

Annotating 12 *Drosophila* genomes for regulatory signals using multiple neural networks and evolutionary history

ULF SCHÄFER
University of Oxford
St. Cross College
ulf.schafer@stx.oxon.org



supervised by:

Dr. Vasile Palade
Oxford University Computing Laboratory
vasile.palade@comlab.ox.ac.uk

and

Prof. Jotun Hein
Oxford University Centre for Gene Function
hein@stats.ox.ac.uk

Dissertation for the degree of Master of Science
in Computer Science

August 30th 2006

Abstract

The application of methods of machine learning is popular in Bioinformatics. Many problems in Bioinformatics can be regarded as pattern recognition problems and well-known methodologies can be used. Until now those methodologies concentrated on the use of a single genome. With the availability of datasets that contain genomes from several closely related species, information that was derived from the evolutionary history of those species can now be utilised in machine learning methods as well.

This project will introduce the well-know paradigm of multi classifier systems and deploy them for the recognition of promoter region in the genome of the fruit fly *Drosophila*. These systems consist of several artificial neural networks whose classification results are merged into a combined statement. The traditional multi classifier system will be extended by incorporating evolutionary parameters. The dataset that is used in this context contains the aligned genomes of 12 different yet closely related species of *Drosophila*.

Keywords: Machine Learning, Bioinformatics, Multi Classifier Systems, Classification, Neural Networks, Promoters, *Drosophila*, Evolution

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 7 |
| 1.1 | Project description | 7 |
| 1.2 | Structure of the document | 8 |
| 2 | Biological background | 8 |
| 2.1 | Structure of DNA | 9 |
| 2.2 | Detection of signals | 11 |
| 2.3 | Promoter structure and function | 13 |
| 2.4 | Data-related background | 16 |
| 3 | Literature review | 18 |
| 3.1 | NNPP | 19 |
| 3.2 | Promoter2.0 | 22 |
| 3.3 | PromoterInspector | 24 |
| 3.4 | Dragon Promoter Finder | 26 |
| 3.5 | McPromoter | 29 |
| 3.6 | Other approaches | 32 |
| 3.6.1 | Zhang (1998) | 32 |
| 3.6.2 | Stormo (1997) | 32 |
| 3.6.3 | Audic and Claverie (1997) | 33 |
| 3.6.4 | Hutchinson (1996) | 34 |
| 3.6.5 | Prestridge (1995) | 34 |
| 3.6.6 | Lawrence (1993) | 35 |
| 4 | Short case study | 35 |
| 4.1 | The example gene | 36 |
| 4.2 | NNPP | 37 |
| 4.3 | Promoter2.0 | 38 |
| 4.4 | Dragon Promoter Finder | 39 |
| 4.5 | McPromoter | 39 |
| 4.6 | Evaluation of the case study | 40 |
| 5 | Description of the method | 41 |
| 5.1 | Neural Networks in classification problems | 41 |
| 5.2 | Using multiple classifiers | 47 |
| 5.2.1 | MCS design and topology | 48 |

| | | |
|----------|--|------------|
| 5.2.2 | Performance of MCSs | 49 |
| 5.2.3 | Combining results | 51 |
| 5.2.4 | Diversity between classifiers | 54 |
| 5.2.5 | Encoding of input data | 55 |
| 5.2.6 | Reducing input dimensions | 57 |
| 5.3 | Incorporating evolutionary history into the MCS | 58 |
| 6 | Account of the work | 61 |
| 6.1 | Training an MCS with data pre-processed with NNPP | 62 |
| 6.2 | Observations regarding evolution in aligned data | 67 |
| 6.2.1 | Relationship of recognition results in related species | 67 |
| 6.2.2 | Examination of more detailed parameters | 69 |
| 6.3 | Reducing dimensionality of ready-to-use dataset | 75 |
| 6.3.1 | Application of property functions | 75 |
| 6.3.2 | Data classification using property values | 78 |
| 6.3.3 | Incorporation of evolutionary data | 80 |
| 7 | Description of the software involved | 83 |
| 7.1 | Java implementations of the property functions | 84 |
| 7.2 | Evolution related Java scripts | 85 |
| 7.3 | MATLAB scripts for combination methods | 87 |
| 8 | Conclusion and future work | 88 |
| A | Java implementations of the property functions | 96 |
| A.1 | GC content | 96 |
| A.2 | Distribution value | 96 |
| A.3 | Fractal value | 98 |
| A.4 | Ficket value | 100 |
| B | Evolution related java scripts | 102 |
| B.1 | Find matches in genome and calculate evolutionary parameters | 102 |
| B.2 | Determination of rate matrices | 106 |
| C | MATLAB scripts | 109 |
| C.1 | Majority voting MATLAB script | 109 |
| C.2 | Maximum MATLAB script | 110 |
| D | Gene CG17759 of Drosophila Melanogaster | 111 |

List of Figures

| | | |
|----|--|----|
| 1 | The structure of the DNA double helix | 11 |
| 2 | Eukaryotic promoter region | 14 |
| 3 | Male <i>Drosophila Melanogaster</i> | 16 |
| 4 | Phylogeny of the <i>Drosophila</i> genus | 17 |
| 5 | NNPP architecture | 21 |
| 6 | Structure of one neural network as used in Promoter2.0 | 23 |
| 7 | Promoter elements and promoter context | 25 |
| 8 | Structure of the Dragon Promoter Finder system | 27 |
| 9 | Accuracy of Dragon Promoter Finder | 29 |
| 10 | Structure of McPromoter | 31 |
| 11 | Annotation of <i>D. Melanogaster</i> gene CG17759 | 37 |
| 12 | Location of promoter in CG17759 according to NNPP | 37 |
| 13 | Predictions of NNPP on CG17759 with threshold setting 0.95 | 38 |
| 14 | Predictions of Promoter2.0 on CG17759 | 38 |
| 15 | Location of promoter in CG17759 according to Promoter2.0 | 38 |
| 16 | Location of promoter in CG17759 according to the Dragon Promoter Finder | 39 |
| 17 | Graphical representation of the results of the Dragon Promoter Finder on CG17759 | 40 |
| 18 | Location of promoter in CG17759 according to McPromoter | 40 |
| 19 | A single neuron with vector input | 44 |
| 20 | Hard-limit transfer function | 44 |
| 21 | Linear transfer function | 45 |
| 22 | Log-Sigmoid transfer function | 45 |
| 23 | Simplified diagram of a small artificial neural network | 46 |
| 24 | Cascading MCS architecture | 48 |
| 25 | Parallel MCS architecture | 49 |
| 26 | Application of multiple classifiers | 50 |
| 27 | Excerpt from the sequence alignment | 59 |
| 28 | Structure of the Multi-Classifer System | 64 |
| 29 | Average mutation rates in 150 promoters and non-promoters | 71 |
| 30 | Average transition/transversion ratio in 150 promoters and non-promoters | 72 |

List of Tables

| | | |
|----|---|----|
| 1 | Core promoter features | 16 |
| 2 | Performance of the NNPP | 22 |
| 3 | Performance of Promoter2.0 | 24 |
| 4 | Performance of PromoterInspector | 26 |
| 5 | Performance of McPromoter | 31 |
| 6 | Predictions in CG17759 by McPromoter with 65% sensitivity . | 39 |
| 7 | Different schemes used to encode DNA for the input to a neural network | 56 |
| 8 | Configurations of four neural network based classifiers | 64 |
| 9 | Results of networks A-D on test set | 66 |
| 10 | Combined results on test set with three different combination methods | 66 |
| 11 | Results of the MCS on the aligned training set | 68 |
| 12 | Mean promoter rate matrix | 73 |
| 13 | Mean non-promoter rate matrix | 74 |
| 14 | Table 12 - table 13 | 74 |
| 15 | Configurations and training sets for four networks | 79 |
| 16 | Results of networks A-D on test set | 79 |
| 17 | Combined results on test set with three different combination methods | 80 |
| 18 | Compared mean vales of evolutionary parameters | 82 |

1 Introduction

1.1 Project description

Methods of machine learning are popularly employed in the area of Bioinformatics. The reason for this is that the amount of data that is to be processed is vast, with more and more genomes being sequenced almost on a daily basis. Furthermore many problems in Bioinformatics such as gene finding and RNA secondary structure prediction can be regarded as pattern recognition problems in Computer Science, so that well-known methodologies can be applied.

Recently, the fruit fly *Drosophila* has been moved into the centre of attention for research in the field of Bioinformatics. This fly has been continuously examined by geneticists since 1910. It can be regarded as one of the key organisms for genetic study. Previous successes of genomic studies with this organism and its very promising nature have motivated the sequencing of 12 complete genomes of 12 different species of *Drosophila*. These sequences are now of major interest for comparative genomic studies. It is the aim of this project to address one of the large number of open problems that the data provides.

Previously, the authors of [1] developed a system which was successfully applied to a closely related problem in Bioinformatics. This system recognises regulatory signals in DNA of *E.Coli* bacteria and humans by means of computational intelligence. Promoters are localised by using multiple classification modules. The results of each individual classifier are merged into an overall statement about whether a certain input pattern is a promoter or not. It was experimentally proven that this statement is always more accurate than that of any individual classifier. The classifiers involved can be neural networks, genetic algorithms, hidden Markov models, stochastic context-free grammars and possibly others. In this project I will apply the same principle to the above mentioned 12 *Drosophila* genomes.

However, this methodology can be enhanced. The main difference between the *Drosophila* dataset and previously sequenced DNA is that the *Drosophila* dataset provides complete genomes for 12 different species which are closely

related. The evolutionary connection between these species is well-known, which makes additional information available. This is where the main advantage of the *Drosophila* dataset lies. Finding a way of exploiting this additional information in the context of multi classifier systems will be the second and most important part of my project. Finally it will be possible to design a system which is able to incorporate evolutionary history into the process of recognising promoters. The eventual outcome of this project is a system comprising multiple classifiers. Additionally it makes use of the additional information that can be obtained from the comparison of all 12 *Drosophila* genomes. It is expected that the results of this system will be even more reliable than those acquired from the analysis of the individual sequences independently.

1.2 Structure of the document

In chapter 2 of this document the problem underlying this thesis is explained in detail. Biological and data-related background will be elucidated. Chapter 3 gives a brief review of how this problem was tackled by others. Chapter 4 contains a short study how these methods perform on some showcase data. Chapter 5 gives a detailed description of the specific methods that are used to attack this problem in the context of this dissertation. After that follows chapter 6 that gives an account of the concrete work that was done to solve the problem. This chapter refers to the appendix of this document which contains samples of the coding used for the implementation. The results that I obtained are presented. Chapter 7 is a short section with explains the code in the appendix. This document finishes with chapter 8, which sums up the results and summarises the reasoning that can be made on the basis of these results and on the basis of the conducted work in general. It also gives an outlook what future work can be done in this particular field.

2 Biological background

This dissertation documents my final project for the degree of Master of Science in Computer Science. The subject of this project is an application of

methodologies of machine learning to the field of Bioinformatics. Bioinformatics is a relatively new field of study that employs methods of mathematics, statistics, computer science and others to solve biological problems. These methods are mainly employed to the analysis of DNA sequences.

Bioinformatics is a major application area for techniques of automated examination. This is because the amounts of data that are to be handled for the purpose of DNA analysis are so huge that they cannot be processed with traditional manual methods. Recently, two developments in genetics and computer science could be observed that condition and complement each other. Together these two developments further promote the fact that computerised analysis is the method of choice when working with DNA. On the one hand there has been a sharp increase in the availability of DNA data for genetic analysis. On the other hand advances in computer technology have made it possible for huge amounts of data to be processed within an acceptable amount of time. As a matter of fact the amount of base pairs in a single genome is so vast (3 billion bases in the case of humans) that a non computerised analysis will be unlikely to deliver results within a sufficiently short period of time. The simultaneous emergence of these two developments opens up great opportunities for computer-aided DNA analysis. Thus the field of Bioinformatics has been the subject of huge research efforts in the past decade and remains a very hot topic today and for the future.

2.1 Structure of DNA

As more and more genomes¹ become sequenced, the need for precise and reliable methods of automated analysis rapidly increases. There are a number of problems that are relevant in the context of the analysis of a DNA sequence where only an automated approach will present a solution in a timely manner. These problems are e.g. the identification of genes and signals within the DNA string, the alignment of different strings of DNA to each other and the prediction of the secondary structure of RNA (Ribonucleic acid) molecules. Since DNA strings are so extremely data-rich, acceptable results can only be obtained with a fast and reliable algorithm.

¹Genome: The collective hereditary information of an organism encoded in its DNA

DNA (Deoxyribonucleic acid) consists of a two very long chains of nucleotides². These two chains are wrapped around each other, thus forming the characteristic helix. Each of the nucleotides in the two chains consists of a sugar molecule called deoxyribose, which is connected to a phosphate molecule. This sugar-phosphate chain makes up the backbone of the DNA molecule. The sugar molecule is also connected to a base. This base can be one of the four options, adenine (A), cytosine (C), guanine (G) and thymine (T). The order in which these bases appear on the chain constitutes the way in which information is encoded within the DNA molecule. This order is often referred to as the genetic code of an organism. The computerised analysis of this code is one of the pivotal elements of research in Bioinformatics. The four bases adenine, cytosine, guanine and thymine can be divided into two groups according to their chemical characteristics. The first group is called purines and comprises the bases adenine (A) and guanine (G) among others that do not play a role in genetics. The second group is called pyrimidines and contains among others the remaining two bases cytosine (C) and thymine (T). The two members of each group further differ in the amount of hydrogen bonds they can establish. Adenine and thymine can establish only two, whereas cytosine and guanine can establish three. This is important, because the two chains of nucleotides in a DNA molecule are complementary to each other in such a way that for each nucleotide on one chain there is only one possible complement on the other chain. This complement is on the one hand always from the opposite group and can on the other hand establish an equal amount of hydrogen bonds. The effect of this is that adenine always bonds with thymine and guanine always bonds with cytosine.

²Nucleotide: A compound in organic chemistry that consists of several integral parts

The structure of the DNA molecule is shown in Figure 1.

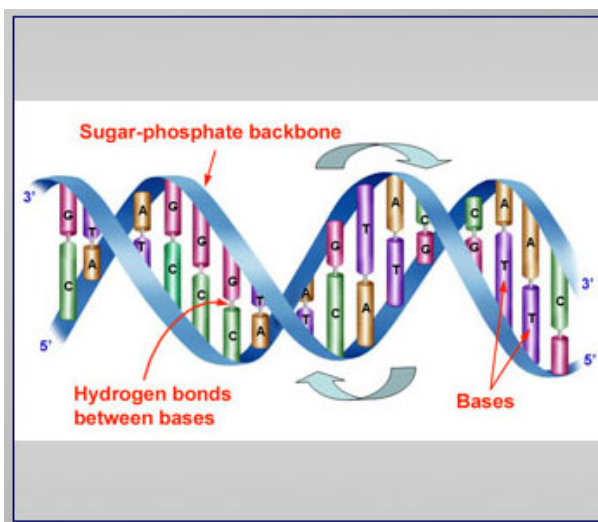


Figure 1: The structure of the DNA double helix

2.2 Detection of signals

An important part of an organisms DNA are regulatory signals. These parts of DNA control the gene and protein expression within the cells. The particular problem within this area that is being dealt within this dissertation is the detection and identification of a special kind of regulatory signal within the genetic code, the so-called promoter.

Promoters play an important role in the transformation from DNA into a protein. Proteins are the vital element of all living cells and define the characteristics of all life. Their production is a multi-staged process. It begins with DNA transcription and translation. This is followed by folding, post-translation modification and targeting. DNA transcription is the process by which DNA gets copied in order to produce complementary RNA. This RNA is in turn responsible for the translation of genes into proteins. The copying from DNA into complementary RNA is done by a polymerase called RNA polymerase (RNAP). The result of this copying process is mRNA (Messenger RNA), which is the first element necessary for a protein production process.

In fact mRNA is used to carry the information encoded in the DNA to the location of the protein production within the cell. Hence the name "messenger RNA".

Simplistically speaking, DNA consists on genes and intergenic DNA sequences. Only the genes are in fact transcribed and later translated into proteins. Genes themselves consist of introns and coding regions. Coding regions are those parts of the gene that actually get transformed into proteins. Introns are non-coding regions within a gene. They get cut out of the genetic code during transcription before mRNA is produced and thus play no role in the further protein production process. Promoters, the regulatory sequence that this project is attempting to locate, are short sequences of DNA located upstream from every gene, that allow the gene to be transcribed. The role of the promoter is to be recognized by the RNA polymerase, by which the transcription of the DNA is initialised.

The promoter sequence is always located immediately before the gene. Since the promoter, because of its inherent characteristics, is in general easier to detect than the gene itself, the approach that is chosen here is to use the location of the promoter in order to find the gene in the DNA sequence. This is very similar to the actual process in nature where the promoter is the part of the DNA sequence that indicates the existence of a coding region to the RNA polymerase.

The location of the promoter automatically yields the location of the gene. For the purpose of biological analysis it is of prime importance to have identified the location of the genes, because only the DNA within the genes is transformed into proteins and thus only the DNA within the genes is relevant to the behaviour of the cell. The identification of the gene locations is the first and most important step in comprehending the characteristics of a genome once it has been sequenced. For the *Drosophila* organism, whose genome is about 140 million base pairs long, it is expected to find approximately 13,000 genes.

2.3 Promoter structure and function

Since the recognition of promoters is the paramount goal of this project this section will give details about the structure of promoters and describe their role in DNA transcription.

Biology distinguishes two types of organisms, eukaryotes and prokaryotes. The main difference is that the cells of eukaryotes have nucleuses, whereas the cells of prokaryotes do not. All higher organisms, such as mammals, insects and plants are eukaryotes. Prokaryotes are mainly bacteria. This is mentioned here, because there are huge differences between promoters in eukaryotes and prokaryotes. All that is said here refers to eukaryotic promoters only. For simplicity the adjective "eukaryotic" was omitted in most cases.

As was explained in section 2, a promoter serves as a marker for the start of the coding region within the DNA sequence. Its function is to guide an RNA polymerase molecule to the correct position for it to start producing complementary mRNA from the DNA sequence. Unfortunately promoters do not possess a straightforward structure. There is no fixed length and no clear pattern of nucleotides that characterises promoters. Instead promoters consist of a number of various parts that are located upstream and downstream³ from the transcription start side (TSS) of the gene. These parts might be of variable distance to each other and the space in between is filled with nucleotides that do not play a role for promoter functionality. For protein coding genes these parts of the promoter are the core promoter, a proximal promoter region and a number of distal enhancers, which play auxiliary roles in transcription initiation. These auxiliary roles might only have to do with the spatial structure of the DNA molecule. They might also be more directly involved with the biochemical processes at transcription initiation. These enhancers can be as far as several kilobases away from the TSS. Interwoven with elements of the promoter are other regulatory signals as e.g. silencers that have an inhibitory effect on transcription initiation or simply junk DNA⁴.

³DNA is traditionally written from the upstream direction to the downstream direction. The upstream end is also called 5' (five prime) end and the downstream end is also called 3' (three prime) end

⁴DNA whose function is for the time being unknown.

The rough structure of a complete eukaryotic promoter region is illustrated in Figure 2.

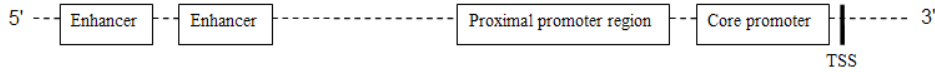


Figure 2: Eukaryotic promoter region

All of the parts of a promoter interact with so-called transcription factors (TF) in the cell. A range of transcription factors are required to interact with a number of promoter elements in order to make the RNA polymerase bind to the promoter sequence and start transcription. Transcription factors are proteins that are omnipresent in the cell. A group of 6 of them are the most important. They are abbreviated TFIIA, TFIIB, TFIID, TFIIIE, TFIIF and TFIIH. When the right promoter elements interact with the right transcription factors in order to attract the attention of a RNA polymerase molecule, one speaks of the establishment of a preinitiation complex (PIC). The interaction of one transcription factor with the DNA string might cause the string to fold back upon itself, which has the effect that even distant regions of the sequence might play a part in the local process.

Out of the three components of a promoter, most is known about the core promoter, which is located immediately before the TSS. It has been observed that many core promoters have an AT-rich region around position -30^5 . This region is called the TATA box and one of the most essential elements of a promoter. Its consensus sequence is T-A-T-A-A-A-(A). The TATA box is the target for TFIID. It is important to have TFIID attached to the DNA string, because this is the starting point of a whole range of other reactions that eventually lead to the construction of the preinitiation complex.

Regrettably not all promoters have TATA boxes. In such cases the PIC is established by the interaction of transcription factors with other regulatory parts of the promoter. In 1995, Arkhipova [2] found out that less than 50% of all promoters in *Drosophila* have TATA boxes, which is a lot less frequent than in vertebrates. In such cases the TFIID might bind to an initiator

⁵In this context the TSS serves as a point of orientation from where positions of base pairs are counted. The TSS is at position 1 with increasing numbers further into the gene and decreasing numbers the other way.

sequence. This initiator is usually located immediately around the TSS and the consensus sequence is T-C-A-G/T-T-C/T. It is almost needless to say that promoters do not necessarily have a TATA box or an initiator sequence, but that they might or might not have both or none. It has also recently been discovered that there exists a so-called downstream promoter element (DPE), which is as widely used in the construction of a PIC as the TATA box, but which is far less well-preserved. The DPE is located between positions 28 and 33 and it usually looks like this: A/G-G-A/T-C/T-A/C/G⁶. In an analysis of 205 core promoters in *Drosophila*, it was estimated that 29% of the promoters contain a TATA box only (no DPE), 26% contain a DPE only (no TATA box), 14% contain both TATA and DPE elements, and 31% do not appear to contain either a TATA or a DPE (Kutach and Kadonaga, 2000 [3]). Thus, it appears that the DPEs are about as common as the TATA box in *Drosophila*.

There are two more elements of the core promoter that are worth being mentioned. One is the so-called CCAAT box, which is located about 75-80 base pairs upstream of the TSS. It is however frequently not present. In combination with TATA boxes one might also encounter a TFIIB recognition element (BRE). The consensus sequence for this element is G/C-G/C-G/A-C-G-C-C. About 15% of all promoters that possess a TATA box also possess a BRE.

It should be added at this point that what exactly happens when transcription is initiated using RNA polymerase, a collection of transcription factors and various bits of DNA in the promoter region, is subject to massive ongoing research efforts in biology. The processes are very complex and to describe them exhaustively would go far beyond the scope of this dissertation.

To summarise, it can be said that there are mainly three elements of the core promoter that one might be able to exploit for the purpose of recognition. These are the TATA box, the initiator sequence and the downstream promoter element. It seems to be the case that a combination of those three elements always plays a role in the establishment of a PIC. It must however also be said that due to many possible variations eukaryotic promoters are extremely unequal and difficult to classify. For this reason their recognition

⁶The consensus sequences given here are for *Drosophila* core promoters. They are different for other eukaryotic organisms.

is a challenging task. The features discussed here are shown in table 1.

| Feature | Approx. position | Consensus sequence | Approx. occurrence in <i>Drosophila</i> |
|-----------|------------------|---------------------|---|
| TATA box | -30 | T-A-T-A-A-A-(A) | 43% |
| Initiator | 0 | T-C-A-G/T-T-C/T | ? |
| DPE | 28 - 33 | A/G-G-A/T-C/T-A/C/G | 40% |

Table 1: Core promoter features

2.4 Data-related background

This project was conducted with DNA data from 12 different species of the *Drosophila* genus (the common fruit fly). Figure 3 shows a male of the most well-know *Drosophila* species, *Drosophila Melanogaster*.

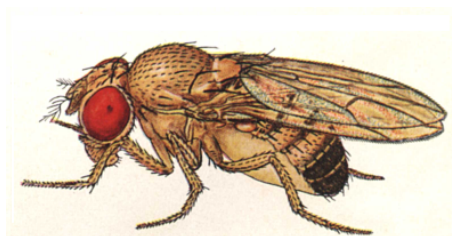


Figure 3: Male *Drosophila Melanogaster*

Recently, a proposal has been made to use the genus of *Drosophila* as a model system for comparative genomic studies [4]. The data that has been provided for this purpose consists of DNA sequences of 12 different species of *Drosophila* and it only became fully available in March 2006. As was put forward in the proposal whitepaper [4] *Drosophila* data is ideal for comparative genomics studies. The *Drosophila* family has been continuously researched by geneticists since 1910 and is one of the most well-studied organisms today. The sequences of 12 *Drosophila* species at hand are of extremely high quality. This characteristic together with the amount of existing knowledge about the *Drosophila* species make it a primary research interest. The existence of data for 12 different *Drosophila* species allows studies that are comparative to a

high degree. The presence of evolutionary information in this dataset gives rise to better opportunities for gaining deeper knowledge about the mechanisms behind evolution and the development of species than the study of a single large organism.

Figure 4 shows the phylogeny of the *Drosophila* genus. No other organism provides information for research in genetics and molecular biology to an extent, that would be comparable to the information available from the *Drosophila* genus. The exploitation of this information, especially of the known interrelationships between the individual species, is one of the essential elements of all research conducted with *Drosophila* data today. Without this exploitation the *Drosophila* dataset would be utilised far under its scope. How exactly this information is going to be used in this project for the detection of promoters is explained in detail in section 5 of this document.

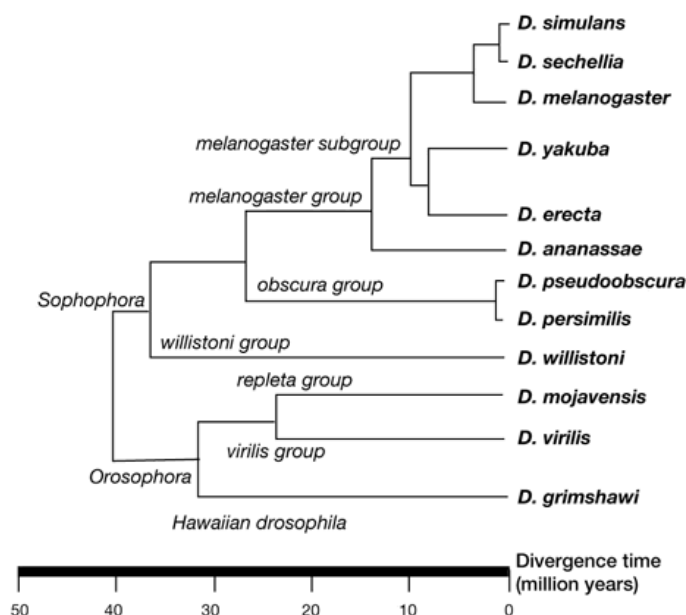


Figure 4: Phylogeny of the *Drosophila* genus (taken from <http://rana.lbl.gov/drosophila/>)

Since the *Drosophila* dataset is far more detailed and advanced than any other dataset that has been used for genetic research it is expected that it will yield better results and will lead to a better understanding of the principal

procedures that determine the structure of the DNA of every organism.

Discoveries about evolutionary mechanisms as well as gene functions and gene regulations made through the study of the *Drosophila* species have a direct influence on human genetics. Understanding the underlying processes in *Drosophila* allows us to advance our knowledge of the general mechanisms of evolution and to draw conclusions relevant to the examination and treatment of human genetic disorders. Changes in gene regulation and gene expression are often observed in oncogenes⁷. Studies of the processes involved in gene expression and regulation result in a better understanding of normal cell behaviour and abnormal behaviour in cancer cells. *Drosophila* is a suitable model for this kind of research, because about 60% of known human disease genes have a recognisable match in the genome of *Drosophila*.

3 Literature review

The problem of promoter recognition is as old as Bioinformatics itself and over the last decade and a half various systems have been developed to tackle this challenge. This chapter will describe some of the more prominent publications on the subject and describe the computational approaches for promoter recognition that have been made. The main approaches are:

- NNPP - Martin g. Reese - Berkeley *Drosophila* Genome Project
- Promoter2.0 - Steen Knudsen - Technical University of Denmark
- PromoterInspector - Matthias Scherf et al. - GSF-National Research Center for Environment and Health
- Dragon Promoter Finder - Vladimir B. Bajic et al. - Kent Ridge Digital Laboratories
- McPromoter - Uwe Ohler - Massachusetts Institute of Technology

⁷Oncogene: A modified normal gene that is in part responsible for a healthy cell to turn into a tumour cell

Details will be given about their methodology, the data used and it will be reported back on the results obtained. Each section of this chapter will introduce one popular standard methods for promoter prediction that has been published on the web. Chapter 5 introduces the method that has been developed for this project and comparisons are made. Another comparison concentrating on the approaches introduced in this chapter can also be found in [5].

Results are usually given in reference to a system's sensitivity, which refers to the percentage of all positive samples labelled as positive. The counterpart is a system's specificity referring to the percentage of all negative samples labelled as negative.

3.1 NNPP

NNPP (Neural Network for Promoter Prediction) was developed by the Berkeley Drosophila Genome Project [6]. The online version of this tool can be found in [7]. The system was developed in 1999, but was updated with new training data in 2002.

This system makes use of a time-delay neural network. A time-delay neural network is basically identical with a standard multi-layer perceptron (MLP). The only difference is that a neuron in a given layer can be connected to a neuron in the subsequent layer by more than one connection. If there is more than one connection the connections will have different weights and a time delay is associated with them, which means at different points in time the neuron fires with different intensities. This adds a memory to the neural network.

The time-delay network used in NNPP basically consists of two independent neural networks, one for recognizing the TATA-box and one for recognising the Initiator. Both of these features of the core promoter have been discussed in the section on the biological background of this project (section 2.3). The results of both of these networks are then combined into one output unit, which gives output scores between 0 and 1. The two networks that make up the NNPP are however not completely independent. The NNPP only has one input layer and one output layer. It furthermore features two hidden

layers. These hidden layers are not connected in series but in parallel, which means that each neuron from the input layer is connected to each neuron in both hidden layers and all neurons in both hidden layers are connected to the neurons in the output layer. In this situation, there are no connections between the neurons of the two hidden layers. Each one of the hidden layers is used to detect one of the two basic elements of the eukaryotic promoter mentioned above. In such an architecture the hidden layers are also called feature detection layers.

The input layer of the NNPP has 200 neurons. This is because the input window spans over 50 base pairs (from position -40 to +10) and each base is encoded using 4 bits (A:1000 C:0100 G:0010 T:0001). Not each of these input neurons is connected to all neurons in both feature detection layers. For the detection of the TATA-box a window of 30 base pairs has been selected (positions -40 to -10). Only the neurons in these positions in the input window are connected to the neurons in the feature detection layer that is responsible for the detection of the TATA-box. For the detection of the Initiator a window of 25 base pairs was chosen (positions -14 to +11). The neurons at those positions in the input window are connected to the neurons in the layer that is used for the detection of the Initiator. All neurons from both feature detection layers are connected to the output layer, which in this architecture consists of only one neuron. The output neuron implements a threshold. If the combined output of both feature detection layers is greater than or equal to this threshold the NNPP predicts the input sequence to be a promoter. Otherwise it is predicted not to be a promoter. This threshold can be determined by the user of the system. The web interface of the NNPP can accept sequences of any length between 51 bases and 100 kilobases. A sliding window of length 50 is applied to the input sequence. In order to avoid multiple matches in immediate neighbourhood to each other the application only returns local maxima in the sequence. If a 50 base input sequence is predicted to be a promoter it is assumed that the transcription start side (TSS) is exactly the 10th base from the right (from direction of the 3'-UTR⁸).

⁸Three prime untranslated region - the untranslated region downstream from the stop codon

The architecture of the NNPP is illustrated in Figure 5.

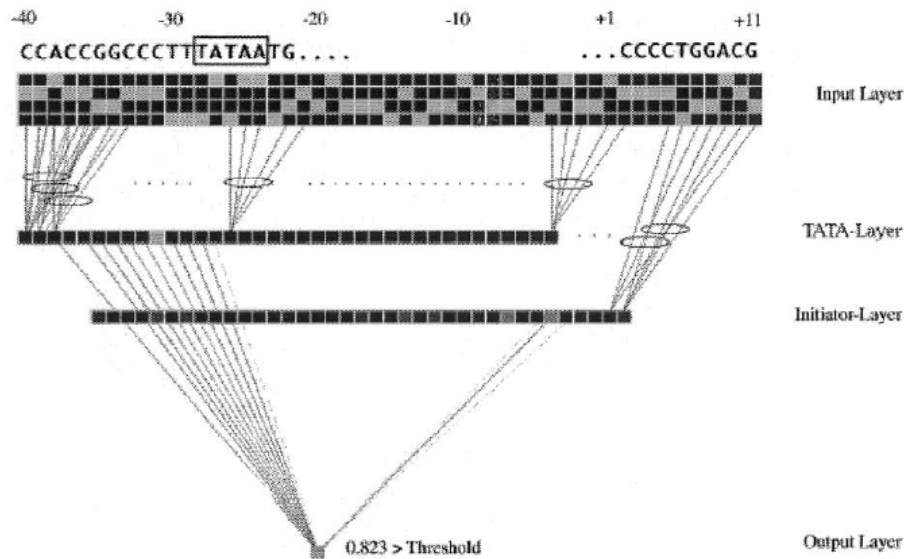


Figure 5: NNPP architecture (taken from [8])

One of the advantages of this architecture is that it allows a relatively precise detection of the two core promoter elements in relative independence from slight variations of the spacing in between them and variations in their positions within the input sequence. More details on the implementation of NNPP can be found in [8], [9] and [10].

Depending on the output neuron's threshold that is used in a given run of the system the performance of the system varies broadly. The higher the threshold is the less false positives are produced by the system. With a high threshold the system does however only detect a small proportion of the promoters that are actually in the data. This proportion can be significantly increased by lowering the threshold. Unfortunately this also increases the number of false positive recognitions that the system delivers.

Table 2 summarises the performance of the NNPP.

| Threshold | % Promoters recognised | False positives(in %) | Correlation coefficient |
|-----------|------------------------|-----------------------|-------------------------|
| 0.99 | 10 | 0.0 | 0.38 |
| 0.97 | 20 | 0.0-0.1 | 0.38 |
| 0.92 | 30 | 0.1-0.3 | 0.50 |
| 0.85 | 40 | 0.1-0.4 | 0.60 |
| 0.70 | 50 | 0.8-1.0 | 0.65 |
| 0.38 | 60 | 1.0-3.1 | 0.61 |
| 0.20 | 70 | 2.2-5.3 | 0.58 |
| 0.12 | 80 | 5.1-12.5 | 0.52 |

Table 2: Performance of the NNPP

The correlation coefficient that is given here is determined using the equation

$$CC = \frac{(TP * TN) - (FN * FP)}{\sqrt{(TP + FN) * (TN + FP) * (TP + FP) * (TN + FN)}} \quad (1)$$

The NNPP methods will be used later in this project for the production of a test and training set. The results presented in above table suggest the system to be used for this purpose. More details on this issues will be given in section 6.1.

3.2 Promoter2.0

Promoter2.0 [11] is an earlier approach to promoter recognition. It has been published as long ago as 1999. It is however important in the context of the present project, because it describes a different method of utilising multiple neural networks for promoter recognition. The present piece of work also utilises multiple neural networks, but in a different way. The Promoter2.0 is available to the public on the website of the Technical University of Denmark [12].

The system consists of four small neural networks. Each nucleotide is encoded using 4 bits. Each of the four neural networks is shown 6 nucleotides at one time, which make a total of 24 input neurons. Each network possesses only one hidden and one output neuron. There are also 2 extra input neurons, which feed the network with input from the previous network. Each of those networks scans over the input sequence in turn. For each of the positions in the input sequence the output of the output neuron is recorded. After the network has been shown all positions within the sequence only the maximal output activity is kept. Thus it can be fed to the next network scanning the sequence. Needless to say that for the first network scanning the sequence does not have any input from any previous scans.

The input from previous scans is the highest output for each of all previous networks, if any, on the same sequence and it is multiplied by a separation function. This separation function is linear and it is normalised to return 1 at the minimum and 0 at the maximum distance of the current position to the position with the previously recorded maximum output within the sequence.

The structure of one neural network with input neurons for input from previous networks is shown in figure 6. For brevity only two instead of 6 input nucleotides are displayed.

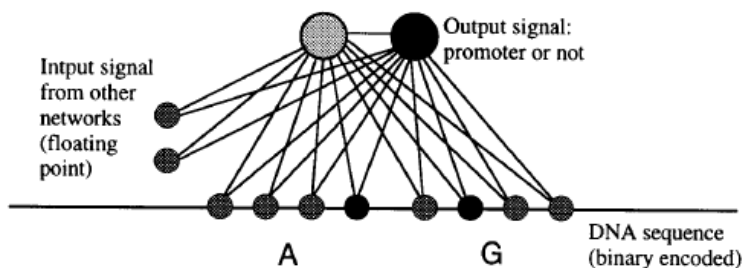


Figure 6: Structure of one neural network as used in Promoter2.0 (taken from [11])

It ought to be mentioned that the training of those neural networks is conducted in an unsupervised way, which means that the training target is not known beforehand. This is why the standard back-propagation algorithm usually used to train neural networks is not applicable to this particular problem. Instead an algorithm of random optimisation is used. Optimisation is thus performed on the ability to classify into promoter sequence or not. Here it does not play a role where in the test sequence the locations

of the signals are. This approach to optimise the parameters of the neural networks is very similar to the use of a genetic algorithm.

The reason why multiple neural networks are run on the sequence and their maximum output is recorded and passed into the next networks to scan the sequence, lies within the structures of the promoters to be recognised. As was explained in the section on the biological background of this project (section 2), the elements of the core promoter are of variable distance to each other. The approach presented in Promoter2.0 overcomes this problem elegantly. The solution is somewhat similar to the approach presented in the NNPP system (see 3.1), where two feature detection layers were used for the same reason.

The results of a limited test on a set of 100 vertebrate promoters is summarised in table 3. These numbers are rough estimates based on this limited test set. On average, the system picks up about 80% of all promoters. For a favourable comparison of this systems to other promoter prediction methods, see [13].

| Positions scoring | True TTSs |
|-------------------|-----------|
| 0.5-0.8 | about 65% |
| 0.8-1.0 | 80% |
| above 1.0 | 95% |

Table 3: Performance of Promoter2.0

3.3 PromoterInspector

PromoterInspector is an algorithm that was published in 2000 by the German Institute of Mammalian Genetics in collaboration with a company called Genomatix GmbH. The PromoterInspector algorithm focuses on the genomic context of promoters rather than their exact location. It also makes use of more than one classifier, but again in a different way from the Multi Classifier System proposed here. Details of the PromoterInspector algorithm were published in [14]. A web based version of the software is available to the public at [15].

The algorithm that PromoterInspector is based on relies in its predictions on context specific features which were extracted from mammalian training sequences. These context features are defined as oligonucleotides⁹ that have a wildcard, typically noted as "N" in the nucleotide sequence, in several positions. These are grouped into two classes, promoter and non-promoter. Using an unsupervised learning algorithm input sequences are classified as belonging to one of those classes if they have more matches in one class than in the other. This analysis of feature frequencies is the basis of the prediction. The classification process into the promoter and the non-promoter class is as follows. Starting from a set of training sequences consisting of promoters and non-promoters, an input sequence is assigned to the class promoter, if the ratio between the number of matches in the promoter and non-promoter training sequences exceeds a certain threshold. The opposite is true for an assignment to the non-promoter class.

PromoterInspector uses three distinct classifiers. Each classifier is specialized to differentiate between promoter and one of the following non-promoter sequences: exons, introns and 3'-UTRs. The prediction system assigns a sequence to the class promoter only if all three classifiers decide that the sequence belongs to this class. The algorithm will not yield an exact position of the transcription start side. It will only hint to the genomic context in which the promoter region is supposed to be residing. It is therefore to be considered more as a pre-processing step that is applied on a large genome, before techniques for the exact localisation and analysis of promoters are applied. The general idea behind the detection of single promoter elements and the detection of the genomic context of the promoter is illustrated in figure 7.

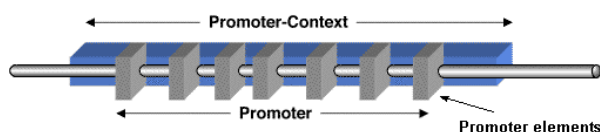


Figure 7: Promoter elements and promoter context (taken from <http://www.genomatix.de>)

The system is applied to large genomes using a sliding window technique. A promoter region is reported if a number of subsequent windows delivers

⁹Oligonucleotides are short sequences of nucleotides typically with twenty bases or less

a match. This creates three more parameters that need to be estimated in order to maximise the performance of the system. These are the size of the window, the distance between consecutive windows and the number of consecutive matches that make the system report the detection of a promoter region. The designers of the algorithm used a cross-validation brute force approach that suggested that the optimal values for these parameters are 100, 4 and 24 respectively.

PromoterInspector was applied to several large genomic sequences with a total volume of 1.3 million base pairs. This led to the discovery that PromoterInspector is able to predict promoter regions with a high degree of specificity. The experimental results are summarised in table 4.

| Genome | Sequence length | Number of TSS | True positives | False positives | %TP of total predictions | %Coverage |
|--------|-----------------|---------------|----------------|-----------------|--------------------------|-----------|
| mouse | 227538 | 17 | 5 | 1 | 83.3 | 29 |
| human | 219447 | 11 | 6 | 14 | 30.0 | 55 |
| human | 301692 | 1 | 1 | 2 | 33.3 | 100 |
| human | 101569 | 1 | 1 | 0 | 100.0 | 100 |
| mouse | 204635 | 4 | 4 | 1 | 66.7 | 50 |
| human | 324816 | 1 | 1 | 1 | 50.0 | 100 |

Table 4: Performance of PromoterInspector

The value in the column "Coverage" describes the percentage of true promoters in a sequence, which have been predicted correctly.

3.4 Dragon Promoter Finder

Another algorithm that utilises an approach similar to the one used in PromoterInspector is used in the Dragon Promoter Finder system. This system was proposed in 2002 [16]. It is available to the public on the internet [17]. The overall structure of this promoter recognition system is elucidated in figure 8 overleaf.

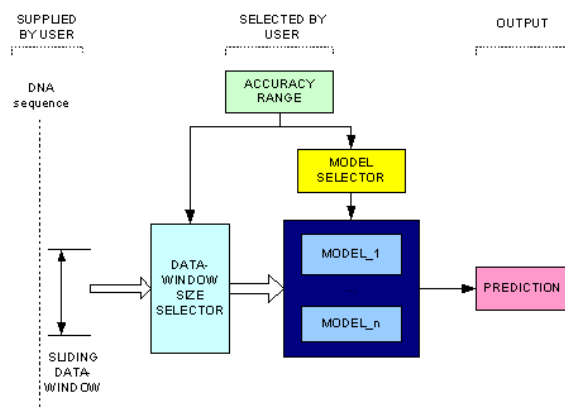


Figure 8: Structure of the Dragon Promoter Finder system (taken from <http://research.i2r.a-star.edu.sg>)

Users are requested to make two inputs to the system. On the one hand they supply the DNA sequence to be analysed. On the other hand they select a level of accuracy from a given range. This level of accuracy is given in the form of a specificity range. Depending on the chosen accuracy level the system selects two parameters. Firstly the size of the window that is used to read in a subsequence of nucleotides from the input data is optimised according to the desired accuracy level. Secondly the system chooses one out of a given number of models for data analysis. Each individual model was trained for a narrow specificity range. The model itself is now responsible for the actual recognition by analysing the content of the data windows. First the data passes through three parallel sensors. Each sensor is set up to recognise a particular functional region of a gene. This can be either a promoter, an exon or an intron. The system further processes the output of those three sensors by feeding them into a neural network. Each model was trained to separate promoter from non-promoter regions. All scores that make the output of the neural network greater than a selected threshold are considered to be promoters.

The individual models for the different functional regions of the gene were derived from positional distributions of overlapping pentamers¹⁰ in a region. By calculating the statistical relevance of the pentamers, only those were used which contribute most significantly to the separation between promoter

¹⁰A sequences of five consecutive nucleotides

and non-promoter regions. Only the 256 most significant ones out of the 1024 possible pentamers were selected for each model. The positional distributions of the selected pentamers were represented by their positional weight matrices (PWM). By comparing the content of the input window to the PWMs a representational score for each input was calculated. The score takes a value between 0 and 1, where it is assumed that the higher the score, the more likely the data window represents the respective functional region, i.e. contains either a promoter, an exon or an intron.

The scores for all three functional regions were fed to a neural network. If the output of the neural network is greater than a given threshold it was assumed the a) the input sequence is a promoter and b) that the transcription start site (TSS) of the respective gene is at the position 50 base pairs before the end of the input window.

More details of this algorithm can be found in [18] and [19]. The papers also contain detailed comparisons between this system and the three systems discussed above. They come to the conclusion that Dragon Promoter Finder outperforms them by a wide margin. An overview of those comparisons is given in figure 9 overleaf.

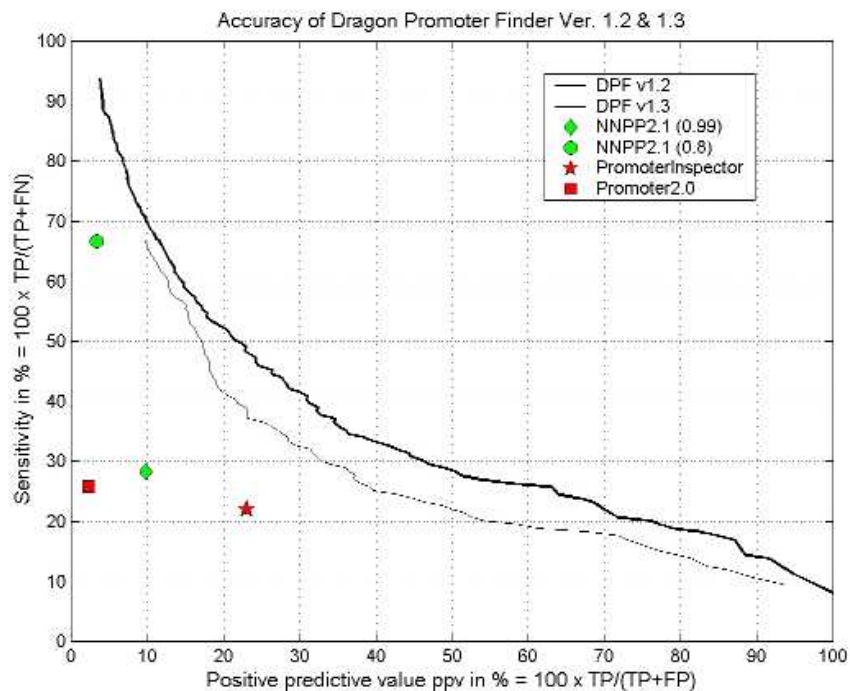


Figure 9: Accuracy of Dragon Promoter Finder expressed in terms of sensitivity and positive predictive value (ppv). The ppv is sometimes considered a specificity measure in Bioinformatics literature. See [20] (Image taken from [18])

It was the aim of this implementation to come up with a system that works with a high specificity. Previous systems displayed a high rate of false positives, which made them less valuable for practical usage.

3.5 McPromoter

This promoter recognition system was developed by Uwe Ohler at the Technische Universität Erlangen and the University of California in Berkeley. Most of the methodologies used are described in his PhD thesis [21] and in another paper of his [22], which proposes extensions to the system. The online version of McPromoter can be found in [23].

One of the proposed extension is of special interest because it proposes to in-

tegrate physical properties of the DNA sequence into the recognition process. These properties include bendability, assumed 3-dimensional structure and the ratio of G and C in the string. The paper in fact shows that such properties can be used in order to boost recognition results. This is interesting in so far that it shows that existing recognition systems can be improved through the incorporation of information that was gained from sources other than the strict sequence of nucleotides. The incorporation of evolutionary information into an existing system, which will be discussed in a later chapter of this document, is in this sense a very similar approach.

Another interesting feature of the basic McPromoter system is that it exists in two versions. One version uses the default algorithm, which is suited for the analysis of the DNA of humans and other vertebrates. The other one is a special version of the same algorithm with adaptations for the use with *Drosophila* DNA. This is done in order to accommodate the fact that there is knowledge about special characteristics of *Drosophila* DNA. This extra knowledge proves beneficial for the recognition process. As was mentioned in section 2 *Drosophila* promoters are known to display a relative infrequency in the appearance of strong TATA-boxes in comparison to other organisms.

The basic McPromoter system for humans and other vertebrate without any extensions is in itself a probabilistic system. It uses a hybrid system with two different models. These models make predictions using interpolated Markov chains and they output a likelihood whether a certain input belongs to a specific class or not. The system contains a background model consisting of states for coding and non-coding sequences and a promoter model. This promoter model recognises promoters by dividing them into six consecutive parts. All of these models are applied to a sliding window of size 300. In the *Drosophila* system the output likelihoods of the background and the promoter model are fed into a neural network along with likelihoods representing the DNA structure in the six segments. Any output of this neural network that is above a preset threshold is considered to represent a detected promoter. McPromoter does not strictly require certain features to be present at certain places in the sequence. It suffices for a positive detection if the combination of all features is good enough. That means that in case the TATA box is insignificant, there can still be a positive prediction if the other features deliver a high enough score.

A rough outline of the structure of the system is shown in figure 10. The basic system for non-Drosophila data is shown.

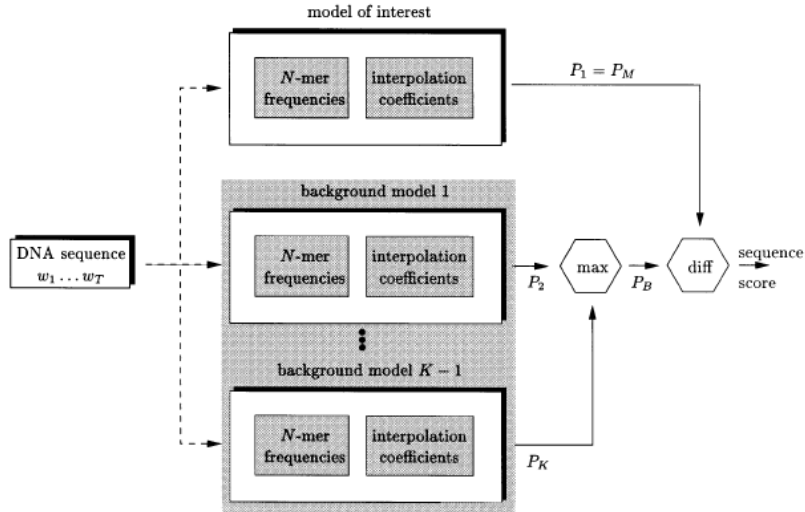


Figure 10: Structure of McPromoter (taken from [24])

Details of the implementation of the Drosophila McPromoter System can be found in [25] by Uwe Ohler et al. Details on the use of interpolated Markov chains for promoter recognition can be found in one of his other papers [24].

Tested on the Adh region in Drosophila Melanogaster the McPromoter system delivers results according to table 5

| Threshold | Sensitivity | Specificity | FP-rate |
|-----------|-------------|-------------|-----------|
| 0.98 | 15.5 | 69.2 | 1/106,647 |
| 0.95 | 36.9 | 50.7 | 1/25,853 |
| 0.9 | 52.1 | 40.3 | 1/12,016 |
| 0.8 | 65.2 | 29.3 | 1/5.884 |

Table 5: Performance of McPromoter

3.6 Other approaches

Previous sections of this document have documented rather recent and popular system for promoter recognition. All of those are publicly available on the internet and were used in a short comparative study that is described in section 4 of this thesis. The following short paragraphs illustrate further approaches to promoter recognition some of which have been published earlier.

3.6.1 Zhang (1998)

This system was published in 1998 by Michael Q. Zhang of the Cold Spring Harbor Laboratory in New York. Details of the implementation can be found in [26].

The novelty in the approach is that Zhang proposes a stepwise strategy when looking for promoters. In a first step he detects what he calls "extended promoters" from a large genome. These "extended promoters" are sequences of 1-2kilobases which contain the complete promoter context including the core promoter, the proximal promoter regions and possibly a number of the remote enhancers. After those extended promoter regions have been localised, he proposes to further localise the transcriptional start site (TSS) within the core promoter region. This is done using positional dependent 5-tuple measures for the implementation of a quadratic discriminant analysis (QDA) method.

Using this approach it is possible to locate the TSS to a 100-bp interval 60% of the time.

3.6.2 Stormo (1997)

This system was proposed by Gary Stormo et al. in 1997. Details of the implementation of the system can be found in [27].

The system uses training sets of promoters and non-promoters. In a first step patterns of 5-10 base pair long strings are identified among all possible permu-

tations. Only those strings are selected that are significantly over-represented in the promoter set. The program also searches matrices that have a significantly higher presence in the promoter set. The results of the searches are stored in the PromFD database, and the program PromFD scores input DNA sequences according to their content of the database entries. The results are reported to be a detection rate of 71% in the training set and 47% in the test set with a false positive rate of 1/13,000bp and 1/9,800bp respectively. Should the program classify an input sequence as a promoter it will try and locate the promoters TATA-box and output its location as the result of the analysis. This is of course problematic, because as was stated earlier not all promoters possess a TATA-box. In *Drosophila* e.g. it has been shown that only less than 50% of the promoters have one.

At the time this algorithm was implemented however the approach of using an Information Matrix Database was new and the false positive rate was lower than previous attempts to solve the same problem.

3.6.3 Audic and Claverie (1997)

This system was proposed by Stephane Audic and Jean-Michel Claverie from the Institute of Structural Biology and Microbiology in Marseille, France. Their paper [28] describes the proposed algorithm in detail.

Using the Eukaryotic Promoter Database (EPD), an extensive collection of sequences proven to be promoter regions, as a source of knowledge the approach chosen by Audic and Claverie was to detect promoters based on a Markov transition matrix that was built using the EPD data. Thus no prior biological knowledge about promoter structure was incorporated into the algorithm, because it restricts itself to the knowledge of positive and negative examples. The algorithm analyses a 250 base pair window upstream from the transcription start site. Despite its simplicity and speed the Markov algorithm was, according to the authors, able to achieve on the training set performances similar to that previously reported for more sophisticated programs.

3.6.4 Hutchinson (1996)

This system was proposed by G. B. Hutchinson in 1996 from the Department of Medical Genetics, University of British Columbia in Vancouver. Details about the system can be found in his paper [29].

In contrast to the approach by Stormo and the approach by Audic and Claverie, Hutchinson attempted to develop a system that detects promoters without the use of an extensive database that contains positive and negative patterns. In order to do this he determined hexamer¹¹ frequencies, which he got from known promoter regions, coding regions and non-coding regions in the DNA of vertebrate. Using those he created a discriminant measure that compared promoter regions with coding and non-coding sequences.

This algorithm is able to detect 61% of all promoters in an independent test set with a false positive rate of 1 in 208,714 base pairs.

3.6.5 Prestridge (1995)

Another early approach to promoter recognition was published by Dan S. Prestridge from the University of Minnesota in 1995. His approach makes use of Transcription Factor Binding Sites (TFBS). Details can be found in his paper [30].

The paper reports the development of a computer program named PROMOTER SCAN designed to detect Polymerase II promoters with a low false positive rate. For this purpose test sets of promoter and non-promoter data were analysed for the comparative density of each unique Transcription Factor Binding Site. From that a ratio of density of each transcriptional element in promoters and non-promoters was derived. The combination of all of those binding site ratios was then used to construct a scoring profile called a Promoter Recognition Profile. Apart from this scoring profile the system also uses a weighted matrix for scoring a TATA-box. Both of these elements are then used in PROMOTER SCAN to discriminate between promoter and non-promoter regions.

¹¹A sequence of 6 consecutive nucleotides

When the program is calibrated to detect 70% of the promoters in a sequence the false positive rate is 1 in 5,600 base pairs.

3.6.6 Lawrence (1993)

Another important piece of work in this field was done by Charles Lawrence et al. in 1993. Their approach is not limited to the detection of promoters, but applies to the identification of subtle patterns that occur analogously in multiple sequences. The algorithm is described in detail in their paper [31].

It is not the aim of the proposed system to find common pattern in multiple sequences that have a lot of similarity due to common ancestry. On the contrary the algorithm attempts to find common patterns in sequences that differ greatly. It is assumed that these similarities reflect structural and functional constraints, thus revealing information about the structure and the functionality of the analysed sequences. The algorithm works in three steps. First a relatively small number of patterns each consisting of one ungapped segment from each of the input sequences is sought. Secondly a single pattern is described by a probabilistic model of residue frequencies at each position. Thirdly the location of the pattern within the sequences is described by a set of probabilistically inferred position variables. A similar approach using multiple sequence alignment was chosen by Hertz and Stormo in 1999 [32].

The algorithm is able to solve difficult multiple sequence input without any prior expert knowledge. It displays accurate performance in the identification of weak but biological significant patterns in a timely manner and with high sensitivity.

4 Short case study

In this chapter, the systems that have been introduced in section 3 of this report will be examined with the help of a practical example. For this purpose a gene will be taken from the FlyBase GadFly Genome Annotation Database of the Drosophila Genome project at the University of California in Berkeley.

The systems that will be evaluated using this gene are:

- NNPP
- Promoter2.0
- Dragon Promoter Finder
- McPromoter

These systems are available to the public on the internet and I will be using these online tools and present the results obtained. The system "PromoterInspector" that was examined in the section "Literature Review" can unfortunately not be evaluated, because it has been commercialised since its development and is no longer freely available on the internet. The results will be compared with the information derived from the annotation which was provided by the FlyBase Annotation database.

4.1 The example gene

To the best of my knowledge there is no special gene in *Drosophila* that has more significance than any of the others. For this reason I selected an annotated gene from the database at random. The gene I have chosen has the identification number CG17759 and it is located on the 2R chromosome. The FlyBase Annotation database offers the transcript sequence of this gene plus 2000 nucleotides upstream and downstream of the mRNA boundaries. The data is annotated for intron/intergenic sequences, untranslated regions (UTR) and exon/coding sequences (CDS).

The annotated gene is taken from the database with a total length of 11,561 nucleotides. A written out form of the whole gene can be found in appendix D. Figure 11 shows the annotation of the gene as it was taken from the flybase database.

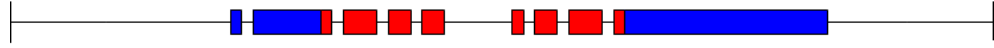


Figure 11: Annotation of *D. Melanogaster* gene CG17759

This figure shows the CG17759 *D. Melanogaster* gene including a region of intergenic sequence upstream and downstream from the gene itself. A simple black line indicates either intron or intergenic sequence, red boxes represent coding DNA and blue boxes stand for untranslated regions. The figure is only roughly drawn to scale.

4.2 NNPP

The sequence of 11,561 nucleotides is run through the web interface of the NNPP system. Depending on the selected threshold value, the system reports different numbers of transcriptional start sites. With a value of 0.99 only one match is reported between position 4125 and 4175. The sequence is

```
aactattgttaaaaaacgcttcattgcgccaattgcgtgcCctgcccct
```

The system assumes the TSS to be at position 10 from the left in this string. It is shown in larger font. This makes the location of the TSS to be exactly 882 nucleotides upstream from the beginning of the first blue-coded UTR in the gene above. This seems to be a reasonable guess for the location of the core promoter. Figure 12 shows the location of the promoter as a yellow box.

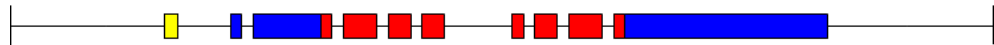


Figure 12: Location of promoter in CG17759 according to NNPP

With a threshold setting of 0.95 the system reports 12 more possible TSS. These are shown in figure 13:

| Start | End | Score | Promoter Sequence |
|-------|-------|-------|--|
| 2635 | 2685 | 0.97 | tttttgggcttaaaacgggtgggtaaccacaaaaatgaaa2aaaaatgg |
| 3208 | 3258 | 0.99 | aaactttatattaaaaagcaccagtaaaaatatataacc2actacacac |
| 3385 | 3435 | 0.97 | ctaaaaatatttaaacagacggaacatoggcaggggaacc2aaaaacaaa |
| 3679 | 3729 | 0.97 | tggttgccactaaaaactggccgaaatatattataatcca2aaatcgaat |
| 4125 | 4175 | 0.99 | aaactattgttaaaaaacgcttcattgcccattgctgpc2Coctgcccct |
| 4442 | 4492 | 0.96 | catcactcogatatataaaactaggggttgaaaaagagagga2atagcacca |
| 4960 | 5010 | 0.97 | ccatgggcaatctatatccgacgtcgtcatcgacgtcgc2dgctagtggg |
| 5485 | 5535 | 0.97 | CAITTTATATACGGAACGGTCTGTGAGCGTAGTGGCAGC2AGCAGCGGAA |
| 5762 | 5812 | 0.98 | cttatggctatataatcttccaatcctataatccccag2GCActGGCGAGT |
| 9206 | 9256 | 0.99 | TACTTTTCGTAAAAATCCCCGAAATATATATATAT2CAAGAATTATA |
| 9663 | 9713 | 0.98 | tacgtgtttgtaaaaaatgtogataaccaccatgcaaca2tgcaacatgc |
| 11022 | 11072 | 0.97 | ccccatatttatataggacagactccaaatgggtact2Ccaaaacttga |
| 11097 | 11147 | 0.99 | atggagcctgttaaaagccccagacattcagattatcag2Catgtgtgta |

Figure 13: Predictions of NNPP on CG17759 with threshold setting 0.95

4.3 Promoter2.0

The same sequence run through the web interface of the Promoter2.0 sequence delivers the results shown in figure 14.

Sequence, 11561 nucleotides

| Position | Score | Likelihood |
|----------|-------|--------------------------|
| 700 | 0.672 | Marginal prediction |
| 1600 | 0.723 | Marginal prediction |
| 2100 | 0.558 | Marginal prediction |
| 4900 | 1.120 | Highly likely prediction |
| 5900 | 0.699 | Marginal prediction |
| 7900 | 0.701 | Marginal prediction |

Figure 14: Predictions of Promoter2.0 on CG17759

Here the highly likely prediction lies 148 nucleotides upstream from the beginning of the first blue-coded UTR in the gene above. Figure 15 shows the location of the promoter as a yellow box.



Figure 15: Location of promoter in CG17759 according to Promoter2.0

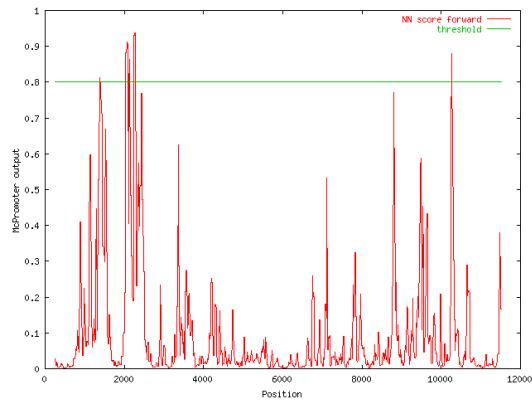


Figure 17: Graphical representation of the results of the Dragon Promoter Finder on CG17759

The location of the most likely prediction is illustrated in figure 18 below.



Figure 18: Location of promoter in CG17759 according to McPromoter

4.6 Evaluation of the case study

The analysis was performed with a single example gene which was chosen at random from the FlyBase Genome Annotation Database. This gene was run through four systems for promoter recognition that have been studied in detail in a previous section of this document and are available to the public on the internet. This analysis shows that there are as many opinions about the location of the promoter in a gene as there are systems performing the detection. If the annotation taken from the database is correct the location of the core promoter is probably somewhere not too far upstream from the first blue-coloured UTR, i.e. approximately between nucleotides 3500 and 5000. Two of the above evaluated systems (NNPP and Promoter2.0) report a promoter in this region, but they disagree by 775 base pairs about the exact location of the TTS. The other two systems claim to have detected promoters at other positions in the gene.

This striking disagreement in results between different approaches on identical data shows that computational promoter recognition is a very challenging and complex task. Due to the large variance in the biological data and the variety of promoters a reliable and universal recognition is almost beyond reach. Depending on the kind of computational approach chosen, the data that is used to adapt the algorithm for the recognition is a very decisive element that determines the performance of the final system.

5 Description of the method

5.1 Neural Networks in classification problems

The process of classification describes the allocation of previously unknown data into a number of predetermined groups or classes. This is a very common problem and it can be computerised. The resulting system is called a classifier. In the context of this project the data shown to the classifier is strings of DNA of different species of *Drosophila*. The classes that this data is being allocated into are "promoter" and "non-promoter".

The algorithms that are used in classification problems are numerous. Basically they can be split in two groups, rule-based classification and computational intelligence based classification. Rule-based classification generally relies on rules that have been installed by the engineer creating the system. These rules are used to interpret the presented information. They generally make direct connections between properties the data exhibits and the classification result. When the data to be classified is sufficiently complex, rule-based classifiers have a big disadvantage, i.e. that the creator of the system might not fully comprehend the coherence between characteristics of the data and the data's assignment to a certain class. Implications that lead from properties of the data to information about the class the data belongs to, might not be fully recognised. If the system's designer implements the system erroneously, the classification results of the rule-based classifier will be flawed from the very beginning. If the system's designer e.g. installs a rule that states that the presence of a TATA-box in a DNA sequence means that the sequence is a promoter, the classification system will perform

badly on *Drosophila* DNA. This is because the actual relationship between TATA-boxes and promoters is much more complicated than this basic rule can describe. For simple classification problems, where the data displays only a small degree of complexity rule-based classifiers may however be the appropriate choice.

The counterpart of rule-based classification is computational intelligence based classification. Here the designer is not responsible for the creation of the rules, which are used for the interpretation of the input. Instead s/he only sets up a fundamental framework. The rules for the correct classification of the data are after that automatically created by a learning or training algorithm. This algorithm abstracts from the characteristics of the input data automatically and thus approximates the classification function. For complicated classification problems in Bioinformatics this approach is in most cases better suited, because it is less error-prone since it does not have to rely on a human designer to fully comprehend the data.

Computational intelligence based classification methods can further be subdivided into those using supervised and those using unsupervised learning paradigms. Supervised systems require the presence of a "teacher". This "teacher" supervises the automated learning process of the classifier. The "teacher" presents the algorithm with a so-called training set, which comprises a sufficiently large number of samples from the input data and the information into which class each of the individual pieces of input data should be correctly classified. The "teacher" also informs the classifier how far a generated output is from the correct answer. The supervised training algorithm requires this input to modify parameters within the classification system. These modifications purpose improvement of the classification performance. After training, the classifier is expected to classify previously unseen, but similar, inputs to the correct classes. The best example of such a supervised learning technique is the backpropagation algorithm used in conjunction with artificial neural networks.

In contrast to systems utilising supervised learning, unsupervised classifiers are expected to be trained without prior information about the data. The presented data is inspected and assigned to different classes using similarity measures and clustering factors. Self-organizing maps, which are also used with neural networks, are an example of such a paradigm.

As can be seen from the two exemplifications for algorithms of supervised and unsupervised learning above, artificial neural networks are one of the most popular implementations of the computational intelligence based classification paradigm. The idea behind the design of these artificial neural networks is to imitate natural neural networks as they occur in the brains of living creatures. Their smallest element is an artificial neuron. These neurons consist of a theoretical arbitrary number of inputs. These inputs are all to be multiplied with a weight that is specified for each of these inputs separately. The sum of all weighted inputs is the argument for a transfer function, which creates the output of the neuron. Assuming an input argument vector $i = i_1, i_2 \dots i_n$ and a vector of weights of equal size $w = w_1, w_2 \dots w_n$ and assuming the transfer function to be denoted f and the output itself to be denoted o , the neuron is defined by equations 2 and 3.

$$o = f(X) \tag{2}$$

$$X = b + \sum_{k=1}^n i_k w_k \tag{3}$$

The neuron also has a so-called bias b which is a constant that is added before the transfer function is evaluated. The layout of a single neuron is shown in figure 19 overleaf.

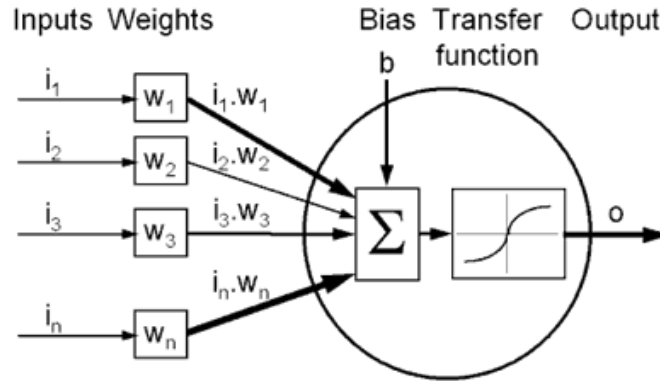


Figure 19: A single neuron with vector input (taken from [33])

Three of the most commonly used transfer functions are briefly described below.

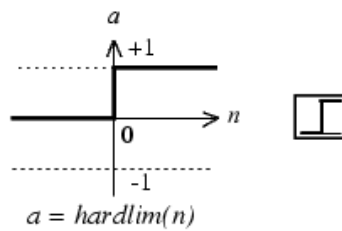


Figure 20: Hard-limit transfer function (taken from [33])

The hard-limit transfer function limits the output of the neuron to either 0, if the input argument n is less than 0 or 1 otherwise.

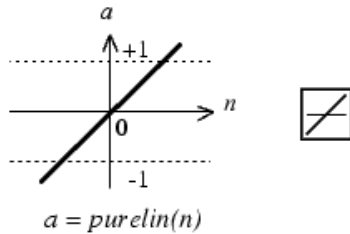


Figure 21: Linear transfer function (taken from [33])

The linear transfer function behaves neutrally.

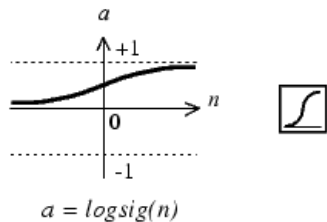


Figure 22: Log-Sigmoid transfer function (taken from [33])

The log-sigmoid transfer function takes any input between plus and minus infinity and maps it to the output range between 0 and 1.

These individual neurons are arranged into distinct layers and interconnected in order to construct a network. Apart from the input and output layer a network might have a theoretically arbitrary number of hidden layers. In real-life applications the number of hidden layers rarely exceeds five, because the network would become too large and cumbersome to train. Figure 23 illustrates the structure of a simple neural network.

The weights of the individual connections between the neurons are randomly initialised. The collection of all connection weights of a given network is called the weight matrix W . During learning the weight matrix is constantly updated, so that the output of the network represents the desired output as exactly as possible. If we imagine the set of possible input values to be I and the set of possible output values to be O , the network is supposed to approximate a function $f : I \rightarrow O$ with as little error as possible. For this purpose an algorithm called "back-propagation algorithm" is used. This algorithm

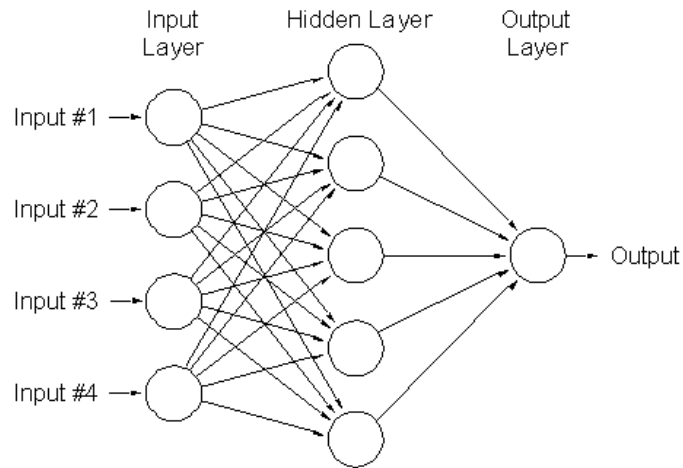


Figure 23: Simplified diagram of a small artificial neural network (taken from <http://smig.usgs.gov>)

adapts the weight matrix of the network depending on the mean square error that the network produces for a given run of the training set. By traversing from output toward input through the neural network the back-propagation algorithm calculates how much "blame" a single neuron takes for the produced error and adjusts the weights for this neuron accordingly. In more mathematical detail, the back-propagation algorithm is used to calculate the gradient of the error of the network with respect to the network's adjustable weights. Subsequently this gradient is used in a stochastic gradient descent algorithm, which will modify the weight matrix so that the networks error is minimised. A more detailed explanation of this algorithm can be found on [33].

As mentioned above, the purpose of network training is to enable the network to approximate the correct input to output mapping function as precise as possible. There are however two problems that system designers face.

1. Most data that is presented to any neural network based classifier for the purpose of training will not be perfect. It will on the contrary contain a lot of imperfections and noise. Very clear and noise-free data for training is however the most essential prerequisite for a successful teaching of the network and an accurate approximation of the

related input-output-mapping function. If the presented training data is of very high quality the approximation performance will be excellent. Real-life situations however are usually not that user-friendly. Most datasets are of poor quality and contain a substantial quantity of noise. These errors can cause the training algorithm to produce a network that exhibits a bad overall performance.

2. The amount of data presented to the network for training needs to be sufficiently large. Generally the bigger the network the more data is needed for training. Similarly the more complex the problem, the bigger the network needed to classify the input data correctly. If the network is not presented with enough data, the training algorithm might reach a local minimum in the error function and subsequently display a bad overall classification performance. It is also possible that when presented with not enough training data the algorithm "overfits", which means it does not generalise rules from the given input, but instead "learns the input data by heart". This will also lead to a subsequent bad classification performance. Unfortunately networks that are used for classification of real world data are often so large that enough data for training is simply not available.

These two problems constitute the main difficulties when attempting to build a well-performing classifier for a specific problem and they are to blame for most of the imperfections that neural network based classifiers exhibit. The persistence of those imperfections gives rise to the ideas described in the next section of this document (5.2).

5.2 Using multiple classifiers

The previous section of this document (5.1) noted that there are limitations to the use of neural networks for the classification of data. In fact it has been experimentally perceived that it is often impossible to build a perfect neural network based classifier for a certain problem. The best networks available still display a certain error rate in approximating the input-output mapping function.

It seems obvious that when different neural networks are used to approximate

the same function, the approximations will differ as well. This is due to the fact that the individual networks will make different interpretations of the training data and of the noise it contains. Due to this variety in the approximation it is likely that data that is classified falsely by one classifier, will be classified correctly by another. This has given rise to the idea of using multiple classifiers for a single classification task. The basic concept of these so-called Multi Classifier Systems (MCS) is that shortcomings of one classifier will be compensated by several others, so that the combined classification result will be more accurate than that of a single classifier by itself. The application of Multi Classifier Systems for DNA analysis has been exhaustively researched in [34].

Such a multi-classifier system will be used in this project to detect promoters in Drosophila DNA. There are several issues related to such systems that need to be discussed in order to provide an understanding of how these systems work in detail. The subsequent sections of this document will deliver this discussion.

5.2.1 MCS design and topology

There are two different basic designs that can be applied when using several classifiers for evaluating the same piece of data. They describe the way in which the constituent parts of the system are put together in order to obtain a single result. The first architecture is illustrated in figure 24.

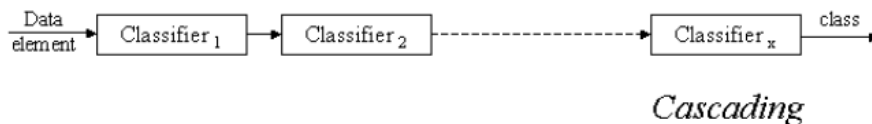


Figure 24: Cascading MCS architecture (taken from [35])

The data element that is to be examined is fed to the first classifier. The result of this system is fed to the next classifier and so forth until the data has passed through all systems in sequence. The result of the last classifier in this sequence is regarded as the final result of the overall system. A system of such a design is called a cascading Multi Classifier System.

In contrast to that stands the architecture that is illustrated in figure 25.

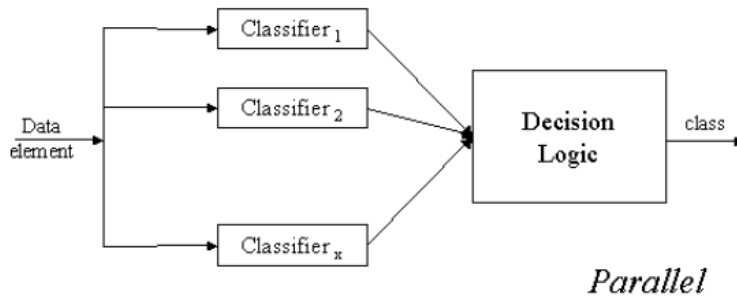


Figure 25: Parallel MCS architecture (taken from [35])

The data element that is to be examined is handed to all constituent systems of the MCS at the same time. The results of the classifiers make up the input to a further important part of the MCS. This part is the so-called decision logic, which a system that utilises the cascading architecture does not have. This decision logic takes all the results from the individual classifiers and combines them into the final result of the overall system. The design of this result combiner is a major part of the design of the system as a whole and it is further discussed in section 5.2.3. For this project only MCSs with a parallel architecture are deployed.

5.2.2 Performance of MCSs

Due to the limitations mentioned above any neural network based classifier will always be only an approximation of the function that transfers elements from the input space into elements of the output space. Thus no classifier will be 100%ly accurate. The use of multiple models to overcome this problem is self-evident. The situation is comparable to a decision that is reached by a corporate board. No individual member of the board will always make correct decisions, but the consensus that the board reaches will in general be more correct than an individual opinion. This analogy will be further exhausted in the next chapter on various methods how to combine the results of the individual classifiers (sections 5.2.3).

There are two conditions that have to be met in order for the application of

multiple classifiers to be successful. These conditions have been stated and are explained in detail in [35].

1. The performance of all classifiers individually needs to exceed 50%. This was stated by [36] and it means that all systems are more often than not correct in their recognition. If false recognitions are in the majority the ensemble of several classifiers cannot correct the mistakes.
2. The individual classifiers need to be sufficiently different from each other. It is obvious that if everybody makes the same mistakes the decision of the group cannot be suddenly correct. For this purpose a measure of diversity between classifiers has been introduced, which is discussed in a separate section of this document (5.2.4).

The idea of how multiple classifiers can cover a larger portion of the input space than a single classifier is graphically illustrated in figure 26.

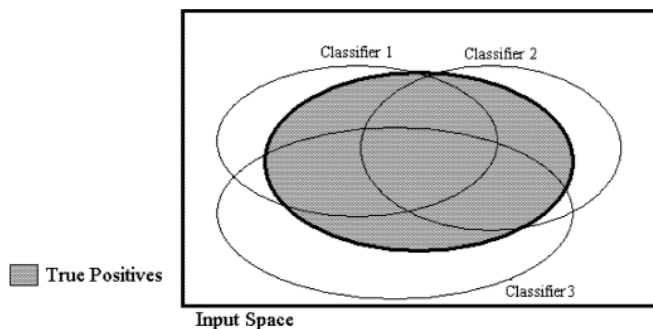


Figure 26: Application of multiple classifiers (taken from [1])

The grey shaded region in this figure represents the true positives within a given output space. The other three ellipses represent the true positive spaces as they are given by three individual classifiers. As can be seen each of the single recognition systems produces a certain percentage of false positives. It can also be seen that most - yet not all - of the true positives of the output space are classified correctly by at least two of the three classifiers. Thus more correct classifications are made using three systems together instead of any one of the three systems alone.

The meaning of above mentioned conditions for the success of a Multi Classifier System can also be identified in figure 26. For once each of the three classifiers covers more than half of the grey shaded area, which means that they all perform better than 50%. Would this not be the case there would be no positive effect of using several recognition systems. On the other hand do all three classifiers cover different areas of the output space. It is very obvious that if that would lay very close to or even exactly on top of each other that the use of multiple classifiers would make no difference.

5.2.3 Combining results

The results of the individual classifiers need to be combined in order to deliver a single result. There are a number of ways to do this. The most frequently used ones are explained below. All combination methods can be categorised into one of two categories, confidence based and non-confidence based. The difference between those categories is that the results delivered by the single classifiers are multiplied by a weight, which represents the level of confidence that is placed in the model in question. If there is no such weight the combination method is categorised as non-confidence based. This distinction was introduced by [37].

The most popular combination methods are:

- **Maximum** - The MCS will follow in its prediction the single classifier that is most certain to be correct. This means that the prediction result of the classifier with the highest score is chosen. The MCS obeys equation 4

$$MCS_{MAX}(x) = \max(f_{C_1}(x)w_1, \dots, f_{C_R}(x)w_R) \quad (4)$$

where $MCS_{MAX}(x)$ is the prediction result of the Multi Classifier System using the maximum combination method, $f_C(x)$ is the result delivered by a single classifier and R is the number of classifiers in the system. w denotes the weight of confidence which might, in the simplest form of this method, always be equal to 1.

- **Summation** - The summation method uses the sum of all scores achieved by the single classifiers. The result is obtained using equation 5

$$MCS_{SUM}(x) = \sum_{i=1}^R f_{C_i}(x)w_i \quad (5)$$

where $MCS_{SUM}(x)$ is the prediction result of the Multi Classifier System using the summation combination method, R is the number of classifiers and f_{C_i} is the score delivered by classifier i . The confidence weight is here denoted as w_i . In the most basic version of this method there are no weights, which means w_i is always equal to 1.

- **Average** - The method uses an approach similar to the summation method. However the final result of the MCS is the result of the summation method divided by the number of classifiers. Equation 6 gives a formal definition of this method.

$$MCS_{AVG}(x) = \frac{\sum_{i=1}^R f_{C_i}(x)w_i}{R} \quad (6)$$

where $MCS_{AVG}(x)$ is the prediction result of the Multi Classifier System using the average combination method. Other variables are used identical to equation 5.

- **Linear Average Predictor** - As a variation of the two basic methods average and summation described above this method is often found in literature. The methods is defined by equation 7 for a given class j

$$MCS_{LAP}^j(x) = \sum_{i=1}^R f_{C_i}^j(x)a_i \quad (7)$$

where $MCS_{LAP}(x)$ is the prediction result of the Multi Classifier System using the linear average prediction method, R denotes the number

of classifiers and $f_{C_i}^j$ denotes the probability that input i belongs to class j . Here the confidence weight is of such a form that

$$\sum_{i=1}^R a_i = 1 \quad (8)$$

- **Majority voting** - The combined result of the MCS is obtained following the opinion of the majority of the classifiers. For this combination paradigm each system outputs an array of numbers between 0 and 1 giving its certainty that an input belongs to a certain class. The result obtained using majority voting is then defined by equation 9

$$MCS_{MV}(x) = \underset{j=1}{\operatorname{argmax}} \frac{1}{R} \sum_{i=1}^R w_i f_{C_i}^j(x) \quad (9)$$

where $MCS_{MV}(x)$ is the prediction result of the Multi Classifier System using the majority voting combination method, Q denotes the number of classes, R denotes the number of classifiers in the system and $f_{C_i}^j(x)$ denotes the certainty of classifier i that input x belongs to class j . Again the classifiers might or might not have a weight w_i associated with them.

Apart from the categorisation into confidence based and non-confidence based combination methods [37] also offers a further more refined breakdown according to the mathematical background of the combination function. The classes that [37] uses are:

- **Linear combination methods** - These methods comprise all methods that use linear approaches to combine the results of several classifiers. From the above list maximum and summation are linear combination methods.
- **Non-linear combination methods** - In this group all classifiers are pooled that use non-linear approaches for combination. These methods usually work with classifiers that rank the possible classes according to the likelihood of the input sequence being a member of a certain class. Majority voting is an example taken from above list.

- **Statistical combination methods** - These methods use statistical approaches such as Bayesian combination to combine the results of multiple systems.
- **Combination methods using computational intelligence** - These systems rely on the usage of further artificial intelligence to combine the results. An example would be to employ a neural network. Its input would be the outputs of the individual classifiers and its output would be the overall classification result. Other AI-based systems such as genetic algorithms can also be utilised.

A more complete list of various combination methods and a more detailed discussion about their properties can be found in [35]. There is no general rule which combination method should be used for which system. This strongly depends on each Multi Classifier System itself and the properties of the classifiers it consists of. To gain the optimal performance of an MCS it is necessary to experimentally determine the optimal combination method. How this was done for the present system and what combination paradigm was eventually chosen will be described in the following section (section 6) of this document.

5.2.4 Diversity between classifiers

As was stated above, one of the vital prerequisites for a Multi Classifier System to be successful is that the individual classifiers need to disagree with each other. This prerequisite was discussed by [38]. [38] further states that not only disagreement, but a sufficient level of disagreement between the classifiers is necessary. If the approximations of the transfer function, which transfers elements from the input space into element of the output space, produced by different classifiers is too similar the performance gain of using a combination of those classifiers will be minimal to non-existent.

Since a sufficient disagreement between individual classifiers is such an important criteria for the selection of the classifiers that are used in any ensemble a measure of their diversity is being discussed. While it is straight-forward to measure the diversity between two classifiers, the problem becomes much more complex when three or more classifiers are supposed to be compared.

Finding the best possible measure for diversity among classifiers has hence been a subject of intensive research in the past years. Actually the *Journal of Information Fusion* published a special edition in 2005 which is titled *Diversity in multiple classifier systems*([39]). The topic is exhaustively discussed in this special edition. For a list of the ten most popular diversity measures currently under discussion see [40]. More detailed information on the subject of diversity can also be found in section 3 of [35].

There are several different sources of diversity for neural network based classifiers. As a matter of fact no two neural networks will deliver identical result for all input data, even if the networks have been trained with the same training set. This is due to the fact that the set of weights in neural networks usually gets initialised at random before training. Depending on the training goal¹² the disagreement that is derived from random initialisation might however not be very significant. Another obvious source of diversity is the configuration of the networks, such as the number of hidden layers and the number of neurons in each layer. Yet another source are different ways of encoding the input data. More information on encoding input data is given in the next section 5.2.5. Finally the diversity of the classifiers can also be achieved by using different training sets to train them. This is of course under the condition that the initial training set was big enough, so that splitting it up will leave the subsets with enough examples to train the networks properly.

How it will be ensured that the classifiers used in the present system are sufficiently diverse from each other is further discussed in the next section (section 6) of this document.

5.2.5 Encoding of input data

Obviously the input neurons of a neural network cannot accept anything else but a number, be it a natural or a floating point number. DNA sequences, that serve as the inputs to a neural network in the present scenario, however do not consist of numbers, but instead of four distinct nucleotides, *A*, *C*, *G* and *T*. In order for the network to be able to accept input of such kind the four nucleotides need to be represented by a certain number or a sequence of

¹²The acceptable mean square error

numbers. The way in which this representation is made is called the encoding scheme.

Table 7 illustrates four different ways of performing this encoding. Using different encoding schemes for different classifiers of the MCS is a primary way of obtaining a suitable level of diversity between the constituent classifiers of the overall system.

| Scheme | A | T | C | G |
|--------|------|------|------|------|
| Enc1 | -2 | -1 | 1 | 2 |
| Enc2 | -1 | -1 | 1 | 1 |
| Enc3 | 00 | 01 | 11 | 10 |
| Enc4 | 1000 | 0100 | 0010 | 0001 |

Table 7: Different schemes used to encode DNA for the input to a neural network

Encoding schemes *Enc1* and *Enc2* were first used by [1]. *Enc3* and *Enc4* were introduced by [41] and [42] respectively. There are two things to remark about this. Firstly, when several numbers are used to encode a single nucleotide, as it is done in the binary encoding schemes *Enc3* and *Enc4* the number of input neurons required to accept the sequence in question will be higher than the number of nucleotides in the sequence. The number of input neurons will be the length of the input sequence multiplied by the number of digits used to encode a single base. Secondly, the order in which nucleotides are encoded in above schemes is not chosen arbitrarily. It is not a coincidence that the encoding of *A* and *T* as well as the encoding of *C* and *G* is often quite close to each other (or even equal as in *Enc2*), whereas the numerical distinction between those two groups is quite sharp. This way a chemical property of the bases is reflected that was discussed in section 2, namely the fact that *A* and *T* as well as *G* and *C* are equal in the number of hydrogen bonds they can establish and thus always pair with each other in the DNA double helix. It is expected that reflecting this property in the encoding scheme proves beneficial for the recognition performance. [41] however comes to the conclusion that *Enc4* performs best. In *Enc4* all nucleotide representations are of equal distance to each other.

5.2.6 Reducing input dimensions

Toward the end of section 5.1 it was already mentioned that the size of the dataset that is used to train a neural network is an important factor determining the performance of this network. The training set must not be too small because this would mean that the network will not be able to generalise over the data. Instead the network will only learn all the presented cases. The effect is that the network performs very well on the training set, but the performance on previously unseen data of the same type is unsatisfactory. This behaviour is called "over-fitting". This can be avoided by training the network with a training set that is large enough. The required size for the training set to avoid over-fitting is directly proportional to the size of the neural network itself. There is a "golden rule" for neural network designers stating that to safely avoid over-fitting the training set should contain at least 10 times as many elements as the neural network has internal connections.

Therein lies a problem. The task of promoter recognition has a level of complexity that makes it necessary for the network to possess at least two hidden layers. The more neurons and layers a network possesses the more difficult the classification problems that the network can deal with. A smaller number of layers has a negative impact on the network's ability to constitute a valid model for promoter recognition due to the inherent intricacy of this problem. Furthermore the number of output neurons for any such network is always going to 1, whereas the number of input neurons is determined by the length of the sequence that is to be examined. A typical sequence length that is found in ready-made training sets for promoters in *Drosophila* is 300 bases.

A network that is designed to accept a sequence of this length will be very large, even if the 2 hidden layers are reasonably small. It will have a number of internal connections that lies far over 1000. The problem is aggravated greatly if more than one number is used to encode a single nucleotide of the sequence. Apart from the fact that such long sequences would result in a network that takes a very long time to train, we do more importantly not have enough data to train this network in the first place. With the species of *Drosophila Melanogaster* having about 13,000 genes and with that about 13,000 promoters there is a natural upper limit to the size of the training set anyway. The training set that is available for promoters in this species is

significantly smaller than the natural limit. It contains only less than 2000 promoters and about twice as many non-promoter sequences. According to the "golden rule" cited above this limits the neural networks trainable with this data to a size of fewer than 400 internal connections.

A solution to this problem was offered by [43]. The paper introduces seven functions that are applied either to single sequences or to the whole training set. These functions calculate a certain property of a given input sequence (e.g. the proportion of G's and C's in the string). [43] suggests to use these properties instead of the original DNA sequences as input to neural network based classifiers. This has the advantage that the dimensionality of the input is reduced drastically. With that the size of the resulting network is also reduced drastically. That means that on the one hand we can train the network in a more reasonable amount of time. On the other hand it means that even with the present training set we are able to design a network that actually generalises on the data and does not learn any of the cases.

A more detailed discussion on the seven property functions can be found in [43]. They are also described in depth in Romesh Ranawana's DPhil thesis, which is, as I type, not yet published.

5.3 Incorporating evolutionary history into the MCS

The methodology described in the previous section (section 5.2) is used here. However this methodology works only with the DNA sequence from a single species. Predictions are made on the basis of this DNA sequence given to the system and nothing else. As was explained in section 2.4 the data that is used for this project consists of the sequenced genomes of 12 different closely related species. This constitutes additional information that can be exploited for the recognition process.

The dataset does not only provide us with 12 complete genomes of high quality, it also delivers a complete alignment of those 12 genomes to each other. This means that the sequences are in such an order that for each nucleotide in each genome the corresponding counterpart in every one of the other genomes can be found. This is achieved by formatting the strings in such a way that corresponding bases are written in the same column. A short

example of this notation is illustrated in figure 27.

Each line represents the genome of a different species of *Drosophila*, the first line the genome of *D.Melanogaster*, the second line the genome of *D.Simulans* etc. The dashes indicate that at this position in the sequence a nucleotide was inserted or deleted respectively, depending from what perspective the process is being looked at.

```
GAAATCAACAAACAAATAGCTGCGCAACAAATGGAAATTGCCGGCCTGCTTAAAGTGGAGCCACAGTAAAGTGTGACCTC
GAAATCAACAAACAAATAGCCGCGCAGCAAAATGGAAATTGCCGGCCTGCTTAAAGTGGAGCCACAGTAAAGTGTGATCCC
GAAATCAACAAACAAATAGCCGCGCAGCAAAATGGAAATTGCCGGCCTGCTTAAAGTGGAGCCACAGTAAAGTGTGACCTC
GAAATCAACAAACAAATAGCCGCGCAGCAAAATGGAAATTGCCGGCCTGCTTAAAGTGGAGCCACAGTAAAGTGTGAGCAC
GAAATCAACAAACAAATAGCCGCGCAGCAAAATGGAAATTGCCGGCCTGCTTAAAGTGGAGCCACAGTAAAGTGTGACCTC
GAAATAAACAAACAAATCGCAGCGCAGCAAAATGGAAATCGCAGGCTTACTCAAAGTAGAGCCGAAAGTAAAGTTCGAATTGA
GAAATAAACAAACAGATTGCCGCGCAGCAAAATGGAAATTGCCGACTTGCTCAATGTGGAACCTTCTGTAA-----
GAAATAAACAAACAGATTGCCGCGCAGCAAAATGGAAATTGCCGACTTGCTCAATGTGGAACCTTCTGTAA-----
GAAATCAATAAGCAAATTTGCTGCCAGCAAAAGAAATTTGCTGGATTGCTCAATGTGGAACCTTCTGTAGGT-----
GAAATAAATAAACAAATCGCTGCGCAGCGCATGGAAATCGCTGGTCTTCTAAATGTAGAGCCCTTCGGTGAGTG-----
GAAATTAATAGCAAATCGCAGCGCAGCGCATGGAAATAGCCGGCCTCTTAAATGTAGAGCCATCTGTGAGTGC---CCCG
GAAATTAATAAGCAAATCGCTGCACACGCATGGAAATTGCCGGTCTGTAAATATGGAACCATCGGTAAGTGT---CATC
```

Figure 27: Excerpt from the sequence alignment

Thus we do not only know what nucleotide resides in a genome at a certain position, but we also know what became of this exact nucleotide in the course of evolution. In addition to this we also know how much time has passed since any two of the 12 species diverged from each other. This information can be taken from the phylogenetic tree of the 12 species which was shown above in figure 4 in section 2.4. With all this additional information it is then possible to examine the example sequences from the training set for characteristics that further distinguish between promoter and non-promoter regions. The following list names a few ideas of what examinations can be conducted in order to find further evolution related distinguishing marks in the training set. The list is incomplete. For the purpose of the present exemplification it is nevertheless sufficient.

- **Total number of insertions/deletions** - It is counted how many insertions or deletions occurred for a given example sequence.
- **Total number of events** - In a certain sequence it is determined how many times evolution has changed each one of the nucleotides in the other species. This change is called a single nucleotide polymorphism (SNP). The number of SNPs is summed up over the entire length of the example sequence.

- **Transition/Transversion ratio** - As was explained in section 2 nucleotides can be divided into two groups, purines and pyrimidines. A transition is a SNP within one group, a transversion alters the group of the nucleotide. For each example sequence the ratio between transitions and transversions can be examined.
- **Mutation rate** - Since it is known what the evolutionary distance¹³ for any set of two species from the dataset is known, it can be determined what the overall mutation rate for a given example sequence is, i.e. the number of changes between the species of reference and one of the other species over the evolutionary distance of those two species, summed over all 11 possible pairs.
- **Mutation rate matrix** - Not only the overall mutation rate can be looked at, it can also be determined with what rates each nucleotide changed into any other nucleotide for each given example sequence. The result is a four by four matrix.

Each of these numbers can subsequently be used in the recognition process. One or possibly all of the neural networks used in the Multi Classifier System are going to have an additional input neuron, which is fed with the information derived from the evolutionary history of the region to be processed. Since this means more information about the sequence this also means a more accurate precision.

There is however one drawback. The recognition rate of the system can only be improved in this way if the evolutionary characteristics of a sequence actually convey information about whether this sequence is a promoter or not. The numbers would have to be sufficiently different between promoters and non-promoters in more cases than not in order to aid the recognition process. If e.g. the mutation rate is approximately equal for all positive and negative examples, then feeding this rate to a neural network for the purpose of classification would be pointless.

If one argues from a biological point of view it is however reasonable to assume that the behaviour of promoter and non-promoter regions with respect to evolution will be different. This is because promoters play a vital roll in the

¹³Evolutionary distance: the amount of time that has passed since two species diverged

transcription of a gene. Details about the function of promoters were given in section 2.3. If a promoter mutates so much that it becomes no longer functional, the whole gene will no longer get transcribed. This means that the whole functionality of this gene is lost for the organism. Therefore the organism is much more likely to die out after a mutation in a promoter than in most of the other regions in its genome. Hence it is hypothesised that the evolutionary activity is lower in promoter regions than it is in non-promoter regions. This lesser extent of activity will help the neural networks in their classification decisions.

With these things in mind the work that is described in the following section (section 6) of this document has been conducted.

6 Account of the work

The reading of this dissertation up to this point should have made very clear that this project has a strong research orientation. For this reason the implementation of the project has not been a straight effort. It was my aim to be able to utilise information from evolutionary history for the benefit of promoter recognition. The method used for this is computational intelligence based on multiple neural networks. This should also have become clear at this point.

To the best of my knowledge nobody has attempted this before. This is probably due to the relative novelty of suitable datasets for this kind of studies. Naturally I did not know how to achieve my aims when I started experimenting with Multi Classifier Systems and the *Drosophila* dataset. Hence the progression of this project was not like those of other projects where the path of action is obvious and there is no uncertainty about the outcome of various stages. In this project on the other hand the outcome of experiments was not clear. The process was, as it is typical for research work, characterised by hypothesising about the possible results of the work and trying to verify these assumptions. Based on the outcome of single experiments it needed to be reasoned about an explanation for the results before further steps could be undertaken. For this reason the project work is not as cohesive and not as straightforward as it is in a project that resides

entirely on known territory, e.g. a well-defined software project according to a customer's specifications.

The following section of the chapter will describe the path of experiments that was taken and present and interpret the results obtained.

For the implementation, training and simulation of neural networks the Neural Network toolbox that is part of the MATLAB 7 distribution has been used. Additional tools for data processing have been programmed in Java. The most relevant MATLAB and Java code is cited in the appendix to this document. Section 7 of this document will deliver a detailed explanation of this code.

6.1 Training an MCS with data pre-processed with NNPP

The first problem that one encounters when setting up a neural network based classifier is to find appropriate data to train and test the network. This data needs to be available in a sufficient amount and it needs to contain positive and negative sample. Both of them need to be numerous enough to allow proper training of the network.

Unfortunately the dataset that is supposed to be used for this project as described in section 2.4 is not commented in any way. It consists solely of the raw *Drosophila* DNA strings aligned with each other. A small excerpt of this data was shown above in figure 27. The data is however very extensive. The genome of a single species of *Drosophila* consists of approximately 132 million nucleotides and there are 12 different species in the set. This does not account for insertions and deletions in the alignment. The fact that this data is not commented in any way means that it is unknown which regions in the genome have which functionality and with that it is unknown where in the DNA string the promoters are. To know this is however necessary for two reasons. On the one hand the neural networks need a training set for which they are also given the expected results to be delivered for the presented samples. On the other hand the networks need to be evaluated and that is only possible if there is a ground to compare the performance of the network against.

All of this means that the dataset cannot be used to set up neural network based classifiers without being pre-processed. For doing this the system NNPP was chosen. This system was discussed in detail in section 3.1 of this document. This system was chosen, because according to its designers it has a false-positive rate of 0.0%, when the threshold value is chosen high enough. Accordingly it has a relatively low false-negative rate when the threshold value is chosen to be low. This means it is absolutely certain that a reported promoter, which has a high enough score is in fact a promoter and not a false recognition. Equally it is quite unlikely that the system misses a promoter when the threshold is set to a low value. False-positives and false-negatives in the training and test set must be avoided otherwise the performance of the newly trained neural network is flawed from the very beginning.

NNPP was used in the following way to create training and test sets for neural networks. The genome of *Drosophila Melanogaster* was part by part fed to the web interface of NNPP [7]. For these runs the threshold value was set to 0.95. Regions that were predicted to be promoters with a score above this threshold were extracted and added as positives to the test and training set. After that the same genome was fed to the NNPP interface again. This time the threshold value was set to 0.7. Regions that were not reported to be promoters with this setting were added to test and training set as negatives. Using this technique a training set was created that contained 2000 elements, each 1000 *Drosophila Melanogaster* core promoter regions and 1000 randomly chosen regions that are not core promoters of the same species. A test set was also created. It contained 500 promoters and 500 non-promoters. The length of all these positive and negative sequences was 50 bases. The transcriptional start side is expected to be 10 bases from the 3' end of the sequence.

This training set was used to set up 4 different neural networks. Each of those four networks uses one of the encoding schemes that were illustrated in table 7. The networks are named netA, netB, netC and netD. As was put forward in [1] the networks need to have at least two hidden layers due to the complexity of the problem.

The configuration that was chosen for all four network can be seen in table 8 overleaf.

| Network | Encoding scheme | Input neurons | Neurons hidden layer 1 | Neurons hidden layer 2 | Output neurons |
|---------|-----------------|---------------|------------------------|------------------------|----------------|
| netA | Enc1 | 50 | 50 | 25 | 1 |
| netB | Enc2 | 50 | 50 | 25 | 1 |
| netC | Enc3 | 100 | 100 | 50 | 1 |
| netD | Enc4 | 200 | 100 | 50 | 1 |

Table 8: Configurations of four neural network based classifiers

All neurons in both hidden layers have Log-sigmoid transfer functions. The output neuron has a purely linear transfer function. The relations in size between the individual layer of the networks have been suggested in [35]. They prove beneficial for the recognition process. It has been experimentally shown that a further increase in the number of neurons in each layer would not improve the performance of the classifier. It would however have a negative impact on the time it takes to train the networks. The additional size would also further increase the risk that the networks overfit on the data.

The overall structure of this system is illustrated in figure 28.

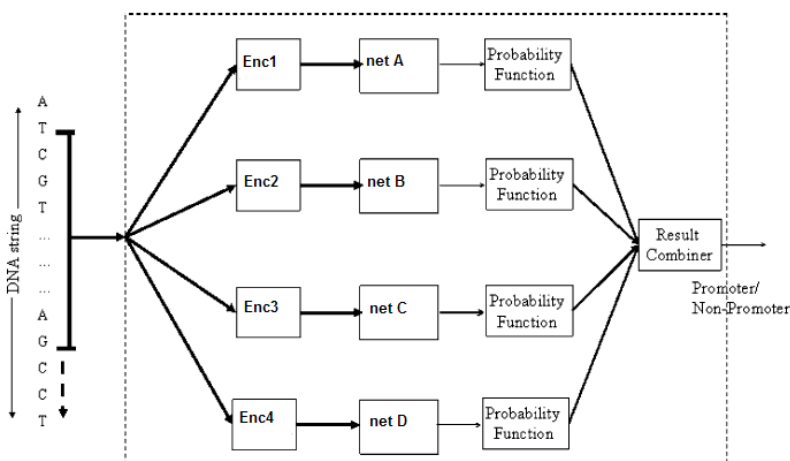


Figure 28: Structure of the Multi-Classifier System (image modified from [1])

The figure also shows the presence of a so-called probability function. This is a function that transfers the output of the network into an assertion of

the form promoter/non-promoter. This function can have various forms depending on the kind of output that the networks produce. In the present system the networks were trained to produce the value 1 when faced with a promoter and the value -1 when faced with a non-promoter sequence. In this case the probability function is a simple hard limiter function that is governed by equation 10.

$$hl(x, l) = \begin{cases} 1 & \text{if } x > l \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

The function hl accepts an argument and a limit value. The argument value is set to 1 if it is greater than the limit l and set to 0 if it is less than or equal to the limit. In the case mentioned above, where networks are trained to produce 1 for promoters and -1 for non-promoters this limit is selected to be 0. The results of the hl function is considered the final assertion, 0 for non-promoter and 1 for promoters.

Depending on the method of combination that is used in the system, the property function can be located either before or after the result combiner. More exactly it depends on the type of input that the chosen result combiner requires. One possibility is that the result combiner works with levels of certainties of the individual classifiers. In this case the result combiner would accept the outputs of the neural networks directly and produce a level of certainty itself. This level of certainty is the fed to a property function of the form shown above. An example of this is the combining method "Average" that was defined in equation 7. The other option is that the result combiner processes results that are already of the form promoter/non-promoter. In this case the property function is applied before the result combiner. This situation is shown in figure 28. An example of such a combination method is "Majority voting" which was defined in equation 9.

All four neural networks were trained using the complete training set. Training was said to be complete when the mean square error of the network fell below 0.00001 of the training data. The algorithm that was used for the training of the networks was resilient backpropagation. This algorithm allows timely training of the networks, because it is especially designed and adapted to work well with multilayered networks with sigmoid transfer functions. The general idea of backpropagation training algorithms was introduced in section

5.1 of this report and more detailed information can be found in [33].

After the training of the networks was completed they were tested on the previously unseen data in the test set. The results can be found in table 9.

| Network | Sn | Sp | Total |
|---------|-------------|-------------|-------------|
| net1 | 478 (95.6%) | 396 (79.2%) | 874 (87.4%) |
| net2 | 472 (94.4%) | 376 (75.2%) | 848 (84.8%) |
| net3 | 463 (92.7%) | 444 (88.9%) | 907 (90.7%) |
| net4 | 482 (96.5%) | 395 (79.1%) | 877 (87.7%) |

Table 9: Results of networks A-D on test set

This table shows that the results of the networks on the training set can still be improved. In the following three different methods for combining the results of those four networks have been examined. These three methods were Maximum, Average and Majority voting. The results that were obtained on the test set by combining networks A-D using these methods can be seen in table 10.

| Combination method | Sn | Sp | total |
|--------------------|-------------|-------------|-------------|
| Maximum | 485 (97.0%) | 414 (82.8%) | 899 (89.9%) |
| Average | 482 (96.4%) | 415 (83.0%) | 897 (89.7%) |
| Majority voting | 480 (96.0%) | 433 (86.6%) | 913 (91.3%) |

Table 10: Combined results on test set with three different combination methods

MATLAB code for the combination methods "Maximum" and "Majority voting" can be found in the appendix C.2 and C.1. For the combination method "Average" no code was necessary. It only consists of simple arithmetic that can be handled in MATLAB manually.

Since the performance using the combination method "Majority voting" is best, this method is chosen for the final system. As can be seen from the two tables above the performance of the system is better when the results of several classifiers are combined. This experiment confirms the assumption that Multi Classifier Systems outperform single classifiers.

6.2 Observations regarding evolution in aligned data

6.2.1 Relationship of recognition results in related species

Above table demonstrates that the system works far from perfect. As a matter of fact almost 1 out of 10 sequences that are given to the Multi Classifier System for the purpose of promoter detection will be classified falsely. It must be said that this system still performs remarkably well in comparison with other systems for promoter recognition (see section 3 for details). However the overall error rate is still almost 10%. This project comprises further experiments how this error rate could be reduced.

As was mentioned in section 5.3 the data that was used for this project did not only contain the genome of a single organism, but instead a group of 12 genomes from species that are closely related. The same section lists a number of aspects that could be examined when trying to reduce the error rate of the MCS using information that is derived from the 12 genomes' evolutionary history. The remainder of this section describes the attempts that were made to improve the system that was described in the previous section by using this information.

Since the alignment of the data is given it is relatively easy to use the existing training set to obtain information about the corresponding positions. Figure 27 showed an excerpt from the aligned data. In a first step the question will be answered how the MCS that was introduced in the previous section will perform in the regions of the 11 other species that are aligned with the data in the test and training set. For this purpose the sequences in the corresponding locations to the positive and negative examples have been extracted from the training and test set and were exercised through the MCS.

Table 11 reports the results.

| Species | Sn | Sp | Evolutionary Distance (in million years) |
|------------------|-----|-----|---|
| D. Melanogaster | 96% | 87% | 0 |
| D. Simulans | 79% | 88% | 3.2 |
| D. Sechellia | 79% | 89% | 3.2 |
| D. Yakuba | 77% | 90% | 9.4 |
| D. Erecta | 70% | 92% | 9.4 |
| D. Ananassae | 49% | 86% | 13.3 |
| D. Pseudoobscura | 45% | 91% | 25.9 |
| D. Persimilis | 44% | 91% | 25.9 |
| D. Willistoni | 55% | 87% | 35.4 |
| D. Mojavensis | 51% | 87% | 39.4 |
| D. Virilis | 50% | 89% | 39.4 |
| D. Grimshawi | 51% | 86% | 39.4 |

Table 11: Results of the MCS on the aligned training set

The fact that the recognition rate on negative samples does not significantly vary with increasing evolutionary distance between the species had to be expected. It is easily explained by remembering that negative samples in the training set are randomly selected sequences of DNA from *Drosophila Melanogaster* that were not classified as promoters by NNPP. It is very unlikely that a lot of those random sequences have evolved into promoters in any of the other species.

The behaviour of the recognition rate on the positive samples is however more interesting. The performance on *D. Melanogaster* is 96%. This value can also be found in table 10. In fact line 3 of table 10 and line 1 of table 11 reflect identical experiments. The number of promoters that are recognised in corresponding positions to true promoters in *D. Melanogaster* decrease with growing evolutionary distance between the species. In the closest related species *D. Simulans* and *D. Sechellia* only about 10% of the promoters in *D. Melanogaster* lost their promoter characteristics through mutation. In the most distant species *D. Grimshawi* only about half of the promoters from *D. Melanogaster* remain intact. This observation can be explained by the fact that mutations are a time dependent process. Single nucleotide

polymorphisms (SNP) happen evenly distributed over the whole genome and with a relatively fixed frequency. So when there is less time between the divergence of two regions, there are less mutations and the regions display more similarity.

Another observation can be made from table 11. This is that only the four most closely related species carry relevant information for the recognition in *D. Melanogaster*. These species are *D. Simulans*, *D. Sechellia*, *D. Yakuba* and *D. Erecta*. In those species a sequence that was aligned with a promoter region in *D. Melanogaster* is more often than not a promoter as well. Between 70% and 80% of those strings are still classified as promoters by above system. In all other species the recognition rate on those sequences is around 50%. This yields no information on the same location in *D. Melanogaster*, because it is equally likely that a sequence is a promoter or not.

Those two observations can be used to update above system. Some false-negative recognitions can be avoided by making use of the recognition results in the four most closely related species. Every time a sequence is classified as non-promoter, the aligned sequences in those four species were run through the MCS as well. Depending on the outcome of those classifications the initial result on the *D. Melanogaster* sequence was altered. Experiments with the test set have shown that this alteration of the initial result is only appropriate when all four related species agree that the aligned sequence in their genome is in fact a promoter. In cases where there is less agreement between the four species the alteration of the initial recognition result in *D. Melanogaster* was more often than not incorrect. Using this method it was possible to increase the sensitivity of above system by 1%. This constitutes an intermediate result and a first successful improvement of a classification system using a genome's evolutionary history.

6.2.2 Examination of more detailed parameters

As was stated in section 5.3 it is assumed that there is more information than this available from the 12 aligned genomes. The remained of this chapter describes experiments how to obtain and utilise this information.

A first and very simple thing to examine is the mutations per time interval of

a certain sequence. Because of their biological significance it is assumed that promoters mutate less than other parts of the genome. The reasoning behind this assumption is that if a promoter loses its functionality through mutation the whole gene that this promoter belongs to is lost for the organism because the gene's TSS can no longer be located. Compared with other sequences of equal length this impact is quite severe for the organism and it is much more likely that the organism will perish because of a mutation in a promoter.

A reduced test set has been processed to obtain the mutation rate for promoter and non-promoter sequences. The set contained 150 positive and 150 negative samples from *D. Melanogaster* and the 11 sequences aligned with those samples. The mutation rate was derived following equation 11.

$$mrate(x) = \frac{1}{S} * \left(\sum_{i=1}^S x_i * \frac{1}{d_i} \right) \quad (11)$$

where S denotes the number of species used, x_i denotes the count of non-identical aligned nucleotides between *D. Melanogaster* and species i and d_i denotes the evolutionary distance between *D. Melanogaster* and species i . $mrate(x)$ is the average mutation rate of sequence x in mutations per million years. The result of this experiment is shown in figure 29 overleaf.

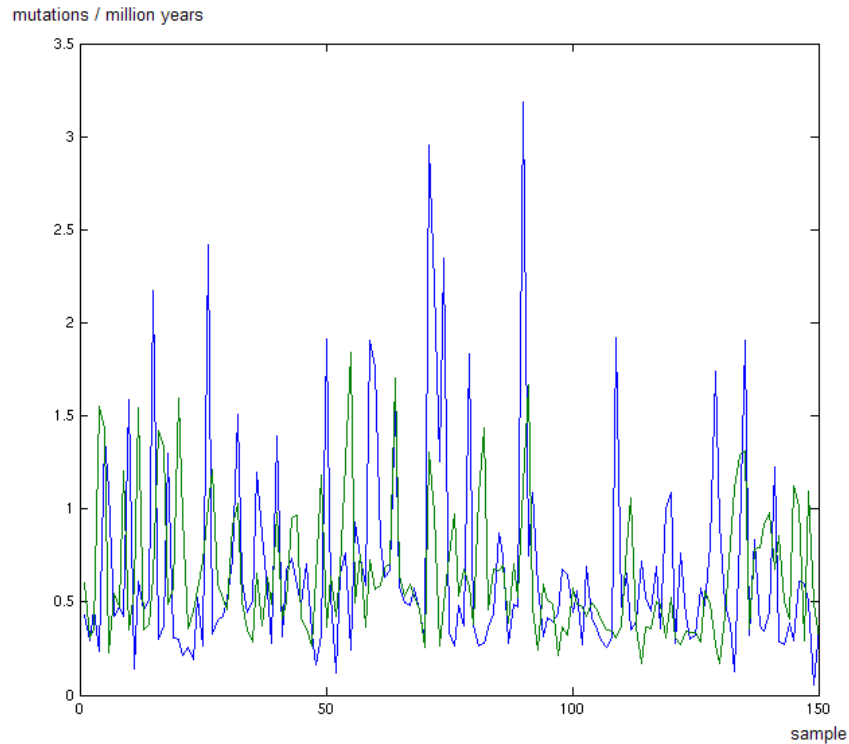


Figure 29: Average mutation rates in 150 promoters and non-promoters

The blue (dark) line in above figure represents the average mutation rate in promoter sequences, the green (light) line the rate for non-promoter sequences. Above graph does in fact show values for 150 discrete samples. A solid lines does however visualise the values better than discrete points and is therefore chosen for display here although mathematically not entirely correct. The mean values for the single nucleotide polymorphisms per million years are 0.68 and 0.65 for positive and negative samples respectively. This means that the average mutation rate is not a clear distinguishing mark to characterise promoter and non-promoters. Further experiments were conducted.

[44] proposes that differences in the ratio between transitions and transversion can be used for predicting RNA secondary structures. Here an experiment has been performed to examine whether the same can be used to recognise promoters. The same reduced set as been processed to obtain the

transition/transversion ratio for promoter and non-promoter sequences. The ratio was derived following equation 12.

$$ttr(x) = \frac{\frac{1}{S} * (\sum_{i=1}^S p_i * \frac{1}{d_i})}{\frac{1}{S} * (\sum_{i=1}^S q_i * \frac{1}{d_i})} \quad (12)$$

where S denotes the number of species, p_i denotes the count of transition single nucleotides polymorphisms between D. Melanogaster and species i , q_i denotes the count of transversion single nucleotides polymorphisms between D. Melanogaster and species i and d_i denotes the evolutionary distance between D. Melanogaster and species i . $ttr(x)$ is the average transition over transversion ratio of a given sequence x . The result of this experiment is shown in figure 30.

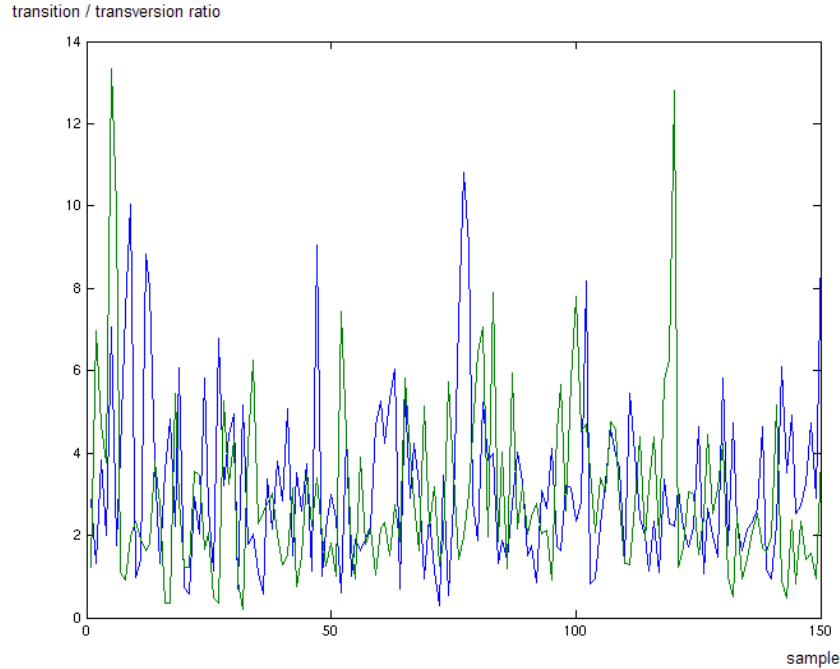


Figure 30: Average transition/transversion ratio in 150 promoters and non-promoters

The blue (dark) line in above figure represents the average transition/transversion ratio in promoter sequences, the green (light) line the ratio for non-promoter

sequences. Again the shown solid line is mathematically not entirely correct, but chosen here to achieve a clearer representation of the result of the experiment. The mean transition/transversion ratio is 3.19 in promoter sequences and 2.98 in others. Unfortunately this means that this ratio is not suitable to distinguish promoter from non-promoter sequences either. A further experiment was conducted to examine mutations in detail.

When looking at mutations the most detailed representation is the rate matrix. This is a 5 x 5 matrix (4 bases + insertion and deletions) contains a mutation rate with which each base was transformed into another base plus the rates with which all bases were inserted or deleted. This rate matrix was attained by using equation 13 which is a variant of equation 11.

$$M(p, q) = \frac{1}{S} * \left(\sum_{i=1}^S x_i^{p \leftarrow q} * \frac{1}{d_i} \right) \quad (13)$$

where p and $q \in A, C, G, T, -$ and $x_i^{p \leftarrow q}$ denotes the number of bases p in D.Melanogaster that turned into base q in species i in a given sequence x . Again S denotes the number of species used and d_i denotes the evolutionary distance between D.Melanogaster and species i . The resulting mean matrices for promoter and non-promoter sequences are given in tables 12 - 14.

| | A | C | G | T | InsDel |
|--------|--------|--------|--------|--------|--------|
| A | - | 0.0233 | 0.0573 | 0.0222 | 0.0003 |
| C | 0.0257 | - | 0.0209 | 0.0502 | 0.0 |
| G | 0.0788 | 0.0225 | - | 0.0224 | 0.0 |
| T | 0.0228 | 0.0464 | 0.0241 | - | 0.0 |
| InsDel | 0.0797 | 0.0548 | 0.0573 | 0.0751 | - |

Table 12: Mean promoter rate matrix

| | A | C | G | T | InsDel |
|--------|--------|--------|--------|--------|--------|
| A | - | 0.0236 | 0.0574 | 0.0256 | 0.0 |
| C | 0.0212 | - | 0.0223 | 0.0570 | 0.0 |
| G | 0.0635 | 0.0222 | - | 0.0291 | 0.0 |
| T | 0.0247 | 0.0555 | 0.0274 | - | 0.0 |
| InsDel | 0.0584 | 0.0468 | 0.0489 | 0.0669 | - |

Table 13: Mean non-promoter rate matrix

| | A | C | G | T | InsDel |
|--------|--------|--------|--------|--------|--------|
| A | - | 0.0003 | 0.0 | 0.0034 | 0.0003 |
| C | 0.0045 | - | 0.0014 | 0.0067 | 0.0 |
| G | 0.0153 | 0.0004 | - | 0.0067 | 0.0 |
| T | 0.0019 | 0.0091 | 0.0033 | - | 0.0 |
| InsDel | 0.0213 | 0.008 | 0.0084 | 0.0082 | - |

Table 14: Table 12 - table 13

Table 12 and 13 show the mean mutation rate matrices for promoter and non-promoter regions. It can be seen that the two are very alike. Table 14 shows the absolute differences between the two tables. The largest disagreement between promoter and non promoter regions is the rate with which A's turn into G's which is on average 0.0153 events per million years more frequent in promoters than in non-promoters. All other mutations are even less distinct. This unfortunately means that even the mutation rate matrices do not yield any relevant information that could be used for the classification of promoters.

How can it be explained that above three experiments deviated in their results so far from the expected outcome? Neither overall mutation rates nor mutation rate matrices nor the transition/transversion ratio can be used to recognise promoters according to these experiments. The values for all those parameters are far too indiscriminative to be of any use as an additional input to a neural network based classifier. It is believed that the explanation for this behaviour lies within the data that is being used. The dataset (training and test) were self-fabricated using results from another classification model. With that all flaws of this system are carried over into the

present system. NNPP was used for the creation of test and training set. If the flaws of NNPP result in a number of positive and negative samples that were falsely classified, the test and training sets end up being unreliable and not sufficiently distinct from each other. This flaw is then reflected in the inconclusive results that the above experiments deliver. Furthermore the negative samples were chosen at random from the sequence. That means their function is unknown. It is merely ensure that they are most likely not core promoter regions. With their function unknown it is uncertain how they behave in the evolutionary context.

For this very reason the subsequent section of this document describes a number of experiments that has been conducted with a different dataset.

6.3 Reducing dimensionality of ready-to-use dataset

6.3.1 Application of property functions

Under [45] a collection of data from the species *D. Melanogaster* is available. This dataset contains core promoter regions, coding sequences and introns. Since these samples are not self-fabricated but are taken from a reliable source on the web it can be taken for granted that are what they claim to be. It can be assumed that the function of those sequences has been verified in a laboratory.

The length of these samples is 300 bases each. As was previously explained in section 5.2.6 of this document, this means that the neural networks designed for the processing of this data would become too large and cumbersome. Training would take too long. Also because samples are not available in sufficient numbers the networks are likely to overfit.

[43] offers a solution to this kind of problem. He introduced four property functions that are applied to each sequence. They result in 7 property values for each sample. One of them delivers a value for each of the four bases. These values are subsequently used as input to the neural network based classifiers. These four property functions are:

1. **GC-content** - This function calculates the proportion of guanine and

cytosine bases within a given string of DNA. The calculation follows equation 14.

$$GCcontent(x) = \frac{N_C + N_G}{L} \quad (14)$$

where N_C and N_G represent the number of cytosine and guanine bases and L denotes the total length of the given sequence. Thus $GCcontent(x)$ is the value for the CG-content of a given DNA sequence x .

2. **Distribution value** - This property function accepts an array containing a set of DNA strings as input. It then calculates the occurrence of each base at each position within all DNA strings. After that it calculates the probability of each base occurring at each position within the array. The calculation follows equations 15 and 16.

$$d(j, b) = \frac{\sum_{i=1}^L Y(i, j, b)}{L} \quad (15)$$

$$Y(i, j, b) = \begin{cases} 1 & \text{if } S(i, j)^T = b \\ 0 & \text{otherwise} \end{cases} \quad (16)$$

where L denotes the length of a given sequence, S^T denotes the complete set of sequences and $S(i, j)^T$ represents the j^{th} base in the i^{th} string. Further the distribution matrix d is defined in equation 15 with $Y(i, j, b)$ being defined in equation 16.

For a given DNA string S the distribution value is then defined by equations 17 and 18.

$$D(S) = \sum_{i=1}^L R(S, i) \quad (17)$$

$$R(S, i) = \begin{cases} d(i, A) & \text{if } S(i) = A \\ d(i, C) & \text{if } S(i) = C \\ d(i, G) & \text{if } S(i) = G \\ d(i, T) & \text{otherwise} \end{cases} \quad (18)$$

3. **Fractal dimension** - This property describes the fractal dimension of a given base for a given DNA sequence. For this reason this function delivers four values for each input string, one for each of the four bases A, C, G and T. The determination of the fractal dimensions is governed by equations 19 and 20.

$$Frac^b(x) = mean(D(X)) \quad (19)$$

$$D(X) = (X(2) - X(1), X(3) - X(2), \dots, X(L) - X(L - 1)) \quad (20)$$

where $Frac^b(x)$ denotes the fractal dimension for a given string x from the dataset with respect to the base b and L denotes the length of the input strings. Given a division of each string into sections of length $l = 1 \dots L$, $X(k)$ then denotes the number of all sequences with length k that contain the base b .

4. **Ficket score** - This property function calculates the ficket score for a given DNA sequence. It expects the complete dataset as input. The dataset be denoted S^T and $b_{i,j}$ is the j^{th} base in the i^{th} string of the set. Equation 21 defines three sets of subsequences of S^T .

$$\begin{aligned} A(i, 1)_n &= (b_{i,(n*3-2)}, b_{i,(n*3-1)}, b_{i,(n*3)}) \\ A(i, 1)_n &= (b_{i,(n*3-1)}, b_{i,(n*3)}, b_{i,(n*3+1)}) \\ A(i, 1)_n &= (b_{i,(n*3)}, b_{i,(n*3+1)}, b_{i,(n*3+2)}) \end{aligned} \quad (21)$$

where n denotes a base position within string i . Using those sets of subsequences three intermediary values are calculated as defined by equation 22

$$A_{int}^b(k, j) = \sum_{i=1}^L count(A(i, k)_j, b) \quad (22)$$

where $k \in 1, 2, 3$, j denotes a position within the sequence, L represents the length of the given sequence and $b \in A, C, G, T$. The final ficket

score of a base b at position j with respect to the whole training set S^T is defined in equation 23.

$$ficket^b(j) = \frac{\max(A_{int}^b(k \in 1, 2, 3, j))}{\min(A_{int}^b(k \in 1, 2, 3, j) + 1)} \quad (23)$$

The final ficket score of a given DNA sequence S is then defined by equations 24 and 25.

$$FICK(S) = \sum_{i=1}^L X(b_j) \quad (24)$$

$$X(b_j) = \begin{cases} ficket^A(\lfloor (j-1)/3 \rfloor + 1) & \text{if } b_j = A \\ ficket^C(\lfloor (j-1)/3 \rfloor + 1) & \text{if } b_j = C \\ ficket^G(\lfloor (j-1)/3 \rfloor + 1) & \text{if } b_j = G \\ ficket^T(\lfloor (j-1)/3 \rfloor + 1) & \text{if } b_j = T \end{cases} \quad (25)$$

A JAVA implementation of these four property functions can be found in appendix A.

The four functions were applied to all sample sequences of the dataset that was taken from [45]. This reduced the input dimensionality of the dataset from 300 down to 7. After that the dataset was divided into two separate training sets A and B and a test set.

6.3.2 Data classification using property values

Two different configurations of neural networks have been constructed, one with a single hidden layer and one with two hidden layers. The configuration with one hidden layer had 7 input neurons, 17 neurons in the hidden layer and one output neuron. The configuration with two hidden layers had the same number of input and output neurons and 12 and 8 neurons in the first and the second hidden layer respectively. All neurons had Log-Sigmoid transfer function except for output neuron who are Linear. Each of those configurations was used to build two neural networks, each of which was trained with either training set A or with training set B . The four networks are called $netA$, $netB$, $netC$ and $netD$.

Table 15 shows which network had which configuration and was trained with which training set.

| | Training set A | Training set B |
|----------|----------------|----------------|
| 7:12:8:1 | netA | netB |
| 7:17:1 | netC | netD |

Table 15: Configurations and training sets for four networks

All four networks were trained until the mean square error on the respective training set dropped below 0.00001. Resilient backpropagation, the same algorithm as before, was used for the training of those networks. Networks were trained to produce 0 for negative and 1 for positive training data. This means that the limit for the final hard limiter function that is applied to the output of the output neuron is set to 0.5 in order to produce a final assertion of the network.

The test set that was used for validating the performance of those four networks contained 500 positive and 500 negative samples. The performance of those networks on the test set can be seen in table 16.

| Network | Sn | Sp | total |
|---------|-------------|-------------|-------------|
| net1 | 493 (98.6%) | 495 (99.0%) | 988 (98.8%) |
| net2 | 478 (95.6%) | 479 (95.8%) | 957 (95.7%) |
| net3 | 499 (99.8%) | 492 (98.4%) | 991 (99.1%) |
| net4 | 498 (99.6%) | 493 (98.6%) | 991 (99.1%) |

Table 16: Results of networks A-D on test set

The three combination methods "Majority voting", "Maximum" and "Average" were exercised on those results. The combined results can be seen in table 17 overleaf.

| Combination method | Sn | Sp | Total |
|--------------------|-------------|-------------|-------------|
| Maximum | 499 (99.8%) | 472 (94.4%) | 971 (97.1%) |
| Average | 498 (99.6%) | 495 (99.0%) | 993 (99.3%) |
| Majority voting | 499 (99.8%) | 493 (98.6%) | 992 (99.2%) |

Table 17: Combined results on test set with three different combination methods

In this case the combination method "Average" delivers the best performance for combining the results for the four classifiers. Thus it is chosen for this system.

These results are very good. In fact they are far better than could have been expected. Naturally there is a significant loss of information when reducing the input dimensions in such a drastic way from 300 down to 7. This should influence the performance of the classifiers negatively. Furthermore the results achieved by previous promoter recognition systems reviewed in section 3 were not nearly as accurate as the neural networks trained here.

An explanation for this behaviour might be found in the data that is used to train the networks. The dataset that was taken from [45] contains sequences that all have a very clear biological significance. The positive samples are core promoter regions the negative samples in this training set consist of introns and coding sequences. It is believed that their clear biological function leads to a characteristic assembly of bases. This assembly is transferred by the property functions into a set of numbers that is well recognisable to the neural networks involved. This feature of the dataset compensates for the loss of information that is inflicted by the reduction of input dimensions.

6.3.3 Incorporation of evolutionary data

The following part of this chapter describes experiments that have been conducted on this dataset in order to improve the performance of above designed classification system with the help of evolutionary data.

The first problem that needs to be overcome in order to include evolutionary

information into this system was to compile a training set that contained aligned sequences to the ones that were taken from [45]. In contrast to the experiments that have been described in section 6.2 this is not as straightforward by comparison. The samples that have been used to train the networks and the data that provides the alignment and therewith the evolutionary information is taken from two different sources. In spite of them being taken from the genome of the same organism, the locating of correspondents between the two is the first obstacle that needs to be overcome.

Apart from the fact that the amount of data that is necessary to be searched is huge, this problem is not simply a matter of basic string matching. Positive and negative sample are given on [45] as a string of 300 bases. An example is given below.

```
>RH54371,.substring (length: 300)
GGCTTTTGGGGTGGGATGAAATTATTAGCCAGCGCGAACCTTAACATACAAATAGCCAGC
GGATTTGCGCAGCGCTTAAAATTGAGATTTACTGGGGCAATATTTATGGCATAACGCAAG
AAGCTGGCGCGAGCTGAACCAATTGGGGAAAGTACTCACTTATGCCAAGACTCAGCTGC
TTATCTAGTGACATTGATAAATGCCCGGGCAATTAGTGCATTGGCTATATAAGGGAAGCT
TCCCTCTCAACTTGCCATACAAATCAACATGCGTTCCCTATTCTGGTCTGTTTGGTCTT
```

These 300 bases are a core promoter region taken from the genome of *D. Melanogaster*. The genome of this organism as it is given in the alignment is a string of 315,438,582 characters. But not all of those characters are bases. The alignment also contains a significant amount of insertions and deletions as could be seen in figure 27 in section 5.3. This is the reason for the discrepancy between the original size of the *Drosophila* genome (about 140 million bases) and the size of the file (about 315 million characters). These insertions and deletions need to be ignored when comparing parts of the genome to the positive or negative sample in the training set on [45]. This and the vastness of DNA data to be searched make this a challenging problem.

In appendix B.1 the JAVA code that was used for locating the data matches is given. An explanation how this algorithm works can be found in the next section (section 7). With this algorithm it was possible to find the location of 100 promoters and the location of 100 coding sequences that were cited on [45] in the complete sequence alignment. The algorithm calculates at

the same time three evolutionary parameters for every match that has been located. These three parameters are

- The number of insertions and deletion occurring at this location in the aligned genomes
- The total number of single nucleotide polymorphisms
- The overall mutation rate at this location in the genome in events per million years

With this information a new training set could be established. It contained 90 positive and 90 negative samples. 10 positive and negative samples were kept as a test set for performance validation. The dataset contained 10 parameters for each sample. Seven of those were the property values described above. In addition to this the dataset contained three parameters taken from the evolutionary history of the respective region in the genome. It is expected that the recognition process benefits from this new data for the simple reason that there is more information available to base the classification decision on.

Table 18 shows a comparison between the mean values of the evolutionary parameters for promoter and coding sequences.

| | Promoter | Coding sequences |
|--------------------------|----------|------------------|
| NOf Insertions/Deletions | 18,295 | 16,140 |
| NOf SNPs | 5,275.7 | 2,411.1 |
| Average mutation rate | 25.77 | 20.16 |

Table 18: Compared mean vales of evolutionary parameters

The experiments that were described in section 6.2 did not show any significant differentiation between parameters acquired from promoter and non-promoters. The mean values of the parameters taken from those two groups were practically identical, which made them useless for the purpose of classification. The parameters that have been examined here are very different in this respect. The mean values display prominent means of discrimination. Especially the number of single nucleotide polymorphisms is very different

for promoter and coding sequences. This makes it very likely that these parameters can be used beneficially as additional input to neural network based classifiers.

The differences are however not consistent with what was hypothesised earlier. These experiments show that the evolutionary activity in promoter regions is higher than in the negative samples and not lower as was assumed before. The reason for this lies probably in the fact that this experiment compares promoters to coding regions only. It is understandable that the mutation rate in coding regions is relatively low, because in such locations the exact order of the bases is decisive for protein production. In promoters however the functionality of marking the TSS of a gene can possibly be maintained after a few minor changes.

The three evolutionary parameters were determined for all 100 promoters and all 100 coding sequences. With this dataset 2 neural networks have been trained. These neural networks were named *EvoNetA* and *EvoNetB*. The configuration of these networks is identical with the configurations shown in table 15 with the exception that the number of input neurons was changed from 7 to 10 to accommodate for the three extra parameters. Again resilient backpropagation was used to train the networks and training was said to be complete when the mean square error on the training set dropped below 0.00001. The networks *EvoNetA* and *EvoNetB* were then shown 10 promoters and coding sequences that were previously unencountered. Both networks managed to recognise all 20 test cases correctly.

Without evolutionary parameters this system did still display a minimal residue error rate. Through the addition of evolutionary parameters it was possible to eliminate this error rate.

7 Description of the software involved

From a software engineering point of view this project has been relatively unambitious. No working piece of software for usage by anyone but me needed to be produced. Nothing needed to be released to a community of users. There were no requirements and no specifications to be followed. There were no usability or security issues to be considered or any other issues

that are most relevant when producing well-written software.

In the context of this project programming was merely a means of processing the DNA data involved. All neural networks were set up and run in MATLAB using the built-in MATLAB Neural Network toolbox. Setup, parameterisation, training and simulation are built-in standard functions of MATLAB. Therefore all the programming that was done for this project was for the purpose of reformatting and analysing the DNA data. Apart from very small and trivial tasks, such as for example interspersing positive and negative sample for the training set, which will not be mentioned any further, the main programming efforts for this project can be broken down into three groups. These groups are:

1. Java implementations of the property functions
2. Evolution related Java scripts
3. MATLAB scripts for combination methods

The code for these can be found in appendix A to C. The choice of Java for most of these tasks was made, because I could resort to previous know-how. Also the processing of strings and characters and the reading and writing of files can be comfortably done with the API classes "String", "BufferedReader", "FileReader", "BufferedWriter" and "FileWriter". It also proved an advantage that classes that were written and compiled on my own computer running Microsoft Windows could be run remotely on the ECS Lab machines running Sun Solaris operating systems without recompilation.

7.1 Java implementations of the property functions

Well-commented code implementing the four property function can be found in appendix A.1 - A.4. In principle all four classes function very similarly. They all expect an input file in the same format and output results to a file in the same way. The input needs to be formatted in such a way that the file contains one sample per line. The next line contains the next sample and so on. The files that were taken from [45] were prepared with a simple java script to meet these requirements.

The main method of each of the four classes contains a *for*-loop that reads the sample into a *String* object. Each sample gets processed individually and the result is written to the output file. This output file contains only one result per line and the line numbers between input and output file correspond to each other. That way sample and respective property value can be related to each other.

The main method of each class will further implement the equation that defines the property function. All equations were given in section 6.3. If a property value is defined by several functions naturally this function will be implemented in the main method that accepts the sample as a string object directly as an argument. For the distribution value this is equation 17, for the Fractal dimension this is equation 19 and for the ficket position this is equation 24.

All other supportive functions that are part of the definition of a property value have been implemented in Java as close as possible to the definition given in the equations. Their names have been preserved. For example equation 23 has the Java definition *public static double fick(char cBase, int j)* where *char cBase* represents the base *b* in the equation and *int j* stands for the position index *j*.

All other functions have been implemented according to the same principle so that it should be straightforward to associate equations with Java code.

7.2 Evolution related Java scripts

There are two pieces of programming that have been done for the purpose of processing evolutionary information that are worth mentioning here.

The first is the location of matches between the data taken from [45] which consists of samples taken from *D. Melanogaster* alone and the complete alignment of the 12 *Drosophila* species. The code for doing this is shown in appendix B.1. Basically this problem is the simple problem of finding the location of small substrings in a single large string. There are however two issues that make this task more complicated than that. For once the data samples taken from [45] do not contain any insertions and deletions. In contrast to that the

aligned genomes contain a great deal of them. This means that the insertions and deletions have to be ignored when comparing strings. Another issue is the size of the *Drosophila* genome. The genome consists of 315,438,582 characters (nucleotides + insertions/deletions). This means that it would take more than 300MB to store it in memory.

The latter problem has been tackled by reading the genome character by character instead of reading it into a *String* object. This way the whole genome is not kept in memory in its entirety at any point. The former problem was solved by putting together a substring for comparison with the sample. This substring will be assembled ignoring the insertions and deletions. Each time a new base from the genome is added to the substring the substring is compared to the start portion of the sample. This start portion will have the same length as the current substring from the genome. A match is found if the substring from the genome (disregarding insertions and deletions) is equal to the sample sequence.

Well-commented code for this algorithm can be found in appendix B.1. Because of the large size of the *Drosophila* genome¹⁴ the execution of this algorithm is very time consuming. The search was executed separately for coding sequences, promoters and introns on separate machines in the ECS lab. After several days of execution time only about 100 promoters and coding sequences could be located. At this point execution was interrupted. The samples located up to this point did suffice to conduct the experiments concerning evolutionary parameters.

The second piece of code that is found in appendix B is Java code for determining the mutation rate matrices for a given set of samples. It shows how the mutation rate matrices have been determined for a given dataset. The class expects the aligned data to reside in a single file. One sample consists of 12 lines the first of which is the positive or negative sample in *D.Melanogaster* and the 11 others are the data from the other species aligned with this sample. In between the data for individual samples an empty line is expected. The algorithm will output one rate matrix for each sample, again all of them in a single file and in the same order than the samples in the input file.

¹⁴For comparison: A file containing an uncompressed text version of *War and Peace* by *Leo N. Tolstoy* is about 3MB in size compared to about 300MB for the *D. Melanogaster* genome

7.3 MATLAB scripts for combination methods

Not all programming for this project has been done in Java. Since the setup of the neural networks involved has been done in MATLAB some programming needed to be done here as well. An example is the implementation of the hard limiter function that is used to convert the output of the neural networks output neuron into an assertion about whether a given input is a promoter or not. This code was already shown in section 6.1.

Apart from this and several other small and trivial tools the main programming effort that was made in the MATLAB environment was the implementation of the combination methods for several classifiers. Code for the "Majority vote" and for the "Maximum" combination methods can be found in appendix C.1 and C.2.

Both of these scripts accept four arrays as input. It is assumed that these arrays contain the results reported by the four individual classifiers. Naturally the "Majority vote" script expects these results to be already hard limited, i.e. they can be only 1 or 0, and the "Maximum" script expects these results to be any floating point number that the neural network outputs.

Both scripts return a fifth array that is of the same size as the input arrays. In the case of the "Majority vote" this array contains in each place either a 1 or a 0 depending what the majority of the input arrays shows at the same position. In the case of a tie the place will contain a 1, which is arguably completely arbitrary. In the case of the "Maximum" combination method this array contains in each place that number that by its absolute value is the biggest from the four input arrays at the corresponding position.

The third combination method "Average" did not need any coding. It consists only of very simple arithmetic that can be done manually in MATLAB very efficiently.

8 Conclusion and future work

There are several conclusions that can be drawn from above experiments. As we have seen it is advantageous for the classification process not to rely on the results of one single classifier alone. The performance is always improved when combining the results of several classification systems. Several classifiers that are sufficiently different from each other through the use of disjoint training sets or through varying methods of input data encoding have the ability to cover different sections of the input space. Thus their combination can cover a greater portion of the input space than any classifier alone, provided they are sufficiently different from each other and all in more than 50% of the cases correct. Which combination method is most suitable for which problem needs to be determined experimentally from case to case.

We have also seen that classifiers based on artificial neural networks perform very well on classification tasks in Bioinformatics. This is under two conditions. Firstly the training data needs to be available in a sufficient amount to enable the network to generalise on the problem and to prevent it from overfitting. If there is not enough training data available to train a network that would be large enough for the level of complexity of the problem, the size of the network needs to be reduced by decreasing the dimensionality of the input data as was demonstrated above. Secondly the training data needs to be of good enough quality. If it contains too many falsely classified elements the outcome of the classifiers will be flawed.

More importantly than this however are the observations made here regarding information obtained from the evolutionary history of the genomes. It was shown that evolution does in fact provide information that can be exploited for the annotation of genomes. More specifically it provides information that can be used as input parameters for neural network based classification systems in order to reduce recognition error rates. In a first instance it was shown that false positive classifications can in part be remedied when considering the corresponding locations of a sequence in a number of related species. After that a number of evolutionary parameters were successfully used to improve recognition results further. These parameters were the total number of evolutionary events for a given region of a genome, the total number of insertions and deletions in a specified region and the average mutation rate of the sequence over time.

In the context of this dissertation the observation was made that data derived from the evolutionary history of a genome can be used for the benefit of genome annotation. This is a very general statement. The experiments that were performed here regarding promoter detection through neural network based Multi Classifier Systems are only one example out of a great multitude of opportunities for possible research in this field. Since the work done in this MSc project yielded the presented results it is straightforward to assume that other approaches that are based on similar assumptions will be successful as well.

The incorporation of evolutionary data into artificial intelligence based tools in Bioinformatics in a relatively new and vast field of research. One tiny exemplary fraction of this field has been examined in this project. Subsequently more advanced problems in the same field can be attacked. It can be expected that one will be able to use evolution for the benefit of those problems as well. More advanced problems in Bioinformatics that could be tackled by the utilisation of artificial intelligence based systems and evolutionary data are among many others:

- RNA gene finding and classification
- Finding and characterisation of regulatory signals
- Characterisation of substitution processes
- Functional characterisation of protein genes
- Prediction of protein interaction
- Local and global sequence alignment on a sequence and gene level respectively.

Not only the type of problem can be varied, it is also self-evident that other techniques than neural networks such as Stochastic Context Free Grammars, Genetic Algorithms and Hidden Markov Models will profit from the incorporation of evolution.

The right time for conducting more research in this field is only now that enough aligned genomes become available for research. Only based on a

strong pool of data can the exploitation of evolutionary history for algorithmic solutions in Bioinformatics be followed to its full scope. The 12 *Drosophila* genomes dataset that was introduced here constitutes such a data pool.

References

- [1] Romesh Ranawana and Vasile Palade. A neural network based classifier for gene identification in dna sequences. *Neural Computing and Applications, Volume 14, Number 2*, pp. 122-131, July 2005.
- [2] Irina Arkhipova. Promoter elements in drosophila melanogaster revealed by sequence analysis. *Genetics, Vol. 139*, pp. 1659-1669, 1995.
- [3] Alan Kutach and James Kadonaga. The downstream promoter element dpe appears to be as widely used as the tata box in drosophila core promoters. *Molecular and Cellular Biology, Vol. 20, No. 13*, pp. 4754-4764, July 2000.
- [4] Clark, Gibson, Kaufman, Myers, and O'Grady. Whitepaper proposing drosophila as a model system for comparative genomics. web resource, <http://flybase.net/.data/docs/CommunityWhitePapers/GenomesWP2003.html>.
- [5] Uwe Ohler and Heinrich Niemann. Identification and analysis of eukaryotic promoters: recent computational approaches. *TRENDS in genetics, Vol. 17, No.1*, pp. 56-60, February 2001.
- [6] Berkeley Drosophila Genome Project. web resource, <http://www.fruitfly.org/>.
- [7] Brekeley Drosophila Genome Project. Neural network promoter prediction. web resource, http://www.fruitfly.org/seq_tools/promoter.html.
- [8] Martin G. Reese. Application of a time delay neural network to promoter annotation in the drosophila melanogaster genome. *Computers and Chemistry, Vol. 26*, pp. 51-56, May 2001.
- [9] Martin G. Reese. Computational prediction of gene structure and regulation in the genome of drosophila melanogaster. Phd thesis, 2000.
- [10] M. G. Reese and F. H. Eeckman. Novel neural network algorithms for improved eukaryotic promoter site recognition. *The seventh international Genome sequencing and analysis conference, Hyatt Regency, Hilton Head Island, South Carolina*, September 16th - 20th 1995.

- [11] Steen Knudsen. Promoter2.0: for the recognition of polii promoter sequences. *Bioinformatics*, Vol. 15, No. 5, pp. 356-361, February 1999.
- [12] Center for Biological Sequence Analysis Technical University of Denmark DTU. Promoter 2.0 prediction server. web resource, <http://www.cbs.dtu.dk/services/Promoter/>.
- [13] J.W. Fickett and A.G. Hatzigeorgiou. Eukaryotic promoter recognition. *Genome Research*, Vol. 7, No. 9, pp. 861-878, 1997.
- [14] Matthias Scherf, Andreas Klingenhoff, and Thomas Werner. Highly specific localization of promoter regions in large genomic sequences by promoterinspector: A novel context analysis approach. *J. Mol. Biol.*, Vol. 297, No. 3, pp. 599-606, 2000.
- [15] PromoterInspector highly specific promoter region prediction. Prediction of promoter regions in mammalian genomic sequences. web resource, <http://genomatix.gsf.de/cgi-bin/promoterinspector/promoterinspector.pl>.
- [16] Vladimir B. Bajic, Seng Hong Seah, and Allen Chong et al. Dragon promoter finder: recognition of vertebrate rna polymerase ii promoters. *Bioinformatics*, Vol.18, No.1, pp. 198-199, 2002.
- [17] Knowledge Extraction Lab Singapore. Dragon promoter finder version 1.5. web resource, <http://research.i2r.a-star.edu.sg/promoter/promoter1.5/DPF.htm>.
- [18] Vladimir B. Bajic, Seng Hong Seah, and Allen Chong et al. Computer model for recognition of functional tss in rna polymerase ii promoters of vertebrates. *Journal of Molecular Graphics and Modelling*, Vol. 21, pp. 323-332, 2003.
- [19] Vladimir B. Bajic, Seng Hong Seah, and Allen Chong et al. An intelligent system for vertebrate promoter recognition. *IEEE Intelligent Systems*, Vol. 17, Issue 4, pp. 64-70, August.
- [20] Vladimir B. Bajic. Comparing the success of different prediction software in sequence analysis: A review. *Briefings in Bioinformatics*, Vol. 1, No. 3, pp. 214-228, 2000.

- [21] Uwe Ohler. Computational promoter recognition in eukaryotic genomic dna - all you (n)ever wanted to know about finding promoters. *PhD Thesis - submitted to the Technische Universitaet Erlangen*, 2001.
- [22] U. Ohler, H. Niemann, G. Liao, and G. M. Rubin. Joint modeling of dna sequence and physical properties to improve eukaryotic promoter recognition. *Bioinformatics*, Vol. 17, pp. 199-206, 2001.
- [23] Massachusetts Institute of Technology. Mcpromoter: Promoter prediction server. web resource, <http://genes.mit.edu/McPromoter.html>.
- [24] U. Ohler, S. Harbeck, H. Niemann, E. Noeth, and M. G. Reese. Interpolated markov chains for eukaryotic promoter recognition. *Bioinformatics*, Vol. 15, No. 5, pp. 362-369, 1999.
- [25] Uwe Ohler. Computational analysis of core promoters in the drosophila genome. *Genome Biology*, Vol. 3, research0087.1-0087.12, 2002.
- [26] M. Q. Zhang. Identification of human gene core-promoters in silico. *Genome Research*, Vol. 8, pp. 319-326, 1998.
- [27] Gary D. Stormo, Qing K. Chen, and Gerald Z. Hertz. Promfd 1.0: a computer program that predicts eukaryotic pol ii promoters using strings and imd matrices. *CABIOS*, Vol. 13, No. 1, pp. 29-35, 1997.
- [28] Stephane Audic and Jean-Michel Claverie. Detection of eukaryotic promoters using markov transition matrices. *Computers Chemistry*, Vol. 21, No. 4, pp. 223-227, 1997.
- [29] G. B. Hutchinson. The prediction of vertebrate promoter regions using differential hexamer frequency analysis. *CABIOS*, Vol. 12, No. 5, pp. 391-398, 1996.
- [30] Dan S. Prestridge. Predicting pol ii promoter sequences using transcription factor binding sites. *J. Mol. Biol.*, Vol. 249, pp. 923-932, 1995.
- [31] Charles E. Lawrence et al. Detecting subtle sequence signals: A gibbs sampling strategy for multiple alignment. *Science New Series*, Vol. 262, No. 5131, pp. 208-214, 1993.

- [32] Gerald Z. Hertz and Gary D. Stormo. Identifying dna and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, Vol. 15, Nos 7/8, pp. 563-577, 1999.
- [33] MatLab documentation (Neural Network Toolbox). Backpropagation. web resource, <http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/backprop.html>.
- [34] Romesh Ranawana and Vasile Palade. A neuro-genetic framework for multi-classifier design: An application for promoter recognition in dna sequences. *International Journal of Hybrid Intelligent Systems*, Volume 3, Number 1, pp. 35-61, 2006.
- [35] Romesh Ranawana and Vasile Palade. Multi-classifier systems - review and a roadmap for developers. *University of Oxford Computing Laboratory, Internal report*, January 6th 2006.
- [36] T.G. Dietterich. Machine-learning research: Four current directions. *AI Magazine*, Vol. 18, No.4, pp. 97136, 1998.
- [37] Anne Magaly de Paula Canuto. Combining neural networks and fuzzy logic for applications in character recognition. *University of Kent, PhD Thesis*, 2001.
- [38] L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, pp. 993-1001, October 1990.
- [39] L.I. Kuncheva (editor). Diversity in multiple classifier systems. *Journal of information fusion*, Vol 6, Oktober 1990.
- [40] L.I. Kuncheva and C.J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, Vol. 51, pp. 181-207, 2003.
- [41] Borries Demeler and Guangwen Zhou. Neural network optimisation for e.coli promoter prediction. *Nucleic Acids Research*, Vol. 19, No. 7, pp. 1593-1599, February 1991.
- [42] Soeren Brunak, Jacob Engelbrecht, and Steen Knudsen. Prediction of human mrna donor and acceptor sites from the dna sequence. *Journal of Molecular Biology*, Vol. 220, pp. 49-65, 1991.

- [43] Romesh Ranawana and Vasile Palade. Mvgen - ensemble learning for mcs majority voting with a genetic algorithm. *University of Oxford Computing Laboratory, Internal report*, November 27th 2005.
- [44] B. Knudsen and J. Hein. Rna secondary structure prediction using stochastic context-free grammars and evolutionary history. *Bioinformatics*, Vol. 15, No. 6, pp. 446-454, 1999.
- [45] Berkeley Drosophila Genome Project. Collection of data of d. melanogaster core promoter regions. web resource, http://www.fruitfly.org/seq_tools/datasets/Drosophila/promoter/.

A Java implementations of the property functions

A.1 GC content

```
import java.io.*;

public class GCcontent
{
    public static void main(String args[]) throws Exception
    {
        // open input file
        BufferedReader inFile = new BufferedReader(new FileReader("promoterClean.fa"));

        // open output file
        BufferedWriter outFile = new BufferedWriter(new FileWriter("promoterGCcontent.txt"));

        // for all lines of the input file
        for(int ii=1; ii<=1941; ii++)
        {
            int iNOfG = 0,
                iNOfC = 0;
            double dResult = 0.0;

            // read one sample
            String s = inFile.readLine();

            // count the number of G's and C's
            for (int j=0; j<300; j++)
            {
                if (s.charAt(j) == 'C')
                {
                    iNOfC++;
                }
                if (s.charAt(j) == 'G')
                {
                    iNOfG++;
                }
            }

            // and divide by the length of the sample
            dResult = (((double)(iNOfC + iNOfG)) / 300.0);

            // round result to 5 digit accuracy
            dResult += 0.000005;
            int i = (int)(dResult * 100000.0);
            double d = (((double)i)/100000.0);

            // write result to output file
            outFile.write(d + "\n");
        }
        outFile.close();
    }
}
```

A.2 Distribution value

```
import java.io.*;

public class distribution
{
```

```

// array for the whole training set
public static String sSet[] = new String[1942];

public static void main(String args[]) throws Exception
{
    // open input file
    BufferedReader inFile = new BufferedReader(new FileReader("promoterClean.fa"));

    // open output file
    BufferedWriter outFile = new BufferedWriter(new FileWriter("promoterDistribution.txt"));

    // read in whole training set
    for(int ii=1; ii<=1941; ii++)
    {
        sSet[ii] = inFile.readLine();
    }

    // for all sample in the set
    for(int ii=1; ii<=1941; ii++)
    {
        double dDist = 0.0;

        // for each index in the sample
        for (int i=0; i<300; i++)
        {
            // sum up R(s,i)
            dDist = dDist + R(sSet[ii], i);
        }

        // round result to 5 digit accuracy
        dDist += 0.000005;
        int iTmp = (int)(dDist * 100000.0);
        dDist = (((double)iTmp)/100000.0);

        // write result to output file
        outFile.write( dDist + "\n");
    }

    outFile.close();
}

// return d(j) for each respective base
public static double R(String S, int j)
{
    char cBase = S.charAt(j);

    switch(cBase)
    {
        case 'A':
            return d(j, 'A');
        case 'C':
            return d(j, 'C');
        case 'G':
            return d(j, 'G');
        default:
            return d(j, 'T');
    }
}

// implementation of function d
public static double d(int j, char cBase)
{
    double dResult = 0.0;

    double a = 0.0;

    // sum up Y values for all samples
    for (int i=1; i<=1941; i++)
    {
        a = a + Y(i, j, cBase);
    }

    dResult = a / 1941.0;

    return dResult;
}

```

```

// implementation of function Y
public static double Y(int i, int j, char cBase)
{
    double dRes = 0.0;

    // return 1 if S[i][j] equals cBase
    if (sSet[i].charAt(j) == cBase)
        dRes = 1.0;
    else
        dRes = 0.0;

    return dRes;
}
}

```

A.3 Fractal value

```

import java.io.*;

public class fractal0
{
    public static String sSeq = "";

    public static void main(String args[]) throws Exception
    {
        // open input file
        BufferedReader inFile = new BufferedReader(new FileReader("promoterClean.fa"));

        // open output file
        BufferedWriter outFile = new BufferedWriter(new FileWriter("promoterFractal.txt"));

        // for all samples
        for(int ii=1; ii<=1941; ii++)
        {
            // read in sample
            sSeq = inFile.readLine();

            double dA = 0.0,
                   dC = 0.0,
                   dG = 0.0,
                   dT = 0.0;

            // execute once for each of the four bases
            for (int jj=0; jj<4; jj++)
            {
                char cBase = 'A';
                switch(jj)
                {
                    case 0:
                        cBase = 'A';
                        break;
                    case 1:
                        cBase = 'C';
                        break;
                    case 2:
                        cBase = 'G';
                        break;
                    case 3:
                        cBase = 'T';
                        break;
                }

                int iFrac = 0;
                double dResult = 0.0;

                // calculate mean value of X(i,B)
                for (int i=0; i<299; i++)
                {
                    iFrac = iFrac + (X(i+2, cBase) - X(i+1, cBase));
                }
            }
        }
    }
}

```

```

    }

    dResult = (((double)iFrac) / 299.0);

    // round result to 5 digit accuracy
    dResult += 0.000005;
    int iTmp = (int)(dResult * 100000.0);
    double dTmp = (((double)iTmp)/100000.0);

    switch(jj)
    {
        case 0:
            dA = dTmp;
            break;
        case 1:
            dC = dTmp;
            break;
        case 2:
            dG = dTmp;
            break;
        case 3:
            dT = dTmp;
            break;
    }

    }

    // write results to output file
    outFile.write(dA +"\t"+ dC +"\t"+ dG +"\t"+ dT + "\n");

}

outFile.close();

}

// returns the number of substrings with a given length containing a given base
public static int X(int iLength, char cBase)
{
    int iResult = 0,
        iNOfSubstr = 0;

    String s = "";

    iNOfSubstr = 301 - iLength;

    for (int i=0; i<iNOfSubstr; i++)
    {
        s = sSeq.substring(i,i + iLength);

        if (contains(s, cBase) == true)
            iResult++;
    }

    return iResult;
}

// return true if string contains character
public static boolean contains(String s, char c)
{
    boolean bResult = false;

    for (int i=0; i<s.length(); i++)
    {
        if (s.charAt(i) == c)
            bResult = true;
    }

    return bResult;
}

}

```

A.4 Ficket value

```
import java.io.*;

public class fick0
{
    // array for the whole training set
    public static String sSet[] = new String[1942];

    public static void main(String args[]) throws Exception
    {
        // open input file
        BufferedReader inFile = new BufferedReader(new FileReader("promoterClean.fa"));

        // open output file
        BufferedWriter outFile = new BufferedWriter(new FileWriter("promoterFicket.txt"));

        // read whole training set to array
        for(int ii=1; ii<=1941; ii++)
        {
            sSet[ii] = inFile.readLine();
        }

        // for all samples
        for(int ii=1; ii<=1941; ii++)
        {
            double dFicket = 0.0;

            // sum up value for X(B, j)
            for (int j=0; j<300; j++)
            {
                dFicket = dFicket + X(sSet[ii].charAt(j), j);
            }

            // round result to 5 digit accuracy
            dFicket += 0.000005;
            int iTmp = (int)(dFicket * 100000.0);
            double dTmp = (((double)iTmp)/100000.0);

            // write result to output file
            outFile.write(dTmp + "\n");
        }

        outFile.close();
    }

    // return value of fick(B, j) for each respective base
    public static double X(char cBase, int j)
    {
        switch(cBase)
        {
            case 'A':
                return fick('A', ((int)((j-1)/3)) + 1);
            case 'C':
                return fick('C', ((int)((j-1)/3)) + 1);
            case 'G':
                return fick('G', ((int)((j-1)/3)) + 1);
            case 'T':
                return fick('T', ((int)((j-1)/3)) + 1);
            default:
                return 0.0;
        }
    }

    // implementation of fick-function
    public static double fick(char cBase, int j)
    {
        double dMax = 0.0,
            dMin = 0.0,
            dRes = 0.0;

        // get maximum of three A values
        dMax = max(A(1, j, cBase), A(2, j, cBase), A(3, j, cBase));

        // get minimum of three A values
    }
}
```

```

    dMin = min(A(1, j, cBase), A(2, j, cBase), A(3, j, cBase));

    // divide max by min+1
    dMin += 1.0;
    dRes = dMax / dMin;

    return dRes;
}

// implementation of A function
public static double A(int k, int j, char cBase)
{
    double dResult = 0.0;

    // sum up all the count values for all samples in the set
    for (int i=1; i<=1941; i++)
    {
        dResult = dResult + count(i, k, j, cBase);
    }

    return dResult;
}

// implementation of count
public static int count(int i, int k, int j, char cBase)
{
    int iResult = 0;

    // return the number of occurrences of cBase in the subset given by i, j and k
    switch(k)
    {
        case 1:
        {
            if ((j*3)-3 < 300)
                if (sSet[i].charAt((j*3)-3) == cBase) iResult++;
            if ((j*3)-2 < 300)
                if (sSet[i].charAt((j*3)-2) == cBase) iResult++;
            if ((j*3)-1 < 300)
                if (sSet[i].charAt((j*3)-1) == cBase) iResult++;
            break;
        }
        case 2:
        {
            if ((j*3)-2 < 300)
                if (sSet[i].charAt((j*3)-2) == cBase) iResult++;
            if ((j*3)-1 < 300)
                if (sSet[i].charAt((j*3)-1) == cBase) iResult++;
            if (j*3 < 300)
                if (sSet[i].charAt(j*3) == cBase) iResult++;
            break;
        }
        case 3:
        {
            if ((j*3)-1 < 300)
                if (sSet[i].charAt((j*3)-1) == cBase) iResult++;
            if (j*3 < 300)
                if (sSet[i].charAt(j*3) == cBase) iResult++;
            if ((j*3)+1 < 300)
                if (sSet[i].charAt((j*3)+1) == cBase) iResult++;
            break;
        }
        default:
        {
            System.out.println("error");
            System.exit(0);
            break;
        }
    }

    return iResult;
}

// helper functions for maximum and minimum of three
public static double max(double a, double b, double c)
{

```

```

        return (Math.max(Math.max(a, b), c));
    }

    public static double min(double a, double b, double c)
    {
        return (Math.min(Math.min(a, b), c));
    }
}

```

B Evolution related java scripts

B.1 Find matches in genome and calculate evolutionary parameters

```

import java.io.*;

public class evoPro0
{
    public static void main(String args[]) throws Exception
    {
        // open file containing samples
        BufferedReader bfrDataFile = new BufferedReader(new FileReader("promoterClean1.fa"));

        String sProm = "",
            sTemp = "";

        int iC = 0,
            iCnt = 0,
            iInsDels = 0;
        long lIndex = 0,
            lMatchIndex = 0,
            lMem = 0;
        boolean bCntInsDels = false;

        char cC = 'x';

        // for all samples in the file
        for (int iProm=1; iProm<=1000; iProm++)
        {
            // open whole genome of D.Melanogaster
            BufferedReader bfrDroMel = new BufferedReader(new FileReader("line1.txt"));

            // read one sample
            sProm = bfrDataFile.readLine();

            // guiding output
            System.out.print("searching promoter " + iProm + "-> ");

            lMatchIndex = -1;
            iCnt = 0;
            lIndex = -1;

            // endless loop
            while (true)
            {
                // read genome character by character
                cC = (char)bfrDroMel.read();

                // remember index of character read
                lIndex++;

                // leave loop at end of genome
                if (cC == '\n') break;
            }
        }
    }
}

```

```

// do or do not count the number of insertions and deletions
if (bCntInsDels == true)
{
    if (cC == '-')
    {
        iInsDels++;
        // continue to next characters
        continue;
    }
}

// count length of current string
iCnt++;

// add current character to current string
sTemp = sTemp + cC;

// compare current string to part of the genome
if (!(sTemp.equals(sProm.substring(0, iCnt))))
{
    // reset and continue if not equal
    if (iCnt > 1)
    {
        lIndex = lMem;
        bfrDroMel.reset();
    }

    sTemp = "";
    iCnt = 0;
    iInsDels = 0;
    bCntInsDels = false;
    continue;
}
else // if equal continue adding characters
{
    if (iCnt == 1)
    {
        bfrDroMel.mark(10000000);

        lMem = lIndex;

        bCntInsDels = true;
    }

    // if current string has length 300 a match is found
    if (iCnt == 300)
    {
        lMatchIndex = lIndex - (300+iInsDels);

        break;
    }
}
}

if (lMatchIndex == -1)
{
    System.out.print("not found\n");
}
else
{
    // guiding output
    System.out.print(lMatchIndex + " match\n");

    // create file for alignment of the match
    String sOutFileName = "AlignedPromoterPro" + iProm + ".txt";

    BufferedWriter outFile = new BufferedWriter(new FileWriter(sOutFileName));

    // copy strings from the 11 other genomes in the respective location to the new file
    for (int i=1; i<=12; i++)
    {
        String sFileName = "line" + i + ".txt";
        BufferedReader bfrDroOther = new BufferedReader(new FileReader(sFileName));

        for (int k=0; k<lMatchIndex; k++)
            bfrDroOther.read();
    }
}

```

```

        String sS2 = "";

        for (int j=1; j <= (300 + iInsDels); j++)
            sS2 = sS2 + ((char)bfrDroOther.read());

        // write result to output file
        outFile.write(sS2 + "\n");
    }

    outFile.close();
}

} // end for

// when all samples have been processed calculate evolutionary parameters
countInsDels();
countNOfEvents();
getMutRate();

} // end main

// =====
public static void countInsDels() throws Exception
{
    // open output file for insertions and deletions
    BufferedWriter outFile = new BufferedWriter(new FileWriter("insDels0.txt"));

    // for all samples
    for (int index=1; index<=1000; index++)
    {
        String sFilename = "AlignedPromoterPro" + index + ".txt";

        BufferedReader bfrFile = null;

        // proceed only for those where a result file exists
        try
        {
            bfrFile = new BufferedReader(new FileReader(sFilename));
        }
        catch (FileNotFoundException ex)
        {
            continue;
        }

        // count the numbr of insertions and deletions
        String saDro[] = new String[12];

        long lInsDels = 0;

        for (int j=0; j<12; j++)
        {
            saDro[j] = bfrFile.readLine();

            for (int k=0; k<saDro[j].length(); k++)
            {
                if (saDro[j].charAt(k) == '-')
                    lInsDels++;
            }
        }

        // write filename and result to output file
        outFile.write(sFilename + " : " + lInsDels + "\n");

        bfrFile.close();
    }

    outFile.close();
}

// =====

public static void countNOfEvents() throws Exception

```

```

{
// open output file for total number of events
BufferedWriter outFile = new BufferedWriter(new FileWriter("nOfEvents0.txt"));

// for all samples
for (int index=1; index<=1000; index++)
{
    String sFilename = "AlignedPromoterPro" + index + ".txt";

    BufferedReader bfrFile = null;

    // proceed only for those where a result file exists
    try
    {
        bfrFile = new BufferedReader(new FileReader(sFilename));
    }
    catch (FileNotFoundException ex)
    {
        continue;
    }

    // count the number of events
    String saDro[] = new String[12];

    long lNOfEvents = 0;

    saDro[0] = bfrFile.readLine();

    for (int j=1; j<12; j++)
    {
        saDro[j] = bfrFile.readLine();

        for (int k=0; k<saDro[j].length(); k++)
        {
            if (saDro[j].charAt(k) != saDro[0].charAt(k))
                lNOfEvents++;
        }
    }

    // write filename and result to output file
    outFile.write(sFilename + " : " + lNOfEvents + "\n");

    bfrFile.close();
}

outFile.close();
}

// =====
public static void getMutRate() throws Exception
{
    // set up array with evolutionary distances from D.Melanogaster
    double[] dEds = new double[12];
    dEds[0] = 0.0;
    dEds[1] = 3.2;
    dEds[2] = 3.2;
    dEds[3] = 9.4;
    dEds[4] = 9.4;
    dEds[5] = 13.3;
    dEds[6] = 25.9;
    dEds[7] = 25.9;
    dEds[8] = 35.4;
    dEds[9] = 39.4;
    dEds[10] = 39.4;
    dEds[11] = 39.4;

    double[] dMutRate = new double[12];

    // open output file for mutation rates
    BufferedWriter outFile = new BufferedWriter(new FileWriter("mutRates0.txt"));

    for (int index=1; index<=1000; index++)

```

```

{
    String sFilename = "AlignedPromoterPro" + index + ".txt";
    BufferedReader bfrFile = null;

    // proceed only for those where a result file exists
    try
    {
        bfrFile = new BufferedReader(new FileReader(sFilename));
    }
    catch (FileNotFoundException ex)
    {
        continue;
    }

    // determined the mutation rate
    String saDro[] = new String[12];

    for (int i=0; i<12; i++)
    {
        saDro[i] = bfrFile.readLine();
    }

    for (int j=1; j<=11; j++)
    {
        String sDroMel = saDro[0];
        String sDroOther = saDro[j];

        double dMut = 0;

        for (int k=0; k<sDroMel.length(); k++)
        {
            if (sDroMel.charAt(k) != sDroOther.charAt(k)) dMut++;
        }

        dMutRate[j] = dMut / dEds[j];
    }

    // determine the mean value of 11 mutation rates
    double dOverallRate = 0.0;
    for (int l=1; l<=11; l++)
    {
        dOverallRate += dMutRate[l];
    }

    dOverallRate /= 11.0;

    // round the result to 5 digit accuracy
    dOverallRate += 0.000005;
    int i = (int)(dOverallRate * 100000.0);
    double d = (((double)i)/100000.0);

    // write filename and result to output file
    outFile.write(sFilename + " : " + d + "\n");
}

outFile.close();
}
}

```

B.2 Determination of rate matrices

```

import java.io.*;

public class getRateMatrices
{
    public static void main(String args[]) throws Exception

```



```

        default:
            ddMutMatrix[0][4] += 1.0;
            break;
    }
    break;
case 'C':
    switch(cOther)
    {
        case 'A':
            ddMutMatrix[1][0] += 1.0;
            break;
        case 'G':
            ddMutMatrix[1][2] += 1.0;
            break;
        case 'T':
            ddMutMatrix[1][3] += 1.0;
            break;
        default:
            ddMutMatrix[1][4] += 1.0;
            break;
    }
    break;
case 'G':
    switch(cOther)
    {
        case 'A':
            ddMutMatrix[2][0] += 1.0;
            break;
        case 'C':
            ddMutMatrix[2][1] += 1.0;
            break;
        case 'T':
            ddMutMatrix[2][3] += 1.0;
            break;
        default:
            ddMutMatrix[2][4] += 1.0;
            break;
    }
    break;
case 'T':
    switch(cOther)
    {
        case 'A':
            ddMutMatrix[3][0] += 1.0;
            break;
        case 'C':
            ddMutMatrix[3][1] += 1.0;
            break;
        case 'G':
            ddMutMatrix[3][2] += 1.0;
            break;
        default:
            ddMutMatrix[3][4] += 1.0;
            break;
    }
    break;
default:
    switch(cOther)
    {
        case 'A':
            ddMutMatrix[4][0] += 1.0;
            break;
        case 'C':
            ddMutMatrix[4][1] += 1.0;
            break;
        case 'G':
            ddMutMatrix[4][2] += 1.0;
            break;
        case 'T':
            ddMutMatrix[4][3] += 1.0;
            break;
        default:
            break;
    }
    break;
} // end switch(cCurr)

```

```

    } // end for (int j=0; j<50; j++)

    // divide over the respective evolutionary distance
    for (int ll=0; ll<5; ll++)
    {
        for (int kk=0; kk<5; kk++)
        {
            ddMutMatrix[kk][ll] /= dEds[i];
            ddRateMatrix[kk][ll] += ddMutMatrix[kk][ll];
        }
    }

} // end for (int i=1; i<=11; i++)

// get rate matrix
for (int ll=0; ll<5; ll++)
{
    for (int kk=0; kk<5; kk++)
    {
        ddRateMatrix[kk][ll] /= 4.0;

        // round the results to 5 digit accuracy
        ddRateMatrix[kk][ll] += 0.000005;
        int i = (int)(ddRateMatrix[kk][ll] * 100000.0);
        double d = ((double)i)/100000.0;

        // write results to output file
        out.write(d + "\t");
    }
    out.write("\n");
}

out.write("\n");

} // end for (int index=1; index<=300; index++)

out.close();

} // end main
}

```

C MATLAB scripts

C.1 Majority voting MATLAB script

```

function y = mv(A,B,C,D)
[a b] = size(A);
K = [1:b];
o = 0;
z = 0;
for i = 1 : b;
    if A(1,i) == 0
        z = z + 1;
    else
        o = o + 1;
    end
    if B(1,i) == 0
        z = z + 1;
    else
        o = o + 1;
    end
    if C(1,i) == 0
        z = z + 1;
    else
        o = o + 1;
    end
end

```

```

end
if D(1,i) == 0
    z = z + 1;
else
    o = o + 1;
end
if o >= z
    K(1,i) = 1;
else
    K(1,i) = 0;
end
z = 0;
o = 0;
end
y = K;

```

C.2 Maximum MATLAB script

```

function y = mx(A,B,C,D)
[aa bb] = size(A);
K = [1:bb];
a = 0;
b = 0;
c = 0;
d = 0;
for i = 1 : bb;
    a = A(1,i);
    b = B(1,i);
    c = C(1,i);
    d = D(1,i);
    if a < 0;
        a = -a;
    end
    if b < 0;
        b = -b;
    end
    if c < 0;
        c = -c;
    end
    if d < 0;
        d = -d;
    end
    X = [a b c d];
    x = max(X);
    if x == a;
        K(1,i) = A(1,i);
    end
    if x == b;
        K(1,i) = B(1,i);
    end
    if x == c;
        K(1,i) = C(1,i);
    end
    if x == d;
        K(1,i) = D(1,i);
    end
end
y = K;

```

D Gene CG17759 of *Drosophila Melanogaster*

Functional regions are colour-coded: introns and intergenic sequences are in lowercase letters, UTRs are blue and CDSs are red.

```
gttgcttaccctttgcagatgatgtaattataaactcaagctacttcaatgttaatgggattgctgctcatttccataaaaattccatcaatt
aagtaagcctaaacttgtagattctcatttgcagggtataaactcggaaatcggagatcacattcctttgagctttgcacgctagttccccct
ccgcgcaaatgcagatctcagcacttccctgctatattgagttcagttcttattgccaatttacaggaggaaggctggctgatgggtcagaaggagg
caccacagagaaggactcttcccccaactcagcgtcccatctgaagcgacaaactggggaacaactgggctagcaacgaacctcagcaacttca
gatcaacataaaaactttacgaacggagggtccaaatcacaactttggcacaatttcttggtagcgtgctttgctttgtccaattaa
taaatctgttttggtaagtattcgtaactggcaactcatttatagcagatgttctgtaggcagtgctccaacataaaaattgtattgttaac
acaaagaagaacgaacacagatgaggaatgagagagagaatattgaaaatgtttcaaacattatttttagttctaaagttcaagaaaactaaag
cataaactcagatacatatattttgtaactctatttggctcataaataatgcgaacaaaactgataaattgtaatttggtaattggaacac
cacaatgtaaaacatacactcgagagctagagtaattgttaacaaaataaaaactcggagagatagccagcctgaacctttgtattttgtgaatt
tgcaaatatgtagtagcagttttttatagtaaatgaaatgtattttggggcgatgcaaaaaggccagatgtaattcctctggtccccgaagtttgg
aaattgttcaagtaacacattaacgtttagcagatagcgtatctttaaattgtactttggttttaattttcgtttgagagactcgaactagctac
agcttataacgcgaacacacatcaatagctactaacctcagatcagatgtttttatatttaactcagattgaaacacagaaaccgaaataat
gttttaattgtgcaacacactgaaagtatttttaatttttaatttaatttaaacactttatctcagatgacaatattgaaacaaatgataata
ttcacagaaggcaataaaaactagatataaataacagcttttgattggcgggcaatgctctgatttgaagatagaaagtgcttgaatttg
acgtatgcagcggcttttacgatattcgcatttttatgttcaaatggtaattatacacacataattagttatttaaaattacgatcatagtcatt
ggttcgggtcacagcttaacgaataggaagtgcaggttaagtgtattcctggtttagagagtgaaaggacacacatttcagcttctcaaaagc
tatcaggcattgccatcgaatattgagaatattggcgactccatccttatttaagatgtctggcgtgccatcctatttgaagatgtaagc
gttcgcaaccttattgctaagaatgtgattgacacacatgaatagaatcctgatggcaacaccgacatgtgttgggtgggggtggcaactaaaaat
accnaagtaataattttcaaatgaagcgaactccaggaatcctgtttttatgtataaacactctatagttattatgctggcaataaaact
ccttcttattgatacaatgaagactttgattgaagcactcttccatcagatatttgcaccactagttcaaatgccgtccgatggttgac
catcgatctcgacgaacgataggccgataggttcgcaacaaataacccctcctgctatcgaaaactcgggtcgtgctcgaactgagcaaaaatgcaaaa
tfaaatgtttgtagtagcagcagatgcaaaatcgaatattgttgagtttcgagcaagatccccctcaaacggactccccgacagcgtgcca
cccgacgctggcagccacccgaaaaagtccccctcgcaaaaaaaaatttagcagcaccgaaacgttccccgaaatgcaagtgtaaaagtga
tttttccgcaaacagctggccttaagcaagcccccataattgttaacaaacggcgaatcgaaatcggcaaatctcgaatcagctgctatca
aaattttctgttaccgctccctctattattgtgtattctgttcaaaagcagtgccagcaagctgctgctgcccagaaaaaacctgc
gaataggaaaaagttgagaatttaaaaaagaatagttgtttaaactcaatgtggtttctttgcccactaaacggtgcatgccagcagagaagttag
tggggtcgtgaactacatagtgatgtgcatattttgggcttaaaacgggtgggtaaccaaaaaatgaaaaaaaatggtagctgagggctggtg
ggtttaccttagccagatgatgatgattttccggaatgcatattcccaacttacacccataataaccaccccccccccatgcttctctctatc
tctctgctctctcttactgcaattccactgcatacagaaaaataaacaataacaagaacataaaatagcagaaaaatatacaaaaaggagcca
caaggtttcgtggaatacaattgaagaccctcgtcgtcgtgctgctgctgatacactttccggttcaattacactcaaatgaatattgtg
aaactcgtttgatttacagactggttttaaaaaacaacaacaagaagaacgttaaaaacaagaacaacaacaacagcactatacacaccg
aaaactagaagcaaaaaggaaaaatataatagatataatctacataataaataagtagtctgatacacacattacattaaaaggagccgaacagacg
ttatcgaattttatataaaaagcaccagtaaaaaatataaaccaactacacagcttcttaaaaaaagttttccaaacgcaaaaccaactaaa
aacgaaatgctttgaaatgatttgaagtaactaaaaaacgtaagaccacaacacagcagcaaaagcaaatcgcggtgattaaactaaaaatattaaa
cagaccgaacatcggcagggcaacaaaaaacaataaaagaacacagatcaatccggaagaactcaatcggcgtactaaaaccgaaatcaaatatt
ttgtataacatagtgtagtcaaaaaaactcgagacgataaccggaacatcttttggagaactcgtcgtgaaatccttagtcccggtctccgtaatt
gtaagtaagtaaacctgaatcgaatgctcattgaatacaagaatcgaatttctggtgtaattttgactccgaccactgttgggtgccataaaaactggc
gaaatataataatacaaaaatcgaatatttccaggcgagtttaattggattgcatcacttattcaaatccatcgaatattttttgtgta
aaagccatgcttccataatgaaagggtactttgtatcaaaaatgagctcataaattctccagcaaaaataggctgagagatgcgataaaggaaactg
catgagtaacgtgaagctatttccgcttagaaatgggttttcggaatgggttgcattagaacagtgatttccagaagagcagtagtctcctcaaat
aagaagcaaatcaaacgttatctcagagttggtttatcatcagcatatacaatattgaagcagataaagcaggttccggtaggttaaatcctcctcga
tttgggtgcatgatatctggcacaactattgttaaaaaacgctcatttgcgcaattcgtgcccctgcaattttgctgcccactgcaatttga
tttgggtgctgctcagcagcaatgtgttccagtgcaaccactaaactcccgaaactactgctgctgcaatcattatccagatcctcaatttcc
cccactgagcccgaatctcctgctcagactttgtcaaaaattgtgtaggtattccatcgatatttggtagtgctcatgtaaacgctactc
cgaccatgtcaatgttgcgcaaacgaaatggattagccatcactcgatataaaactagggttggaaaagagaggaaatgaccagctggttata
ggctatattctataggatccctatgctgaaccatgaatgaaccgaggttccggtgactcattctttaaagataaaaatgacattaaagtttata
aagttatgcaatgatattccagttttgtatagagctgtgttttgcatacaagttcggtagtgactttttttccgaattagaatgatagc
caattctctatcatagactgcttggccccaataggagaacactcctgcaacttggcagcaggaacatattttccaaaatcagcttct
tgcattgaacttggctgaggggcttggggaacgcttctcaatctattgccacctggtagaaggcagatccaggtctcgcagagagcagggcca
gaactagaacaaagcaatgccatgctcagaggaatccccgggtgtagctggtagccatgggcaatctatccgacgtctcagcgtccgca
gctagtgggcagctagcaaatgaaatgttaattgtcagcagg
GACACAACGCGGCAACTCTCGTTGGGCCAACAACTGTGGT
gtcgtgagcaaatagttggatagatcctacagttcgggtatgcaagcactaaatcaaaacttgtctcctgag
ATGTTCTTGGATCGTTTCGATGTGGTCTTTGTTTATTAGTAACCATCACTCTCAGCCTCC
GAGTACCGAAATCTGTTCAAGAACTTAGCGTAGACCGTAATTAGCAATAGCAATAGC
CAAAGAGCGCTGGATCGTAAGCGCTGGAGGAGTCTGCAGAGCGGACAAAGCGAAGT
ACAAATGCACCAATCGAACTACAACAGAATCTATAACTTGAACGGGTGAATCAACG
CCGCCATACAACCAATTAGCTAGGTGTCCGAGGTCAGAGAAGGGAGTCCAGGAGAC
CATCCCACCATCCAAACAGCATACATTTTATATACGGAACGGTCTGTCGAGCGTATGG
CAGCAGCAGCGGAAAGCGTCTATAAATCTAGTTAGC
ATGGAGTGTGTTTATCGGAGGAGGCCAAGGAACAAAAGCGCATCAATCAGGAAATCGAG
AAGCAGTTGCGCCGGACAAAGAGATGCGCCCGGAGCTTAAACTGCTACTACTGG
gtaagtgagcagtgcccaattcagttgcaactagcctgcttaagcgaattgaagccctcctgcaattgcaattg
tttccccgctttaggctatataattttcaactctatattccccag
GCACTGGCGAGTCCGGGAAAGTCCACATTCATCAAAACAGATCGGTATTATCCACGGCAGG
GTTACTCGGACGAGACAAGCGTGGGTACATCAAGCTGGTTTTTCAGAACATATTCAT
GGCCATGCAGTCAAGTATCAAGGCCATGGATATGCTGAAGATTTCCACGGCCAGGA
```

GAGCATAGT

gtaagtgaacattgatctaattgaagtcactgaatcttaacaacatcttacgatccacag
GAACTGGCCGATCTGGTGATGAGCATCGATTATGAGACCGTTACCACGTTFCGAGGATCCATA
CTTGAATGCCATCAAACCGCTTTGGGACGATGCTGGCATCCAGGAGTGCTATGATCGT
CGTAGGGAATATCAGCTGACTGATTACGCCAAATA
gtaagttaacggtggcatgcaaaagggaacggttaaacctcgcgcaaaatgaccgaaaaacgaactgaagagactgaaactaccgaaacgactatgc
aaactaaatcaattatctgtctatgtatag
TTATCTGAAGGATCTCGATCGTGTGGCTCAACCTGCATATTTACCCACTGAGCAAGACATTT
TAAGAGTTCGTGTGCCACAACAGGGATAATTGAGTATCCCTTTGATTAGAAAGAA
TCAGATTTAG
gtacataccatctttctacccttactctgtgcatgcatgacacctgtttgggtatggtaaaatftttttgactttgtaactac
acacttaagaattcttttagcgcctgtctctcaagcaacaattcagggacttagtagggcctcagcgaatgtatggactgagacgggtgcaaaa
ctatctttgcccactgctttggctccgtctatgttttaaaagtattatcttaatttagctacttgacgcatgactgcaattgcaacggctg
attacttgcacacggagcagatattctcgtgctcagtgccgacaactggcattcttgatccattcgattggatggcattggttttaggtatct
atataaaatcaattgtaattaccgcaaggctagttaagttaattggctcattgtaagctagtgaagacgattcttacaatttagttactaatagat
tcgttgcctgctcagaatfaattgctgtatgtttggccttatgtatgtaattgttggtaacttttctacaataatataatcttcaataaa
acag
AATGGTAGACGTCGGTGGTCAGCGATCCGAGAGAAGAAAGTGGATTTCATTGCTTTGGAAGAT
TGACATCAATTATATTTTTTGGTAGCGCTATCGGAGTACGATCAAACTCTTGTTTGAATC
TGATAATGAG
gtatgattcaagttatgaatatatTTTTTtaacaataatcacctgtacgctttcacag
AATCGAATGGAGGAATCTAAAGCTTTATTTTCGTACTATAATTACATACCCTTGGTTTCAAAA
TTCTGTCAGTTATCTTTTCTCTGATAAGAGGACTTGTGGAAAGAGAAAATAATGTAT
TCCGATTTGGTAGACTATTTTCTGAAATCGATG
gtaagtttccgcatgataaaagggtggccataatattgacgattttatgcccgaatgaaccagggtcaaaaacaggatcacgctggcggcaaacagttcgt
ctgaaaaagacttggcctgcaatccagatcccgaacgtcaatgctattcgcatttcacaacggccacaggtgattttcaccttatgacttaagatc
tatgtaattgcttaaatgccaatcatcctgtagattttgctagttttctgtagaacccgctagacgtaactcatattgcctctttatgcaaacag
GTCTCAGCGAGATGCAATAACGGCCCGAGAGTTTATCTGCGAATGTTTGTAGATTTAAAT
CCAGATTCGGAATAAATTTATCTATTCTCATTTCCAGGTGTGCTACAG
gtaagtttgcgacttctgtgtttggtcagaattgtgtgtgtagtatttagtttcagcctaaaaagggactggcctcaaaaggtaa
acttttggtaaaaaactttacatttgcataattgcttttggcctgactagagtagagagtagttgtatcgcctagtcacaaattgcaaacctaaac
gaattgatttcaataaaacacagatccttggtaaacagataccgaaacatcaaacctgtgttctcgcctgcaaaagatacaaatgcaaaatgcaac
tgaagaaattcaatttgggctaaaggcggcagtgccgatacaggaaggtggcaatgcaaaatgattttatataatgtagtattggtttttca
aagtcctcaataaaactaaatagatctgtgaaatctgtaaggtgaaatgtgtgactttactatcatagattattgcaaaaaactaaact
acgctcttttcttttctgtag
ATACGGAAAATATAAGGTTTGTGTTTGCAGCTGTTAAGGACACAATTCTGCAATCGAACCT
TAAGGAATATAATTGGTCTAAA
CTGGATTGGATCGAAAACATATTTTGTGATTTTTATACAAAACACATACACATTCAT
ATAATTCCTTTTTCGCATATTTACTAAAATATAAAGAAGAAATTGAGAAGCGAGCGAAT
GTGGCCGAAAATTAGGCATAAATTAGACCCGAGAAAGTTTCTACGTTGTTCATCA
GGAACCGAAAATTGTAGCATAAATAGAATACTAAACTGAGATGCTAAAACCGCATCAG
CAGAATGTGAAAAGATGTATACATTTAATTTGAGAGAAAATAACACAAAATCTG
ATACAGTGTGACAACAATAATCAGGGCAGAGTTTCTCTACGAAATGTGACGATTCGT
AAAACCTTATTTATATTGCACTACTTCTATTCTTCTAGCGGCAAGTTTTCAGAAA
ACGTTGTGTAATTTTTTTTTTGTTCGACTAGGAAAACCTTTGAAAGTCAAAAAG
ACAGTGGCCAGAAATGATTGGTACCCAATCGATAGAGTTTTTGGTTAGATTGAATAA
ATCTGATGGTAGTTTTTCTTCTACACAAGAAAACATGATTTCTTTGAAAAGTTTTTA
TACAAAATATCGTCCGAATTAGTTGTTGTCACTCTGTAATATTTAAATAAATTTATTA
CTATAGTAACTTTTGTAATAAAATCCGAAAACACAAAATATGAAAACGTAATTTATCA
AAAGCTTGATAAGGAACAACAACAACTGAAAACAACCTTTGTCGTACTGAAAGCATAACATAA
TAAAGCAACATTTACTAAAATAACTACAGAAAACAAAACATACTTTTCCGTAATAA
TCCCAGAAAATATATATATTTCAAAAAGATTTATATGCAAGAAGTTTCTTATAGAA
TTTTCAGGGCATATAGCGGTGTA AAAATAATAAACCGAAAATGATAAGCTGATAAATG
ATAAATGAATGAAAACGAAATCACAGATAACTGTATTTTTTATTAAGCTAAACAG
TAAACCGTAAATATTGATACTATTGTTTACCAGAAATATTAACTATACATGAAAGCAG
ATATAAATTCATAAACGCATATGTAACAATTTAAAATTTGATTAATTTGCATTGCG
CGAGCTCCATTTCATGTAATTTACGATTCGTATTTAATACCATGAGAATGA
aaagaaagaatgaaacgcatgcaaaccaatacaaaaacgatataggagctagattgtatgagaaaagccagaacatttcatggagcattcattaa
atagctgttgtaaaaatgtcgaataaccacccatgcaacatgcaacatgcaacagcagaatccatacaacttattgatatttaacagacatcaggtta
aatccaatcaaatgcatataaccactggcagagaacaacaactatcagaaggaaagtttgcctcagctacttttggccatcaaattcagtcgaa
aacattgttatacaaaaatagcttatttctgacttttacaataactttgttagtttggcttatttatagatcttaattgtaacaaagggactc
gcttggcggctagcnaaacggcagatgctgcccgaaggggtcaggttttggctttttataattttagtttaattatacaaggataact
aaatgagacgcacataaatagaaaacccatcaacaatatttattgtaaaaaatcaataaaaaacaaactctactcagcagatggtgaaacaaaaat
aataaaatgaaanaacaaatgaaacgctacatgtatgtaaaataaatgatgcaaaatgttttcaatattatagtaactattttgttcaaccgagc
gagatccctcaaccacatccccatattggcgtttagttccctcggggccacagagagtagacactttgccagactccaagcgaagacactact
catgggagtggaatgagatattcctatgaccgcaatacaaaaacaaagcagcactcttagtttaagtagcattcaacagtggtgtatataagct
ttaatccctgtaaccacataatgtaaaagtaaaagtaaaataaaccattttatgattaaccaatggattagcttttaaaaagcttcaataag
tcaatgtatttttaaaagatgtagccgagaacaacatgagaaaaaaagcaaatgagtttggggccctgctgtaggataatggaatcggttttaggt
ctttggcaatgctgcaatacaaaaatccagctggcactcaaatggtagctccgatttggcctggccatttttatccagcagcattatagacat
tcacacactgacatacatatataatgtttccgtccaatgaattgcatcagcagcagcagcaaaagctcccatattttgactcctaacatttgg
cgattgttcttcgaaaatccactcctttaaaacaagatttttggcgggttcaatgataaaatgttccgactttgagccacacatacgcacact
taaatgactgatttcgatagatttcccaacattttgtccttccagcataatgaatccccatatttatataaggacagactccaatgggtact
caaaactgattattttttttgtactactaatggcctgttaaaagccccagacattcagattatcagcagtggtgtatggccagattgtgt
gcttgggtttagatagattggttcttaattttatgatacgcacagtagtcagacagttgtatgtaaatatttggcaaatctcgcgagggcattcaaa
aggcctccgagagataacacataatcagcagatgtatgtaagttgacattaaacgcaaaatgaaactataataaaatgagaatttgataaaacat
taagtattttgtattttatatacaattataaattttagttacacttacaattgttagcattataactaaaatccatgataaaaccaaagtaa
cgcagatgaatctactagatatttccaatagagaagccgagtagacataatcacacaaatccgacgattgctaaaggaattcagaatattgtaagg