribose, referred to as 5' end, or a phosphate, referred to as 3' end. The possible direction are now 5'→3' or 3'→5'. Since the two strands are complementary to each other, the direction of one

strand is always the opposite of the other.

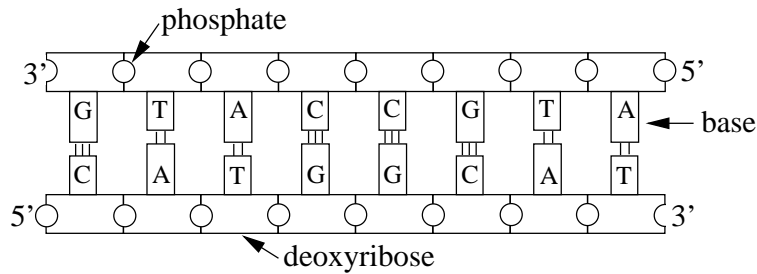


Figure 2.1: Structural view of the DNA

2.3 From the DNA to the genes

Up to 98.5% of the DNA doesn't contain any information or more specificity, isn't known yet to have a function. The human for example has about 3.2 billion nucleotides and approximately about 15000 genes, although the true number isn't known yet.

The DNA is divided into several regions which are shown in Figure 2.2. Following is an overview about the function of the several regions.

2.3.1 Intergenic region

The nucleotides between two genes are called intergenic region. Although this region contains so called promoters, which can announce the possibly start of a gene, it does not contain any hereditary information.

2.3.2 Translation Initiation Sites (TIS)

The TIS indicates the protein machine to start the translation process which is described later. The TIS contains only one codon¹, namely ATG.

2.3.3 Exon

The exon is an integral part of the gene. It contains the codons which are used to form the proteins. The initial exon begins with the TIS. A gene is made of several exons. For example the human DNA contains genes with up to 178 exons. These exons are separated by the introns, which are explained in the next section.

¹The term "codon" will be explained in the next sections

2.3.4 Intron

The introns are similar to the intergenic region, with the only difference that introns are located between the start and the end of a gen. The beginning of an intron is indicated by the sequence GT, the so called donor splice site (5' splice site), while the end of an intron is indicated by the sequence AG, the acceptor splice site (3' splice site).

2.3.5 Translation stop site TSS

The TSS marks the end of gene. The end is always indicated by one of the three stop codons TGA, TAG or TAA. All nucleotides between a start codon and a stop codon are in the so called open reading frame ORF.

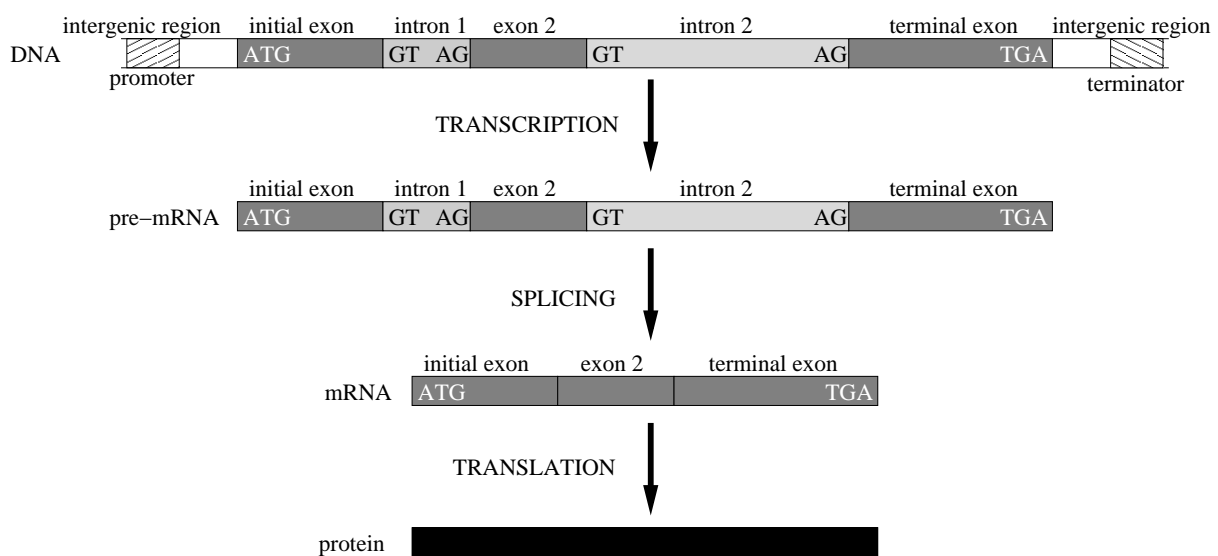


Figure 2.2: Processes from the DNA to the proteins

2.4 Protein machine

With Protein Machine the entire group of processes is meant, which are needed to translate a DNA sequence to a protein. In the following section this processes are explained.

2.4.1 Transcription

During the process called transcription, all the nucleotides between the promoter and the terminator are copied, leading to the pre-mRNA (pre messenger ribonucleic acid). The pre-mRNA is almost a perfect copy of a selected DNA sequence with the difference that the nucleotide thymine is replaced by the nucleotide uracil (U). Since this nucleotide forms exactly like thymine only hydrogen bonds with adenine, every T can just be replaced by an U. For the matter of simplicity, in this thesis U is always represent by T.

2.4.2 Splicing

The step after transcription is called splicing. Splicing removes all introns from the pre-mRNA, leading to the mRNA. The mRNA contains all exons in a row, without a single intron between them. The mRNA is now ready for the next step, the translation.

Alternative splicing

It has been shown, that not always all exons between a start and a stop codon are used to form a gene. It is possible that some exons are skipped and other ones are not. Figure 2.3 shows how different genes can be formed out of 4 exons. These days it is very difficult for biologist to decide which exons are involved to form a protein and which are not. Alternative splicing is a process which needs to be researched in the future.

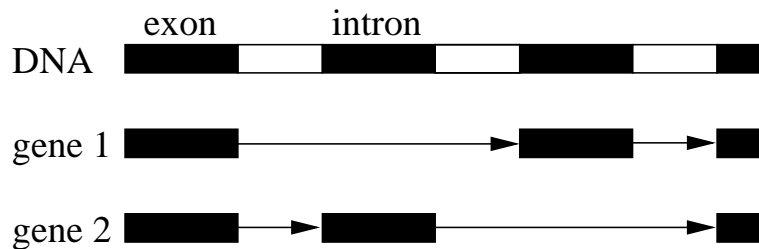


Figure 2.3: Alternative Splicing: Some exons are skipped to build a protein

2.4.3 Translation

Translation is the last step on the way to the protein. During the translation always three nucleotides are taken together to form a codon. This codons are translated in a second step to amino acids. Three out of four nucleotides can form 64 codons. But only 61 codons are translated to amino acids. The remaining three codons represent the stop codons. As exception, the sequence ATG is an start codon and represents also the amino acid Methionine. Table 2.2 shows all possible signaling codons. According to the table 2.1 the codons are translated to amino acids. Since there are only 20 amino acids, some codons are translated to the same amino acid. All the translated amino acids represent now the protein.

Amino Acid			Codon
A	Ala	Alanine	GCA GCC GCG GCT
C	Cys	Cysteine	TGC TGT
D	Asp	Aspartic acid	GAC GAT
E	Glu	Glutamic acid	GAA GAG
F	Phe	Phenylalanine	TTC TTT
G	Gly	Glycine	GGA GGC GGG GGT
H	His	Histidine	CAC CAT
I	Ile	Isoleucine	ATA ATC ATT
K	Lys	Lysine	AAA AAG
L	Leu	Leucine	TTA TTG CTA CTC CTG CTT
M	Met	Methionine	ATG
N	Asn	Asparagine	AAC AAT
P	Pro	Proline	CCA CCC CCG CCT
Q	Gln	Glutamine	CAA CAG
R	Arg	Arginine	AGA AGG CGA CGC CGG CGT
S	Ser	Serine	AGC AGT TCA TCC TCG TCT
T	Thr	Threonine	ACA ACC ACG ACT
V	Val	Valine	GTA GTC GTG GTT
W	Trp	Tryptophan	TGG
Y	Tyr	Tyrosine	TAC TAT

Table 2.1: Translation from codons to amino acids

Codon	Type
ATG	Start codon
TAA	Stop codon
TAG	Stop codon
TGA	Stop codon

Table 2.2: Signaling codons

Chapter 3

Genetic statistics

3.1 Introduction

According to chapter 1 the amount of data is huge. Only a short example. There are about 3 billion amino acids coding the human genome. Printing amino acids on an A0 page with font size 12, prints 70000 amino acids on one page. For the human genome there are about 43'000 pages of size A0 needed. Laying all these pages on the floor, five times the area of football field would be covered. As can be seen there is no chance to analyse every possibility. So the field of gene prediction usually use statistic information. With statistics losing information can not be avoided but it is now of manageable complexity.

In the beginning of bioinformatics, researcher tried to find the exons with "simple" statistic analysis of the genome. We also went this way, due to find the necessary statistic for the GHMM. We have also tried to proof the statement, found in [4] and [16], that before a start codon there should be a promoter.

3.2 Statistic for the GHMM

There are two statistics which are used in the GHMM. These are :

- distribution of the four amino acids over genic and intergenic region
- length distribution of different exons

3.2.1 Amino acid distribution

The distributions of the four amino acids are different in genic and intergenic region. They are nearly equally distributed in the genic region. In the intergenic region the As and Ts are more likely.

This statistic can be found in our plant as well as in the book [16].

These differences are very important for gene prediction. It is used these days as the foundation to build up more complex models.

	exon	intron	intergenic region
A	28.663 %	27.525 %	33.375 %
T	27.175 %	39.607 %	33.379 %
G	23.871 %	17.463 %	16.112 %
C	20.291 %	15.404 %	16.170 %

Table 3.1: Probabilities of the nucleotides in different regions of chromosome one of *Arabidopsis thaliana*

3.2.2 Length distribution

The distribution of the length of different regions is another point of interest. With the GHMM, described in section 9.3, it is possible to model the different length distribution in the prediction. The different lengths are according to [8] :

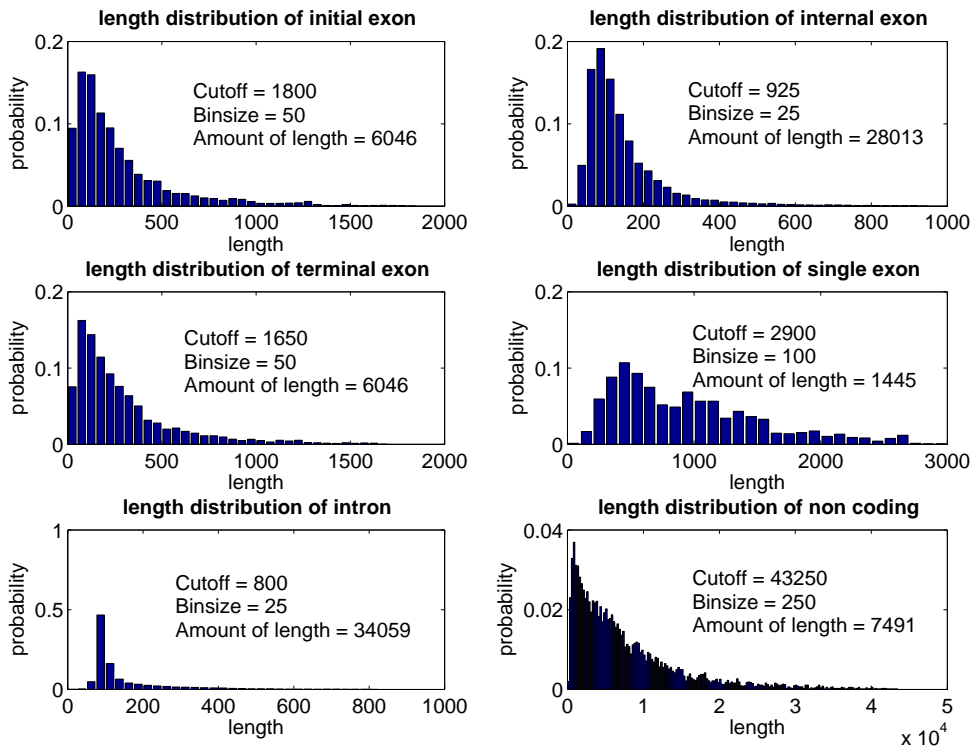


Figure 3.1: Length distribution according to [8]

3.3 Promoters

Promoters are sections in the genome where the protein machine can approach the DNA strand. They should also appear before a start codon. There are two possible promoters

documented in [16] and [4] page 311 ff. One is the TATA box the other is the GC box. The TATA box should be about 30 nucleotides before a start codon. TATA Box has its name because it consists primarily of Ts and As. This constitution makes it easy to break up the double helix due to an A is connected to the T on the reverse strand only by two hydrogen bonds.

In the GC box there are primarily G's and C's. There is no physical information as for the TATA box. It has just been found before a lot of genes. This box is located about 35 nucleotides before the start codon.

These boxes could regrettably not be found in our plant as can be seen in figure 3.3.

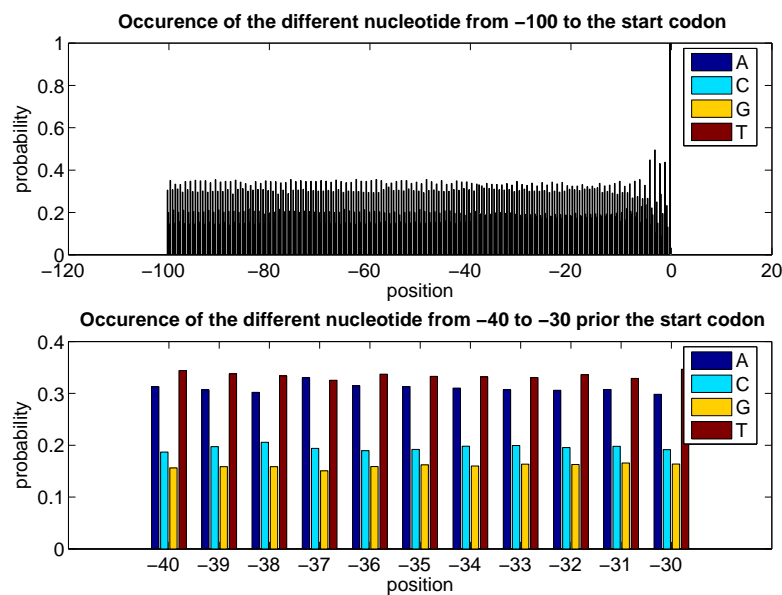


Figure 3.2: Promoter region in front of a start codon

As can be seen, the A of the start codon has to be at the position zero. But there is an almost equally distribution over the other positions. Especially in the supposed promoter region -40 to -30 nucleotides. There can not be seen a clear difference. Due to this circumstance we stopped the promoter approach at this point.

Chapter 4

GHMM

4.1 Introduction

The GHMM is an abbreviation and standing for Generalized Hidden Markov Model. It has been elaborated over several years.

The Markov Model has been used very successfully in lossless compression, such as FAX or Text file compression.

The Hidden Markov Model is an extension in that way that it tries to find the state in which the sequence was emitted.

The Generalized Hidden Markov Model solves the problem that with length the geometrically probability falls. In some application the length probability needs to be set according to physical or other information. In genetic research, the length probability of the exons is known. The probability for an exon to have 50 nucleotide is smaller than to have 500. To be able use this information in the model, it is necessary to use a GHMM.

4.2 From Markov Model to Generalized Hidden Markov Model

4.2.1 Markov Model

The Markov Model was introduced by Andrei Andrevich Markov in 1906. The model reduces the entropy of the symbols emitted by the source by implementing a dependency. For example fax compression. A fax consists of only black and white pixels, especially in the early eighties. Line after line is scanned and transmitted. In that way the source emits a long sequence consisting of zeros and ones.

One stands for white, zero for black. It would be a waste of bits to transmit that over a medium. It is better to make longer symbols and use Huffman to code it.

But there is a better way. Looking at the sequence above we see that the amount of white pixels following another white one is huge. The first order markov model takes care of this circumstances because it looks always one pixel in the “past”.

The model is now able to depend on one previous pixel. This pixel can have two values, zero and one. For the markov model this leads to 2 states. The last pixel was white, or black.

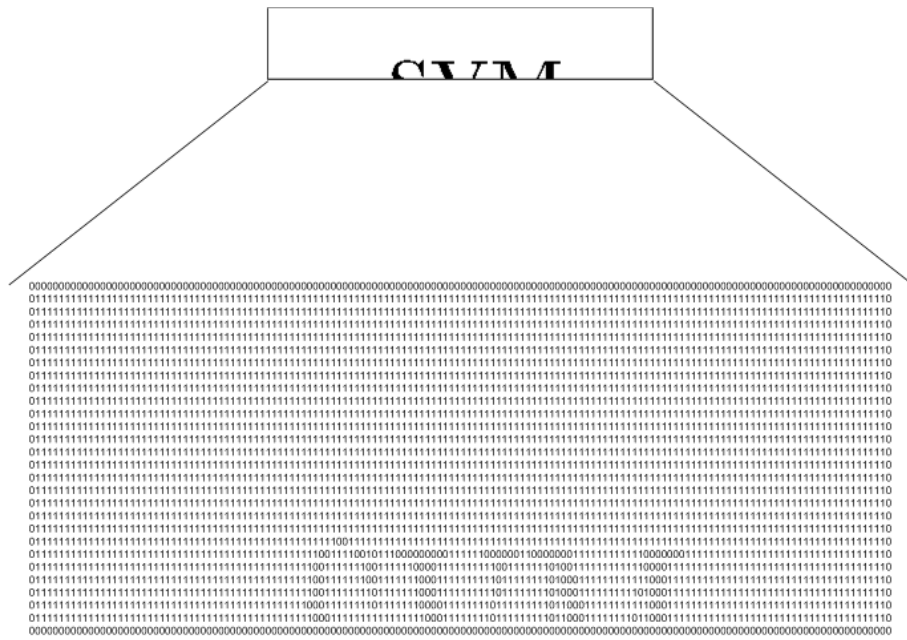


Figure 4.1: Fax compression

Now we now adapt this theory to genetics. In genetics there are four symbols as introduced in section 2.2. This leads to a more complex markov model shown in figure 4.3.

The transition probabilities are noted in form of a matrix and is called A . For example the following sequence of observed symbols

$$O = tattacgaacatttggaacatcttcaggtaactcttctatgttgtacttgt$$

the probability that this sequence was omitted by model above can now be evaluated.

$$P(O|Model) = 1.2544 \cdot 10^{29} \tag{4.1}$$

The exact calculation is not part of this documentation. For the interested reader see

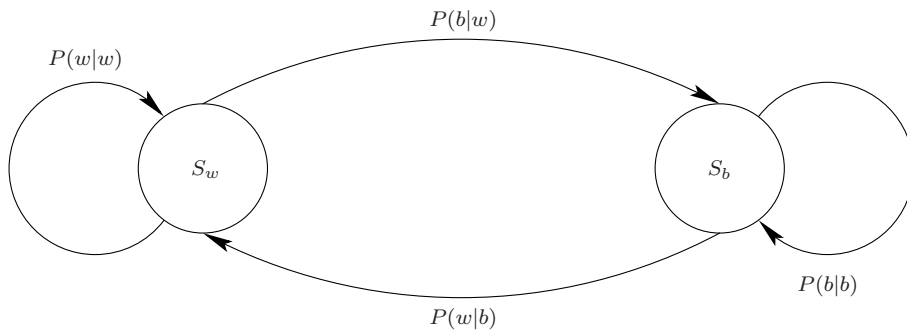


Figure 4.2: Markov Model for fax compression

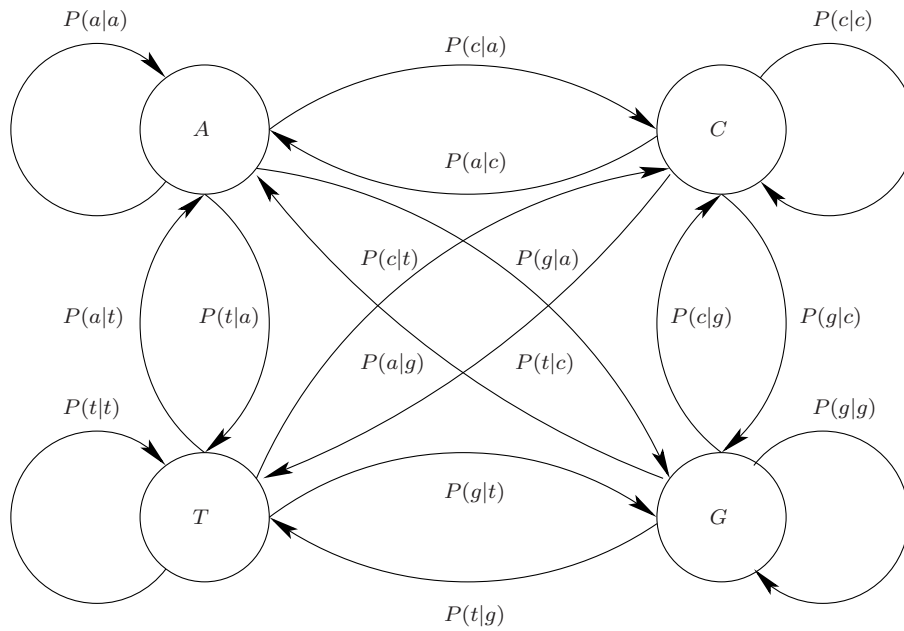


Figure 4.3: Markov model of genetic data

from / to	A	C	G	T
A	0.3399	0.1746	0.1873	0.2982
C	0.3374	0.1822	0.1382	0.3421
G	0.3569	0.1578	0.1889	0.2964
T	0.2239	0.2147	0.1911	0.3703

Table 4.1: Transition probability matrix A of the genetic model

[28].

This is the first step in gene prediction. For a simple predictor, with two states, one coding and one non coding. We make a separately transition table A for both states. The outcome of this are two tables:

	A_{coding}					$A_{nonCoding}$			
	A	C	G	T		A	C	G	T
A	0.3110	0.1790	0.2609	0.2491	A	0.3619	0.1677	0.1857	0.2846
C	0.3188	0.1939	0.1684	0.3189	C	0.3435	0.1989	0.1286	0.3291
G	0.3541	0.1871	0.2292	0.2296	G	0.3549	0.1674	0.1910	0.2868
T	0.1739	0.2510	0.2787	0.2964	T	0.2442	0.2044	0.1946	0.3568

Table 4.2: Specialised transition matrices A_{coding} and $A_{nonCoding}$ of the genetic model

The information on the genebank file NC_003075 was used to calculate the tables above, using coding.mat and noncoding.mat on CD. Now the probability of the observed sequence

O can be evaluated. This leads to the following results

$$P(O|Coding) = 1.3987 \cdot 10^{30} \quad P(O|NonCoding) = 7.6671 \cdot 10^{30} \quad (4.2)$$

Looking at the probabilities the sequence must come from the non coding region. Which in this case is true. The observation sequence is from nucleotide 200000 to 200050 on NC_003075 between gene number 62 and 63.

Even though those results have lower probability than equation 4.1 they are more relevant. The reason is that the transition matrix A has been calculated by using only 50000 nucleotides. whereas the A_{coding} and $A_{nonCoding}$ matrices uses over 5 millions of nucleotides each.

This was the first step in sequencing genetic data. But there are better models. The Hidden Markov Model, introduced in the next section, tries to work with the deviation in the probabilities calculated above.

For more information about Markov Models and there usage see [28] or [33]. [28] is a good source for markov models and their usage in compression.

4.2.2 Hidden Markov Model

The following introduction to HMM follows mainly the book [17].

In the markov model represented in figure 4.2 the state in which it is is clear after recognise one symbol. What if this is not so clearly to keep apart? Mention a coin toss example. Someone is performing the coin toss experiment, and he tells you just the result, head or tail.

You also don't know how many coins he is using. It could be one coin with a fifty-fifty probability for head or tail. But also he could use 2 coins, or 3. You do not know. But you have more flexibility if you say he is using more coins. So 2 coins gives two states in which you have to define the emission probability of a head, denoted as $P(H)$. Tail then gets defined as $1 - P(H)$. Also you have to specify the transition probability, the probability that the person, performing the experiment, changes the coin. So for a 2 coin model, you have four parameters to specify, for a 3 coin model those 9.

For the model with 3 coins and 9 parameter you will need a quite longer sequence to estimate all the parameters. Practical considerations such as calculation speed and others impose some strong limitations on the size of the model.

No we move on to a more complex situation. This example is used in a lot of publications and shows good the problem a HMM tries to solve and the terms used in this part. The Urn-and-Ball model.

There are N big glass urns in a room. Within each urn is a large quantity of colored balls. We assume there are M distinct colors of the balls. The process of the experiment is now that a genius is going into the room. It chooses, according to some random procedure, an initial urn. Out of this urn it takes a ball and the color is recorded as the observation. The ball is then replaced in the urn from which it was selected. A new urn is then selected according to the random selection process associated with the current urn, and the ball selection process is repeated. This entire process generates a finite observation sequence

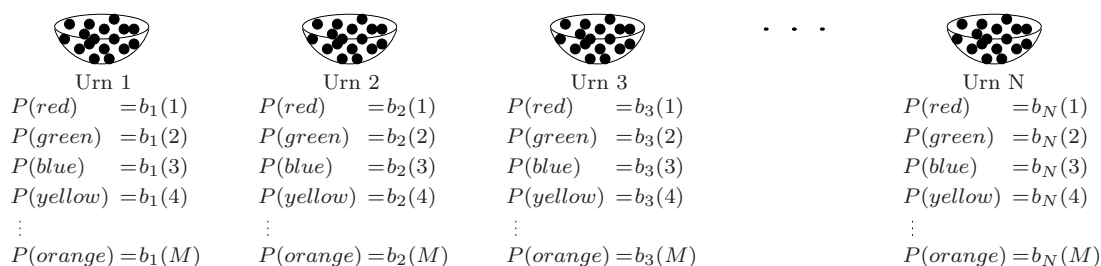


Figure 4.4: A N -state urn-and-ball model illustrating the general case of a discrete symbol HMM

of colors, which we would like to model as the observable output O of an HMM. !!!!!!!!!!!!!!!!

Elements of an HMM

The above example gives a good overview what an HMM is and how it can be applied to some scenarios. We now define the elements of an HMM.

An HMM for discrete symbol observation such as the above urn-and-ball model is characterized by the following:

1. N , the number of states in the model
2. M , the number of distinct observation symbols per state
3. The state-transition probability distribution, A
4. The observation symbol probability distribution, B
5. The initial state distribution, π

It can be seen in the above enumeration, that a complete specification needs the definition of two parameters, N and M , specification of the three sets of probability measures A, B , and π . Often there is seen the compact notation

$$\lambda = (A, B, \pi) \quad (4.3)$$

to indicate the complete parameter set of the model.

HMM challenges

Given the form of the HMM of the previous section, there are three problems that must be solved that the model becomes useful. These three problems are as follows:

1. If you have the observation sequence O and the model $\lambda = (A, B, \pi)$ how can you compute the probability of the information sequence efficiently, given the model. Solved with the so called forward and backward procedure.

2. Given the observation sequence O , and the model λ how do you choose a corresponding state sequence q that is optimal, best "explains" the observation?
Solved with the Viterby algorithm. Also known in mobile communication as the decoder of an forward error correction algorithms.
3. How do you adjust the model parameters to maximize $P(O|\lambda)$?
Solved by the Baum-Welch method. Also know as expectation maximization, EM.

With the solutions above those problems are actually already solved despite of one important point: the baum-welch or EM algorithms. The algorithm converge to a result. This result is only a local maximum and not particularly the global one! This means that with other initial matrices A , B and π the HMM converges to different results. For a deeper insight in HMM topic consider the books [17], [12] or [32].

A simple genetic HMM

Coming back to our aim: define where a genome is coding and where not. We can now define these two states as the hidden states of our hidden markov model. The A , B and π matrices can also be set to an initial value. Its obvious that the better the initial values are the better the results get. The model then looks as in figure 4.5.

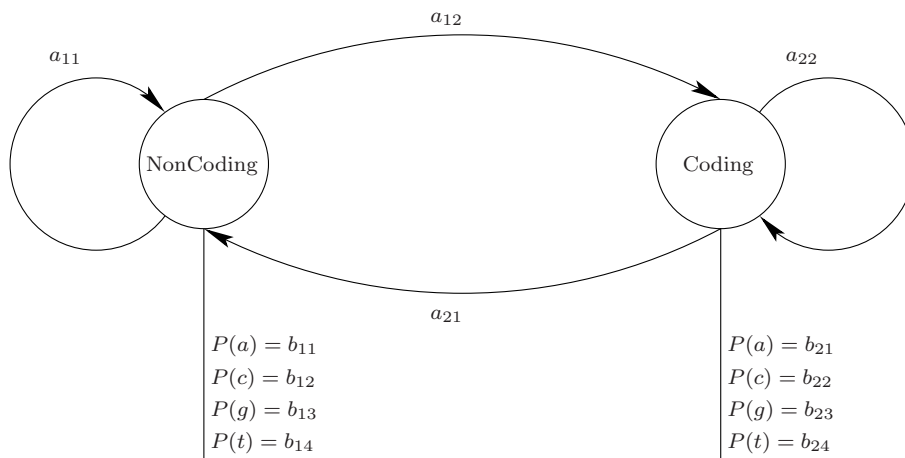


Figure 4.5: A simple HMM for genetic data

The initial parameter specification for the model is now as follows

$$A = \begin{bmatrix} 0.999 & 0.001 \\ 0.001 & 0.999 \end{bmatrix} \quad B = \begin{bmatrix} 0.33375 & 0.33379 & 0.16112 & 0.16170 \\ 0.28663 & 0.27175 & 0.23871 & 0.20291 \end{bmatrix} \quad \pi = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

This parameters are starting parameters. The baum-welch algorithms then specifies the parameters more exactly to get the most likely sequence. If initial state distribution matrix π is initialized in that way, the sequence should start in a non coding region.

4.2.3 GHMM

Having a sequence of nucleotides in the HMM the probability rate declines more and more with growing length. But this is not true for the model of a genetic source. An exon with length of 1 nucleotide is almost impossible. The length distribution of the exons are documented in chapter 3.2.2. This information should be implied in the model. That is the idea of the Generalized Hidden Markov Model.

Taking the length, two points are needed to calculate the difference within, they are called signals. But what defines a signal? To answer this question we take a deeper look at the definition of genetic sequences. As defined in chapter 2 the genetic code can not change between coding or non coding area at every nucleotide. There are only short sequences where a change is possible, called signals. Now we can measure the length between the signals and rate them for example according to a histogram of the length distribution. In this we include more information of the source in our model and should get a model which can uncover the hidden part better than the HMM.

In this approach the states which are to uncover are not clearly seen in the model. The signals are more present. We generate a queue with all our possible signals differentiated by there state. Then we go step by step through this queue and take always the next signal to calculate the probability of the sequence. It does not matter to which signal type it belongs. Important is to go straightforward through the sequence. For each signal we calculate the best predecessor: First we add the transition, length and emission score to the score of the predecessors. Than taking the best predecessor we add the signal score of the actual signal and store it as well as the link to the best predecessor.

The backtracking algorithm is now a Viterbi algorithm. We go back to the last signal and choose his best predecessor. From this predecessor we choose his best predecessor and so on until the first signal is reached.

This was a general overview over the topic of GHMM. In the next section we discuss our implementation of the GHMM. For further information on GHMM see [21].

4.3 The implemented GHMM

The implemented GHMM is based on the diploma thesis [8].

Remarkable is that the model encounters difficulties if we expand the amount of nucleotides which are to be predicted. It shows problems if genes are located on the reverse strand. This comes from the constitution of the model itself. The model is looking for statistical difference in coding or noncoding regions. Looking at the figure 4.5 it is obvious that there is a difference within the probability of the individual nucleotides. Even so the GHMM does not do that in this way it is usable to show the problem. Therefore, if there is a gene on the forward strand, or the reverse one, the probability of the individual nucleotide remains the same. The GHMM of the diploma thesis can not distinguish between forward and reverse strand. This causes wrong predictions on the forward strand as can be seen in figure 4.6.

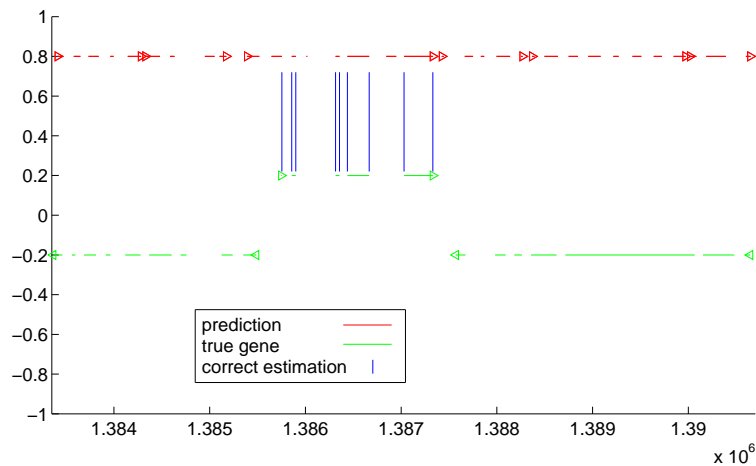


Figure 4.6: DA Model from 167000 to 177000

That's the reason why the model was extended to be capable of searching genes on the reverse strand too. According to [21] the feature mentioned above must lead to a better result.

All probabilities are logarithmic. This because the addition is easier for a computer as an multiplication. The equation

$$\log(p_1) + \log(p_2) = \log(p_1 \cdot p_2)$$

gives us the possibility to do that.

In the next eight sections, the extended model will be introduced.

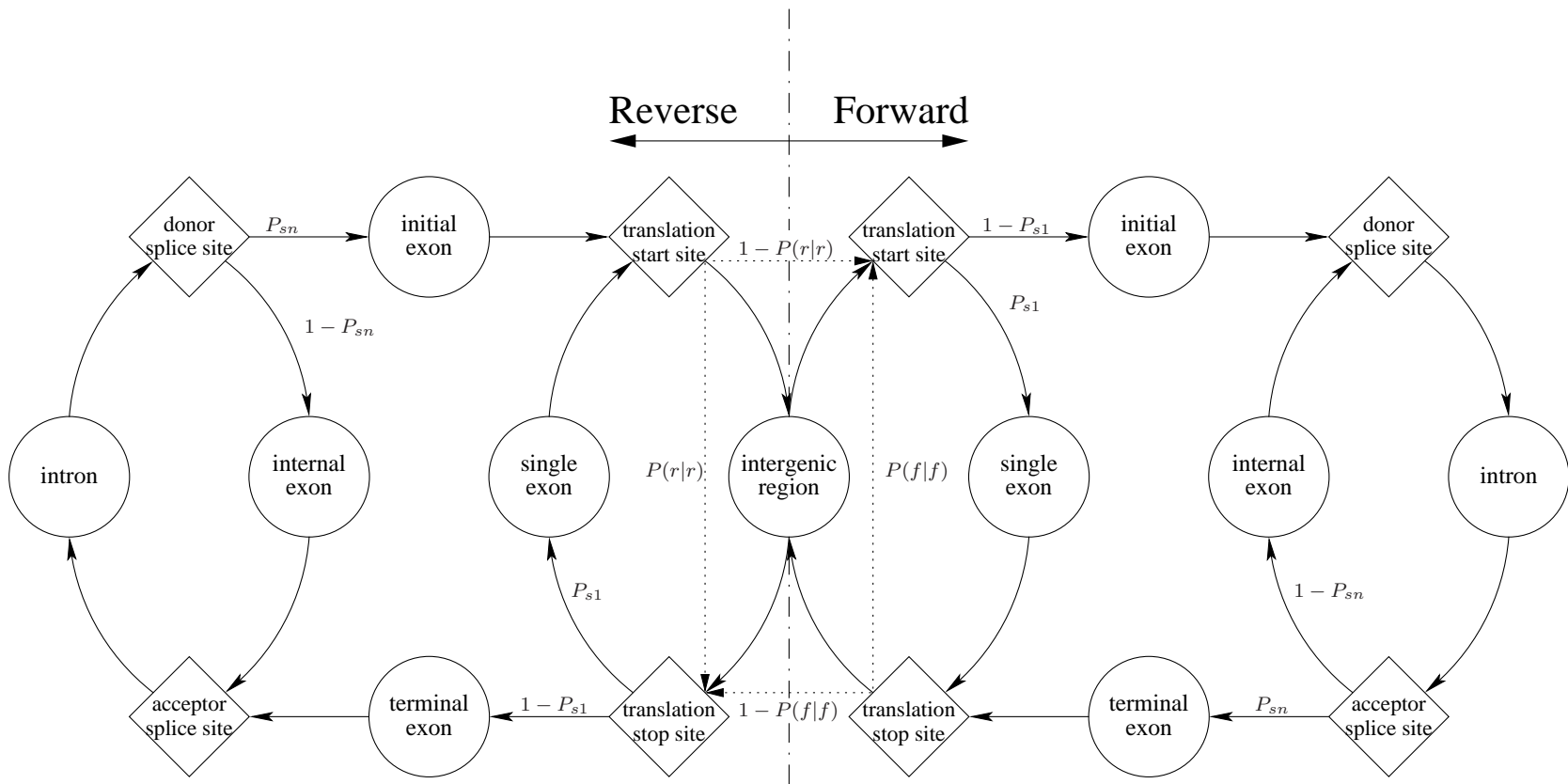


Figure 4.7: Implemented GHMM

4.3.1 Signal states

Apart of the four signals already used four new signals will be introduced in the extended model. In that way there are the following eight signals.

- acceptor splice side, ASSSF
- donor splice site, DSSSF
- translation start site, TISF
- translation stop site, TSSF
- acceptor splice site reverse, ASSR
- donor splice site reverse, DSSR
- translation start side reverse, TISR
- translation stop site reverse, TSSR

The signals belong of a defined mini sequence. This mini sequence is part of the grammar present in the genomic data. The mini and the exact definition of the states can be seen in figure 4.8

In the sequence presented in figure 4.6 there are 1361 signals on the forward strand and 1498 on the reverse strand. Expecting more information around the signals we rate them. And truly there are. Two models were used to show this dependency.

First the weight array model. This model can be seen as a markov model. The states are the four nucleotides. But for every position in the signal there is different transition matrix A . Further information are in [22]. This model is used for all the signals except DSSR and DSSF.

Donor splice sites have a very strong dependency. But they do not depend on the direct predecessor as in the other signals. They depend on the fourth and third predecessor, as can be seen in [8]. As the weight array model is not capable the maximum dependency is used. This is documented in [6].

To point out these dependencies the signals are different length. The length and model type can be seen in table 4.3.1.

	model	length	offset
ASSF	WAM	23	20
DSSF	MDM	9	3
TISF	WAM	12	6
TSSF	WAM	12	6
ASSR	WAM	23	3
DSSR	MDM	9	6
TISR	WAM	12	6
TSSR	WAM	12	6

Table 4.3: models and length of each signal

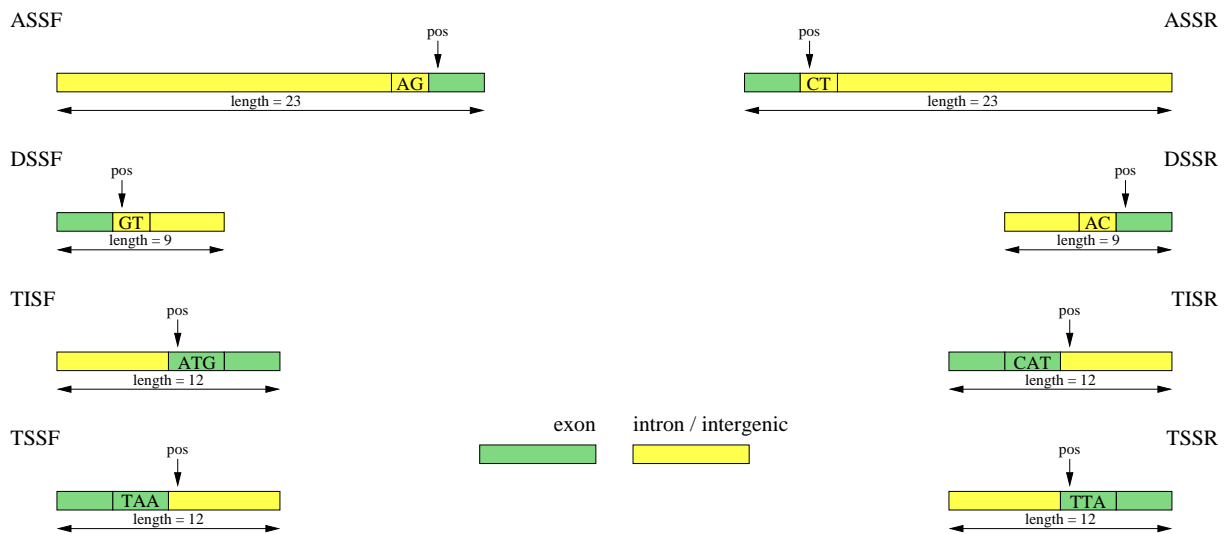


Figure 4.8: Signals

4.3.2 Emission states

The emissions states rate the nucleotides between the signals. They can also be seen as the states defined in an HMM which are to be uncovered. The simplest emission score model is the left side of the model presented in figure 4.5. There only the probability of the individual nucleotide is rated. That is also a kind of markov model of zero order. Increasing the order, the model gets better. This is also implemented in the GHMM. The intergenic and intron states are designed as a fifth order markov model. That means the probability of the nucleotide is calculated depending on the last five nucleotides, see figure 4.9.

The other, more complex model for exons is the interpolated markov model, IMM. If only little training data is available an IMM is useful. Therefore having a markov model 10th order, with an alphabet of four nucleotides, at least $4^{10} = 1048576$ nucleotides are needed. This is the minimum amount of nucleotides needed to get at least one of every possible combination. But this is still not good enough to get a good distribution. And there is no guarantee for every possible combination to appear. This leads to the problem that if a sequence does not appear in the training data it gets rated as impossible in the test data. The interpolated markov model works under this circumstances by reducing the order of the markov model if the probability of a sequence is returned as impossible. This is done until the model does not return the result impossible any longer. The implemented IMM has order eight and is made three times due to its depending on the phase position. The phase problem is discussed in section 4.3.5. More information on IMM see [26].

The emission states are precalculated because otherwise one similar calculation would be computed again and again. The searched probability is then the difference in the precalculated arrays. This is possible because all probabilities are stored in logarithmic scale.

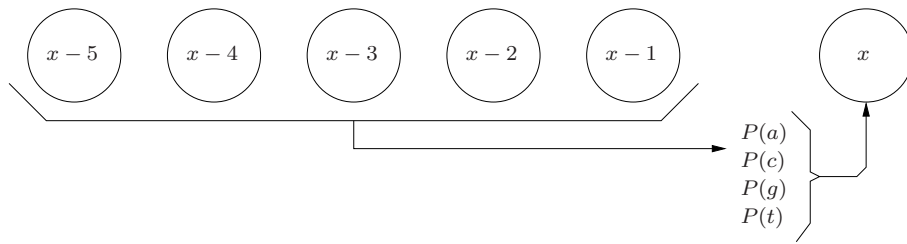


Figure 4.9: A fifth order markov model which uses the last five nucleotides to select the probabilities of the actual nucleotide

4.3.3 Length probability

The length probability is the probability of how long an exon or intron is. There has been found a statistical on the length difference between different exons, introns and intergenic region. On average an initial or terminal exon is longer than an internal exon.

The different length distributions are shown in section 3.2.2. They are taken over from the diploma thesis [8] section 3.3.3. There is no difference made between forward and reverse strand.

4.3.4 Transition probability

Transition probabilities are needed if there are two or more ways out of a state. According to figure 4.7 there are three transition probabilities needed. As in the length probability no difference is made between forward and reverse strand. The values of the probabilities are in the table 4.4.

The P_{s1} and P_{sn} have been copied from [8]. The probability $P(f|f)$ and $P(r|r)$ are calculated from chromosome one.

transition	value
P_{s1}	0.1874
P_{sn}	0.1750
$P(f f)$	0.5644
$P(r r)$	0.5711

Table 4.4: Transition probabilities

4.3.5 The phase problem

The phase problem was one of our greatest challenges. The length of the finished gene has to be able to be divided by three. Due to this it is important to know in every position which of the three phases is relevant. The phase is defined as modulo of the actual length

by three. There are two main problems. First, to know in every position which of the three phases is relevant. Second, due to the precalculation of the phase depending IMM arrays, the actual phase and the precalculated phase must be linked together.

Keep track of the nucleotide count

To keep track of the phase over a possible gene there is an extension necessary. The probability score array, where the summed probabilities are stored, gets beside the dimension for all predecessors another dimension for the phase. Because each signal can have a predecessor of different phase. For an example the nucleotide count between a ASSF and a DSSF is 19. So if the first element is in phase one, then the next element of this sequence is in phase 2. This according to formula

$$phase\ next\ element = ((\pm length + (phase\ first\ element - 1)) \bmod 3) + 1 \quad (4.4)$$

The \pm is a $+$ for the forward strand and a $-$ for the reverse one. That is caused from the "going backward". If it has length x and so the next element gets phase 2 we now increase the length by one and get with the minus phase 1 as next phase. Which is correct. In figure 4.10 you can see what have to count together for the ps and when it is switched over according to the formula 4.4.

The ps array is now calculated according figure 4.10 with the transition box as an implementation of formula 4.4.

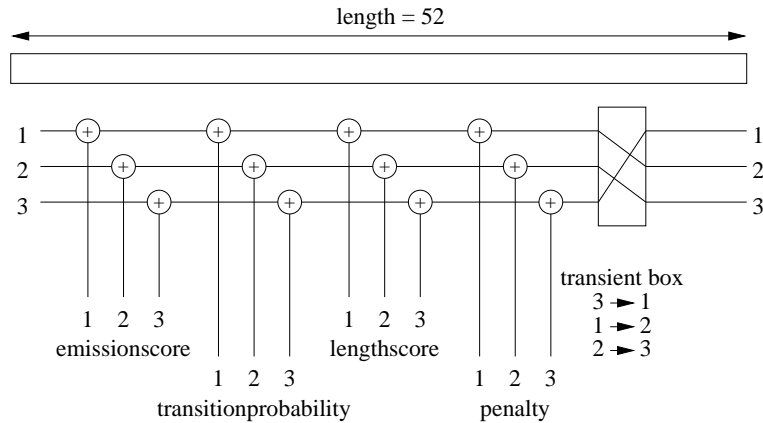


Figure 4.10: How to add the scores and the switch over the phase

Linking of the actual phase and the IMM phase together

The previous section was the first step. Now always the actual phase is known. Linking this phase to the precalculated IMM is the next step. The precalculation of the IMM starts at the beginning or at the end of the sequence. The relative position of the actual signal against the start of the precalculation need to be known. Then it is possible to

calculate which phase of the IMM should be used according to the formula:

$$phase\ IMM\ forward = ((for\ phase - 1 - (pos - 1))\ modulo\ 3) + 1 \quad (4.5)$$

MATLAB

$$phase\ IMM\ reverse = ((for\ phase - 1 + (pos - seq\ len))\ modulo\ 3) + 1 \quad (4.6)$$

For an example using the equation for the forward strand with the *for phase* set to 1 and position set to 29 the result is 3. This means taking the IMM array started with phase three leads to the searched emission score. The minus 1 is for matlab indices only. Due to matlab starts with 1 and not zero.

You can see this result also in figure 4.11 for forward or figure 4.12 for a reverse example.

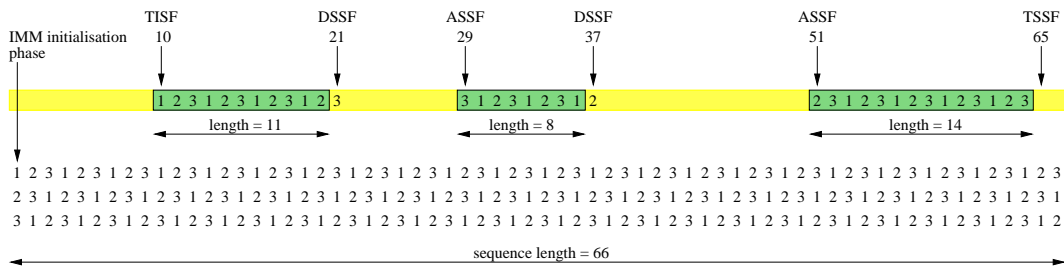


Figure 4.11: An example of a forward strand with signals

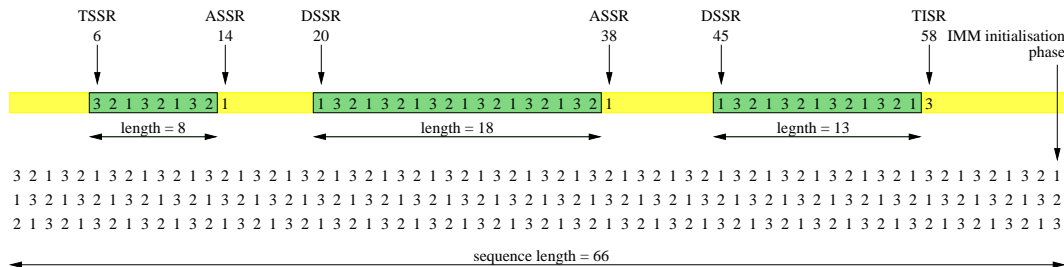


Figure 4.12: An example of a reverse strand with signals

4.3.6 Eclipsing

Eclipsing is making signals improbable. This can be due to grammar information or handling thought.

The genetic code makes it possible to eclipse signals with grammar information. The stop codon always stops a gene. A start codon can be eclipsed if a stop codon appears in phase. Also the phase of an acceptor splice site can be eclipsed which ends after the stop codon in phase. In that way that they do no more participate in the evaluation process of the

best predecessor. So it is possible to save computation time.

The genetic grammar let us eclipse signals at one point. Then when a stop codon appears. Then all start sites and acceptor sites which have a multiple of three between them are stopped. By the acceptor site there is only this phase stopped which ends the exon in phase as showed by the red line in figure 4.13. To this signals we can now add a penalty that they surely do not participate at the evaluation process again. To save computation time, we can check the probability of the indicated score of the signal before we do the process of evaluation the probability. This can truly save computation time.

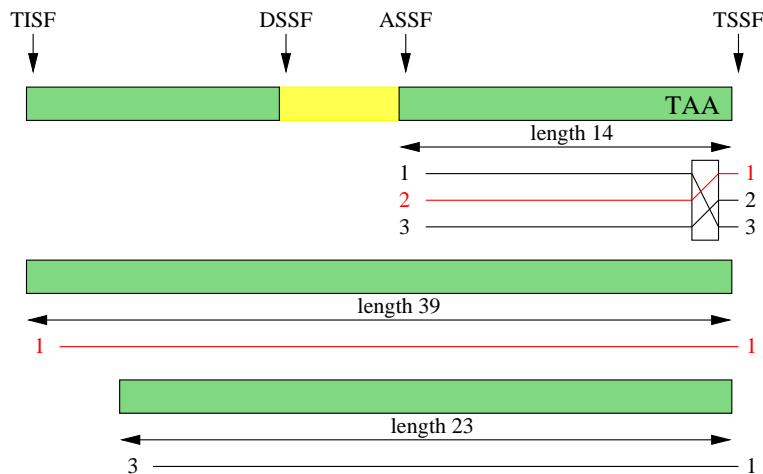


Figure 4.13: Eclipsing examples. The red line shows that a signal or the specific phase can be eclipsed

For long sequences, there must be a possibility to eclipse the signals if they are far away. But here there is no grammar on which the decision can be build. The limit of how far away a signal must be before eclipsing this signal must carefully be set according to training data. The limit was measured in chromosome I. These values were multiplied by 1.5 and set as the limit. The values are showed in table 4.5.

signal	maximal length	eclipsing distance
ASSF	3824	5736
DSSF	4257	6386
TISF	5615	8423
TSSF	76176	114264
ASSR	4948	7422
DSSR	7712	11568
TISR	259842	389763
TSSF	5090	7635

Table 4.5: Distances where the model eclipse the signals

4.3.7 Back tracking

The backtracking algorithm is now a Viterbi algorithm. We go back to the last signal and choose his best predecessor. From this predecessor we choose his best predecessor and so on until the first signal is reached.

To make this work in the implementation there are pseudo states added at the beginning and at the end. This is for better backtracking and for definition of the first and last state which are defined to be in intergenic region.

4.3.8 GHMM step by step

In this section the function of the GHMM and the exact procedure of three sample signals are presented.

First, the signal translation indication start site forward, TISF, is looked at. This signal is simple and shows how basically the GHMM works.

Afterward the signal donor splice site forward, DSSF, will be discussed. Here the phase problem will be introduced. Discussion and solution of the phase problem was in section 4.3.5.

Finally the signal translation indication start side reverse, TISR. This signal shows the problem on the reverse strand.

Translation indication start site, forward direction

```

1 %-----
2 % nc -> start
3 %-----
4
5     if (qn==TISF || theend)
6         %If the actual signal is a TISF then do the following
7
8 %-----
9 % TSSF -> inter genic -> TISF
10     clear ps phaseNextElement
11     %due to the eclipsing, the following index i is not going through all indices.
12     % Therefore set all to -Inf
13     ps = ones(1,idx.to(TSSF) - 1) * -Inf;
14
15     %Calculate the score for the preceeding stop signals which are not eclipsed
16     % In idx.from stands the index of the first signal which is not eclipsed
17     for i=idx.from(TSSF):1:idx.to(TSSF) - 1
18
19         %the featerlen represent here the length of the sequence which is between the signals.
20         % if this length is below zero the signals are to close together and it can not be rated
21         % so it is excluded from the possibility to get the best predecessor
22         featerlen = (tmppos + sigprop(TISF).offset) - ...
23             (queue(TSSF).pos(i) + sigprop(TSSF).offset + sigprop(TSSF).length);
24         if (featerlen <= 0)
25             ps(i) = PENALTY + queue(TSSF).indscore(1,i); % penalty
26             %Next preceeding signal
27             continue;
28         end
29
30     %At the beginning and the end there have been signals added. The length of these signal
31     % can not be rated because there are not true signals
32     if ( (i==1) || theend )
33         lengthscore = 0;
34     else
35         %rate the length of this signal
36         %use lengthdistribution of the intergenic region
37         %calc the bin which it has to use in the histogram
38         lendistidx = floor(featerlen / alpha(1).lendist(2).binsize) + 1;
39         if (lendistidx >= alpha(1).lendist(2).len)
40             %if the length is too long just take the last bin of the histogram
41             lengthscore = alpha(1).lendist(2).dist(alpha(1).lendist(2).len);
42         else
43             %else take the probability of the coresponding bin
44             lengthscore = alpha(1).lendist(2).dist(lendistidx);

```

```

45     end
46 end
47
48 %the transition probability is defined as
49 transprob = log10(stayForward);
50
51 %The emissionscore is evaluated by taking the difference of the precalculated array for
52 % intergenic region at the signal positions.
53 % To get exactly the point after the signal the calculation below is needed.
54 % In tmppos always the position of the actual signal is stored
55 emissionscore = alpha(3).psa(1,tmppos+sigprop(TISF).offset) - ...
56               alpha(3).psa(1,queue(TSSF).pos(i)+sigprop(TSSF).offset+sigprop(TSSF).length);
57
58 %The ps array no gets updated from -Inf where an true symbol is
59 ps(i) = queue(TSSF).indscore(1,i) + emissionweight*emissionscore + ...
60               lengthweight*lengthscore + transprob;
61
62 end
63
64 %The score for each preceding signal is now stored in the ps array
65 % find the most likely preceding signal of this type
66 [predecessor.score(1:3,1), predecessor.index(1:3,1)] = max(ps,[],2);
67 % and save the type
68 predecessor.type(1:3,1) = TSSF;
69
70 %-----
71 % TISR -> inter genic -> TISF
72
73 %The next part is almost the same as above,
74 % changing the signal type and the transition probability
75
76 clear ps phaseNextElement
77 ps = ones(1,idx.to(DSSR)-1) * -Inf;
78
79 for i=idx.from(TISR):1:idx.to(TISR) - 1
80     featurelen = (tmppos + sigprop(TISF).offset) - ...
81               (queue(TISR).pos(i) + sigprop(TISR).offset + sigprop(TISR).length);
82     if (featurelen <= 0)
83         ps(i) = PENALTY + queue(TISR).indscore(1,i); % penalty
84         continue;
85     end
86
87     if ( (i==1) || theend )
88         lengthscore = 0;
89     else
90         lendistidx = floor(featurelen / alpha(1).lendist(2).binsize) + 1;
91         if (lendistidx >= alpha(1).lendist(2).len)
92             lengthscore = alpha(1).lendist(2).dist(alpha(1).lendist(2).len);
93         else
94             lengthscore = alpha(1).lendist(2).dist(lendistidx);
95         end
96     end
97
98     transprob = log10(1 - stayReverse);
99
100    emissionscore = alpha(3).psa(1,tmppos + sigprop(TISF).offset) - ...
101                  alpha(3).psa(1,queue(TISR).pos(i)+sigprop(TISR).offset+sigprop(TISR).length);
102
103    ps(i) = queue(TISR).indscore(1,i) + emissionweight*emissionscore + ...
104            lengthweight*lengthscore + transprob;
105 end
106
107 %find the best predecessor -> TISR
108 % and save the type
109 [predecessor.score(1:3,2), predecessor.index(1:3,2)] = max(ps,[],2);
110 predecessor.type(1:3,2) = TISR;
111
112
113 %find best predecessor for this signal
114 [bestScore, predecessorSelector] = max(predecessor.score,[],2);
115
116 %fill in the queue
117 % first optpre 4 to 6 contain the queue/type of signal which contain the best predecessor
118 % take for every phase the most likely predecessor. This is granted by the predecessor selector.
119 % He points on the corresponding column. The diag command is to take the needed values out of the
120 % array which comes back from the two array access
121 queue(TISF).optpre(4:6,idx.to(TISF)) = diag(predecessor.type([1:3],predecessorSelector));
122 % second optpre 1:3 contain the index in the queue with the best predecessor
123 queue(TISF).optpre(1:3,idx.to(TISF)) = diag(predecessor.index([1:3],predecessorSelector));
124 % and last the new indscore has to be calculated with the best scores and the actual signal score
125 queue(TISF).indscore(1:3,idx.to(TISF)) = bestScore + signalweight*queue(TISF).score(idx.to(TISF));
126
127
128 %increment the index to point to the next signal if not already reached the end
129 if (idx.to(TISF) < lenq(TISF))

```

Listing 4.1: TISF Signal

Acceptor splice site, forward direction

```

1 %-----
2 % exon -> donor ss
3 %-----
4   if (qn==DSSF || theend)
5     %If the actual signal is a DSSF then do the following
6
7 %-----
8 % ASSF -> exon -> DSSF
9
10  clear ps phaseNextElement
11  %due to the eclipsing, the following index i is not going through all indices.
12  % Therefore set all to -Inf
13  ps = ones(3,idx.to(ASSF)-1)*-Inf;
14
15  %Calculate the score for the preceeding stop signals which are not eclipsed
16  % In idx.from stands the index of the first signal which is not eclipsed
17  for i=idx.from(ASSF):1:idx.to(ASSF)-1
18
19      %the featerlen represent here the length of the sequence which is between the signals.
20      % if this length is below zero the signals are to close together and it can not be rated
21      % so it is excluded from the possibility to get the best predecessor
22      featurelen = (tmppos+sigprop(DSSF).offset) - ...
23                  (queue(ASSF).pos(i)+sigprop(ASSF).offset+sigprop(ASSF).length);
24      %In the next line is an idea to not calculate the signal if the inducted score is already -Inf
25      % Test give a look that for calculating this if for all signals lead to longer computation time
26      % that is the reason why it is disabled
27      if (featurelen <= 0) %| (queue(ASSF).indscore == -Inf)
28          ps(1:3,i) = PENALTY + queue(ASSF).indscore(1:3,i);
29          %Next preceeding signal
30          continue;
31      end
32
33      %At the beginning and the end there have been signals added. The length of these signal
34      % can not be rated because there are not true signals
35      if ( (i==1) || theend )
36          lengthscore = 0;
37      else
38          %rate the length of this signal
39          %use lengthdistribution of an internal exon
40          exontype = 2;
41          %calc the bin where it has to use in the histogram
42          lendstidx = floor(featurelen/alpha(2).lendist(exontype).binsize)+1;
43          if (lendstidx >= alpha(2).lendist(exontype).len)
44              %if the length is too long just take the last bin of the histogram
45              lengthscore = alpha(2).lendist(exontype).dist(alpha(2).lendist(exontype).len);
46          else
47              %else take the probability of the coresponding bin
48              lengthscore = alpha(2).lendist(exontype).dist(lendstidx);
49          end
50      end
51
52      %the transition probability is defined as
53      transprob = log10(1-psn);
54
55      %PHASEPROBLEM !!!!!
56      %definition : a donor splice site signal carries in his three prob the following information
57      % phase 1 : the next element is in phase 1 / probability of the hole story
58      % phase 2 : the next element is in phase 2 / probability of the hole story
59      % phase 3 : the next element is in phase 3 / probability of the hole story
60
61      %per definition is the begin at the acceptor site in the corresponding phase
62      % so the range between can directly be rated with the corresponding phase
63      phaseStart = [1:3]';
64
65      %These phases has to be linked together. This is made with equation 4.5
66      phaseIMM = mod(phaseStart-1 - (queue(ASSF).pos(i)-1),3)+1;
67
68      %The emissionscore is evaluated by taking the difference of the precalculated array for
69      % exon region at the signal positions.
70      % As defined in eq 4.5 to get the right precalculated array we need that
71      % stored in the phaseIMM.
72      % To get exactly the point after the signal the calculation below is needed.
73      % In tmppos always the position of the actual signal is stored
74      emissionscore(phaseStart,1) = alpha(2).psa(phaseIMM,tmppos + sigprop(DSSF).offset) - ...
75                                  alpha(2).psa(phaseIMM,queue(ASSF).pos(i)+sigprop(ASSF).offset+...
76                                                  sigprop(ASSF).length);
77
78      %To get the correct next element, another evaluation of the featurelength is needed.
79      % Here the nucleotides between the signal positions are relevant to compute the phase of the
80      % next element
81      featurelen = tmppos - queue(ASSF).pos(i);
82
83      %calculation of the phase of the next element according to equation 4.4
84      phaseNextElement = mod((phaseStart-1) + featurelen,3)+1;
85
86      %Now for the complete probability for this preceeding signal is calculated by:
87      % the preceeding signal score (phase 1 to 3)

```

```

88     % emissionscore between the signal (phase 1 to 3)
89     % lengthscore -> the prob that this length between the two signals can occur
90     % transprob -> the transitionprobability -> much exons are unlikely
91     % at the end store this in the ps array with the new ending phase so that the next
92     % acceptor can take the right phase!
93     % All this is visualized in figure 4.10
94     ps(phaseNextElement,i) = queue(ASSF).indscore(phaseStart,i) + ...
95                             emissionweight*emissionscore(phaseStart,1) + ...
96                             lengthweight*lengthscore + ...
97                             transprob;
98
99     end
100
101     %The score for each preceding signal is now stored in the ps array
102     % find the most likely preceding signal of this type
103     [predecessor.score(1:3,1), predecessor.index(1:3,1)] = max(ps,[],2);
104     % and save the type
105     predecessor.type(1:3,1) = ASSF;
106
107 %-----
108 % TISF -> exon -> DSSF
109
110     %The next part is almost the same as above,
111     % changing the signal type, transition probability, exon type
112     % and the use of the phase information
113
114     clear ps phaseNextElement
115     ps = ones(3,idx.to(TISF)-1)*-Inf;
116
117     for i=idx.from(TISF):1:idx.to(TISF)-1
118
119         featurelen = (tmppos+sigprop(DSSF).offset) - ...
120                     (queue(TISF).pos(i)+sigprop(TISF).offset+sigprop(TISF).length);
121         if (featurelen <= 0) %| (queue(TISF).indscore == -Inf)
122             ps(1:3,i) = PENALTY + queue(TISF).indscore(1:3,i); % penalty
123             continue;
124         end
125
126         if ( (i==1) || theend )
127             lengthscore = 0;
128         else
129             %rate the length of this signal
130             %use lengthdistribution of an start exon
131             exontype = 1;
132             lendistidx = floor(featurelen / alpha(2).lendist(exontype).binsize) + 1;
133             if (lendistidx >= alpha(2).lendist(exontype).len)
134                 lengthscore = alpha(2).lendist(exontype).dist(alpha(2).lendist(exontype).len);
135             else
136                 lengthscore = alpha(2).lendist(exontype).dist(lendistidx);
137             end
138         end
139
140         %the transition probability is defined as
141         transprob = log10(1-ps1);
142
143         phaseStart = [1:3]';
144
145         phaseIMM = mod(phaseStart -1 + queue(TISF).pos(i)-1,3)+1;
146
147         emissionscore(phaseIMM,1) = alpha(2).psa(phaseStart,tmppos+sigprop(DSSF).offset) - ...
148                                     alpha(2).psa(phaseStart,queue(TISF).pos(i)+sigprop(TISF).offset+...
149                                     sigprop(TISF).length);
150
151         featurelen = tmppos - queue(TISF).pos(i);
152
153         phaseNextElement = mod((phaseStart-1) + featurelen,3)+1;
154
155         %Phase penalty (start signal is per definition in phase 1.
156         % The following donor site can therefore only be in one phase depending on the featurelength.
157         phasepenalty = PENALTY*ones(3,1);
158         % The start codon starts per definition in phase 1 therefore no penalty for allowed phase 1
159         phasepenalty(1) = 0;
160
161         %The ps is now extended by the penalty
162         ps(phaseNextElement,i) = queue(TISF).indscore(phaseStart,i) + ...
163                                 emissionweight*emissionscore(phaseStart,1) + ...
164                                 phasepenalty(phaseStart) + ...
165                                 lengthweight*lengthscore + ...
166                                 transprob;
167
168     end
169
170     %The score for each preceding signal is now stored in the ps array
171     % find the most likely preceding signal of this type
172     [predecessor.score(1:3,2), predecessor.index(1:3,2)] = max(ps,[],2);
173     % and save the type
174     predecessor.type(1:3,2) = TISF;
175
176
177     %find best predecessor over all possible predecessor types for this signal
178     [bestScore,predecessorSelector] = max(predecessor.score,[],2);
179

```

```

180 %fill in the queue
181 % first optpre 4 to 6 contain the queue/type of signal which contain the best predecessor
182 % take for every phase the most likely predecessor. This is granted by the predecessor selector.
183 % He points on the corresponding column. The diag command is to take the needed values out of the
184 % array which comes back from the two array access
185 queue(DSSF).optpre(4:6,idx.to(DSSF)) = diag(predecessor.type([1:3],predecessorSelector));
186 % second optpre 1:3 contain the index in the queue with the best predecessor
187 queue(DSSF).optpre(1:3,idx.to(DSSF)) = diag(predecessor.index([1:3],predecessorSelector));
188 % and last the new indscore has to be calculated with the best scores and the actual signal score
189 queue(DSSF).indscore(1:3,idx.to(DSSF)) = bestScore + signalweight*queue(DSSF).score(idx.to(DSSF));
190
191 %increment the index to point to the next signal if not already reached the end
192 if (idx.to(DSSF) < lenq(DSSF))

```

Listing 4.2: ASSF Signal

Translation indication start site, reverse direction

```

1 %-----
2 % exon -> start REVERSE
3 %-----
4
5 if (qn==TISR || theend)
6 %If the actual signal is a TISR then do the following
7
8 %-----
9 % donor ss REVERSE -> exon -> start REVERSE
10
11 clear ps phaseNextElement
12 %due to the eclipsing, the following index i is not going through all indices.
13 % Therefore set all to -Inf
14 ps = ones(1,idx.to(DSSR)-1)*-Inf;
15
16 %Calculate the score for the preceding stop signals which are not eclipsed
17 % In idx.from stands the index of the first signal which is not eclipsed
18 for i=idx.from(DSSR):1:idx.to(DSSR)-1
19
20 %the featerlen represent here the length of the sequence which is between the signals.
21 % if this length is below zero the signals are to close together and it can not be rated
22 % so it is excluded from the possibility to get the best predecessor
23 featurelen = (tmppos+sigprop(TISR).offset) - ...
24 (queue(DSSR).pos(i) + sigprop(DSSR).offset + sigprop(DSSR).length);
25
26 %In the next line is an idea to not calculate the signal if the inducted score is already -Inf
27 % Test give a look that for calculating this if for all signals lead to longer computation time
28 % that is the reason why it is disabled
29 if (featurelen <= 0) %| (queue(DSSR).indscore == -Inf)
30 ps(i) = PENALTY + queue(DSSR).indscore(1,i) ; % penalty
31 %Next preceding signal
32 continue;
33 end
34
35 %At the beginning and the end there have been signals added. The length of these signal
36 % can not be rated because there are not true signals
37 if ( (i==1) || theend )
38 lengthscore = 0;
39 else
40 %rate the length of this signal
41 %use lengthdistribution of an initial exon
42 exontype = 1;
43 %calc the bin where it has to use in the histogram
44 lendistidx = floor(featurelen/alpha(2).lendist(exontype).binsize)+1;
45 if (lendistidx >= alpha(2).lendist(exontype).len)
46 %if the length is too long just take the last bin of the histogram
47 lengthscore = alpha(2).lendist(exontype).dist(alpha(2).lendist(exontype).len);
48 else
49 %else take the probability of the corresponding bin
50 lengthscore = alpha(2).lendist(exontype).dist(lendistidx);
51 end
52 end
53
54 %the transition probability is defined as
55 transprob = log10(psn);
56
57 %PHASEPROBLEM !!!!!
58 %definition : a donor splice site signal carries in his three prob the following information
59 % phase 1 : the next element is in phase 1 / probability of the hole story
60 % phase 2 : the next element is in phase 2 / probability of the hole story
61 % phase 3 : the next element is in phase 3 / probability of the hole story
62
63 %per definition is the begin at the acceptor site in the corresponding phase
64 % so the range between can directly be rated with the corresponding phase
65 phaseStart = [1:3]';
66

```

```

67 %These phases has to be linked together. This is made with equation 4.5
68 phaseIMM = mod(phaseStart - 1 + queue(DSSR).pos(i) - seqlen,3) + 1;
69
70
71 %The emissionscore is evaluated by taking the difference of the precalculated array for
72 % exon region at the signal positions.
73 % For the reverse path the IMM is initialised at the end. As can be seen in figure 4.12
74 % As defined in eq 4.5 to get the right precalculated array we need that
75 % stored in the phaseIMM.
76 % To get exactly the point after the signal the calculation below is needed.
77 % In tmppos always the position of the actual signal is stored
78 emissionscore(phaseStart,1) = alpha(2).psa(phaseIMM+3,seqlen -
79                                     (queue(DSSR).pos(i) + ...
80                                     sigprop(DSSR).offset + ...
81                                     sigprop(DSSR).length) + 1) - ...
82                                     alpha(2).psa(phaseIMM+3,seqlen - ...
83                                     (tmppos+sigprop(TISR).offset)+1);
84
85 %To get the correct next element, another evaluation of the featurelength is needed.
86 % Here the nucleotides between the signal positions are relevant to compute the phase of the
87 % next element
88 featurelen = tmppos - queue(DSSR).pos(i);
89
90 %calculation of the phase of the next element according to equation 4.4
91 phaseNextElement = mod((phaseStart-1) - featurelen,3)+1;
92
93
94 %the start has to be in Phase
95 % Therefore there is only one phase allowed to be finished at the start.
96 % (looking from the opposite direction). So the next element must be in phase 3
97 allowedPhase = find(phaseNextElement == 3);
98
99 %The prediction score then is reduced to a single dimension array.
100 % Because next element has to be in phase 3. See figure 4.12
101 ps(i) = queue(DSSR).indscore(allowedPhase,i) + emissionweight*emissionscore(allowedPhase,1)+...
102                                     lengthweight*lengthscore + transprob;
103
104 end
105
106 %The score for each preceding signal is now stored in the ps array
107 % find the most likely preceding signal of this type
108 [predecessor.score(1:3,1), predecessor.index(1:3,1)] = max(ps,[],2);
109 % and save the type
110 predecessor.type(1:3,1) = DSSR;
111
112 %-----
113 % TSSR -> exon -> TISR
114
115 %The next part is almost the same as above,
116 % changing the signal type, transition probability, exon type
117 % and the use of feature length information
118
119 clear ps phaseNextElement
120 ps = ones(1,idx.to(TSSR)-1)*-Inf;
121
122 for i=idx.from(TSSR):1:idx.to(TSSR)-1
123
124     featurelen = (tmppos+sigprop(TISR).offset) - ...
125                 (queue(TSSR).pos(i)+sigprop(TSSR).offset+sigprop(TSSR).length);
126
127     if (featurelen <= 0) %| (queue(TSSR).indscore == -Inf)
128         ps(i) = PENALTY + queue(TSSR).indscore(1,i) ;
129         continue;
130     end
131
132     if ( (i==1) || theend )
133         lengthscore = 0;
134     else
135         %rate the length of this signal
136         %use lengthdistribution of an single exon
137         exontype = 4;
138         lendistidx = floor(featurelen/alpha(2).lendist(exontype).binsize)+1;
139         if (lendistidx >= alpha(2).lendist(exontype).len)
140             lengthscore = alpha(2).lendist(exontype).dist(alpha(2).lendist(exontype).len);
141         else
142             lengthscore = alpha(2).lendist(exontype).dist(lendistidx);
143         end
144     end
145
146 %the transition probability is defined as
147 transprob = log10(ps1);
148
149 phaseStart = [1:3]';
150
151 phaseIMM = mod(phaseStart - 1 + queue(TSSR).pos(i) - seqlen,3)+1;
152
153 emissionscore_temp(phaseStart,1) = alpha(2).psa(phaseIMM+3,seqlen-(queue(TSSR).pos(i) + ...
154                                     sigprop(TSSR).offset + ...
155                                     sigprop(TSSR).length)+1) - ...
156                                     alpha(2).psa(phaseIMM+3,seqlen-(tmppos+...
157                                     sigprop(TISR).offset)+1);
158

```

```

159 %The stop codon is in phase 3 per definition. See figure 4.12
160 %Therefore the emission score is this one wich starts in phase 3
161 emissionscore = emissionscore_temp(3);
162
163 %The exact feature length is needed, because the grammar information tells that a gene always
164 % must be a multiple of 3
165 featurelen = tmppos - queue(TSSR).pos(i);
166
167 %If the feature length is not i multiple of 3 that is according to grammar information no valid
168 % combination and get a penalty.
169
170 if mod(featurelen,3) == 0
171     %multiple of 3, no penalty
172     ps(i) = queue(TSSR).indscore(3,i) + emissionweight*emissionscore + ...
173             lengthweight*lengthscore + transprob;
174 else
175     %not a multiple of 3, adding a penalty
176     ps(i) = PENALTY + queue(TSSR).indscore(3,i);
177 end
178
179 end
180
181 %The score for each preceding signal is now stored in the ps array
182 % find the most likely preceding signal of this type
183 [predecessor.score(1:3,2), predecessor.index(1:3,2)] = max(ps,[],2);
184 % and save the type
185 predecessor.type(1:3,2) = TSSR;
186
187 %find best predecessor over all possible predecessor types for this signal
188 [bestScore,predecessorSelector] = max(predecessor.score,[],2);
189
190 %fill in the queue
191 % first optpre 4 to 6 contain the queue/type of signal which contain the best predecessor
192 % take for every phase the most likely predecessor. This is granted by the predecessor selector.
193 % He points on the corresponding column. The diag command is to take the needed values out of the
194 % array which comes back from the two array access
195 queue(TISR).optpre(4:6,idx.to(TISR)) = diag(predecessor.type([1:3],predecessorSelector));
196 % second optpre 1:3 contain the index in the queue with the best predecessor
197 queue(TISR).optpre(1:3,idx.to(TISR)) = diag(predecessor.index([1:3],predecessorSelector));
198 % and last the new indscore has to be calculated with the best scores and the actual signal score
199 queue(TISR).indscore(1:3,idx.to(TISR)) = bestScore + signalweight*queue(TISR).score(idx.to(TISR));
200
201 %The start signal reverse is an anchor in the genetic grammar. If a stop codon appears every DSSR
202 % and TSSR which are in phase are set impossible.
203 for i=idx.from(TSSR):1:idx.to(TSSR)-1 % stop REVERSE
204     featurelen = tmppos - queue(TSSR).pos(i);
205     if (mod(featurelen,3) == 0)
206         queue(TSSR).indscore(:,i) = queue(TSSR).indscore(:,i) + PENALTY_ECLIPSE; % penalty
207     end
208 end
209 for i=idx.from(DSSR):1:idx.to(DSSR)-1 % donor ss REVERSE
210     featurelen = tmppos - queue(DSSR).pos(i);
211     phaseNextElement = mod((phaseStart-1) - featurelen,3)+1;
212     endedPhase = find(phaseNextElement == 3);
213     queue(DSSR).indscore(endedPhase,i) = queue(DSSR).indscore(endedPhase,i) + PENALTY_ECLIPSE;
214 end
215
216 %increment the index to point to the next signal if not already reached the end
217 if (idx.to(TISR) < lenq(TISR))
218     idx.to(TISR) = idx.to(TISR) + 1;

```

Listing 4.3: TISR Signal

Chapter 5

Support Vector Machine

5.1 Introduction

The Support Vector Machine (SVM) is a mathematical model for pattern recognition. Although the SVM was first introduced by Vapnik in 1979, the concept only became popular the last decade. SVMs have been used with success for speaker identification, face detection in images and handwritten digit recognition during the last years. The following introduction to the function of SVMs follows strongly the explanation of Christopher J.C Burges [5], Christianini and Taylor [23] and Eitrich [10], which are very good sources for an outline as well as a deeper insight into SVMs.

5.2 Support Vector Classification

The main idea of SVMs is to separate two different classes by a hyperplane. Figure 5.1 illustrates this idea with two dimensional feature vectors.

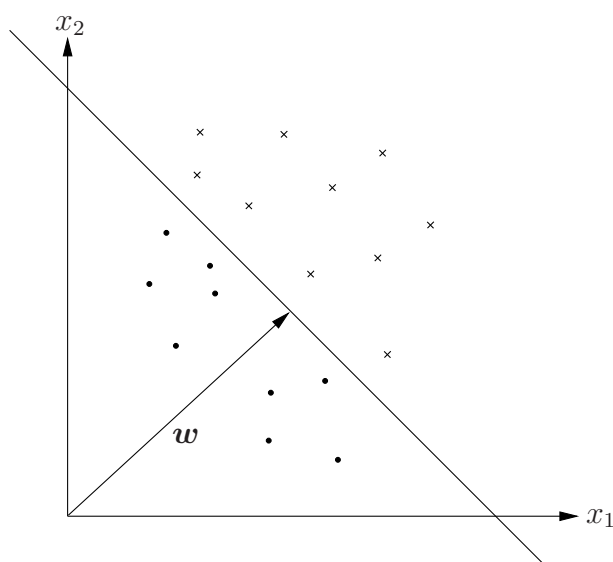


Figure 5.1: Concept of a hyperplane as a linear classifier

Since the SVM is a training based method it needs a training set. This training set contains l feature vectors $X \subseteq \mathbb{R}^n$ and the corresponding labels $Y \in \{1, -1\}$. A hyperplane

in an n dimensional space can now be represented in the following way:

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \tag{5.1}$$

If the hyperplane is known, a new vector can be classified by simply evaluating the function

$$f(\mathbf{x}) = \frac{\langle \mathbf{w}, \mathbf{x} \rangle + b}{\|\mathbf{w}\|} \tag{5.2}$$

which is the signed distance to the hyperplane.

By taking the sign of f one gets the most likely label y corresponding to the input vector \mathbf{x} :

$$y = \text{sign}(f(\mathbf{x}))^1 \tag{5.3}$$

Now the question remains how to find the best hyperplane to make the classification work at its best.

5.2.1 The maximal margin

Hard margin

As one can see, there are endless possibilities of hyperplanes separating the two classes. As one would intuitively say, the best hyperplane is found by maximizing its margin γ to the closest input vector.

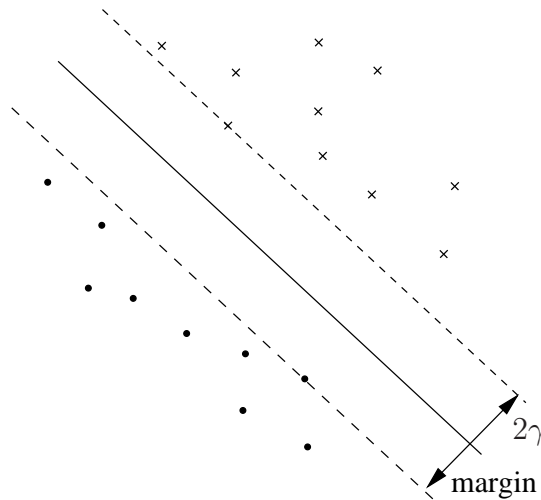


Figure 5.2: Margin around the hyperplane

The margin of the hyperplane to an input vector \mathbf{x}^i is

¹Note, this is a modified sign function with $\text{sign}(0) = 1$

$$\gamma = y_i \frac{\langle \mathbf{w}, \mathbf{x}^i \rangle + b}{\|\mathbf{w}\|}. \quad (5.4)$$

Now, the maximal γ , for which following inequality applies to, is looked for.

$$\begin{aligned} \gamma &\leq y_i \frac{\langle \mathbf{w}, \mathbf{x}^i \rangle + b}{\|\mathbf{w}\|} \\ \gamma \|\mathbf{w}\| &\leq y_i (\langle \mathbf{w}, \mathbf{x}^i \rangle + b) \quad \forall i \end{aligned} \quad (5.5)$$

By setting $\|\mathbf{w}\| = 1/\gamma$ the equation 5.5 simplifies to

$$y_i (\langle \mathbf{w}, \mathbf{x}^i \rangle + b) \geq 1 \quad \forall i, \quad (5.6)$$

which leads to the following maximizing problem:

$$\min_{\mathbf{w}, b} \|\mathbf{w}\| \quad (5.7)$$

$$\text{with the side condition } y_i (\langle \mathbf{w}, \mathbf{x}^i \rangle + b) \geq 1 \quad \forall i$$

Without losing the correct solution of \mathbf{w} , equation 5.7 can be rewritten to

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad (5.8)$$

$$\text{with the side condition } y_i (\langle \mathbf{w}, \mathbf{x}^i \rangle + b) \geq 1 \quad \forall i$$

which makes it easier to form the Lagrangian dual problem.

Soft Margin

Up to now only the case of separable data was considered. However, in practice this is not always the case. There is a real chance to get some data, which can not get separated by an hyperplane. To manage this cases, the SVM is modified in order to allow mistakes. The so called "slack variables" ξ_i represents the error distance of a wrong classified input vector \mathbf{x}^i . As shown in Figure 5.3, the error distance is always positive for misclassified input vectors and zero for correct classified input vectors.

$$\xi_i = \max(0, \gamma - y_i f(\mathbf{x}^i)) \quad (5.9)$$

With this slack variables equation 5.6 changes to

$$y_i (\langle \mathbf{w}, \mathbf{x}^i \rangle + b) \geq 1 - \xi_i \quad \forall i. \quad (5.10)$$

It makes sense to include the error sum of the slack variables ξ_i into the maximizing problem of equation 5.8. This is done in two different ways, with a common penalty variable C .

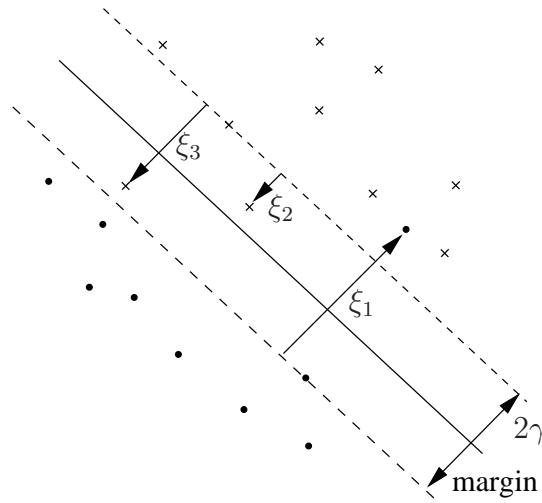


Figure 5.3: Slack variables defining a soft margin

1 Norm Soft Margin

The 1 Norm Vector Machine weights the slack variables linear, so that the minimizing problem can be rewritten to

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^l \xi_i \quad (5.11)$$

with the side conditions $y_i (\langle \mathbf{w}, \mathbf{x}^i \rangle + b) \geq 1 - \xi_i$ and $\xi_i \geq 0 \quad \forall i$

The Lagrangian can now be formulated by

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^l \xi_i - \sum_{i=1}^l \alpha_i (y_i (\langle \mathbf{w}, \mathbf{x}^i \rangle + b) - 1 + \xi_i) - \sum_{i=1}^l r_i \xi_i \quad (5.12)$$

with the side conditions $\alpha_i \geq 0$ and $r_i \geq 0 \quad \forall i$

By differentiating equation 5.12 with respect to \mathbf{w}, ξ_i and b and equate to zero

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^l y_i \alpha_i \mathbf{x}^i = \mathbf{0} \quad (5.13)$$

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r})}{\partial \xi_i} = C - \alpha_i - r_i = 0 \quad (5.14)$$

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r})}{\partial b} = \sum_{i=1}^l y_i \alpha_i = 0 \quad (5.15)$$

the dual Lagrangian problem can be formulated by resubstituting the obtained equations into the primal Lagrangian problem (see [23] for details).

$$W(\alpha) = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l y_i y_j \alpha_i \alpha_j \langle \mathbf{x}^i, \mathbf{x}^j \rangle \quad (5.16)$$

The optimal α can now be found by

$$\max_{\alpha} W(\alpha) \quad (5.17)$$

$$\text{with the side condition } 0 \leq \alpha_i \leq C \text{ and } \sum_{i=1}^l \alpha_i y_i = 0 \quad \forall i$$

This is a quadratic programming problem for which a lot of numeric algorithms are available.

2 Norm Soft Margin

The 2 Norm SVM weights the error of the slack variables ξ_i quadratic. Equation 5.8 changes to

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^l \xi_i^2 \quad (5.18)$$

$$\text{with the side condition } y_i (\langle \mathbf{w}, \mathbf{x}^i \rangle + b) \geq 1 - \xi_i \quad \forall i$$

Whereas the Lagrangian problem is

$$L(\mathbf{w}, b, \xi, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 + \frac{C}{2} \sum_{i=1}^l \xi_i^2 - \sum_{i=1}^l \alpha_i (y_i (\langle \mathbf{w}, \mathbf{x}^i \rangle + b) - 1 + \xi_i) \quad (5.19)$$

$$\text{with the side condition } \alpha_i \geq 0 \quad \forall i$$

By differentiating equation 5.19 again with respect to \mathbf{w} , ξ and b and setting the differential functions equal to zero

$$\frac{\partial L(\mathbf{w}, b, \xi, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^l y_i \alpha_i \mathbf{x}^i = \mathbf{0} \quad (5.20)$$

$$\frac{\partial L(\mathbf{w}, b, \xi, \alpha)}{\partial \xi} = \xi C - \alpha = \mathbf{0} \quad (5.21)$$

$$\frac{\partial L(\mathbf{w}, b, \xi, \alpha)}{\partial b} = \sum_{i=1}^l y_i \alpha_i = 0 \quad (5.22)$$

and resubstituting the solutions into the primal Lagrangian the dual Lagrangian is obtained.

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l y_i y_j \alpha_i \alpha_j \left(\langle \mathbf{x}^i, \mathbf{x}^j \rangle + \frac{1}{C} \delta_{i,j} \right) \quad (5.23)$$

$\delta_{i,j}$ is the Kronecker delta with $\delta_{i,j} = 1$ when $i = j$ and 0 otherwise. The optimal solution is found again by

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) \quad (5.24)$$

with the side conditions $\alpha_i \geq 0$ and $\sum_{i=1}^l \alpha_i y_i = 0 \quad \forall i$

Note: The following explanation are always based on the 1 Norm SVM.

5.3 The Kernel Trick

Although with the soft margin principle an optimal hyperplane can always be found in the input space, there might exist an feature space in which it might be easier to find the optimal hyperplane. As shown in Figure 5.4 the mapping $F = \{\Phi(\mathbf{x}) | \mathbf{x} \in X\}$ maps the input space $X \subseteq \mathbb{R}^n$ to the feature space $F \subseteq \mathbb{R}^m$. Since the mapping $\Phi(\mathbf{x})$ does not have to be linear, it extends the SVM to a nonlinear classifier.

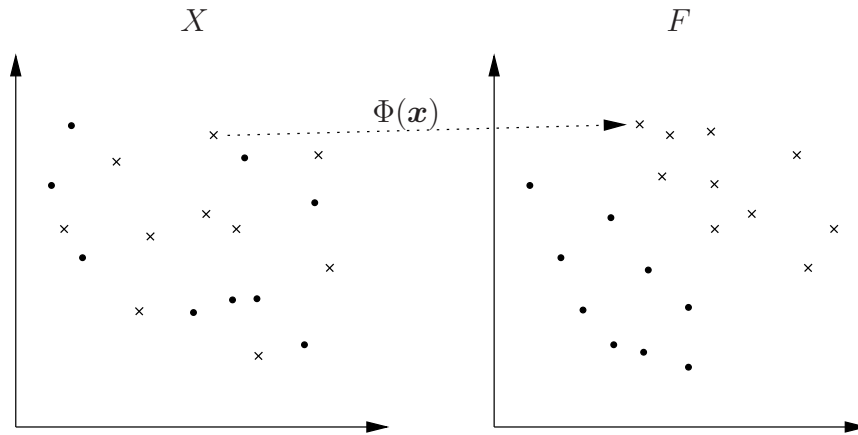


Figure 5.4: Mapping of the input space X to the feature space F

One could now just substitute \mathbf{x}^i and \mathbf{x}^j with $\Phi(\mathbf{x}^i)$ and $\Phi(\mathbf{x}^j)$ respectively in equation 5.16.

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l y_i y_j \alpha_i \alpha_j \langle \Phi(\mathbf{x}^i), \Phi(\mathbf{x}^j) \rangle \quad (5.25)$$

However, for high m this mapping can't be accomplished in reasonable time and for $m \rightarrow \infty$ it's even impossible to compute the mapping. Hence, a new approach needs to be found.

Looking at equation 5.25 it stands out, that the mapping function $\Phi(\mathbf{x})$ appears only in the inner product.

This situation implies to substitute the whole inner product $\langle \mathbf{x}^i, \mathbf{x}^j \rangle$ instead of the individual input vectors \mathbf{x}^i and \mathbf{x}^j :

$$K(\mathbf{x}^i, \mathbf{x}^j) = \langle \Phi(\mathbf{x}^i), \Phi(\mathbf{x}^j) \rangle \quad (5.26)$$

This function is called kernel. The kernel needs to satisfy the Mercer's condition [23] in order to define the dot product in a feature space F . The Mercer's condition states that a kernel can be constructed without knowing the mapping $\Phi(\mathbf{x})$. The following list shows some of the most popular kernels for SVMs:

- Polynomial kernel: $K(\mathbf{x}, \mathbf{z}) = (\langle x, z \rangle + a)^d$
- Radial Basis Function (RBF) kernel: $K(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|x-z\|^2}{2\sigma}\right)$
- Sigmoid kernel: $K(\mathbf{x}, \mathbf{z}) = \tanh(\kappa_1 \langle x, z \rangle + \kappa_2)$

The respective parameters a, d, σ, κ_1 and κ_2 must be found using an iterative process to find the optimal hyperplane (see 5.6.1).

5.4 The Support Vectors

After solving the dual quadratic optimizing problem in equation 5.16 the resulting \mathbf{a}^* stores the information about the hyperplane. All input vectors \mathbf{x}^i with a corresponding $a_i^* \neq 0$ are called Support Vectors. Only these vectors are needed to define the hyperplane. Moreover all input vectors \mathbf{x}^i , for which $a_i^* = C$ holds, are misclassified vectors.

5.5 Decision Function

5.5.1 Linear classifier

If the SVM was used as a linear classifier (no kernel used) than \mathbf{a}^* can be used to compute the base vector \mathbf{w}^* :

$$\mathbf{w}^* = \sum_{i=1}^l a_i^* y_i \mathbf{x}^i \quad (5.27)$$

Equation 5.1 holds for every support vector with $0 < a_i^* < C$. To compute b^* the equation needs only to be reformed to

$$b^* = \frac{1}{y_i} - \langle \mathbf{w}^*, \mathbf{x}^i \rangle = y_i - \langle \mathbf{w}^*, \mathbf{x}^i \rangle \quad \forall i \text{ with } 0 < a_i^* < C \quad (5.28)$$

Now the class y can simply be evaluated by taking the sign of the following decision function:

$$f(\mathbf{x}) = \langle \mathbf{w}^*, \mathbf{x} \rangle + b^* \quad (5.29)$$

5.5.2 Nonlinear classifier

If the SVM was used as a nonlinear classifier, \mathbf{w}^* can't be computed with equation 5.27 since \mathbf{w}^* lives in the feature space F . Furthermore the mapping function $\Phi(\mathbf{x})$ is mostly not known, so it's impossible to compute the direct \mathbf{w}^* . Although, with equation 5.27 the decision function 5.29 can be rewritten to:

$$f(\mathbf{x}) = \sum_{i=1}^l a_i^* y_i K(\mathbf{x}^i, \mathbf{x}) + b^* \quad (5.30)$$

Since the distance of every support vector with $0 < a_j^* < C$ to the hyperplane is one, the bias b^* can be computed² in the following way:

$$b^* = y_j - \sum_{i=1}^l a_i^* y_i K(\mathbf{x}^i, \mathbf{x}^j) \quad \forall i \text{ with } 0 < a_i^* < C \quad (5.31)$$

The class y can now be evaluated by taking the sign of the decision function 5.30.

5.6 Parameter Evaluation

5.6.1 Grid–Search

The penalty parameter C and the kernel parameters a, d, σ, κ_1 and κ_2 are not an essential part of the optimizing problem in equation 5.16. This means that the optimal hyperplane is found depending on these parameters. To find the best parameters, the optimizing problem has to be solved for every combination of these parameters. This is best done by a two level grid–search. In a first step the accuracy of the SVM is measured for a set of wide stepped parameters. Then the parameter set with the highest accuracy is taken and the grid–search is done again, but now only around the maxima of the highest accuracy and with narrow stepped parameters. Then, the parameters corresponding to the highest accuracy are taken for the final SVM.

²Theoretically, for every j the same b^* is obtained, so that the equation must be computed only once. However, due to numerical reasons it makes sense to compute the equation for every j and to take the mean for b^* at the end

5.6.2 Cross-Validation

Since the SVM is a learning machine based on a training set, a test set is always needed to test the prediction of the SVM. However, there is not always enough data to build a reasonable training and test set. The following algorithm, called Cross-Validation, is used to test the prediction of the SVM if there is only one training set available:

```

1 devide TrainingSet into n subsets
2
3 for i=1 to n
4     remove the i-th set
5     train the SVM with the remaining n-1 subsets
6     prediction=prediction+prediction of the i-th subset
7 end
8 return prediction/n

```

Listing 5.1: Cross-Validation algorithm

5.7 Multiclass SVM

Although the SVM is a binary classifier, it's possible to extend its application to multiclass problems. There exist three popular methods to handle multiclass problems, namely “one against one”, “one against all” and “error correction codes” .

5.7.1 One against one

In this method the SVM is trained with every combination of two out of k classes. This leads to $k(k - 1)/2$ different SVMs. Once all this SVMs have been trained, prediction is given after a score system. A new input vector is tested on every SVM and every time a class is predicted, its score is increased. In the end, the class with the highest score is taken to represent the best class corresponding to the input vector.

5.7.2 One against all

This method uses one SVM for every class. To train one SVM, $k - 1$ out of k classes are put together to form a pseudo class. Then the SVM is trained with the remaining class and the new built pseudo class. This is done for every single class. To find the corresponding class to an input vector the distance of the input vector to every hyperplane is calculated and the class with the longest distance is taken to be the best prediction. Vojtěch and Hlaváček [30] developed a SVM using a special kernel, which implemented the “one against all” strategic into a single SVM.

Class	Code Word					
	f_0	f_1	f_2	f_3	f_4	f_5
	vertical line	horizontal line	diagonal line	closed curve	curve open to left	curve open to right
0	0	0	0	1	0	0
1	1	0	0	0	0	0
2	0	1	1	0	1	0
3	0	0	0	0	1	0
4	1	1	0	0	0	0
5	1	1	0	0	1	0
6	0	0	1	1	0	1
7	0	0	1	0	0	0
8	0	0	0	1	0	0
9	0	0	1	1	0	0

Table 5.1: Error Correction Output matrix for digit recognition. Source: [29]

5.7.3 Error Correction Output Code

The use of Error Correction Output Codes (ECOC) to reduce a multi-class problem to several class problems was introduced by T. Dietterich and G. Bakiri [29]. The main idea of this approach is to assign each class to a code word. The code word matrix is generated by training different SVMs. Table 5.1 shows an example for digit recognition. The SVMs f_i are trained to identify if a digit contains different line geometrics.

To predict a new digit the output of every SVM f_i is evaluated and the class with the smallest hamming distance is taken. High accuracy for multi-class prediction can be obtained, by finding SVMs f_i to form error correction matrices like BHC.

5.8 Probabilistic Output

Platt [25] introduced a way to map the geometrical distance from a vector to the hyperplane to probabilities. With this extension it is thinkable to combine the SVM with statistical models like HMM.

5.9 Implementation

A small SVM has been implemented in Matlab with use of Mathworks Optimization Toolbox. Although this Matlab code works well with small training sets, it does not for training sets with thousands of vectors. The amount of memory needed is $O(N^2)$, therefore decomposing algorithms have to be used, which are not implemented.

For this thesis the LIBSVM [7] has been used. LIBSVM is a SVM implemented in C providing a wrapper class, so it can be used from Matlab. Moreover it is a very fast implementation and it works fine with up to 10000 training vectors.

Chapter 6

Application of the SVM on genetic problems

6.1 Introduction

SVMs have been used in a lot of different applications in bioinformatics. In [24] a good overview of these different fields of application are shown.

The most popular approach is to use a SVM to identify Translation Initiation Sites. For this problem several kernel have been developed, of which the best is described below.

6.2 Translation Initiation Sites (TIS)

A Translation Initiation Side is a start codon (“ATG” in the 5’3’ direction or “CAT” in the 3’5’ direction) which is a real start codon, meaning that it is the start of an initial exon. This problem is discussed in several publications. Pedersen and Nielsen [1] used a neuronal network and achieved an accuracy of about 80%.

With specialised kernels for SVMs the accuracy has been improved up to 99% (Salzberg Kernel [2], Edit Kernel [11]).

6.2.1 Edit Kernel

Levenshtein distance

The edit kernel is based on the weighted Levenshtein distance (also called edit distance). The Levenshtein distance is a measure on how many operations are needed to change a string A to a string B. Allowed operations are insertion, substitution or deletion. Each of this operation is associated with a cost, so for example an insertion can be more “expensive” then a substitution. The following listing shows the algorithm.

```

1 string A % input
2 string B % input
3
4 indel=1 % cost for a insertion or deletion
5 subst=1 % cost for a substitution
6
7 % Matrix with the width A+1 and the height B+1
8 distance=zeros(length(a)+1, length(b)+1);
9
10 % Initalise the 1. column of the matrix with the numbers 0 to A
11 distance(:,1)=[0:length(a)]';
12 % Initalise the first row of the matrix with the numbers 0 to B
13 distance(1,:)=0:length(b);
14
15 for i=2:length(A)+1
16     for j=2:length(b)+1
17         if B(j-1)~=A(i-1) % If the two string positions
18                             % aren't the same, the cheapest
19                             % operation is taken
20             vertical=distance(i-1,j)+indel;
21             horizontal=distance(i,j-1)+indel;
22             subst=distance(i-1,j-1)+cost(b(i-1),a(j-1));
23             % Take the cheapest possible operation
24             distance(i,j)=min([vertical horizontal subst]);
25         else % otherwise the cost is zero
26             distance(i,j)=distance(i-1,j-1);
27         end
28     end
29 end
30
31
32 % The weighted Levenshtein distance is
33 % now in the cell distance(length(A)+1,length(B)+1)

```

Listing 6.1: Levenshtein Algorithm

As an example the Levenshtein distance of the two strings “Magma” and “Magazin” is calculated (same cost for all operations). The Levenshtein distance is in the right lower cell (4):

		M	a	g	m	a
	0	1	2	3	4	5
M	1	0	1	2	3	4
a	2	1	0	1	2	3
g	3	2	1	0	1	2
a	4	3	2	1	1	1
z	5	4	3	2	2	2
i	6	5	4	3	3	3
n	7	6	5	4	4	4

Table 6.1: Example of the Levenshtein algorithm

The Edit Kernel uses two different weighted Levenshtein distances. One distance is calculated upstream of the start codon and the other distance is calculated downstream of the start codon. While the upstream distance is directly calculated from the nucleotides, the downstream sequence is translated to amino acids first, and then the Levenshtein distance is calculated. The costs for substitution are based on the PAM Matrix.

PAM–Matrix

The PAM (Point Accepted Mutation) was introduced by Dayhoff [20]. It describes the probability of mutating one amino acid into an other one when 1 percent of the protein has changed in one time unit. So the entries in the PAM1 matrix are of the form $P(b|a, t = 1)$. To get the probabilities $P(b|a, t = x)$, the PAM1 matrix is simply raised by the power of x . The edit kernel uses a modified PAM250. Since the PAM Matrix is asymmetric, and probabilities are too small to define costs, the PAM250 matrix is modified by [2] as follows:

1. $\text{ASCM250} \leftarrow -\log \text{PAM250}$
2. Calculate the average value m of the diagonal of M
3. $\text{ASCM250} \leftarrow \text{ASCM250} - m$
4. Set the diagonal and all elements < 0 to zero

In a similar way a substitution matrix for the upstream Levenshtein distance is calculated, resulting in SCM250. See appendix E for the PAM1, ASCM250 and SCM250 matrices.

With these two Levenshtein distances the final edit kernel looks like this:

$$K_{edit}(\mathbf{x}^i, \mathbf{x}^j) = \exp \left[-\gamma_1 \cdot \text{Levenshtein}(\mathbf{x}_{up}^i, \mathbf{x}_{up}^j) - \gamma_2 \cdot \text{Levenshtein}(\mathbf{x}_{down}^i, \mathbf{x}_{down}^j) \right] \quad (6.1)$$

The two parameters γ_1 and γ_2 are found by a two level grid search.

Chapter 7

Identifying Coding Regions

7.1 Introduction

During the last years several methods have been developed to identify coding regions. Fickett presents about twenty different methods in his paper [14]. A closer look on four methods, which are later used in combination with a SVM, is given in this chapter.

7.2 Fourier Analysis

The Fourier Analysis was introduced by Tiwari *et al* [27]. It uses the fact that coding regions are more likely to have higher energy in the third spectral period.

To transform a given DNA sequence, the first thing needed is some kind of mapping nucleotides to numbers. Several mappings are possible, for example a conjugate complex mapping like $A \rightarrow 1 + j$, $C \rightarrow -1 + j$, $G \rightarrow -1 - j$ and $T \rightarrow 1 - j$. Here a binary mapping introduced by [31] is used. As shown in table 7.1, every base $b \in \{A, C, G, T\}$ is stored in a separate array U_b , with the symbols $\{0, 1\}$, indicating the appearance at a specific position i in a DNA sequence.

x_j	A	C	A	T	C	G
U_A	1	0	1	0	0	0
U_C	0	1	0	0	1	0
U_G	0	0	0	0	0	1
U_T	0	0	0	1	0	0

Table 7.1: Binary mapping

Now the full fourier spectrum is simply the sum of the four transformed binary sequences

$$S\left(\frac{k}{N}\right) = \sum_b S_a\left(\frac{k}{N}\right) = \sum_b \frac{1}{N^2} \left| \sum_{i=1}^N U_b(x_i) e^{j2\pi \frac{k}{N} i} \right|^2 \quad (7.1)$$

In figure 7.1(b)¹ a significant peak is clearly visible at position $S\left(\frac{N}{3}\right)$ in the coding region, while in the non coding region the spectrum is mainly flat.

¹Since a binary mapping is always resulting in a high DC energy, the mean of every sequence U_b has been removed before transformation for visual purposes

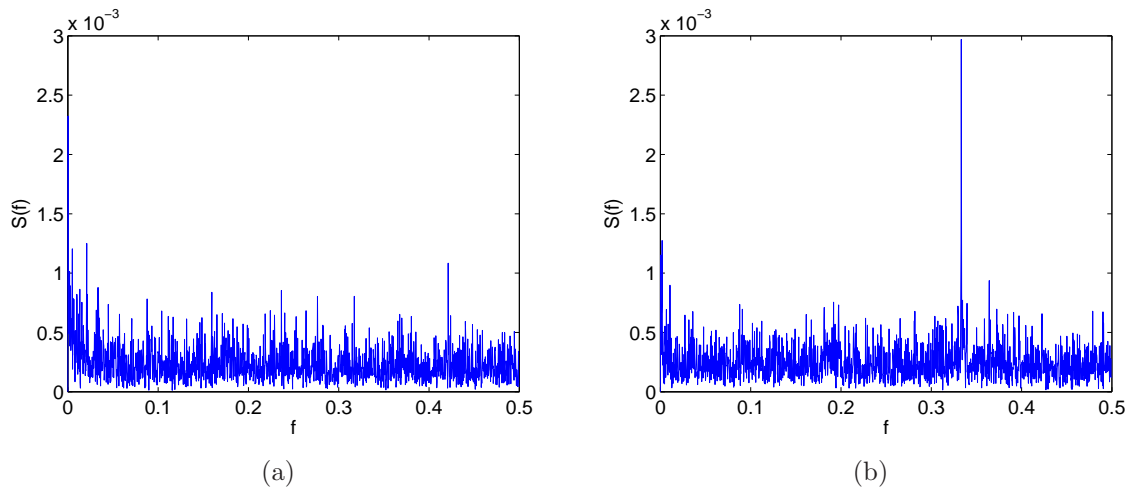


Figure 7.1: Fourier transformation of a DNA region a) noncoding , b) coding

The energy of this peak at $\frac{N}{3}$ is compared to the average energy

$$\bar{S} = \frac{2}{N} \sum_{k=1}^{N/2} S\left(\frac{k}{N}\right) = \frac{1}{N} \left(1 + \frac{1}{N} - \sum_b \rho_b^2 \right) \quad (7.2)$$

in the entire DNA sequence and is taken as an indicator P for coding regions. ρ_b denotes the frequency of occurrence of every base in the sequence of the length N .

$$P = \frac{S\left(\frac{N}{3}\right)}{\bar{S}} \quad (7.3)$$

By sliding a window over a long DNA sequence P can be calculated for every single window. Figure 7.2 shows the idea of this process.

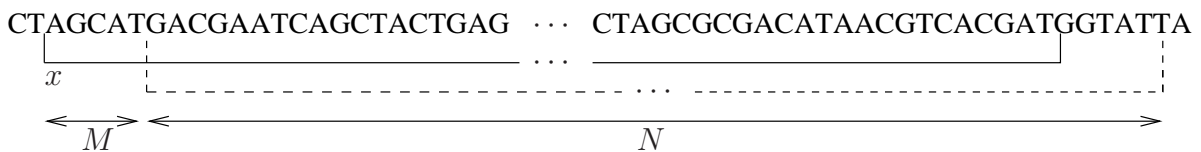


Figure 7.2: Sliding Window

The fourier analysis is done for a window with the length N , beginning at position x . The resulting P is then saved to the corresponding index $j = x + N/2$. Then the fourier window is shifted by M nucleotides downstream and the process begins again. As an example in figure 7.3 a window with length $N = 351$ and the step size $M = 3$ was used to analyze the Arabidopsis Thaliana Chromosome IV from position 167000 to 177000. As one can see long exons are clearly indicated while it is more difficult to identify short exons.

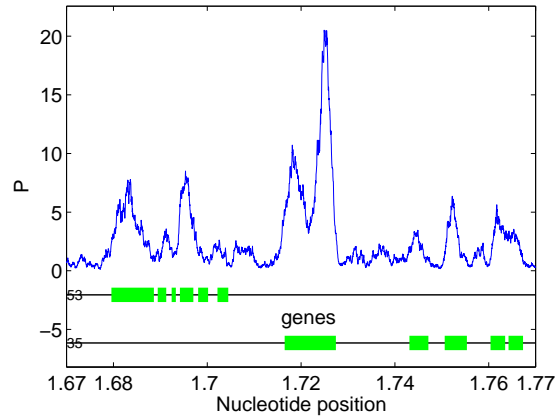


Figure 7.3: Fourier prediction

7.3 Detrended Fluctuation Analysis

The application of the Detrended Fluctuation Analysis (DFA) on genetic problems was introduced by Gao *et al* [13] in 2004. Following a short overview of that publication is given.

First of all, a mapping of nucleotides to numbers is needed. Azbel [3] suggested to code the bases with strong hydrogen bonds with the same symbol and the bases with weak hydrogen bonds with an other one. The mapping used here is A or T $\rightarrow u(i) = -1$ and G or C $\rightarrow u(i) = 1$.

Then a random walk (also called DNA walk) is done . A random walk is simply done by integrating over the sequence $u(i)$, resulting in $y(n)$.

$$y(n) = \sum_{i=1}^n u(i), \quad n = 1, 2, 3 \dots \quad (7.4)$$

Figure 7.4 shows a DNA walk over the chromosome IV of Arabidopsis Thaliana from the position 167000 to 177000.

Now a DFA can be applied to this DNA walk $y(n)$. The DFA works as follows. A window over the DNA walk with the length N is divided into $a = \lfloor \frac{N}{l} \rfloor$ non overlapping segments of the length l . The curve in this segments is then approximated by straight lines $y_a(n)$ with the least squared error. As an example in figure 7.5 a window is shown with the length $N = 512$ and a segment length l of 64. Then the fluctuation $F(l)$ is computed by taking the root-mean-squared error of the straight lines $y_a(n)$ compared to the DNA walk $y(n)$.

$$F(l) = \sqrt{\frac{1}{N} \sum_{n=1}^N (y(n) - y_a(n))^2} \propto l^H \quad (7.5)$$

H is the so called Hurst parameter. By computing F for different segment lengths l ,

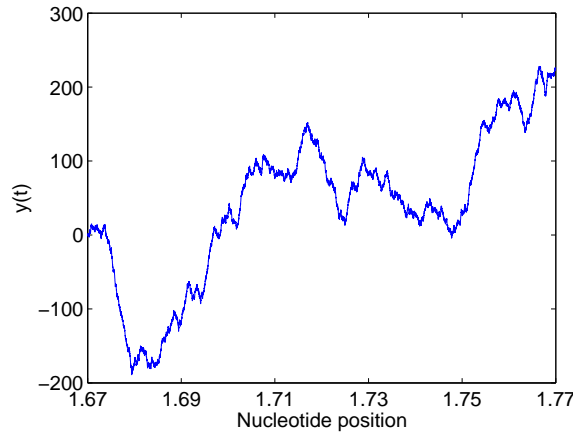


Figure 7.4: DNA walk

H can be estimated as the slope of the best fitting straight line in a log–log plot. When $H > 0.5$, the DNA walk contains long–range correlation, which are very common in non coding sequences. When $H = 0.5$, the DNA walk is completely at random.

The exponential conduct of $F(l)$ is called the fractal scaling law. However, this law does not hold if there is a periodicity in sequence. As seen before, coding regions often have a periodicity of 3. This circumstance can now be used as an indicator for coding regions in the following way:

1. $F(l)$ is calculated for different $l > 3$
2. Now the best fitting straight line $\log \hat{F} = H \log l + b$ needs to be found.
3. Then the period–3 fractal deviation is found by $PFD = |\log \hat{F}(3) - \log F(3)|^2$

Figure 7.6 shows the difference between the coding and noncoding region. The used window length is $N = 512$ and l is stepped from 2^2 to 2^6 .

Since the deviation in a coding region is only measurable when the reading frame is in phase with the start of a coding region, three reading frames, one for each phase are needed. Figure 7.7 shows the values of this three reading frames for the position 167000 to 177000 of the chromosome IV of Arabidopsis Thaliana. It can be seen, that long exons are indicated with a peak in one of the reading frames.

To decide which phase indicates the coding region best, simply the maximum is taken. Furthermore to reduce the distortion the average of M values is taken, which leads to the following codon index:

$$\text{MAXFD} = \frac{1}{\lfloor M/3 \rfloor} \sum_{m=1}^{\lfloor M/3 \rfloor} \max(\text{PFD1}(m), \text{PFD2}(m), \text{PFD3}(m)) \quad (7.6)$$

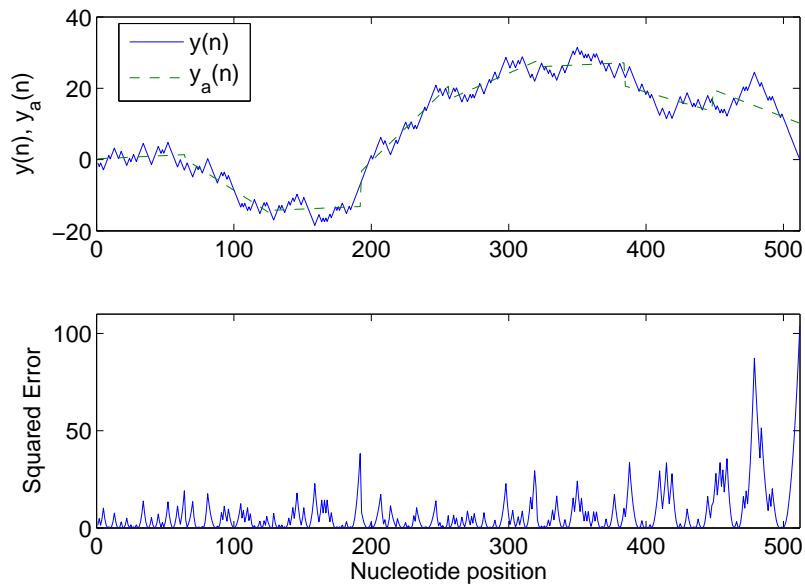


Figure 7.5: DFA on a window with length $N = 512$ and segment length $l = 64$

MAXFD is shown in figure 7.8 for the same nucleotides positions as above. Again, long exons are better indicated than short ones. There is also a problem of selectivity since very small introns and exons are not clearly visible.

Since the Detrended Fluctuation Analysis is a very time consuming operation, a implementation written in C by [19] has been used for testing purposes. This implementation is very fast and provides a Matlab interface.

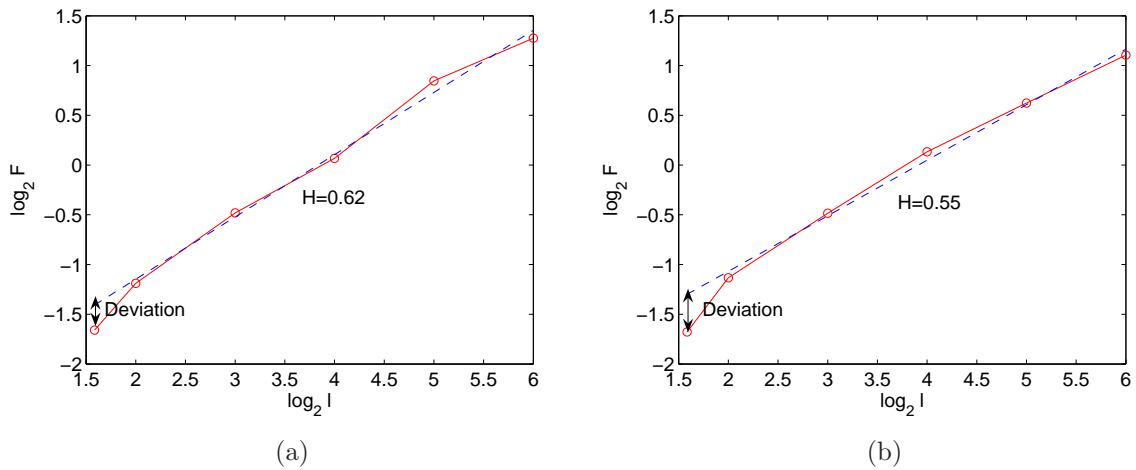


Figure 7.6: Detrended Fluctuation Analysis of a DNA region a) noncoding , b) coding

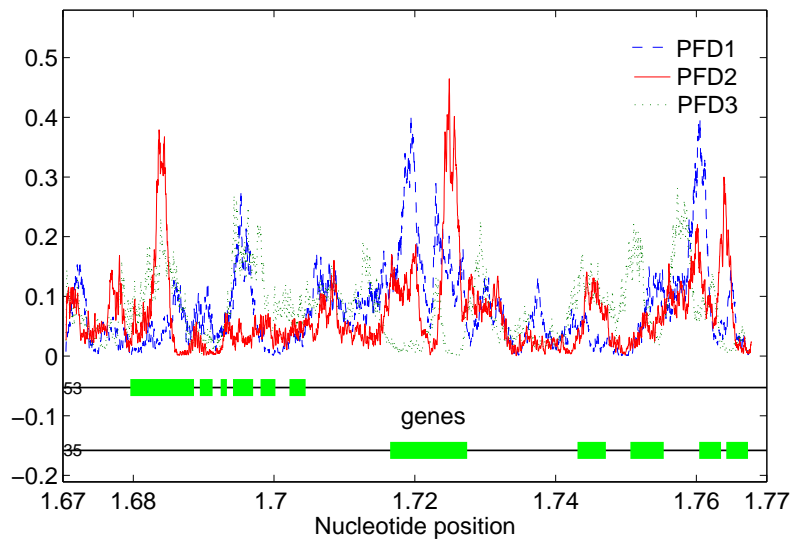


Figure 7.7: Period-3 fractals deviations of the three phases. The curves have been filtered with a fifth order moving average filter. The window size N is 256 and the scaling range l is $[2^2 \ 2^5]$.

7.4 The Z-Curve

The Z-Curve was introduced by Chun-Ting Zhang and Ju Wang [9] in 2000. It has been proved that the Z-Curve is a very powerful tool to identify coding regions. The Z-Curve represents a DNA sequence of the length N in a three-dimensional space. Furthermore, from a given Z-Curve the DNA sequence can be reconstructed easily and uniquely. The Z-Curve is constructed as follow:

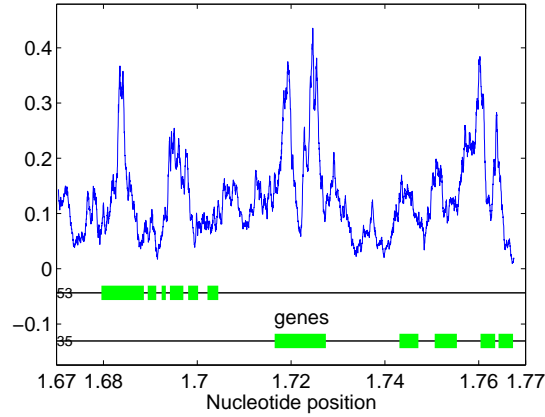


Figure 7.8: MAXFD codon index with a window length $N = 256$ and a average factor $M = 9$.

$$Z_n = \begin{cases} x_n &= (A_n + G_n) - (C_n + T_n) \\ y_n &= (A_n + C_n) - (G_n + T_n) \\ z_n &= (A_n + T_n) - (C_n + G_n) \end{cases} \quad n = 0, 1, 2, \dots, N \quad (7.7)$$

X_n denotes the number of bases in the DNA sequence of the length n . Interestingly, the components of Z represent the biological behavior of the DNA. The component x_n represents the distribution of bases of the types purine/pyrimidine, the component y_n represents the distribution of bases of the types amino/keto and finally the component z_n represents the distribution of bases of the weak and strong H-bond.

As seen above, the periodicity of three in the coding region is a strong indicator. So, the Z-Curve takes advantage of this property too by splitting the Z-Curve into three phase curves $Z_n(1)$ with the nucleotide positions $1, 4, 7, \dots$, $Z_n(2)$ with the nucleotide positions $2, 5, 8, \dots$ and $Z_n(3)$ with the nucleotide positions $3, 6, 9, \dots$. To simplify later calculations the components are approximated by straight lines

$$\begin{aligned} Z_n(1) &= \begin{cases} x_n(1) &\approx k_x(1) \cdot n \\ y_n(1) &\approx k_y(1) \cdot n \\ z_n(1) &\approx k_z(1) \cdot n \end{cases} \\ Z_n(2) &= \begin{cases} x_n(2) &\approx k_x(2) \cdot n \\ y_n(2) &\approx k_y(2) \cdot n \\ z_n(2) &\approx k_z(2) \cdot n \end{cases} \\ Z_n(3) &= \begin{cases} x_n(3) &\approx k_x(3) \cdot n \\ y_n(3) &\approx k_y(3) \cdot n \\ z_n(3) &\approx k_z(3) \cdot n \end{cases} \end{aligned} \quad (7.8)$$

with

$$\begin{aligned}
 k_x(1) &= 3 \cdot x_{N/3}(1)/N \\
 k_y(1) &= 3 \cdot y_{N/3}(1)/N \\
 k_z(1) &= 3 \cdot z_{N/3}(1)/N \\
 k_x(2) &= 3 \cdot x_{N/3}(2)/N \\
 k_y(2) &= 3 \cdot y_{N/3}(2)/N \\
 k_z(2) &= 3 \cdot z_{N/3}(2)/N \\
 k_x(3) &= 3 \cdot x_{N/3}(3)/N \\
 k_y(3) &= 3 \cdot y_{N/3}(3)/N \\
 k_z(3) &= 3 \cdot z_{N/3}(3)/N
 \end{aligned} \tag{7.9}$$

Now every DNA sequence can be represented as a point in the following 10-dimensional space:

$$\begin{aligned}
 u_1 &= k_x(1) \\
 u_2 &= k_y(1) \\
 u_3 &= k_z(1) \\
 u_4 &= k_x(2) \\
 u_5 &= k_y(2) \\
 u_6 &= k_z(2) \\
 u_7 &= k_x(3) \\
 u_8 &= k_y(3) \\
 u_9 &= k_z(3) \\
 u_{10} &= \bar{a}^2 + \bar{c}^2 + \bar{g}^2 + \bar{t}^2
 \end{aligned} \tag{7.10}$$

where \bar{a} , \bar{c} , \bar{g} and \bar{t} are the average occurrence frequencies of the different bases.

In [9] is a Fisher algorithm presented, which computes the Fisher coefficients \mathbf{c} from a trainings set for the following decision function:

$$F(\mathbf{u}) = \mathbf{u} \bullet \mathbf{c} - c_0 = \begin{cases} > 0 & \text{for coding regions} \\ < 0 & \text{for noncoding regions} \end{cases} \tag{7.11}$$

The threshold c_0 is found at the point where the true positive rate is the same as the true negative rate. The YZ-Score can then be computed as follows:

$$\text{YZ} = \frac{F(\mathbf{u}) - F_{min}^-}{F_{max}^+ - F_{min}^-}, \text{YZ} \in [0, 1] \tag{7.12}$$

with $F_{min}^- = -0.3$ and $F_{max}^+ = 0.3$. The decision if a region is coding or noncoding

becomes simply $YZ > 0.5$ / $YZ < 0.5$.

Since the Z-Curve analysis in [9] is used for open reading frames which is not suitable for later use with a SVM machine, a sliding window as by the fourier analysis is taken to analyze a DNA sequence. Interestingly, there is no dependence on the window phase. This leads to a simple window with the length N , which is shifted one by one nucleotide downstream. Figure 7.9 shows the YZ-Score of the chromosome IV of Arabidopsis Thaliana from position 167000 to 177000. By taking 0.5 as decision value, exons are very good indicated.

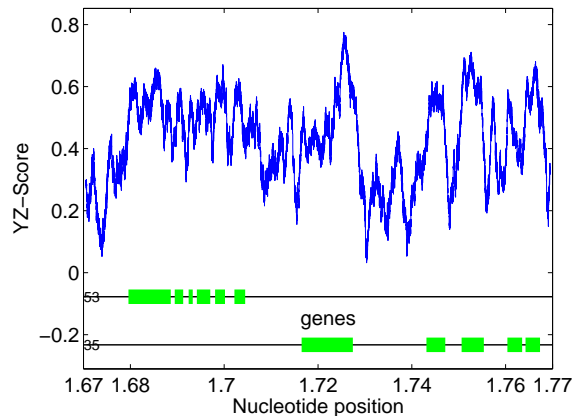


Figure 7.9: YZ-Score for a sequence of 10000 nucleotides. The used window length is $N = 102$.

7.5 Hexacount Analysis

The Hexacount analysis for identification of coding regions was introduced by JM. Claverie and L. Bougueleret in 1986 [15]. The Hexacount analysis is based on the idea, that probabilities of tuples are different in coding alternatively non coding regions. A k -tuple contains k nucleotides. For example the sequence “ATGCGATCGATT” can be divided into four 3-tuples “ATG”, “CGA”, “TCG” and “ATT”. Since a DNA is made of four different bases, the number of possible k -tuples is 4^k . Now the frequency of occurrence of every single k -tuple is counted in a non coding region and in a coding region respectively. For this thesis this has been done for $k = 5$. The two regions were made with exons and introns from the chromosome I of Arabidopsis Thaliana, which were joined together to have two sequences, one with only coding nucleotides and another one with only non coding nucleotides.

With the frequencies of occurrence of the different tuples in these two sequences, the probabilities $P_{coding}(k\text{-tuple})$ and $P_{noncoding}(k\text{-tuple})$ can be estimated.

To analyze a DNA sequence, the codon index with a discriminator threshold of 0.5

$$\frac{P_{coding}(k\text{-tuple})}{P_{coding}(k\text{-tuple}) + P_{noncoding}(k\text{-tuple})} \quad (7.13)$$

is evaluated. This is done by shifting a window with the length $N = k = 5$ one by one nucleotide downstream. Since this approach leads to a very noisy codon index, a simple moving average filter with the length $M = 31$ is applied. In figure 7.10 the sequence 167000 to 177000 of the chromosome IV is analyzed. By taking 0.5 as threshold, some of the exons are clearly indicated.

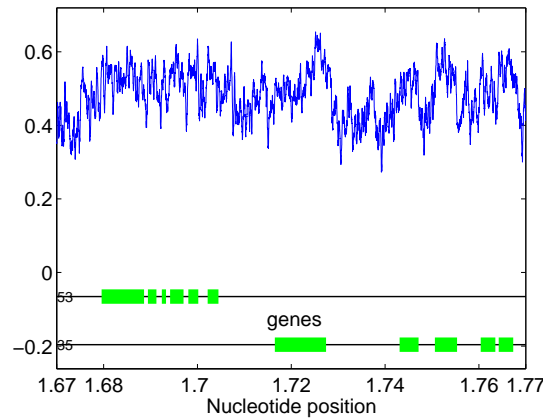


Figure 7.10: Hexacount analysis with 5-tuples from the positions 167000 to 177000 of the chromosome IV. The data has been filtered with a moving average filter with the length $M = 31$.

7.6 Identifying coding regions with help of a SVM

Since the accuracy of a SVM is highly dependent on the provided feature vectors, it makes sense to look for good features. To form a useful feature vector, as much as possible independent features are needed. This leads to the idea of using the different coding region identifying methods as features.

In this implementation the four methods, as discussed before, have been used, namely Z-Curve, Hexacount, Fourier and DFA. This leads to a four-dimensional feature vector. However, since the Z-Curve analysis uses a 10-dimensional vector for its decision function (see 7.4), this vector can directly be used instead of only the YZ-Score. Doing so, the feature vector extends to a 13-dimensional vector:

$$[\text{Fourier} \quad \text{DFA} \quad \text{Hexacount} \quad \underbrace{u_1 \quad u_2 \quad u_3 \quad u_4 \quad u_5 \quad u_6 \quad u_7 \quad u_8 \quad u_9 \quad u_{10}}_{\text{Z-Curve}}]$$

Since the SVM does improve but not degrade its accuracy with more features, it is useful to use as many features as possible. It can easily be seen, that the accuracy does

not degrade, even if the additional feature is only noise, by the following example. Bicycles and cars could hypothetically be separated by using their weight as an feature (although everyone would take the number of wheels). By taking now the color as an additional feature, the accuracy does not get worse, since a bicycle is still lighter than a car. In a two-dimensional graph, they could still be separated by a line.

Chapter 8

Combination of SVM and GHMM

8.1 Introduction

To use the information from the SVM the GHMM needs to be changed. The SVM on the other hand does not need a change.

In the next section the changes in the GHMM are documented. There are lots of possibilities to engage in the decision process of the GHMM. The changes which are implemented in the GHMM changes the emission and the signal probabilities.

8.2 Changes in the GHMM

Table 9.5 shows that the SVM makes less mistakes with the not genic prediction. therefore this information is used in the GHMM.

The implementation gives a penalty to the calculated emission score if the SVM does not say that there is a gene. This is made according to the following equation

$$es_{rated}(i) = es_{rated}(i - 1) + (es_{unrated}(i) - es_{unrated}(i - 1)) * penalty$$

The second change is a change of the signal probability. A look at the results of the SVM prediction shows a dependency on the signals. A closer look shows that the distance is a gauss distribution. This distribution has a variance of 20. An additional Penalty has been added to a signal if there is no change from coding to non coding, or vice versa, in the SVM prediction within 40 nucleotides. The following code example should give a better understanding.

```
1 for i = 2:1:length(svmPrediction)
2     if svmPredictio.value(i) ~= svmPredictio.value(i)
3         estimationArray = [estimationArray svmPrediction.index(i)];
4     end
5 end
6
7 for i=1:1:length(signalQueue)
8     dist = min(abs(estimationArray - signalQueue.index(i)))
9     if dist > 40
10        signalQueue.score(i) = signalQueue.score(i) + log10(PENALTY);
11    end
12 end
```

Listing 8.1: change of the signal probability

The penalties can not be set according to information of the genetic data. It has to be set according to the two dimensional grid search already used for the SVM. The parameters are documented in section 9.4.2.

Chapter 9

Results

9.1 Translation Initiation Sites (TIS)

9.1.1 Validating of the implementation

The implementation of the edit kernel has been tested with the same data as used in the publication [11]. This data set consists of 3312 Translation Initiation Sites of different mRNAs¹ prepared by Pedersen [Personal communication]. Downstream of the TIS there are always 150 nucleotides, while upstream of the TIS there are a minimum of 30 up to a maximum of 150 nucleotides. Since all of this TISs are real, 3988 pseudo TIS out of the intergenic region of *Aridobsis Thaliana* with 150 nucleotides downstream and 150 nucleotides upstream of the pseudo TIS have been generated. This leads to a data set of the size 7300.

In [11] an accuracy as shown in table 9.1 was obtained. By doing a six fold cross-validation with the same parameters ($\gamma_1 = 0.00195$, $\gamma_2 = 0.00781$, $C = 7$, deletion and insertion cost nucleotide/amino acid 0.35/7.), almost the same numeric results were obtained, showing the correctness of the kernel implementation.

	Li et al	own implementation
Accuracy	99.64%	>97%
Specificity	99.73%	>97%
Sensitivity	99.82%	>97%

Table 9.1: Validation of the implemented edit kernel by comparing with the original publication

The small difference can be explained by different set sizes and different DNAs for the pseudo TISs.

9.1.2 TIS recognition in DNA

The edit kernel has been applied on TISs in DNA instead of mRNA. The data set has been build as follows. First real TIS have been extracted of the chromosome IV of *Aridobsis*

¹This is an important point, since all nucleotides downstream of the TIS are in an exon and all nucleotides upstream of the TIS are in the intergenic region.

Thaliana, resulting in 3437 real TISs with 150 nucleotides down- and upstream. Then the pseudo TISs have been built by taking “ATG” as pseudo TIS. Since it is possible that a ”ATG” is less than 150 nucleotide before or after an exon, a lot of the pseudo TISs containing coding nucleotides down- or upstream. On the other hand most of the initial exons are shorter than 150 nucleotides, which means that a lot of the downstream nucleotides after a real TIS are non coding (See figure 9.1). At the end a data set with 3437 real TISs and 7000 pseudo TISs has been built.

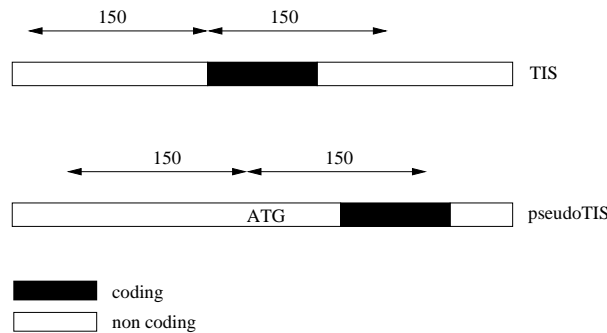


Figure 9.1: Building TIS and pseudo TISs of a DNA

On the basis of this noisy data set the results obtained are not too good. Several six-fold cross-validations with different set sizes have been performed, with the best results as follows.

Accuracy	<78%
Specificity	<83%
Sensitivity	<74%

Table 9.2: Obtained results on TIS prediction on a DNA sequence

This result indicates that the edit kernel can not be used without a modification on DNA sequences. One possibility would be to take significant shorter windows. However with too short windows the statistic behavior is lost. But this would be an approach to test in the future.

9.2 Identifying coding regions with help of a SVM

9.2.1 Training of the SVM

The DNA sequence of the chromosome I of the Arabidopsis Thaliana from position 1550000 to 1610000 has been analyzed with the four coding identification methods Hexa-Count, Z-Curve, Fourier and Detrended Fluctuation Analysis. Table 9.3 shows the used lengths of the respective windows.

	Windowlength N	Stepsize M	Average Filter
Z-Curve	102		-
DFA	256	3	9
Fourier	351	1	-
HexaCount	5	1	31

Table 9.3: Used Windowsizes

Each value of these coding identification methods corresponding to every 9–th nucleotide has been taken to form the training vectors. This has led to 6610 training vectors, 2167 coding and 4443 non coding.

To find the best parameters for the SVM, 600 coding and 600 non coding training vectors have been taken to form a parameter evaluating test set. Then, this set has been used to perform 6 fold cross–validations on it to find the best values C and γ for the RBF kernel. Table 9.4 shows the best occupied accuracy with its corresponding parameters ($C = 194$, $\gamma = 0.01314$).

Accuracy	90.1%
Sensitivity	88.6%
Specificity	91.9%
Correlation Coefficient	0.81

Table 9.4: Occupied Accuracy of the 6 fold cross validation with a RBF Kernel and the parameters $C = 194$, $\gamma = 0.01314$.

With these parameters the final SVM has been trained with all 6610 training vectors. 2279 of this training vectors resulted in Support Vectors, which is one third. This is an indication of a training set which has a very small difference between positive and negative vectors.

9.2.2 Results

Since the SVM has been trained on chromosome I, the tests have been done with chromosome IV of Arabidobis Thaliana to obviate the problem of distorting the results by using the same data for testing and training.

The whole chromosome was analyzed by all four coding identification methods, which are the basis for the later prediction with the SVM as written in section 9.2. Then this data was used to predict every single nucleotide if it is in a coding region or not. As an example in figure 9.2 the SVM prediction is shown from the position 160000 to 180000 of the chromosome IV. It can be clearly seen that especially long exons are very good recognized.

Table 9.5 shows the results over the whole chromosome IV, rated on the nucleotide level.

Accuracy	80.2%
Specificity	61.7%
Sensitivity	73.3%
Correlation Coefficient	0.534
True Positive	61.8%
False Positive	38.2%
True Negative	89.1%
False Negative	10.9%

Table 9.5: Measures over the whole chromosome IV, SVM

As one can see, the main problem is the massive amount of false positives. This is because of the disproportion of coding and non coding regions. 73% of the chromosome IV contains non coding nucleotides while only 27% nucleotides are coding.

For comparison table 9.6 shows the result of using the Z-Curve only, to identify coding regions. It can be seen, that the prediction of the SVM is slightly better as the Z-Curve, showing the potential of combining different coding identification methods with a SVM.

Accuracy	77.2%
Specificity	55.8 %
Sensitivity	58.9%
Correlation Coefficient	0.418
True Positive	58.9%
False Positive	41.1%
True Negative	83.6%
False Negative	16.4%

Table 9.6: Measures over the whole chromosome IV, only Z-Curve

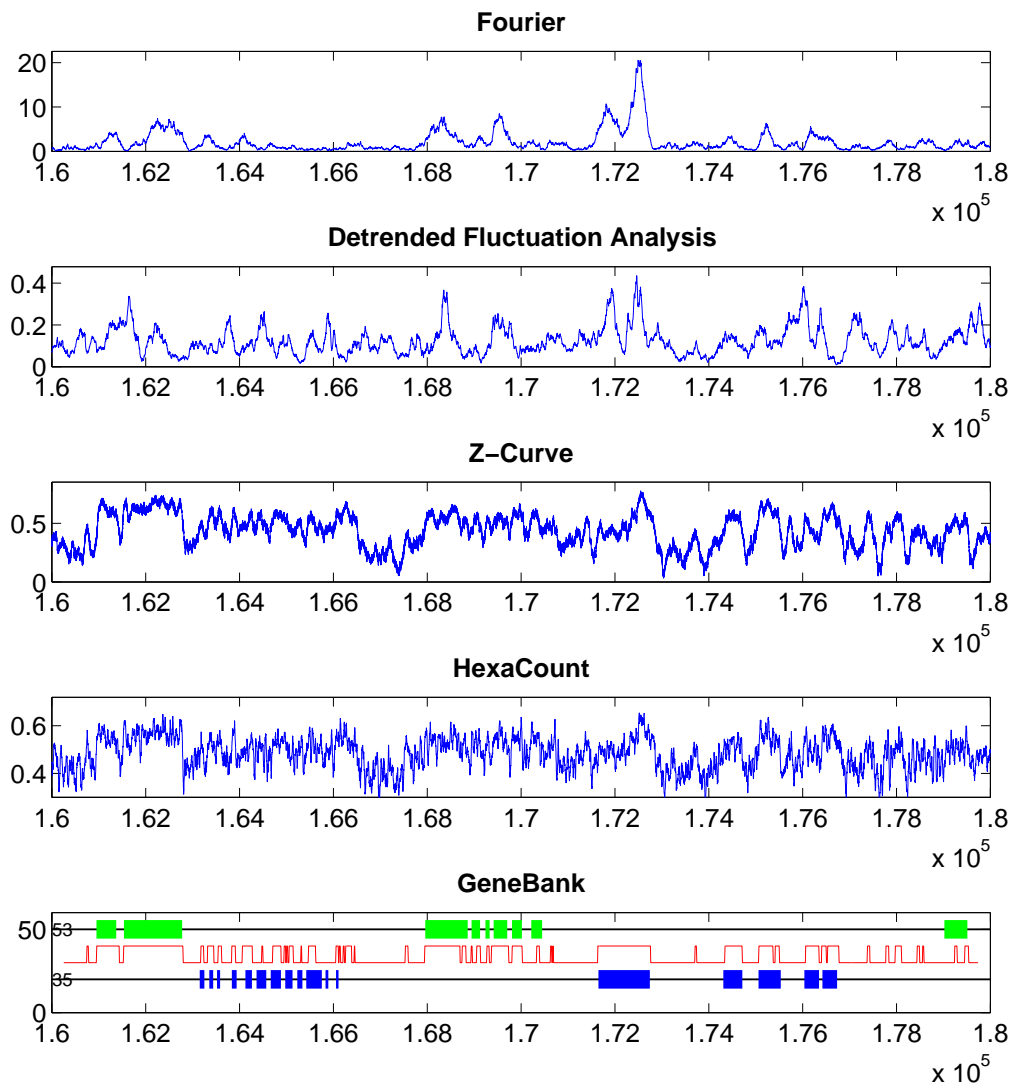


Figure 9.2: SVM prediction based on Z-Curve, HexaCount, DFA and Fourier from position 160000 to 180000 of chromosome IV. The toggling red line in the most lower graph is the SVM prediction.

9.3 GHMM

9.3.1 Training of the GHMM

The models used by the GHMM are all trained using the chromosome I from Genebank file "NC_003070".

The training files are adapted to the reverse strand from the diploma thesis [8]. On the forward strand no changes has been made.

9.3.2 Results

The GHMM trained as written above yields to the following results. In figure 9.3 and 9.4 the results of the GHMM and for comparison the results of the diploma thesis [8] are showed. The table 9.7 then shows the corresponding data. It can be seen, that with implementation of the reverse strand, the prediction is getting better. Just on the forward strand is no difference.

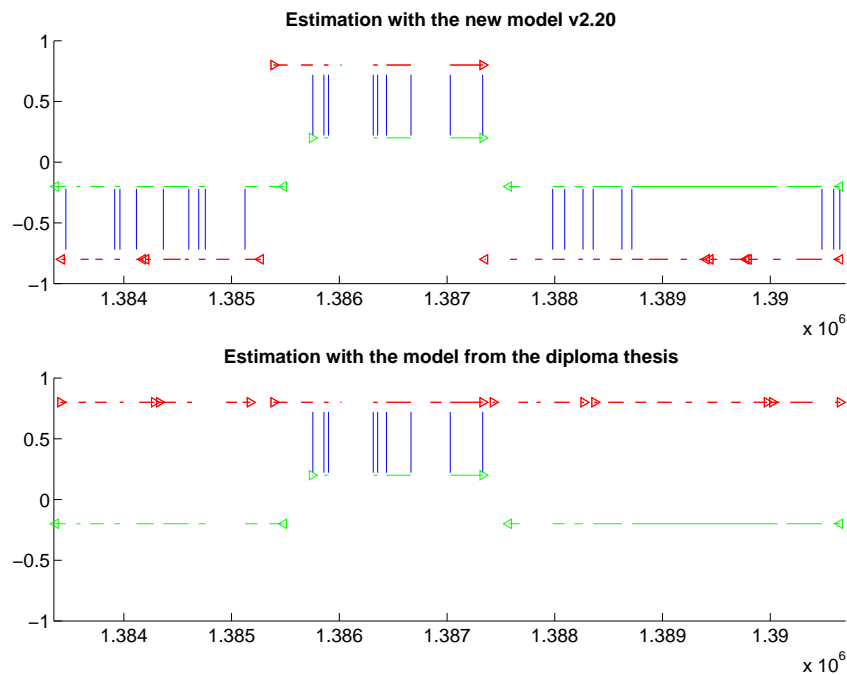


Figure 9.3: GHMM prediction from 1383350 to 1390700.

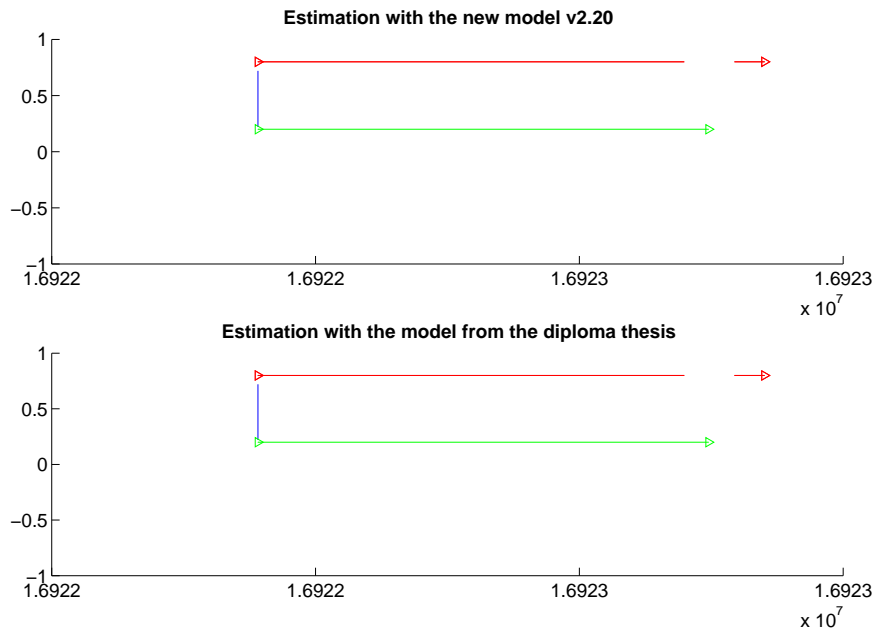


Figure 9.4: GHMM prediction from 16921500 to 16923000.

	1383350 to 1390700		16921500 to 16923000	
	GHMM	GHMM from DA	GHMM	GHMM from DA
Signal Accuracy	39 %	14 %	25 %	25 %
Signal Hits	27	9	1	1
True Positive	2798	2350	809	809
False Positive	475	788	58	58
True Negative	2927	2614	587	587
False Negative	1150	1598	46	46
Nucleotide Accuracy	77.9 %	67.5 %	93.1 %	93.1 %
Nucleotide Sensitivity	70.7 %	59.5 %	94.6 %	94.6 %
Nucleotide Specificity	85.5 %	74.9 %	93.3 %	93.3 %
Nucleotide Correlation Coefficient	0.571	0.367	0.858	0.858

Table 9.7: Results of the GHMMs

The numbers of the predictions above are to handle with care. With cutting out the sequence exactly around the genes and only one gene the model becomes additional information. To avoid this the model has to run over a longer sequence which is not aligned to an exon. This leads to the following result.

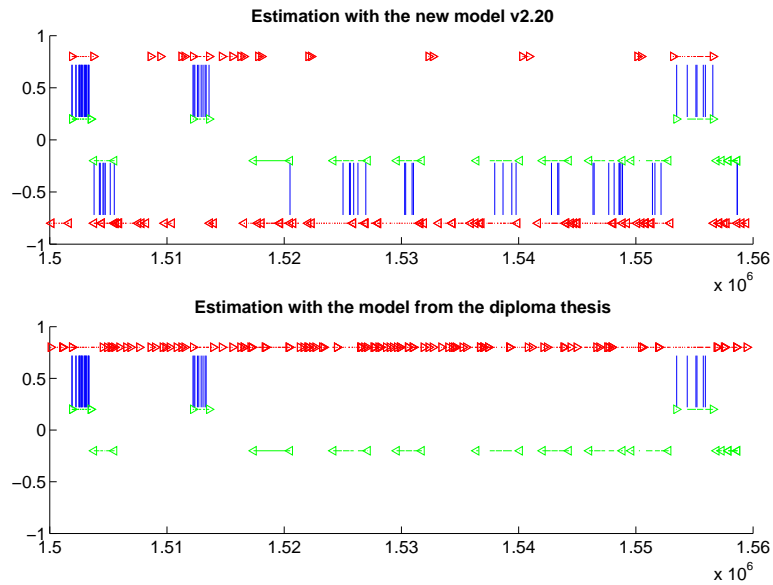


Figure 9.5: GHMM prediction from 1500000 to 1560000. The green arrows are gene from the gene bank file, the red is the prediction. Where a blue line appears, prediction and genebank file have a match.

As it can be seen as longer the sequence gets, it gets more inaccurate.

9.3.3 Speed up with signal eclipsing

Eclipsing the signals as discussed in section 4.3.6 makes the model faster. As an example see table 9.9.

It can be seen that the model from the diploma thesis is the fastest for short sequences. But it does not have to calculate the same amount of signals. The time needed by the model can be reduce by factor 2 with eclipsing. It has to be considered that with this approach mistakes can be done due to eclipsing to early. Although, the time needed is directly dependent on the length of the sequence. The time does not rise linear with the length of the sequence. The time needed grows faster than the sequence length.

	1500000 to 1560000	
	GHMM	DA
Signal Accuracy	16 %	8 %
Signal Hits	77	36
True Positive	15126	12827
False Positive	10113	9349
True Negative	28263	29027
False Negative	6498	8797
Nucleotide Accuracy	72.3 %	69.7 %
Nucleotide Sensitivity	69.9 %	59.3 %
Nucleotide Specificity	59.9 %	57.8 %
Nucleotide Correlation Coefficient	0.424	0.348

Table 9.8: Results of the GHMMs for a long sequence

Model	duration to calculate	
	30000 nucleotide	150000 nucleotides
diploma thesis	950 seconds	20 hours
with reverse without eclipsing	2304 seconds	-
with reverse with eclipsing	1273 seconds	10 hours

Table 9.9: Speedup with eclipsing

9.4 Combination of GHMM and SVM

9.4.1 Training of the SVM for GHMM

The SVM is trained exactly the same way as showed in section 9.4.1.

9.4.2 Parameter evaluation

The two parameter emission score penalty and signal score penalty are evaluated by using the following five sequences.

- from 146000 to 149000
- from 810500 to 815000
- from 15602500 to 15605500
- from 16548500 to 16550500
- from 16921500 to 16923000

The penalty evaluation is done by a 2-level grid search similar to the one used for the SVM. The two parameters are the two penalties.

The output was analyzed to give the best signal accuracy. The penalties in table 9.10 comes from this procedure.

emission score penalty 1.02
signal score penalty 0.50

Table 9.10: Penalty values

9.4.3 Results

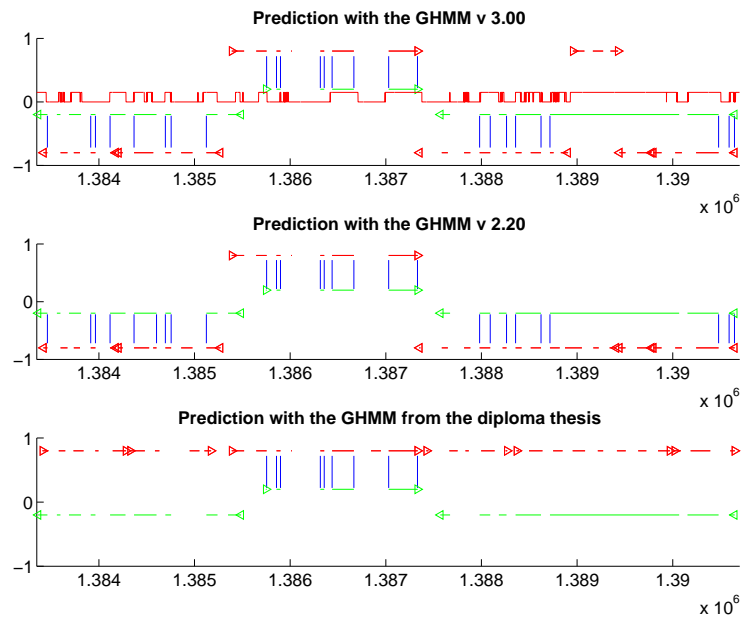


Figure 9.6: GHMM prediction with SVM support from 1383350 to 1390700. The green arrows are gene from the gene bank file, the red is the prediction. Where a blue line appears, prediction and genebank file have a match.

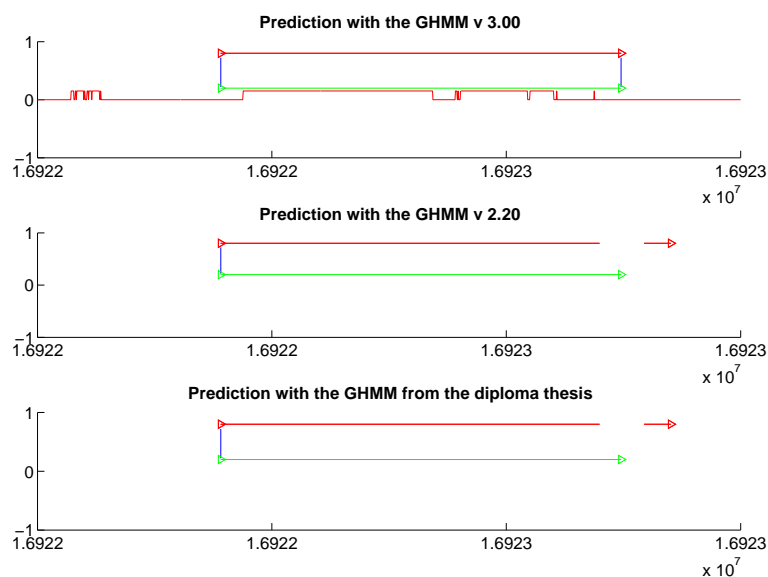


Figure 9.7: GHMM prediction with SVM support from 16921500 to 16923000. The green arrows are gene from the gene bank file, the red is the prediction. Where a blue line appears, prediction and genebank file have a match.

	1383350 to 1390700		16921500 to 16923000	
	GHMM with SVM	GHMM	GHMM with SVM	GHMM
Signal Accuracy	37 %	39 %	100 %	25 %
Signal Hits	26	27	2	1
True Positive	2819	2798	855	809
False Positive	475	475	0	58
True Negative	2927	2927	645	587
False Negative	1129	1150	0	46
Nucleotide Accuracy	78.1 %	77.9 %	100 %	93.1 %
Nucleotide Sensitivity	71.4 %	70.7 %	100 %	94.6 %
Nucleotide Specificity	85.6 %	85.5 %	100 %	93.3 %
Nucleotide Correlation Coefficient	0.576	0.571	1.000	0.858

Table 9.11: Results of the GHMM with SVM support

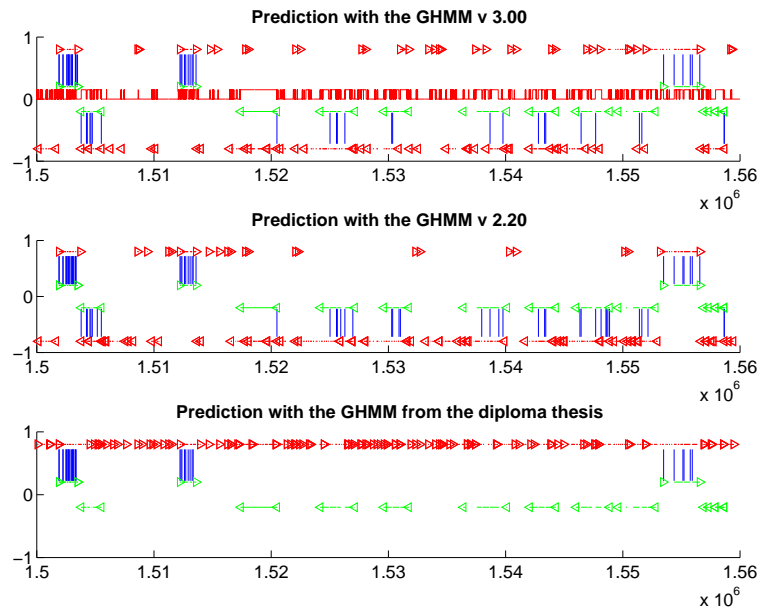


Figure 9.8: GHMM prediction with SVM support from 1500000 to 1600000. The green arrows are gene from the gene bank file, the red is the prediction. Where a blue line appears, prediction and genebank file have a match.

	1500000 to 1650000			
	GHMM with SVM	GHMM	diploma thesis	SVM alone
Signal Accuracy	21 %	17 %	14 %	-
Signal Hits	170	181	138	-
True Positive	31663	31118	28558	506
False Positive	22686	29338	26890	99596
True Negative	86380	79910	82358	48814
False Negative	9089	9634	12194	574
Nucleotide Accuracy	78.7 %	74.0 %	73.9 %	66.9 %
Nucleotide Sensitivity	77.7 %	76.36 %	70.1 %	46.8 %
Nucleotide Specificity	58.1 %	51.5 %	51.5 %	1.0 %
Nucleotide Correlation Coefficient	0.525	0.449	0.449	0.025

Table 9.12: Results of the GHMM with SVM support over a long sequence

Chapter 10

Conclusion

10.1 Review

Identification of genes in DNA is a extremely demanding endeavour. These days, the known differences between coding and non coding regions are quite weak and thus not allowing to get outstanding results.

However, it has been shown in this thesis, that the use of different coding region identification methods in combination with a SVM leads to results which can outperform the stand-alone use of these individual methods.

Since the output of the SVM only yields identification of coding regions without giving useful information on which strand and where exactly the gene starts and stops, its prediction has been used to bias the emissions and signal scores of a GHMM. The implemented GHMM uses four different signal sensors and six different content sensors on each strand.

The obtained results based on the combination SVM/GHMM are comparable with other gene finder implementations available these days. Nevertheless a comparison is difficult, since the implementation has not yet been tested with widely used DNAs like the yeast genome.

However, the main problem of this thesis, as well as of other gene prediction programs. It is the high rate of false positive predictions. Without lowering this rate, it is very difficult to identify correct genes in practice. There still remains the question if it is possible to obtain satisfying results by using statistical methods, since the amount of non coding regions is a lot higher than that of coding regions and therefore, the probability of finding non coding regions with the same statistical properties as the coding regions is quite high.

10.2 Outlook

Several possibilities are supposable to increase the prediction quality. For example using additional coding identification methods as features for the SVM, or rarefy the GHMM by using more states. To lower the false positive rate it is thinkable to translate every predicted gene to mRNA and applying the TIS recognition using the edit kernel on that mRNA.

Appendix A

Measures of prediction accuracy

The measures of the prediction accuracy is an important point to verify a new method and make it possible to compare one method to an other. Nevertheless, often quite different measures are used. To reduce this problem, Burset and Guigo [18] recommend measures for using with gene prediction methods.

A.1 Mathematical basics

All of the following mathematical formulas are based on the four boolean quantities

		Reality	
		positive	negative
Prediction	positive	TP	FP
	negative	FN	TN

Figure A.1: Measures of prediction accuracy

- True Positive (TP): Number of values which are positive in prediction as in reality
- False Positive (FP): Number of values which are positive in prediction but not in reality
- True Negative (TN): Number of values which are negative in prediction as in reality
- False Negative (FN): Number of values which are negative in prediction but not in reality

quantifying the prediction dependent on the reality.

With this quantities the most common measure value, the accuracy, which is the number of all correct predicted values in relation to all predictions, is

$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \quad (\text{A.1})$$

As an other measure value, the sensitivity is

$$S_n = \frac{TP}{TP + FN}, \quad (\text{A.2})$$

which is the proportion of how many positive values are the same in reality and prediction. With $S_n = 1$ the prediction of true values corresponds perfectly to the reality.

The specificity

$$S_p = \frac{TN}{TN + FP} \quad (\text{A.3})$$

denotes the proportion of how many negative values are the same in reality and prediction. However, since in gene prediction TN are much more common, due the amount of intergenic nucleotides, than FP, the specificity gets a noninformative high value. Hence, the specificity has been redefined as

$$S_p = \frac{TP}{TP + FP}. \quad (\text{A.4})$$

The sensitivity or specificity for itself does not give a good global accuracy, since it is possible to have very high specificity and at the same time very low sensitivity. Hence, it makes sense to define one value which gives us a measure for overall accuracy. This is done with the Correlation Coefficient

$$CC = \frac{TP \cdot TN - FN \cdot FP}{\sqrt{(TP + FN)(TN + FP)(TP + FP)(TN + FN)}}. \quad (\text{A.5})$$

CC is one when reality and prediction match perfect and zero when the prediction is completely random.

Now, there exist three different measure methods.

A.2 Nucleotide Level

This measure has been one of the most popular for coding identification methods. Every single nucleotide is compared with its prediction, so the measure is fair for methods which can only identify coding regions instead of donor and acceptor splices sites.

A.3 Exon Level

This measure compares the prediction of the donor and acceptor splice sites. However, often a match is rated perfectly even if the prediction is in some range around the real

splice site instead. In this thesis only the congruent coverage of prediction and reality is rated as a perfect match.

A.4 Protein Level

For this measure all exons of the prediction are translated to an mRNA and compared with the real mRNA. So, this measure is almost the same as on the exon level and is not used in this thesis.

Appendix B

Notation

\mathbf{x}	Vector
\mathbf{x}^i	i-th vector of a set of vectors
\mathbf{x}_i	i-th element of a vector
$\langle \mathbf{x}^i, \mathbf{x}^j \rangle$	Inner Product of two vectors
$\ \mathbf{x}\ $	Euclidean length of the vector \mathbf{x}
$\lfloor x \rfloor$	largest integer that is not greater than x
\mathbf{A}'	Transpose of the matrix \mathbf{A}

Appendix C

Abbreviations

ASS	Acceptor Splice Site
DFA	Detrended Fluctuation Analysis
DNA	Deoxyribonucleic Acid
DSS	Donor Splice Site
ECOC	Error Correction Output Code
GHMM	Generalized Hidden Markov Model
HMM	Hidden Markov Model
IMM	Interpolated Markov Model
MAXFD	Maximum Fluctation Deviation
MDM	Maximum Dependency Model
MM	Markov Model
mRNA	Messenger Ribonucleic Acid
PAM	Point Accepted Mutation
pre-mRNA	pre messenger Ribonucleic Acid
RBF	Radial Basis Function
SVM	Support Vector Machine
TIS	Translation Initiation Site
TSS	Translation Stop Site
WAM	Weighted Array Model

Appendix D

Matlab files

D.1 Gene prediction with a SVM

main.m	
Parameter	none, uses a console as user interface
Return Value	Training Modus: stores the files <code>scale.mat</code> , <code>threshold.mat</code> and <code>machine.mat</code> on the harddisk Prediction Modus: stores the predicted DNA sequence <code>prediction.mat</code> on the harddisk
Description	This is the main file for the whole SVM prediction code. It is used to train the SVM as well as to predict gen sequences

genfft.m	
Parameter	<code>seq</code> : the DNA sequence as a string <code>windowSize</code> : size of the analysis window
Return Value	<code>fourier</code> : struct with index and value of the fourier transformation
Description	Does a fourier analysis over the given sequence

analyzeDFA.m	
Parameter	<code>seq</code> : the DNA sequence as a string <code>windowSize</code> : size of the analysis window <code>kWindowSize</code> : Size of the moving average filter
Return Value	<code>PFD</code> : struct with index and value of the DFA transformation
Description	Does a Detrended Fluctuation Analysis over the given sequence

analyzeZcurve.m	
Parameter	seq : the DNA sequence as a string windowSize : size of the analysis window kWindowSize : step size of the analysis window
Return Value	Z :struct with index and YZ-Score of the Z-Curve analysis
Description	Does a Z-Curve analysis over the given sequence

hexacount.m	
Parameter	seq : the DNA sequence as a string
Return Value	hexa :struct with index and value of the Hexacount analysis
Description	Does a Hexacount analysis over the given sequence

plotAnalysedGen.m	
Parameter	gene : gene structure, given from the function <code>get_gene</code> startnuc : startposition of the DNA sequence stopnuc : stopposition of the DNA sequence pred : prediction results of the SVM DFA : results of the DF analysis Fourier : results of the fourier analysis Zcurve : results of the Z-Curve analysis hexa : results of the Hexacount analysis
Return Value	none
Description	Plots all results together with the gene positions out of the gene bank file

accuracy.m	
Parameter	prediction : Prediction of the SVM
Return Value	none
Description	Calculates the accuracy, sensitivity, specificity and the correlation factor. The prediction of the SVM is compared with the chromosome IV.

D.2 Own SVM implementation

SVM/svm.m	
Parameter	kern : Kernel type, 'linear', 'rbf', 'poly' norm : 1norm 1, 2norm 2 C : penalty for missclassified vectors parameter : optional parameters for kernels
Return Value	machine : struct with informations about the SVM
Description	initializes a new SVM, which is used for the training

SVM/svmtrain.m	
Parameter	machine: The machine which is initialized with svm.m X: learning vectors Y: labels, corresponding to the learning vectors
Return Value	machine: struct with the learned SVM parameters
Description	Trains the SVM machine with a training set

SVM/svmtest.m	
Parameter	machine: the trained machine X: test vectors Y: labels, corresponding to the test vectors
Return Value	machine: accuracy of the machine
Description	Tests a set of vectors and computes the accuracy, sensitivity, specificity and the correlation factor

SVM/svmcrossval.m	
Parameter	machine: the trained machine X: test vectors Y: labels, corresponding to the test vectors num: number of crossvalidations to perform
Return Value	machine: accuracy of the machine
Description	Tests a set of vectors and computes the accuracy, sensitivity, specificity and the correlation factor based on a n-fold cross-validation

D.3 GHMM prediction

estimatepenalty/main_v_3_00.m	
Parameter	none, uses a console as user interface
Return Value	stores the result of the prediction in a mat file.
Description	The user is being asked for a section to predict. After that the predictions runs with the sequence defined. The prediction is made with the to models v3.00 and v2.20. After that the prediction is displayed.

estimatepenalty/ghmm_v_3_00.m	
Parameter	seq: the DNA sequence as a string.
Return Value	result: The prediction of the sequence. queue: The modified signal queue through out the model alpha: The precalculated emission score array.
Description	This model makes the prediction with the support of the SVM. After that the results are given back.

estimatepenalty/ghmm_v_2_20.m	
Parameter	seq : the DNA sequence as a string.
Return Value	result : The prediction of the sequence. queue : The modified signal queue through out the model alpha : The precalculated emission score array.
Description	This model makes the prediction without the support of the SVM. After that the results are given back.

All other GHMM versions on CD are made throughout the process of implementing different features. Version 2.20 and version 3.00 are the most common and should therefore be used.

estimatepenalty/changeSignalProb.m	
Parameter	queue : The signal queue. signalPenalty : Penalty is loaded from the file penalty.mat.
Return Value	queue : The changed signal queue
Description	This function changes the signal queue with support of the SVM according to section 8.2. The penalty comes from the file penalty.mat

estimatepenalty/change_emission_prob	
Parameter	alpha : The emission score emissionscorePenalty : Penalty is loaded from the file penalty.mat.
Return Value	alpha_new : The changed emission score
Description	This function changes the emission score with support of the SVM according to section 8.2. The penalty comes from the file penalty.mat

D.4 GHMM penalty estimation

estimatepenalty/main_estimatePenalty.m	
Parameter	none, uses a console as user interface
Return Value	stores the genauigkeit.mat and the results of the prediction with different penalties.
Description	Eight sequences are predicted with the three models , diploma thesis, version 2.20 without SVM support and version 3.00 with SVM support. The results are then shown to the user and he is able to make the decision of the best penalty set.

estimatepenalty/calresult_good_medium_bad_iteration.m	
Parameter	<p>result_new: The result of the GHMM v 3.00</p> <p>svmPrediction: The prediction of the SVM</p> <p>result_1: The result of the GHMM v 2.20</p> <p>gene: gene structure, given from the function <code>get_gene</code></p> <p>interval: Consists of all intervals looked at.</p> <p>intervalIndex: Points to the position in the interval where the actual prediction has started</p> <p>result_old: The result from the GHMM of the diploma thesis</p>
Return Value	<p>signalHits: Amount of signals which are the same</p> <p>SignalAccuracy: The signal accuracy, multiplied with 100</p> <p>Accuracy: Nucleotide accuracy</p> <p>Sensitivity: Nucleotide sensitivity</p> <p>Specificity: Nucleotide specificity</p> <p>CC: Correlation coefficient</p> <p>TP: The amount of true positive nucleotides</p> <p>FP: The amount of false positive nucleotides</p> <p>TN: The amount of true negative nucleotides</p> <p>FN: The amount of false negative nucleotides</p>
Description	According to appendix A the return values are calculated.

estimatepenalty/printAccuracy	
Parameter	none, uses a console as user interface to select the penalty
Return Value	none. First it displays a lot of figures with information calculated with the 2-dimensional grid search. After the entering of specific penalties the results of the prediction at this penalties are displayed.
Description	It displays figures which help to determine the best penalties. After a selection of the best penalties the results at this penalties are displayed.

estimatepenalty/plotresult_good_medium_bad.m	
Parameter	<p>result_new: The result of the GHMM v 3.00</p> <p>svmPrediction: The prediction of the SVM</p> <p>result_1: The result of the GHMM v 2.20</p> <p>gene: gene structure, given from the function get_gene</p> <p>interval: The interval in which the prediction has been made as a matrix [from , to].</p> <p>result_old: The result from the GHMM of the diploma thesis</p> <p>figureNumber: The number of the figure in which it is being displayed.</p>
Return Value	none, it displays a figure with the information of the prediction
Description	Plots the prediction with the true genes and the prediction of the three models.

Appendix E

Substitution tables

	Ala A	Arg R	Asn N	Asp D	Cys C	Gln Q	Glu E	Gly G	His H	Ile I	
Ala	A	0	1.4205	1.0007	0.89492	1.6503	1.1855	0.86352	0.36028	1.4727	1.2241
Arg	R	1.4205	0	1.4473	1.67	2.3207	1.1877	1.5876	1.6733	1.1813	1.9179
Asn	N	1.0007	1.4473	0	0.8822	2.3766	1.2934	1.0004	0.94791	1.179	1.8892
Asp	D	0.8949	1.67	0.8822	0	2.6448	1.0225	0.47423	0.81884	1.3164	1.9474
Cys	C	1.6503	2.3207	2.3766	2.6448	0	2.7982	2.6545	2.0412	2.4113	2.1128
Gln	Q	1.1855	1.1877	1.2934	1.0225	2.7982	0	0.79777	1.3544	0.90344	1.9811
Glu	E	0.8635	1.5876	1.0004	0.47423	2.6545	0.79777	0	0.88969	1.3025	1.8471
Gly	G	0.3602	1.6733	0.94791	0.81884	2.0412	1.3544	0.88969	0	1.6674	1.6759
His	H	1.4727	1.1813	1.179	1.3164	2.4113	0.90344	1.3025	1.6674	0	2.2002
Ile	I	1.2241	1.9179	1.8892	1.9474	2.1128	1.9811	1.8471	1.6759	2.2002	0
Leu	L	1.1072	1.7261	1.7279	1.8883	2.5484	1.4884	1.6972	1.5438	1.6996	0.56192
Lys	K	0.9617	0.31822	0.85336	0.99395	2.4087	0.94674	0.99329	1.0958	1.1999	1.5321
Met	M	1.8246	2.0232	2.3344	2.4842	3.2269	2.1959	2.3535	2.24	2.5657	1.4824
Phe	F	1.7758	2.3077	2.1302	2.5468	2.5096	2.4675	2.4837	2.0412	1.8853	1.2654
Pro	P	0.6525	1.361	1.3933	1.3969	1.9844	1.2655	1.2831	1.0067	1.4602	1.8322
Ser	S	0.5214	1.2424	0.99353	1.0041	1.3032	1.311	1.045	0.51904	1.4667	1.5364
Thr	T	0.5788	1.4354	1.1351	1.1891	1.816	1.4556	1.2162	0.83912	1.6563	1.275
Trp	W	3.0758	1.6503	3.1814	3.662	3.9265	3.3573	3.6719	3.3298	3.072	3.5179
Tyr	Y	1.9716	2.6378	2.0388	2.4793	1.6231	2.5525	2.4273	2.3734	1.7032	1.8925
Val	V	0.7873	1.7576	1.6018	1.6199	1.7176	1.6865	1.5222	1.1327	1.8823	0.40509

	Leu L	Lys K	Met M	Phe F	Pro P	Ser S	Thr T	Trp W	Tyr Y	Val V	
Ala	A	1.1072	0.96179	1.8246	1.7758	0.65252	0.52143	0.57881	3.0758	1.9716	0.78732
Arg	R	1.7261	0.31822	2.0232	2.3077	1.361	1.2424	1.4354	1.6503	2.6378	1.7576
Asn	N	1.7279	0.85336	2.3344	2.1302	1.3933	0.99353	1.1351	3.1814	2.0388	1.6018
Asp	D	1.8883	0.99395	2.4842	2.5468	1.3969	1.0041	1.1891	3.662	2.4793	1.6199
Cys	C	2.5484	2.4087	3.2269	2.5096	1.9844	1.3032	1.816	3.9265	1.6231	1.7176
Gln	Q	1.4884	0.94674	2.1959	2.4675	1.2655	1.311	1.4556	3.3573	2.5525	1.6865
Glu	E	1.6972	0.99329	2.3535	2.4837	1.2831	1.045	1.2162	3.6719	2.4273	1.5222
Gly	G	1.5438	1.0958	2.24	2.0412	1.0067	0.51904	0.83912	3.3298	2.3734	1.1327
His	H	1.6996	1.1999	2.5657	1.8853	1.4602	1.4667	1.6563	3.072	1.7032	1.8823
Ile	I	0.56192	1.5321	1.4824	1.2654	1.8322	1.5364	1.275	3.5179	1.8925	0.40509
Leu	L	0	1.3371	0.78757	0.67283	1.4789	1.4411	1.2487	2.7949	1.4416	0.41099
Lys	K	1.3371	0	1.4516	2.226	1.1856	0.83903	0.88435	2.619	2.3822	1.3742
Met	M	0.78757	1.4516	0	1.9285	2.2958	2.0271	1.8593	3.852	2.6957	1.2864
Phe	F	0.67283	2.226	1.9285	0	2.3172	1.8448	1.8904	1.9495	0.06035	1.4886
Pro	P	1.4789	1.1856	2.2958	2.3172	0	0.81234	1.0283	3.3161	2.5915	1.3754
Ser	S	1.4411	0.83903	2.0271	1.8448	0.81234	0	0.65851	2.3704	1.9853	1.1638
Thr	T	1.2487	0.88435	1.8593	1.8904	1.0283	0.65851	0	3.1136	2.0631	0.95199
Trp	W	2.7949	2.619	3.852	1.9495	3.3161	2.3704	3.1136	0	2.1109	3.5123
Tyr	Y	1.4416	2.3822	2.6957	0.06035	2.5915	1.9853	2.0631	2.1109	0	1.9272
Val	V	0.41099	1.3742	1.2864	1.4886	1.3754	1.1638	0.95199	3.5123	1.9272	0

Table E.1: The ASCM250 Matrix

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val	
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	
Ala	A	9867	2	9	10	3	8	17	21	2	6	4	2	6	2	22	35	32	0	2	18
Arg	R	1	9913	1	0	1	10	0	0	10	3	1	19	4	1	4	6	1	8	0	1
Asn	N	4	1	9822	36	0	4	6	6	21	3	1	13	0	1	2	20	9	1	4	1
Asp	D	6	0	42	9859	0	6	53	6	4	1	0	3	0	0	1	5	3	0	0	1
Cys	C	1	1	0	0	9973	0	0	0	1	1	0	0	0	0	1	5	1	0	3	2
Gln	Q	3	9	4	5	0	9876	27	1	23	1	3	6	4	0	6	2	2	0	0	1
Glu	E	10	0	7	56	0	35	9865	4	2	3	1	4	1	0	3	4	2	0	1	2
Gly	G	21	1	12	11	1	3	7	9935	1	0	1	2	1	1	3	21	3	0	0	5
His	H	1	8	18	3	1	20	1	0	9912	0	1	1	0	2	3	1	1	1	4	1
Ile	I	2	2	3	1	2	1	2	0	0	9872	9	2	12	7	0	1	7	0	1	33
Leu	L	3	1	3	0	0	6	1	1	4	22	9947	2	45	13	3	1	3	4	2	15
Lys	K	2	37	25	6	0	12	7	2	4	1	9926	20	0	3	8	11	0	1	1	1
Met	M	1	1	0	0	0	2	0	0	5	8	4	9874	1	0	1	2	0	0	0	4
Phe	F	1	1	1	0	0	0	0	1	2	8	6	0	4	9946	0	2	1	3	28	0
Pro	P	13	5	2	1	1	8	3	2	5	1	2	2	1	1	9926	12	4	0	0	2
Ser	S	28	11	34	7	11	4	6	16	2	2	1	7	4	3	17	9840	38	5	2	2
Thr	T	22	2	13	4	1	3	2	2	1	11	2	8	6	1	5	32	9871	0	2	9
Trp	W	0	2	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	9976	1	0
Tyr	Y	1	0	3	0	3	0	1	0	4	1	1	0	0	21	0	1	1	2	9945	1
Val	V	13	2	1	1	3	2	2	3	3	57	11	1	17	1	3	2	10	0	2	9901

Table E.2: The PAM1 Matrix. Entries are scaled with 10000

	A	C	G	T
A	0	0.3009	0.0626	0.3009
C	0.3009	0	0.3009	0.0626
G	0.0626	0	0	0.3009
T	0.3009	0.0626	0.3009	0

Table E.3: The SCM250 Matrix

Bibliography

- [1] A. Pedersen and H. Nielsen. Neural network prediction of translation initiation sites in eukaryotes: Perspectives for EST and genome analysis. Technical report, Center for Biological Sequence Analysis, University of Denmark, 1997.
- [2] A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer and K.-R. Müller. Engineering support vector machine kernels that recognize translation initiation sites. *Journal of Bioinformatics*, 16:799–807, 2000.
- [3] M. Azbel. Random two-component one-dimensional ising model for heteropolymer melting. *Phys Rev Lett.*, 31:589–592, 1973.
- [4] B. Alberts, A. Johanson, J. Lewis, M. Raff, K. Roberts and P. Walter. *Molecular biology of the cell*. Garland Science, fourth edition edition, 2002.
- [5] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.
- [6] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic dna. *J Mol Biol*, 268:78–94, 1997.
- [7] C. Chang and C. Lin. LIBSVM: a library for support vector machines. 2001. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [8] C. Straessle and M. Boss. Prediction of genes in eukaryotic dna. Master’s thesis, HSR Hochschule für Technik Rapperswil, 2006.
- [9] C. Zhang and J. Wang. Recognition of protein coding genes in the yeast genome at better than 95 *Nucleic Acids Research*, 28:2804–2814, 2000.
- [10] T. Eitrich. *Support-Vektor-Maschinen und ihre Anwendung auf Datensätze aus der Forschung*. Forschungszentrum Jülich, 2003.
- [11] H. Li and T. Jiang. A class of edit kernels for SVMs to predict translation initiation sites in eukaryotic mRNAs. *Journal of Computational Biology*, 12:702–718, 2005.
- [12] H.-P. Hutter. *Comparison of classic and hybrid HMM approaches to speech recognition over telephone lines*. PhD thesis, ETH Zürich, 1996.
- [13] J. Gao, Y. Qi, Y. Cao and W. Tung. Protein coding sequence identification by simultaneously characterizing the periodic and random features of DNA sequences. *Journal of Biomedicine and Biotechnology*, 2:139–146, 2005.

- [14] J. W. Fickett and C. Tung. Assessment of protein coding measures. *Nucleic Acids Research*, 20:6441–6450, 1992.
- [15] JM. Claverie and L. Bougueleret. Heuristic informational analysis of sequences. *Nucleic Acids Research*, 14:179–196, 1986.
- [16] G. Karp. *Molekulare Zellbiologie*. Springer Verlag, 2005.
- [17] L. Rabiner and B. Juang. *Fundamentals of speech recognition*. Prentice Hall, 1993.
- [18] M. Burset and R. Guigo. Evaluation of gene structure prediction programs. *Genomics*, 34:353–367, 1996.
- [19] M. Little, P. McSharry, I. Moroz and S. Roberts. Nonlinear, biophysically-informed speech pathology detection. In *Proceedings of ICASSP 2006, IEEE Publishers*, 2006.
- [20] M. O. Dayhoff, R. M. Schwartz and B. C. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5:345–352, 1978.
- [21] M. Stanke and S. Waack. Gene prediction with a hidden markov model and a new intron submodel. *Bioinformatics*, 16:ii215–ii225, 2003.
- [22] M.O. Zhang and T.G. Marr. A weight array method for splicing signal analysis. *Oxford Journal Life Science Bioinformatics*, 9 number 5:499–509, 1992.
- [23] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 1 edition, 2000.
- [24] W. S. Noble. Support vector machine applications in computational biology. Technical report, Department of Genome Sciences, University of Washington, 2003.
- [25] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. Technical report, Microsoft Research, 1999.
- [26] S. Salzberg, D. Searls and S. Kasif. *Computational Methods in Molecular Biology*. Elsevier Science, 1998.
- [27] S. Tiwari, S. Ramachandran, A. Bhattacharya, S. Bhattacharya and Ramakrishna Ramaswamy. Prediction of probable genes by fourier analysis of genomic sequences. *Computer Applications in the Biosciences*, 13:263–270, 1997.
- [28] K. Sayood. *Introduction to data compression*. Morgan Kaufmann Publishers, 3rd edition, 2006.
- [29] T. Dietterich and G. Bakiri. Solving multiclass learning problems via error–correction output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.
- [30] V. Franc and V. Hlaváě. Multi–class support vector machine. Technical report, Czech Technical University, Faculty of Electrical Engineering, 2002.

- [31] R. Voss. Evolution on long-range fractal correlation and $1/f$ noise in DNA base sequences. *Physical Review Letters*, 68:3805–3808, 1992.
- [32] K. Weber. *HMM mixtures (HMM2) for robust speech recognition*. PhD thesis, EPFL Lausanne, 2003.
- [33] Wikipedia article. Markov chain. 2006. http://en.wikipedia.org/wiki/Markov_chain.