EUKARYOTIC RNA POLYMERASE II START SITE DETECTION USING ARTIFICIAL NEURAL NETWORKS

by

Gerbert Myburgh

Submitted in partial fulfilment of the requirements for the degree
Master of Engineering (Computer Engineering)

in the

Faculty of Engineering, the Built Environment and Information Technology

UNIVERSITY OF PRETORIA

March 2005

SUMMARY

EUKARYOTIC RNA POLYMERASE II START SITE DETECTION USING ARTIFICIAL NEURAL NETWORKS

Author: Gerbert Myburgh
Study leader: Prof. E Barnard

Master of Engineering (Computer Engineering)

Faculty of Engineering, the Built Environment and Information Technology

An automated detection process for Eukaryotic ribonucleic acid (RNA) Polymerase II Promoter is presented in this dissertation. We employ an artificial neural network (ANN) in conjunction with features that were selected using an information-theoretic approach.

Firstly an introduction is given where the problem is described briefly. Some background is given about the biological and genetic principles involved in DNA, RNA and Promoter detection.

The automation process is described with each step given in detail. This includes the data acquisition process, how the different samples were split into different sets and statistical information gathering, feature generation, and the full ANN process. The ANN section of the project is split up in a generation process, a training section as well as a testing section.

Lastly the final detection program was tested and compared to other promoter detection systems. An improvement of at least 10% in positive prediction value (PPV) in comparison with current state-of-the-art solutions was obtained.

Note: A Companion CD should accompany this report that contains all the program code and some of the source data that was used in this project. All the references to “Companion CD”, reference number [18] are references to these programs.

Keywords: Polymerase II, Automated promoter detection, Artificial Neural Network application.

OPSOMMING

EUKARYOTIC RNA POLYMERASE II BEGIN POSISIE DETEKSIE DEUR KUNSMATIGE NEURALE NETWERKE

Deur: Gerbert Myburgh
Studie leier: Prof. E Barnard

Meester van Ingenieurswese (Rekenaar Ingenieurswese)

Fakulteit Ingenieurswese, Bou Omgewing en Inligtings Tegnologie

In dié verslag word 'n outomatiese opsporingsproses vir Eukaryotiese "ribonucleic acid" (RNA) Polymerase II Promoter gegee. Die projek gebruik 'n kunsmatige neurale netwerk (KNN) (ANN in Engels) tesame met kenmerke, wat gekies is deur van inligtingsteoretiese beginsels om die oplossing te benader. In die inleiding word die probleem beskryf. Volgende word die biologiese en genetiese beginsels, wat betrekking het op die projek (DNA, RNA en Promoters), kortliks bespreek. Die outomatisasie proses word beskryf met elke stap in besonderhede verduidelik. Dit sluit in 'n beskrywing in van hoe die verskillende klasvoorbeeldmonsters verkry is en hoe die verskillende stelle opgestel is. Die metode waarvolgens statistiese inligting, as ook hoe die verskillende opsporingskenmerke vekry is word verduidelik. Dan volg die verduideliking van die KNN werking. Die afdeling is verder opgedeel in afdelings oor KNN bou, leer proses en toetsing.

In die laaste gedeelte word die finale promoter identifikasie program getoets en vergelyk met ander promoter identifikasie stelsels. Verbetering ten opsigte van die huidige standaardstelsels van 10% in positiewe identifikasie waarde (PIW) (PPV in Engels) is verkry.
Nota: Saam met die verslag behoort daar 'n CD te wees wat die program kode sowel as van die oorspronklike promoter data bevat. Alle verwysings na die "Companion CD", verwysing [18] is na programme op die CD.

Sleutelwoorde: Polymerase II, Outomatiese promoter identifikasie, Kunsmatige Neurale Netwerke

ABBREVIATIONS

DFP –	Dragon Promoter Finder
DNA –	Deoxyribonucleic Acid
E2I –	Exon-to-Intron
EPD –	Eukaryotic Promoter Database
FD –	False Detections
FR –	False Rejections
GpC –	Sequence segment that contains primarily G and C bases
I2E –	Intron-to-Exon
NCBI –	National Center for Biotechnology Information
PPV –	Positive Predictive Value
RNA –	Ribonucleic Acid
SSE –	Sum of Squared Error
TD –	True Detections
TR –	True Rejections
TSS –	Transcription Start Site

TABLE OF CONTENTS

1 CHAPTER 1: BACKGROUND

1.1 Introduction

The field of pattern recognition is expanding rapidly with productive applications interfacing with various other disciplines. Bio-informatics is an area where pattern recognition comes into contact with disciplines such as genetics and biochemistry. The resulting study of the principles of biological function and organisation is currently one of the most exciting domains of scientific enquiry; although work in this field dates back to before 1860, current knowledge is still very sketchy. Given the vast quantities of data that are being generated, it is crucial that pattern-recognition algorithms be developed to assist in the analysis of genetic data. The specific problem studied here is gene expression; that is, how and why particular genes in a cell are activated. One of the problems with gene expression that currently receives much attention within the pattern-recognition community is that of promoter and TSS[1] detection.

1.2 Brief Problem Definition

The problem addressed in this project is developing a *reliable automated* detection process for Eukaryotic RNA[2] polymerase II promoters. Promoter detection is a very complex and time-consuming process that can potentially be sped up significantly through automation. The first technical problem addressed was the automation of this complex process. The transcription process and the difficulties of automating are described in more detail in the following section of this document. The next challenge was to program reliable detection. As with every pattern recognition application the final system suffers from limited accuracy. The reliability of the system is directly associated with the accuracy of the final detection.

[1] TSS – Transcription Start Site
[2] RNA – Ribonucleic Acid

1.3 Genetic Principles

1.3.1 Deoxyribonucleic Acid (DNA[1])

Lodish, Darnell and Baltimore [1] describe DNA as a cellular storehouse required to build a cell or an organism. DNA contains all the genetic information that describes how and why certain cells function. All known DNA is made up of long sequences of molecules called nucleotides. There are only four different nucleotide types, all differing only in a single part called the base. Because of this nucleotides are commonly referred to as bases, in other words there are four different bases found in DNA. These four bases are adenine (A), guanine (G), cytosine (C) and thiamine (T). It is common practice in genetics when writing down a DNA molecular sequence to write down only a single letter representing the base contained in the nucleotide. In other words a typical DNA sequence might be …CCATCTAGATCGGTAGCATGCTAGTGTCGTAG…

Obviously this is only a small section of the sequence since full sequences are millions of bases long, and will cover thousands of pages if given in full in this document.

A nucleotide has a very specific molecular structure, shown in Figure 1 below.

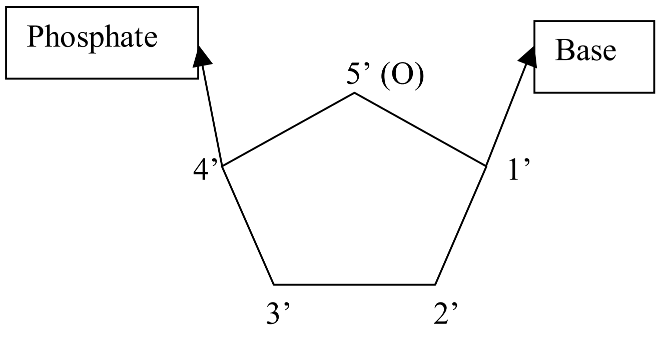

Figure 1. Diagram of a nucleotide
– the digits indicate carbon atoms in the pentose component of DNA

Binding between nucleotides that form a DNA sequence always takes place at the same carbon molecule. The phosphate or 5' carbon always binds with a free hydroxyl at the 3' carbon. To standardise written sequences it is common practice to write down a sequence from the 5' end to the 3' end. In other words in a written sequence the first base or nucleotide will have a free

[1] DNA – Dioxyribonucleic Acid

carbon at the 5' end, and the last nucleotide in the sequence will have a free carbon at the 3' end. The typical DNA sequence given earlier becomes:

`(5′) …CCATCTAGATCGGTAGCATGCTAGTGTCGTAG… (3′)` For the remainder of this document it can be assumed that all given DNA sequences will be in this format. When smaller sections of DNA are compared with regards to their location in the DNA they are said to be upstream or downstream of one another. For example taking the sample DNA once again, the section **ATC** is upstream of <u>GCTA</u>, while <u>GCTA</u> is downstream of **ATC**. `(5′) …CC`**`ATC`**`TAGATCGGTAGCAT`<u>`GCTA`</u>`GTGTCGTAG… (3′)` It is also standard practice to use numbers to represent how far upstream or downstream a specific section is with relation to another section or centre point. For example with transcription the TSS will start at position 1, with everything upstream of the TSS having negative numbers, and everything downstream of the TSS positive. If, for example, the base T marked as 1 in the sample sequence is the TSS then the sequence <u>GCTA</u> starts at position 11, while **ATC** starts at position –7.

```
         -987654321 1 2345678901234567890123
(5′) …    CCATCTAGA T CGGTAGCATGCTAGTGTCGTAG… (3′)
                          ↑
```

Note that there is no position 0. This is because the reference base is base 1, and the one directly upstream of it is base –1, and the base directly downstream of it is base 2.

1.3.2 Transcription

The previous section states that DNA is only the storehouse of genetic material and does not control the functionality of cells directly. Before cells can function correctly the DNA must first be changed to RNA, and the RNA must be changed to proteins. Proteins are directly responsible for cell functionality. DNA directs RNA synthesis, RNA directs protein synthesis and proteins catalyse DNA and RNA synthesis. This circular process - as shown in Figure 2 - is known as the central dogma of genetics [2].

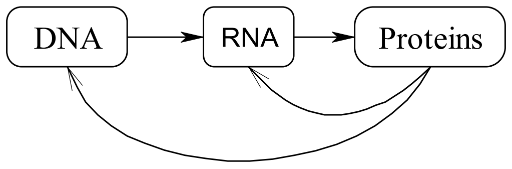

Figure 2: DNA, RNA and Protein relationship. (Central dogma of genetics)

Transcription is the process where RNA is formed from DNA, while the process where proteins are formed from RNA is called translation. The focus of this project is on the transcription process. The full genetic transcription process will not be explained here, but can be found in great detail in the literature, for example [3,4,5,6]. Broadly, what happens is that RNA polymerase (Polymerase are the actual molecules responsible for the transcription) follows the DNA downstream (5' end to 3' end) until a promoter region is recognised. Then binding between the RNA polymerase and the DNA takes place. The polymerase continues its downstream movement until the transcription start site is reached where the transcription process starts. Obviously this is a very elementary explanation of a process that is quite complex in reality, but it does show one of the most important aspects of transcription, which is that each protein or gene contains at least one promoter region upstream of the TSS for the polymerase to bind with. The TSS is the actual position at which the Polymerase binds, and is contained in the promoter segment, usually close to the 3' end. The basic process is the same for both prokaryotic and eukaryotic organisms. (Eukaryotic organisms have cells that contain true nuclei [2], while prokaryotic organism cells do not.) There is more than one type of polymerase, each transcribing different gene types. Polymerase II is responsible for the biggest percentage of gene transcription and thus is the focus of this project.

The biggest difference between prokaryotic and eukaryotic transcription lies in the fact that eukaryotic gene coding information is split up in multiple sections. With prokaryotic transcription polymerase searches the DNA downstream until a promoter region is reached. The proteins required for transcription bind with the DNA and transcription takes place until the end of the gene, or the tail of the gene, is reached. Each base on the way to the promoter region is transcribed. Genes in eukaryotes are separated into intron and exon sections. The introns stay in the nucleotide, and are not transcribed, while exons are transcribed to RNA. Figure 3 displays how a typical gene might look in a eukaryotic organism.

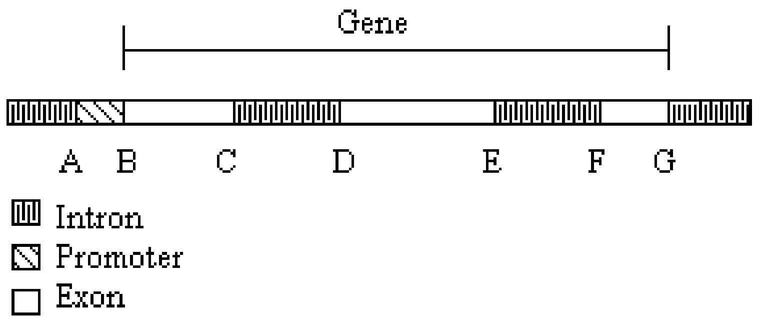

Figure 3: Eukaryotic gene split up into intron and exon regions.

In this figure the total length of the gene is from point B to point G, A being the upstream or 5' end of the DNA, and G being the downstream or 3' end. But the only sections that actually contain coding DNA data for the gene are the exon sections, B-C, D-E and F-G, with section C-D and E-F being non-coding intron areas. The gene coding DNA starts at point B, the TSS, with the area A-B directly upstream of B the promoter area. Once again this is a simplified approximation of the promoter; the next section of this document will provide some insight as to why this is only an approximation. G is the tail, or end of the DNA that will be transcribed. Every letter given in the figure except for A and B represents the place where the DNA changes from intron to exon, or exon to intron. These places are called splices, or splice sites. Points C and E are splices where the exon changes over to intron, exon-to-intron splice, and points D and F splices where the intron changes over to exon, intron-to-exon splice. These are very important features and for the rest of this document they will be abbreviated as I2E[1] and E2I[2] for simplicity.

1.3.3 Complexities of DNA

Now that the transcription process has been briefly explained, we will investigate why this is such a complex process to automate or to model mathematically.

[1] I2E – Intron to exon splice
[2] E2I – Exon to intron splice

The first aspect to consider is the enormous amount of data contained in a single DNA sequence. For example the sequenced human chromosome 20 (used as part of the sample collection process for this project) spans several million bases. This immediately gives some very important implications when writing automated systems that do feature detection. Firstly the final system must work very quickly. It is useless to have a system that is 95% or 100% accurate, but takes too long per nucleotide to do a reliable detection. Secondly the system has to be able to handle large amounts of textual data. This is not such a big problem since programs can generally handle large text inputs in the form of files.

The second problem is that DNA sequences are not entirely unambiguous. When comparing DNA sequences that have the same function it can be found that they differ significantly. The ambiguity comes from the fact that amino acids are formed by three bases. For three bases there are a total of 4x4x4 = 64 total combinations, but there are only 20 different amino acids. The encoding for some of the amino acids can take on several forms. Furthermore this is because not every base in a given DNA section is relevant. This means that sequences might be the same in some parts, and differ in other parts but still perform the same biological function. This becomes even worse where sections differ significantly, but still have the same function. Since not every base in a given section is relevant it can thus be assumed that there are important and garbage DNA parts. (Think of how a gene consists of relevant exon sections as well as non-coding intron parts.) This means for any functional DNA feature a consensus sequence has to be generated that gives the functioning parts with meaningless, or even random, parts in between. The problem arises when the random parts in between functional sections are not the same length. For example consider the following two sequences.

If the regions **ATGCTA** and **AGGCTA** have some function, but are separated by random length DNA garbage they could look like:

CTA**ATGCTA**ATCTGATCGA**AGGCTA**

and

TCCA**ATGCTA**CTCGATC**AGGCTA**CA

Doing a base-by-base comparison, these two look totally different, since the random sections are of different length, causing difficulty in developing an exact model.

The next, and most important aspect addressed, is the promoter region itself. As explained in the previous section it can be assumed that every gene has at least one promoter upstream of the TSS for polymerase binding and thus for the transcription to take place. The problem is that promoter regions are not very well understood by geneticists. For example in eukaryotic transcription it is an accepted fact that there is a promoter region called the TATA-Box. This can be found everywhere in literature on this subject [3,4,5,6]. When these four sources were consulted a disturbing discovery was made. Each one of the four mentioned the TATA-Box and gave a consensus sequence of how it should look, but they all differed on its composition. Although all four sources said that the TATA-Box is upstream of the TSS, the exact location given for it differs from source to source. Each of the sources contains a small sequence, called the consensus sequence, which represents the exact form of the TATA-Box. The consensus sequences and position given in all four is shown below. The notation ($\frac{X}{Y}$) means that either base X or base Y can occur.

[3] Gave the TATA-Box as $TATA\frac{A}{T}A\frac{A}{T}$ at position –30 from the TSS.

[4] Gave it $TATAT\frac{A}{T}AT\frac{A}{T}$ at position –25.

[5] Had it as ATATAA at position –25 to –30.

[6] Said it should be $TATA\frac{A}{T}A\frac{A}{T}$ at –25.

This suggests there is a TATA-Box, containing only bases A and T, somewhere between 25 and 30 bases upstream of the TSS, but its exact form and location is not known and can thus not be accurately modelled. If a simple, well-known promoter like the TATA-Box causes such disagreements, it is easy to understand why more complex promoters give an even bigger problem.

1.4 Full Problem Definition and Project Aims

Simply put, the problem addressed in this project is to develop a *reliable automated* detection process for Eukaryotic RNA polymerase II promoters.

The first aim, then, is to acquire samples of both promoter and non-promoter DNA that can be compared, used to gather statistics, to train and to test a detection system. The main goal of the project is to write a program that can read a small section (of 250 to 256 bases long) of DNA and determine whether this sequence represents a promoter region or not. The system, called NNPromoterFind1.0, must be able to overcome the complexities and the underlying relationship between smaller parts in this 256-base window by means of an artificial neural network. The resulting neural network must be able to work through large amounts of DNA data in a short enough period to still be useful. For actual data, with sequences spanning millions of bases, the detection should at the very least be able to check and classify 500 sequences per second. The accuracy of the system should improve on current systems so that promoters detected from unknown DNA can be trusted and used reliably in other genetic work.

Aims of NNPromoterFind1.0 are to:

be a working system,

have reliable detection (improved accuracy when compared to current systems),

maintain fast detection,

be able to handle a large amount of DNA data,

be expandable for future use.

1.5 Previously Proposed Solutions

Previous attempts have been made to solve this particular problem such as in Bajic, Seah, Chong, Krishnan, Koh and Brusic [7] and Knudsen [8]. Most of the work done in this project was based on the process Bajic et al. [7] used for their program, Dragon Promoter Finder (DPF[1]).

In short, DPF takes a sequence of 250 bases and determines the number of G and C bases to see whether this sequence is CpG rich, or CpG poor. (As explained by Cross [9] and Larsen et. al [10], CpG islands can be used as an indication of the presence of a promoter.) Two parallel processes are then applied, one if the sequence has been determined to be CpG rich, and the other if it is CpG poor. The sequence is broken up into smaller 5-base windows. Sensors are then applied to these windows to determine whether it is a promoter, intron or exon. The outputs of the three sensors are then combined by means of a neural network that does the prediction.

Promoter 2.0 written by Knudsen [8] uses a neural network that was trained by a genetic algorithm to do detection of well-known promoters like the TATA-Box.

[1] DPF – Dragon Promoter Finder

Other approaches include PromoterInspector [11], and NNPP2.1 [12]. All these systems showed potential, but suffered from limited accuracy (as can be expected from any automated detection process). When looking at the way these systems work it can be seen that the full 256-length sequence cannot be used as is. It has to be broken into smaller sections to determine which parts are relevant, and which are not.

1.6 Implemented Solution

1.6.1 How it was done

NNPromoterFind1.0 was designed, implemented and trained to look at a single DNA sample that is 256-bases long and to determine whether the sample is a promoter or not. Obviously before a network could be set up a lot of pre-processing and reliable data collection had to be done. After data collection, statistical methods were used to determine which sections in the 256-base sample contained the information that determines whether it is a promoter or not. These features were then used to train an ANN that does detection directly on the DNA sample.

1.6.2 Contrasts between the current and earlier approaches

NNPromoterFind1.0 does not determine whether a given sample is CpG-rich or CpG-poor. This was excluded in an attempt to build a single system that could be adapted and expanded in the future to do more than polymerase II promoter detection, and the CpG islands lose relevance in other research areas. However, nothing prevents us from doing CpG detection first and then training two ANNs, one for CpG-rich, the other for CpG-poor DNA.

The second difference between NNPromoterFind1.0 and previous solutions is that the entire 256-base sample was not used. After the pre-processing step smaller windows are identified that contain the useful information for detection. Only these smaller windows are used and combined by the ANN. This reduces the computational cost of promoter detection, as only relevant data is calculated; no time is wasted on in-between sections. Also, it allows us to gain insight into the promoter regions, and is useful in improving generalization.

A third difference is that much pre-processing is done for feature selection. The ANN is used to do detection directly, based on the selected features. Other programs like Dragon Promoter

Finder [7] use mathematical and statistical methods to do detection, and then simply combine the detectors with an ANN. With NNPromoterFind1.0 system the ANN is the detector.
Yet another difference is the fact that five classes will be defined instead of two or three. DPF [7] for example has a detector for introns, exons and promoters. NNPromoterFind1.0 includes splice sites in both the training and testing process - I2E and E2I splice sites were both included. As shown later in this document, splice sites share some features with promoters, but can be detected more reliably. Hence, if splice detection is done, and thus the splice samples eliminated as possible promoters, it means that more accurate promoter detection can take place.

1.6.3 Contribution

The work done in the project yields several useful contributions to this research field. At the very least the groundwork has been laid out for future work. All the work done to get samples of the different classes was done. A lot of statistics were also generated that can be used in future work on promoter detection. Lastly, and most importantly a complete ANN was written, trained and tested (NNPromoterFind1.0) that is a fast, reliable promoter detection program.

When compared to the current state of the art promoter detection programs a total performance increase of 10% was obtained. This 10% increase is the worse case obtained. Some of the experiments obtained a performance increase of up to 30%.

2 CHAPTER 2: TASK OVERVIEW

This chapter contains the functional breakdown of the project. The project is broken down into functional units, and it is shown how the functional units fit together. A brief description of what was done to accomplish each function is also given. The sequence of functional units also provides the sequence in which the greater problem was solved.

2.1 Overview of the Entire Process

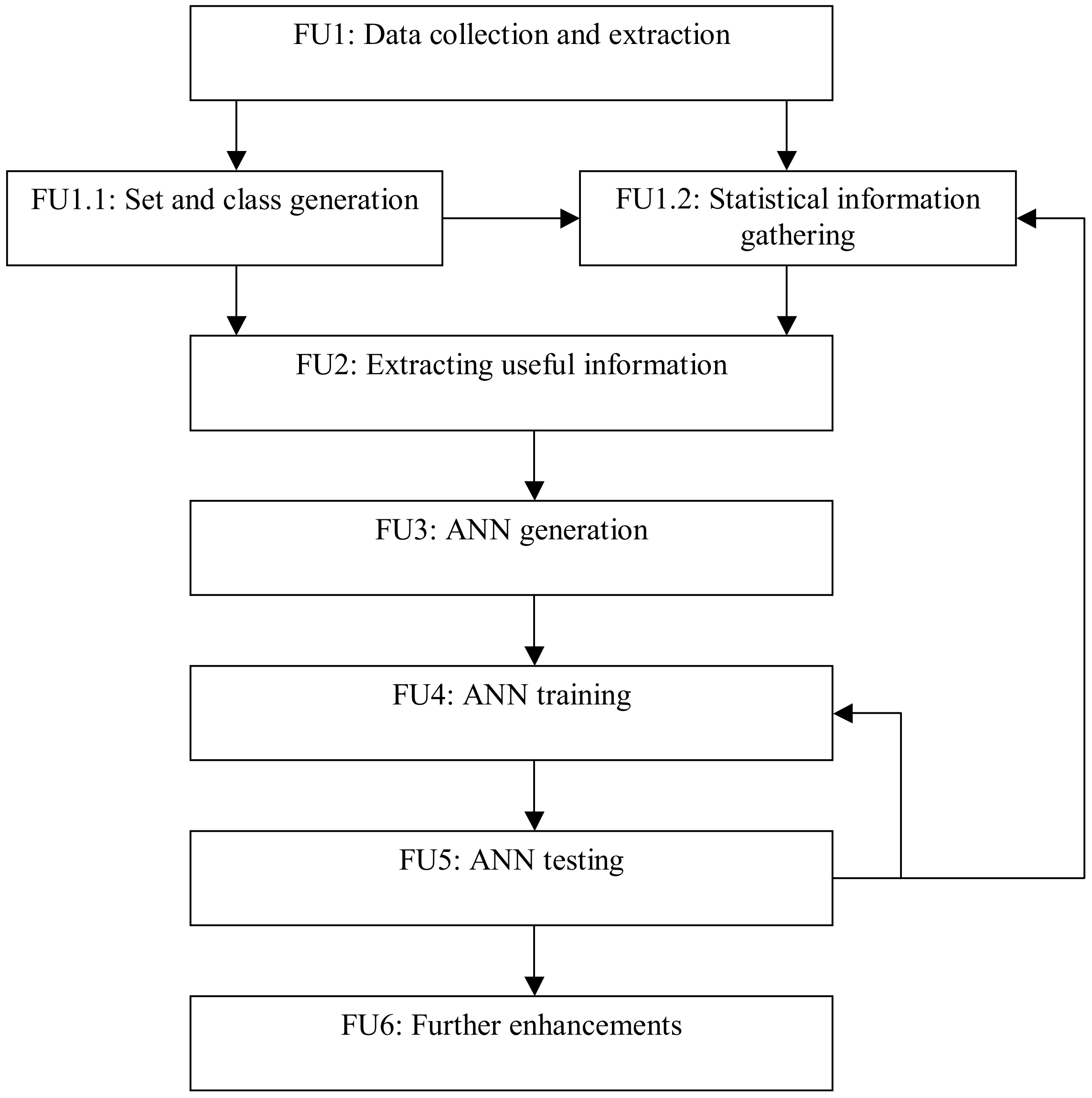
FU1: Data collection and extraction
FU1.1: Set and class generation
FU1.2: Statistical information gathering
FU2: Extracting useful information
FU3: ANN generation
FU4: ANN training
FU5: ANN testing
FU6: Further enhancements

Figure 4: Process flow diagram.

2.2 FU 1: Data Collection and Extraction

This functional unit handles the process for generating reliable, usable samples of five different classes. It is one of the most important units of this project since all the other units are built directly on the data obtained here. Incorrect data collection would have led to incorrect statistical conclusions. This would result in incorrect network set-up, incorrect training parameter selection and in the end incorrect detection. Care has been taken in this functional unit to obtain samples with varying features. Such samples provide improved generalisation, although some training-set accuracy might be lost. The final system should be able to recognise and detect a wide variety of promoters, even if some of them are not detected as easily or strongly as others. The alternative is to select fewer promoters and try to get a stronger detection on them. Better generalisation means more expandability with regards to different applications as well as different promoter types. Since promoter features are not 100% known, as seen in Section 1.3.3 where the TATA-Box is discussed, it is more desirable to design a system that does good promoter finding by looking at various aspects than to design one that can only detect a certain "known" box. The data collection functional unit consists of class generation, set generation, as well as statistical calculation processes.

2.3 FU 1.1: Set and Class Generation

In this functional unit the task of generating different classes and sets is handled. The main aim of this project was to differentiate between promoter features and non-promoter areas in DNA. Non-promoter areas include every possible DNA section that does not function as a promoter, including intron, exon, I2E- and E2I-splices. Samples of each of these classes have to be collected and stored so that they can be compared and studied. All the samples have to be the same length, 256 bases long, with distinguishing features at a specific position to ensure that they can be compared with one another. Comparisons had to be made in a given class as well as between classes.

The second step was to generate three different sets, each with a very specific function. A training set had to be made containing most of the available data, as defined by Bishop [13]. This set was used for the statistical analysis, and also for training the detection neural network. A second, smaller testing set had to be generated that could be used for designing the neural network. To see whether changes in network topology or training methodology changes system performance a set that was independent of the training set was required. The test set fulfilled this function. Lastly a validation set was generated. The validation set contained less data than the other two sets, and was used only once. The only function of the validation set is to test the final system on data that originates from the same source as the training and testing sets, but to the moment of testing has remained undetected by the system. This means that the validation set has no influence on the training or tuning process of the system.

2.4 FU 1.2: Statistical Information Gathering

The raw DNA data starts out as sequences of millions of bases long. These were processed into different samples of various classes and divided into different sets. The next step was to gather meta-data on samples to attach some mathematical meaning to each class. Statistical methods were employed to determine where certain bases occur in the sequences. The statistics were calculated for each of the five different classes separately so that they could be compared with one another. The average chance for each base type to occur at each of the 256 positions in the sample sequences had to be calculated and then compared amongst the classes to determine where certain bases occur more in one class than the others. One of the biggest problems with the statistical information is that there is practically no useful information when one looks at single base occurrence only. This is due to the fact that each base type occurs with a likelihood of about 25%, as shown in greater detail in Chapter 4. This implies that any one of the four possible base types is as likely to occur as any one of the remaining three. Hence, we studied the occurrence of base combinations, instead of single bases. Bases were grouped together in short sequences called n-tuples, where n represents the number of bases grouped together (for example a 3-tuple is three bases strung together). Neighbouring n-tuples were selected such that they overlapped, shifting a single base at a time. This drastically increased the number of statistical calculations made. When looking at four single-base occurrences, there are 256 (number of positions) multiplied by 4 (number of possible bases), hence 1024 numbers that have to be calculated. The number of possible n-tuples, on the other hand, is 4^n). This meant that a very large number of statistics had to be calculated, but also implied a larger number of values that could be used to search for distinguishing features. Note that the statistics are gathered only on the training set, to make sure that the test set and validation set were completely isolated from the training methods of the system.

2.5 FU 2: Extracting Useful Information

The previous section briefly described what statistics were generated, but still doesn't explain how this large quantity of numbers is of any practical use. A reliable, mathematically sound method had to be used to compare the statistics of the different classes to one another in such a way that distinguishing features for the promoter class could be identified. Various methods were attempted: some direct comparisons were made, the number of occurrences were counted and compared, weighted averages were taken etc. The most useful extraction method was found to use entropy as described by Bishop [13], and originally developed by Shannon (1948). Entropy measures the information content of data. The entropy was calculated for each possible position (1-256) in the sequence, and each n-tuple (1-4^n). The lower the entropy value, the more significant a specific feature. Using entropy, the statistical data was transformed from numbers to useful information, determining which n-tuples at which positions could be used to identify promoter sequences. These features were extracted and stored, for use in the neural network as explained in the following section.

2.6 FU 3: Generate an Artificial Neural Network

As mentioned in Section 1.4, one of the aims of this project was to develop efficient promoter detection. One of the well-known pattern recognition methods for this purpose uses an artificial neural network (ANN). The full mathematical background on ANNs is not given here, but can be found in various sources such as Bishop [13], Negnevistky [14] and Russell & Norvig [15]. An ANN had to be designed that could take a 256 base sequence, compare it with the entropy–based features and give a simple yet useful indication whether the sequence is a promoter or not. A simple 2-layer, feed-forward ANN was developed that was trained using the back-propagation training method. To speed up the system an adaptive learning rate as well as a momentum term, both described by Negnevistky [14] was implemented. The inputs to NNPromoterFind1.0 are the entropically extracted features. The hidden and output layers use sigmoid transfer functions. The output is a single binary output that approximates a 1 for a promoter sample, and a 0 for a non-promoter. Both the input and the hidden layer have biasing neurons to make sure that the outputs are biased correctly. NNPromoterFind1.0 is described in more detail in Chapter 5.

2.7 FU 4: Training Process

NNPromoterFind1.0 was trained with the training set generated in FU 1.1. Each sample of the training set was shown to the ANN, and the error was calculated at the output neuron. The error was combined for all the samples in the training set, and then after each sample was introduced to NNPromoterFind1.0 one time, the error was propagated backwards, and all the necessary weights updated. This process was repeated until a sufficient reduction in the combined error was obtained, or a constant minimum error was found.

2.8 FU 5: Testing of the System

The last step was to test the ANN to check whether accurate detection is possible. For this step, the test set was used for the first time. Note that no sample in the test set was used to determine statistics, generate features with entropy or train NNPromoterFind1.0, thus providing total isolation. Each sample in the test set was introduced to the ANN, and the ANN then determined whether the sample is a promoter or not. The number of mistakes made was noted to determine how accurate the system is. As with any automated detection process there are various ways to calculate how well the system performs. The method used was originally designed by Bajic et al. [7] to compare the performance of a promoter detection system. This method calculates a sensitivity value as well as a positive prediction value to determine accuracy.

As can be seen in the functional flow diagram (given at the start of this section) the entire process had to be repeated from various levels: sometimes it had to be repeated from as far back as statistical extraction and other times only more training was required. This was done to make sure that different network topologies and different selected features were used to get the best final results.

2.9 FU 6: Further Enhancements

The last step of this project was to investigate possible future improvements to make the system more accurate and useful. Some of these methods were tested whilst others were simply motivated. These include using multiple networks, getting the system to read entire DNA strings and doing full gene-finding instead of just promoter detection. Chapter 7 discusses these options in more detail.

3 CHAPTER 3: DATA EXTRACTION

3.1 Introduction

In this chapter a full description of the data extraction process is given, as well as an explanation of how the source data files should be read and interpreted. Obtaining reliable data is one of the most important steps for any automated detection program because the entire behaviour of the system is based on the data samples. Faulty samples, or samples that do not represent the whole truth will lead to an inaccurate or even totally useless detection system. For that reason great care was taken to obtain accurate samples for generating the detection system.

3.2 Attaining promoter data

Because the system is designed to detect promoters, the first step was to get good promoter data. There is one fundamental problem in getting promoter samples, and that is that it cannot be extracted from raw DNA data using another automated program. If other automated systems were used to extract the promoter sequences, any system trained with that data would only look for the same distinguishing features that the source system used instead of looking for distinguishing features of true promoters. The point is that any system trained on data extracted by another automated system will be at the best as accurate as the previous system, and thus of no additional use. Therefore, promoters that have been experimentally determined by biochemical means should be used. The Eukaryotic Promoter Database (EPD[1]) [16] contains exactly that: promoters that were experimentally determined. This also happens to be the source of the promoter data used to train the Dragon Promoter Finder system developed by Bajic et al. [7]. The EPD provides eukaryotic promoter sequences of different species with very specific formatting. The length of the sequence is defined by giving the number of bases upstream and downstream of the TSS that has to be extracted. Since it was already decided that 256-base sequences would be used for this system, only the position of the TSS had to be chosen. Since there might be some promoter information downstream of the TSS it was

[1] EPD – Eukaryotic Promoter Database

decided to include 55 bases downstream in the samples, and thus 200 bases upstream. 200 upstream plus 1 TSS plus 55 downstream gives a total sample length of 256 bases. As explained in Section 1.3.3, the TATA-Box that is the most commonly accepted promoter is found at position –50 to –20, but the full 200 bases were extracted to make sure that possible unknown promoter features will be captured. A total of 1871 *Homo sapiens* (human) promoter sequences were downloaded from the EPD. These 1871 samples were all stored in a single text file called "HSV2.txt". The downloaded samples are available on the CD that accompanies this dissertation, and four examples are given below:

```
>EP17030 (-) Hs snRNA U1 (pU1-6); range -200 to 55
GGGGCTGACGTCTTCGCCACTGGCTGTTTCACCACGAAGGAGCTCCCGTGCCGTGGGAGC
GGGTTCAGGACCGCTGGTCGNACCTGAGGGTCCCAGCTGTGTGTCAGGGCTAGGAAGGCT
CGGGGGTGCGCGGGGCAAGTGACCATGTGTGTAAAGGGTGAGGTATATGGAGCTGTGACA
GGGCAGAAGTGTGTGAAGTCATACTTACCTGGCAGGGGAGATACCATGATCACGAAGGTG
GTTTTCCCAGGGCGAG
>EP17031 (-) Hs snRNA U1 (pHU1-1); range -200 to 55
GAGGCTGCTGCTTCGCCACTTGCTGCTTCACCACGAAGGAGTTCCCGTGCCCTGGGAGCG
GGTTCAGGACCGCTGATCGGAAGTGAGAATCCCAGCTGTGTGTCAGGGCTGGAAAGGGCT
CGGGAGTGCGCGGGGCAAGTGACCGTGTGTGTAAAGAGTGAGGCGTATGAGGCTGTGTCG
GGGCAGAGGCCCAAGATCTCATACTTACCTGGCAGGGGAGATACCATGATCACGAAGGTG
GTTTTCCCAGGGCGAG
>EP17036 (+) Hs snRNA U2; range -200 to 55
CCGGGAACGCCGAAGAAGCACGGGTGTAAGATTTCCCTTTTCAAAGGCGGGAGAATAAGA
AATCAGCCCGAGAGTGTAAGGGCGTCAATAGCGCTGTGGACGAGACAGAGGGAATGGGGC
AAGGAGCGAGGCTGGGGCTCTCACCGCGACTTGAATGTGGATGAGAGTGGGACGGTGACG
GCGGGCGCGAAGGCGAGCGCATCGCTTCTCGGCCTTTTGGCTAAGATCAAGTGTAGTATC
TGTTCTTATCAGTTTA
>EP15024 (+) Hs histone H3.3; range -200 to 55
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNGCGAGCCTTCCCT
CCATTGTGTGTGATTGGCTGCGCGCGGCGGGGGCGGGGCGGCGTGTGTTGGGGGATAGCC
TCGGTGTCAGCCATCTTTCAATTGTGTTCGCAGCCGCCGCCGCGCCGCCGTCGCTCTCCA
ACGCCAGCGCCGCCTC
```
Extract 1: Example of promoter samples.

The first line of each sample, starting with a ">" gives the name of the gene where the promoter was found. Each following line contains the base sequence of the DNA, with each line containing a maximum of 60 bases. This 60-base maximum per line format was adopted as the standard throughout the rest of the project, since it was easier simply to make sure all the generated samples look the same than to attempt to reformat each promoter sample. These adopted formatting rules also included the first line starting with a ">" where any description text could be inserted.

The fourth sample starts with a long sequence of "N" characters, which is obviously not one of the four possible base types A, C, G or T. This is how the EPD indicates sequences that are unknown or not transcribed. Many of the 1871 downloaded promoter samples started or ended with these “N” characters. There was no way to get the true sequence values for those positions, so a method had to be devised that could generate statistics but could ignore the segments where the sequence was not known.

3.3 Obtaining non-promoter data

For the promoter data no actual work was required, since they could simply be downloaded. No such database exists for the non-promoter data that was required, so a different approach had to be taken: an automated method was used to extract the non-promoter data from sequenced DNA data. The first step was to simply get raw DNA data from the NCBI download site [17]. From this site the entire human chromosome DNA sequences could be downloaded. This is an enormous amount of data as can be seen from the table below:

Table 1: Chromosome file sizes.

Chromosome	Zipped File size (MB)
1	71.6
2	70.3
3	81.3
4	78
5	74.2
6	70.9
7	129
8	59.8
9	49.6
10	55.3
11	55
12	54.1
13	39.5
14	35.9
15	34.5
16	33.1
17	32.1
18	30.7
19	23
20	25.7
21	14.8
22	14.7
X	62.9
Y	7.76
Total	1203.76

The smallest chromosome – excluding the X and Y-chromosomes – chromosome 21, spans over more than 15000 pages, and over more than 28 million bases, and some of the longer chromosomes are up to eight times this length.

3.3.1 A closer look at the GBK file format.

The downloaded files are in GBK format. Such files contain not only DNA sequence data, but also some source, variation and annotation information. For illustration, a closer look will be taken at chromosome 21. The file starts with header information, describing the source of the data as well as when, where and how the sequencing was done. Different sections were sequenced by different institutes as shown in the example "source" sections.

```
LOCUS       NT_011512           28602116 bp    DNA     linear   CON 07-OCT-2003
DEFINITION  Homo sapiens chromosome 21 genomic contig.
ACCESSION   NT_011512
VERSION     NT_011512.9  GI:37558541
KEYWORDS    .
SOURCE      Homo sapiens
  ORGANISM  Homo sapiens
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
            Mammalia; Eutheria; Primates; Catarrhini; Hominidae; Homo.
REFERENCE   1  (bases 1 to 28602116)
  AUTHORS   International Human Genome Sequencing Consortium.
  TITLE     The DNA sequence of Homo sapiens
  JOURNAL   Unpublished (2003)
COMMENT     GENOME ANNOTATION REFSEQ:  Features on this sequence have been
            produced for build 34.0 of the NCBI's genome annotation [see
            documentation].
            On Oct 7, 2003 this sequence version replaced gi:29806267.
            The DNA sequence is part of the second release of the finished
            human reference genome. It was assembled from individual clone
            sequences by the Human Genome Sequencing Consortium in consultation
            with NCBI staff.
            COMPLETENESS: not full length.
FEATURES             Location/Qualifiers
     source          1..28602116
                     /organism="Homo sapiens"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:9606"
                     /chromosome="21"
     source          1..106559
                     /organism="Homo sapiens"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:9606"
                     /clone="21B49A22"
                     /note="Accession AP001464 sequenced by RIKEN Genomic
                     Sciences Center"
     source          106560..218367
                     /organism="Homo sapiens"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:9606"
                     /clone="RP1-133G21"
                     /note="Accession AJ239321 sequenced by Max Planck
                     Institute for Molecular Genetics"
```

Extract 2: Example of GBK file start.

The header information gives authentication information, and is not of direct use for the extraction process. The next section in the file contains all the possible variations:

```
variation       13212
                /allele="A"
                /allele="G"
                /db_xref="dbSNP:4086405"
variation       13230
                /allele="C"
                /allele="A"
                /db_xref="dbSNP:7509718"
variation       13374
                /allele="A"
                /allele="G"
                /db_xref="dbSNP:4086404"
variation       13380
                /allele="C"
                /allele="A"
                /db_xref="dbSNP:7510510"
```

Extract 3: Example of GBK variation of bases.

This means that the sequence is given with a specific base type, but that a different base type can possibly occur at that position. This information could be very relevant to the extraction process, since a sequence might be extracted as …TTGTC… but the "G" base in the middle might have a "C" and an "A" variation. This complicates the extraction process and it was decided that the variations would not be used for the sake of simplicity. The main aim of this project is to get accurate promoters and the assumption has been made that if there are no variations on the promoter data some small variations on the non-promoter samples can be tolerated.

The next section in the file contains the coding gene and mRNA sequences. mRNA is a complementary copy of a gene according to Tamarin [2]. In other words the mRNA sequence is the sequence that should be transcribed to RNA for the transcription process and the position where the mRNA starts can also be seen as the TSS. Although they technically differ, for this project "mRNA", "a gene" and the "section that has to be transcribed" are seen as equivalents.

```
Gene encoding section:
gene            complement(1405307..1417342)
                     /gene="STCH"
                     /note="Derived by automated computational analysis using
                     gene prediction method: BestRefseq. Supporting evidence
                     includes similarity to: 1 mRNA"
                     /db_xref="GeneID:6782"
                     /db_xref="LocusID:6782"
                     /db_xref="MIM:601100"
```

Extract 4: Example of GBK gene positioning.

Followed by the mRNA encoding section:

```
     mRNA            complement(join(1405307..1408476,1409844..1410011,
                     1412391..1412604,1415395..1415735,1417287..1417342))
                     /gene="STCH"
                     /product="stress 70 protein chaperone,
                     microsome-associated, 60kDa"
                     /note="unclassified transcription discrepancy; Derived by
                     automated computational analysis using gene prediction
                     method: BestRefseq. Supporting evidence includes
                     similarity to: 1 mRNA"
                     /transcript_id="NM_006948.2"
                     /db_xref="GI:24431965"
                     /db_xref="GeneID:6782"
                     /db_xref="LocusID:6782"
                     /db_xref="MIM:601100"
```

Extract 5: Example of GBK mRNA positioning.

It can easily be seen from this example that the gene start- (1405307) and stop- (1417342) position are the same as the first and last positions given in the mRNA section. The gene section gives the locations of the entire gene, while the mRNA section gives the exon sections. This means that exon, intron, I2E and E2I positions can be established from these numbers. In the example above the TSS is at base position 1405307 in the sequence. As displayed all the sequence sections are given as a pair of numbers separated by two full stops: "XXXX..YYYY". XXXX is the start of the exon region and YYYY is the end. This also gives the I2E and E2I splice regions accordingly. The first number is the TSS. The second number will then be the first exon-to-intron splice site. From the third number every second number given is an I2E splice, and every other number an E2I splice.

In our notation, the format is therefore ((TSS..E2I,I2E..E2I,I2E..E2I)).

The next section in the file is the most important one, containing all the actual sequence information that spans most of the file:

```
ORIGIN
        1 catgtttcca cttacagatc cttcaaaaag agtgtttcaa aactgctcta tgaaaaggaa
       61 tgttcaactc tgtgagttaa ataaaagcat caaaaaaaag tttctgagaa tgcttctgtc
      121 tagtttttat gtgaagatat ttccattttc tctataagcc tcaaagctgt ccaaatgtcc
      181 acttgcagat actacaaaaa gagtgtttca aaagtgctca atgaaaagga atgttcagct
      241 ctgtgagtta aatgcaaaca tcacaaataa gtttctgaga atgcttctgt ctagtttta
      301 tgggaagata attccgtgtc cagcgaaggc ttcaaagctt tcaaaatatc cacttgcaaa
      361 ttctacaaaa agagtgtttc aaagctgctt tatcaaaaga aagtttcaac tctgtgagtt
      421 gaatgtgcac atcacaaaga agtttctgag aatgccttca gtctggtttt tatgtgaaga
      481 tattcccttt tccaacgaaa gcctcgaagc tgtccaaata tccacttgta agtgctgcaa
```

Extract 6: Example of GBK actual sequence.

Each line starts with the base position followed by 60 base types. The 60-bases per line is exactly the same as the format used by the EPD for storing the promoters as was seen in Section 3.2, with the only exception that the EPD did not start each line with a base number.

3.3.2 Developing an extraction program.

Since gene and mRNA data was available directly in the GBK files a program could now be written that can read through these enormous files and generate samples for each of the non-promoter classes. Five classes were extracted: Introns, Exons, I2E, E2I as well as mRNA start samples. The mRNA start class was extracted, and some statistics were computed from it, but it was not considered a regular non-promoter region since it could contain a promoter. It was also not used as a promoter, because of the automated annotation process used to mark it. It can be seen that most of the gene annotation was done by a program called BestRefSeq. This might cause concerns, as one now has to assume once again that this program is accurate and correct. It was assumed that this program could be trusted to generate splice sites that are accurate enough for the purposes proposed as it is only used for the non-promoter data. Also, a very large set of non-promoter sequences were generated, far more than the 1871 promoter samples, and therefore if some of them are not 100% correct the average error should be low enough for the construction of a reliable system.

The samples extracted with the program Extract.exe (extract.c - Addendum 1) will always be 256 bases long. The sample will also have the same format as the promoters downloaded from the EPD, in other words it will be –200 to +55 from the “marking” feature. mRNA start positions, I2E and E2I splices will always be exactly at position 0. Introns and Exons don't have specific distinguishing features at position 0, but will still be 256 bases long.

The samples extracted for the different classes should not contain the same base information; therefore a length detection process was created to determine whether an 256-length sample can be extracted without crossing into neighbouring samples. The main reason for keeping the regions non-overlapping is to prevent the same features from occurring in different classes. Although the positions will differ it is still preferable to keep the samples non-overlapping for future use where the position information might not be used, and only the actual base information will be used. This is best described with the help of a diagram, and is also given in slightly more detail in Addendum 2. Figures 5-A and B show the DNA (nucleotide detail excluded) and the intron and exon regions. The splice markings in Figure 5-B are the values read directly from the GBK file. In Figure 5-C the mRNA sample is shown. The sample starts 200 bases upstream from the actual mRNA value given in the GBK file, and ends 55 bases downstream of it. The same applies for E2I (Figure 5-D) and I2E (Figure 5-E) splice sites. The intron sample (Figure 5-F) and the exon sample (Figure 5-G) are any areas that are at least 256 bases long between two following splice sites.

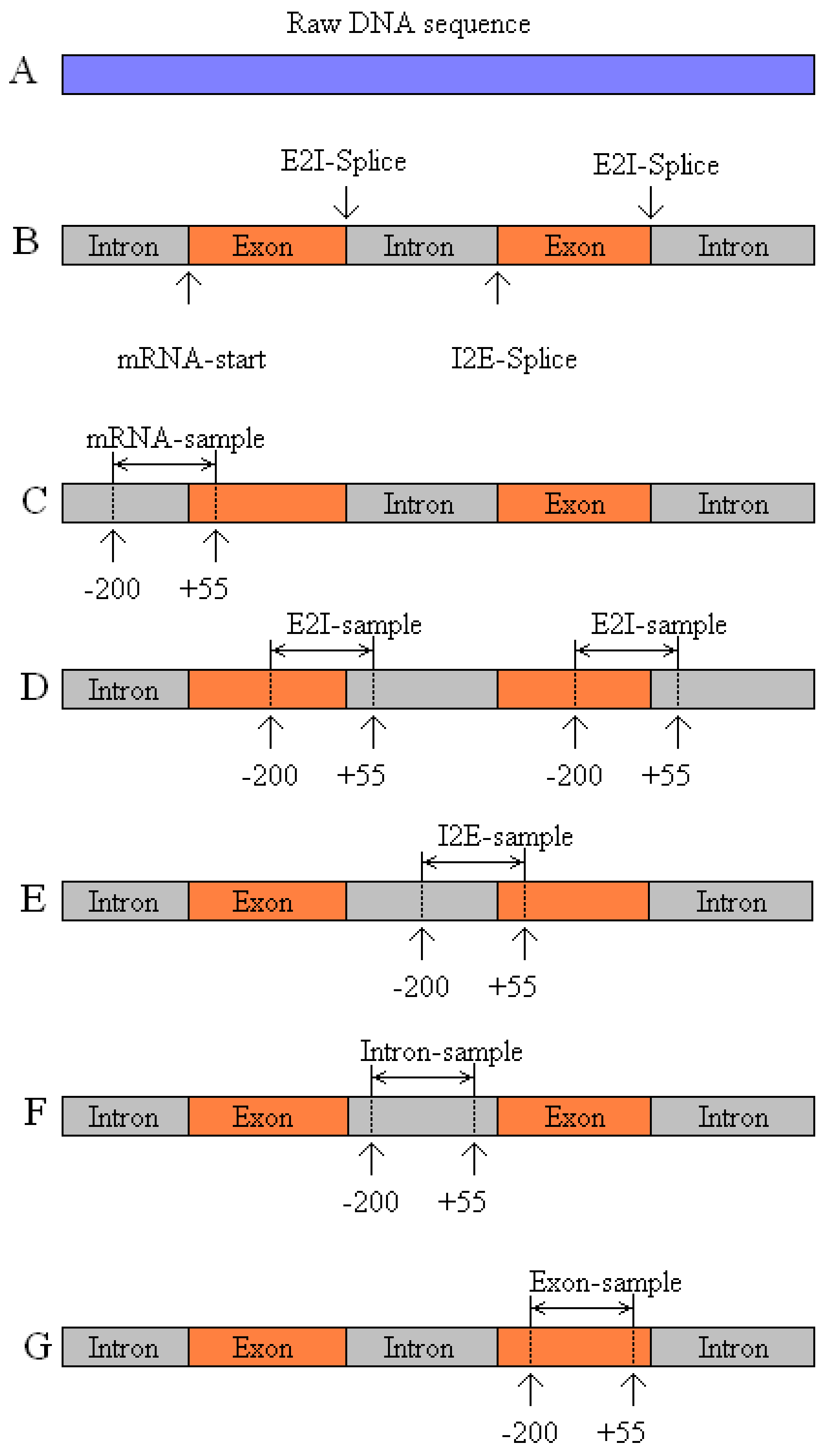

Figure 5: Different 255 base sequence selections for different sample types.

The extraction program is a multi-step process, with the simplified flow diagram, Figure 6 showing the broad steps taken.

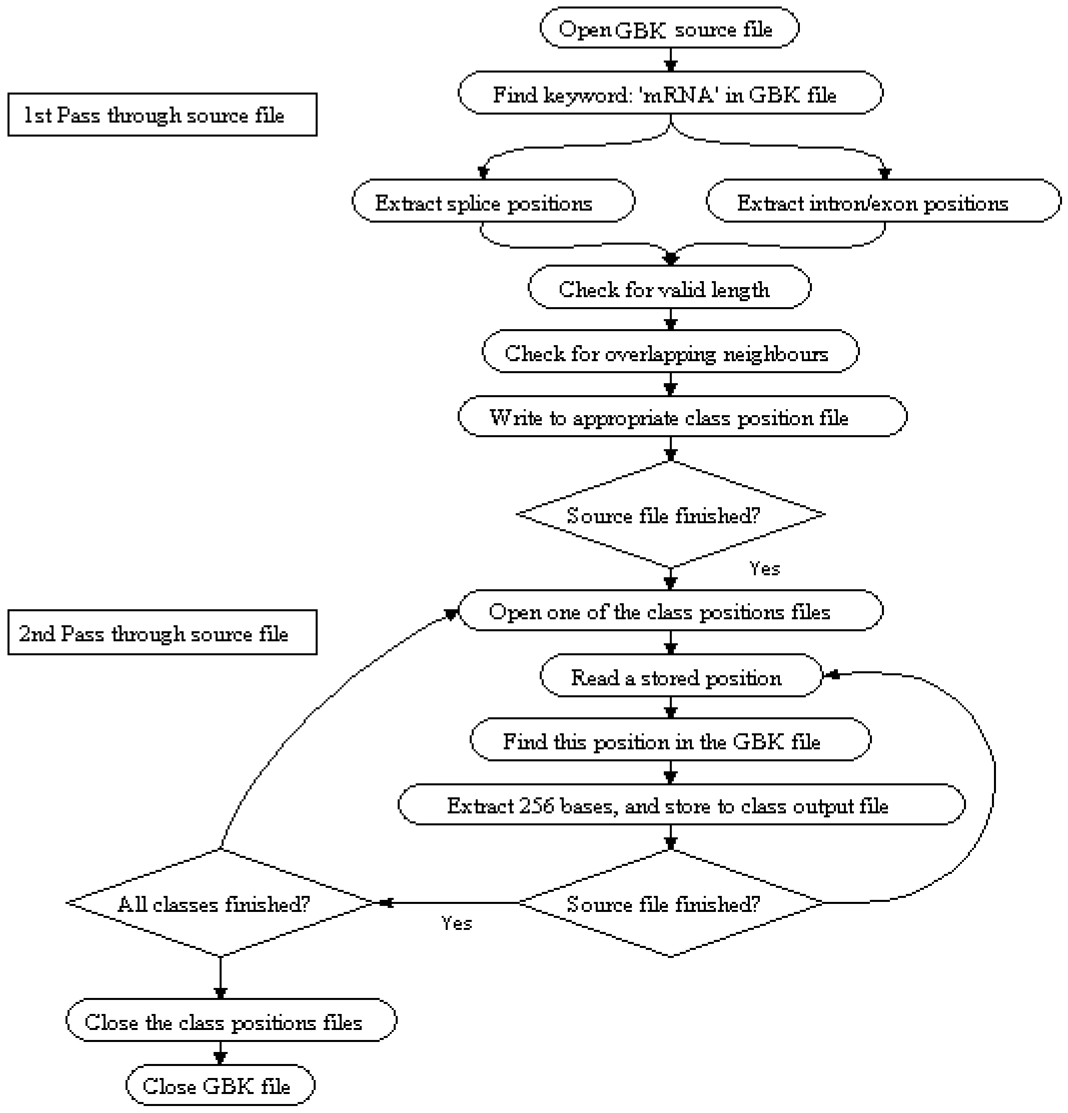
1st Pass through source file
2nd Pass through source file
Open GBK source file
Find keyword: 'mRNA' in GBK file
Extract splice positions
Extract intron/exon positions
Check for valid length
Check for overlapping neighbours
Write to appropriate class position file
Source file finished?
Yes
Open one of the class positions files
Read a stored position
Find this position in the GBK file
Extract 256 bases, and store to class output file
Source file finished?
Yes
All classes finished?
Close the class positions files
Close GBK file

Figure 6: Extraction program flow diagram.

As seen in this diagram the entire GBK file is processed twice during the extraction process. Initially, the mRNA and splice positions are read from the file, and stored in their adapted form (position –200) to five different files, one for each class. Positions are only stored if a valid 256 base length sequence can be extracted without passing over into neighbouring samples. The second pass is actually 5 passes, once for each class type, where the positions read from the first set of files are located in the sequence section of the GBK file. The samples are then read from the correct position and stored in a class-sample file in the same format as the promoter file hsv2.txt. The first line of each sample (where the promoter starts with a “>” symbol and the gene origin location) gives the position in the GBK file from where the sample was extracted.

The five files are:

“intronD.txt”, “exonD.txt”, “i2esplice.txt”, “e2isplice.txt” and “startseq.txt”

The extraction program extracts samples from one of the 24 (22 + X + Y) available chromosome GBK files at a time. The position text files that are created during the first run through the GBK file are rewritten every time the program is run. The sample files are expanded every time the program is run so that the total samples come from all the available source files. Table 2 gives the numbers of each sample type that was extracted from each of the used source files. The files for chromosome 1-5 and 12 as well as chromosome X and Y were not used, because these files were either corrupted during download, or had some unknown parameters that caused extraction to fail. Chromosome 9, 15 and 20 were also skipped for use as a final, second validation set. This second validation was never actually implemented as no further promoter samples were available that were not part of the training or testing set. These numbers were not the final number of samples used, because sample duplicates had to be removed first to ensure that the sets do not contain duplicates. After duplicate removal, and excluding the samples of chromosome 9, 15 and 20 there were 50450 samples left.

Table 2: Number of each sample for each class extracted from the different chromosomes.

Chromosome	Intron	Exon	I2E Splice	E2I Splice	mRNA start	Totals
5	1022	895	946	404	1016	4283
6	1017	974	1128	589	1165	4873
7	2400	1785	1996	954	2226	9361
8	823	653	695	309	824	3304
9	784	684	804	382	915	3569
10	1103	749	933	447	931	4163
11	969	970	1146	557	1327	4969
13	651	384	473	237	740	2485
14	708	620	766	391	974	3459
15	747	659	724	369	711	3210
16	934	863	893	464	1286	4440
17	1057	1078	1249	580	1627	5591
18	584	369	464	230	640	2287
19	862	1122	1389	715	1686	5774
20	648	578	655	293	912	3086
21	254	234	241	107	252	1088
22	505	510	545	225	726	2511
Totals	15068	13127	15047	7253	17958	68453

3.4 Set generation.

The final step before statistical data could be extracted was to split the extracted data into the three separate sets: a training set containing most of the data, a test set used to fine-tune the detection process and a validation set for final system testing. At this point there were exactly 1871 promoter samples in the file hsv2.txt and 50450 other samples in five different files. This suggests another interesting problem, namely that the ratio of promoter samples compared to the non-promoters. is 1871:50450, or about 1:27. The problem with this value is that the true ratio between promoters and non-promoters in the source Eukaryotic DNA is not known at all, and in all likelihood the ratio should be even larger, meaning much more non-promoters for each promoter. Furthermore, when designing a detection system or classifier like this the ratio of samples can influence the final decision boundaries. Usually the method used to overcome ratio problems is to pre-bias the system or to include ratio calculations into the training process, but for that the true ratio needs to be known. To help overcome this problem two separate sets were generated: the first set contained all the available samples, while the second set was generated so that there were exactly 1871 samples of each class (1871 being the smallest number of samples between all six class types). The first group was generated to get the more accurate representation of the promoter:non-promoter ratio, while the second group was generated to have equal numbers of samples for each class. From here on the first set will be referred to as the large set, and the second set will be referred to as the equal sized set.

Table 3: Number of samples per set (2 different groupes used).

	Train	Test	Validate	Total
Promoter	1294	383	194	1871
Intron	7669	2286	1006	10984
Exon	3760	1755	442	5289
I2E Splice	8096	2309	956	11338
E2I Splice	6398	1087	834	8987
mRNA	9857	2742	1253	13852
				50450

Promoter	1300	350	221	1871
Intron	1300	350	221	1871
Exon	1300	350	221	1871
I2E Splice	1300	350	221	1871
E2I Splice	1300	350	221	1871
mRNA	1300	350	221	1871
				9355

The program used to split up the extracted source files into the three different sets is given in Addendum 3, with the flow diagram given below in Figure 7.

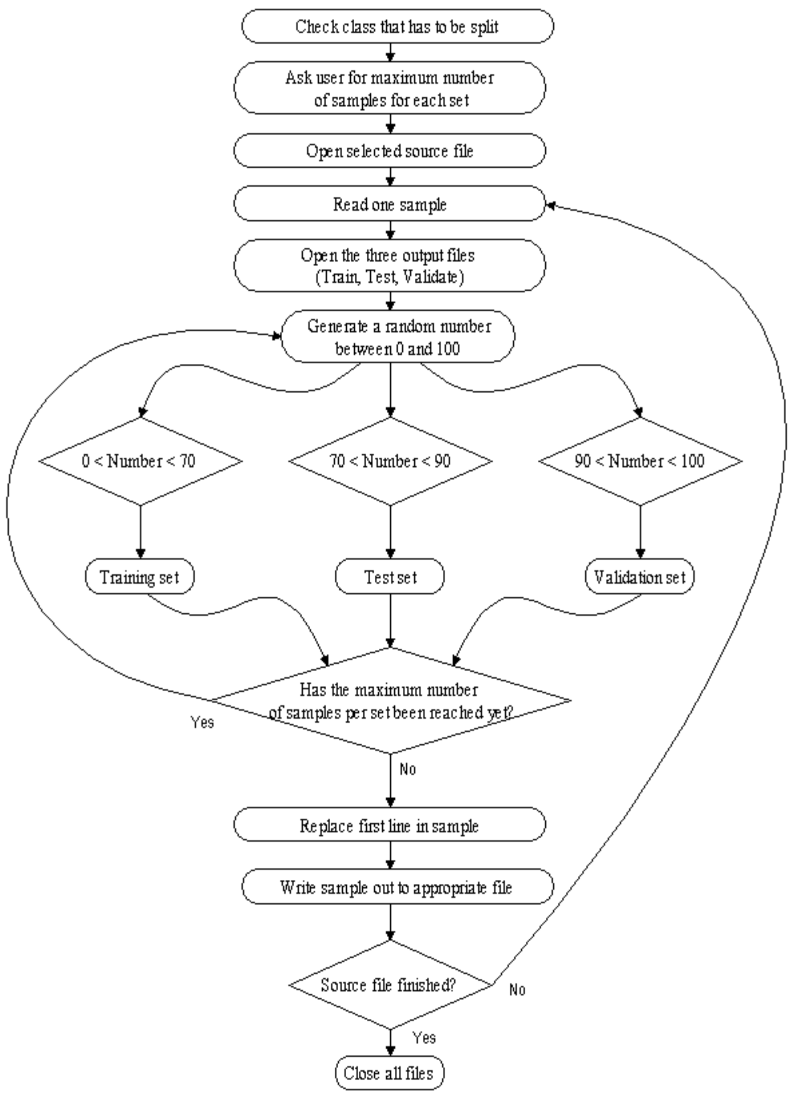
Check class that has to be split
Ask user for maximum number of samples for each set
Open selected source file
Read one sample
Open the three output files (Train, Test, Validate)
Generate a random number between 0 and 100
0 < Number < 70
70 < Number < 90
90 < Number < 100
Training set
Test set
Validation set
Has the maximum number of samples per set been reached yet?
Yes
No
Replace first line in sample
Write sample out to appropriate file
Source file finished?
No
Yes
Close all files

Figure 7: ClassSets program flow diagram.

This program only operates on one class file at a time, in order to minimize its complexity. At the start of the program the user selects the class that has to be split into sets. The samples are read one by one from the source file and randomly distributed between three files, one for each set. The program automatically creates a subdirectory for each of the three classes, and creates a text file with the same name as the original source text file. A random number is generated and used to select the output set in such a way that 70% of all the samples go to the training set, 20% go to the test set and the remaining 10% go to the validation set. If the maximum number of samples for each set has already been reached a new number is selected. The first line of each sample contains descriptive information: for promoter, the gene that they activate and for non-promoters, the position in the chromosome file from which they were extracted. This information is subsequently replaced by a single number, showing the class to which the sample belongs (0 indicates non-promoters while 1 indicates promoters).

3.5 Outputs of extraction.

The final outputs of the extraction process were eighteen files, six classes in three sets. The file names used for each class amongst the three different sets were identical, but the files were stored in three different directories.

The three directories:

Train, Test and Validate,

each containing the six text files:

hsv2.txt, intronD.txt, exonD.txt, i2esplice.txt, e2isplice.txt and startseq.txt.

Each text file contained a number of samples, ranging between 221 and 9875 samples per file. Each sample in the format:

```
> <Promoter / Non-promoter>
Nucleotide 1   to 60
Nucleotide 61  to 120
Nucleotide 121 to 180
Nucleotide 181 to 240
Nucleotide 241 to 256
> 0
tattacaagaaatggtttgaggggcaccaaatagctcagcaccacaagctcaatgtgttc
ttcactctcgcctaattggaatagtgcacggcaccagtaagattccccatcttcctccaa
aagttgtgttatcttcagctttgtttcgttagcatatgttccattgatcacatatttatt
catttgaacaccaacaggaacctaatatgaggagacattaaaatccattcctatcatagc
acataaaagatacatg
```

Extract 7: Final sample format.

Each file also starts and stops with a new line, which is required for compatibility with programs using these files later to determine statistics.

4 CHAPTER 4: STATISTICAL INFORMATION GATHERING

4.1 Data reduction.

After data extraction, the number of bases was reduced by several orders of magnitude. The raw chromosome data contained more than 26 million bases each, so in total there were more than 26 000 000 * 24 = 624 million bases. The largest training set (see Table 3, Section 3.4) contained only 37074 samples, so only 37074 * 256 = 9490944 bases. This is already an immense reduction in data, but it still does not provide any directly useful information as to what the differences between promoters and non-promoters are. Doing a base-by-base comparison between the promoter samples shows that there is not a very good correlation between them (see Section 1.3.3. for the reason) and the same is true for the non-promoter samples.

To be able to find the sections in the 256-base window that describes promoters it was required to do relevant statistical analysis on the occurrence of the four different nucleotides at the 256 different positions in the file.

4.2 Base-per-position statistics.

A program was written that could read in the six text files created in the data extraction process and could generate a number of arrays that contain the occurrence percentages for each n-tuple (from lengths n=1 to n=8) at each position in the sample. The program consists of a Matlab section as well as a C-section, which are given in Addendum 4 and 6 respectively. The Matlab program, pullData.m, generates the arrays, and calls the C–compiled program statistics.c, as a Matlab function. Most of the actual computation is done by the C-section because the file handling and computation time in C is much faster than in Matlab. Along with the arrays a very large text file is also created that simply stores all of the base count information. The number of times each base occurs at each position in each of the samples is stored one after the other in the text file. The reason why this very large linear file is generated is because the actual base extraction process takes a long time, so storing it helps when re-using the data. This text file is called "baseCnt*N*.txt" where *N* is the n-tuple length (1 for base-by-base). A second Matlab program pullHalf.m (given in Addendum 5) was written so that it is almost identical to pullData.m, with the exception that pullHalf.m does not run the C program, but instead extracts the required arrays from the baseCnt*N*.txt text file. Note that the size of this text file increases exponentially with n, resulting in very large files for n larger than six. The second program cannot be used unless pullData.m had already been used at an earlier stage to create the text file. (IN) When calling the Matlab programs pullData.m or pullHalf.m the program prompts the user to enter the length of n. This is required to ensure that the arrays are the correct size.

All the outputs generated by the Matlab programs are temporary, and stored as Matlab variables in memory only; they are never stored in file. The arrays generated by these two programs are identical, the only difference is that one (pullData.m) first generates the baseCnt*N*.txt text file from the class text files generated in the extraction process, while the second (pullHalf.m) only uses the baseCnt*N* .txt text files.

The following six arrays were created, one for each class:
PositionDataIntron, PositionDataExon, PositionDataE2I, PositionDataI2E, PositionDataPromoter, PositionDataMRNA.

Each Array has 4^n+1 rows and 256-n+1 columns. Each row keeps one of the 4^n possible combinations of base length n. All the data are stored as percentage of occurrences, not total number of occurrences. The last row (4^n + 1) is used simply as a double-checking measure, containing the sum of each column. This value should be 100 for every entry since every combination is stored and added; this is not true for the promoter class, however, because some of the promoter samples contain unknown bases (marked as N in the sequence) that do no contribute to the percentage. The 256-n+1 rows correspond to the possible positions at which an n-tuple of length n can occur. For example, there are 256-1+1 = 256 possible positions to store a single base, but there are only 256-5+1 = 252 possible starting positions for a 5-tuple.

Further arrays generated were CountIntron, CountExon, CountI2E, CountE2I, CountPromoter and CountMRNA, each with one row only and 4^n columns. They contain the total number of times (as a percentage) each n-tuple occurs, disregarding specified position in each of the classes. This was used to see if any n-tuple occurs more often, across all positions, in one of the classes.

The number of samples used from each of the classes to generate the statistics is also stored in the values a-e. The program flow diagram is given in Figure 8 below.

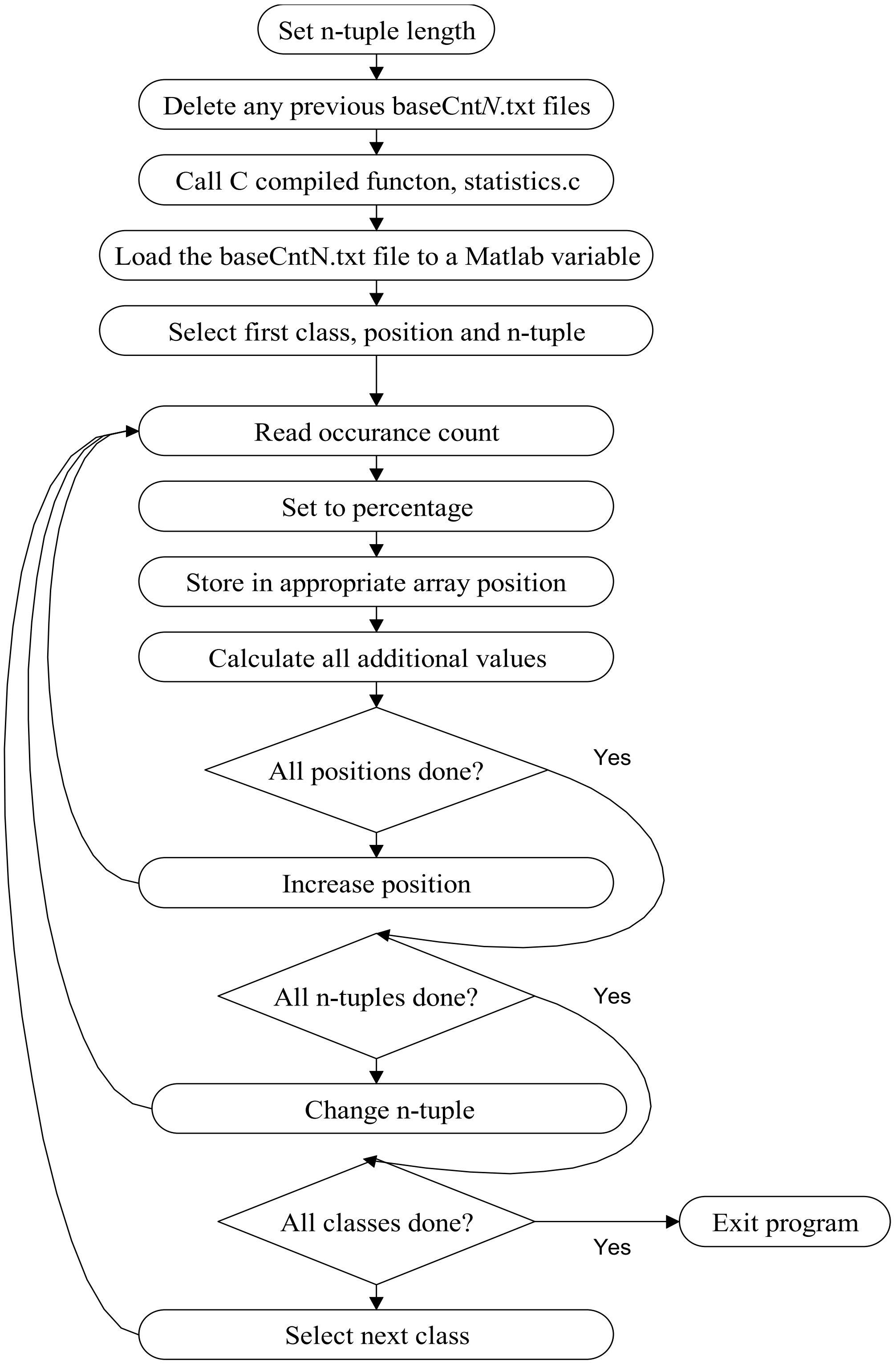
Set n-tuple length
Delete any previous baseCnt*N*.txt files
Call C compiled functon, statistics.c
Load the baseCntN.txt file to a Matlab variable
Select first class, position and n-tuple
Read occurance count
Set to percentage
Store in appropriate array position
Calculate all additional values
All positions done?
Yes
Increase position
All n-tuples done?
Yes
Change n-tuple
All classes done?
Yes
Exit program
Select next class

Figure 8: Matlab statistics extraction flow diagram.

4.3 Indexing and assigning n-tuple number.

The flow diagram for the statistics.c program is not given, but the full program can be seen in Addendum 6. NNPromoterFind1.0 works through the samples of each class one by one, counting the number of times each n-tuple occurs at each position. While the program is active the values are stored in a large memory array, and after the program exits it is stored as the text file baseCnt*N*.txt. This text file is then loaded into Matlab memory and the separate class statistics are extracted to the arrays in the format described earlier. As mentioned previously, the statistics program was developed to work with n-tuples ranging from length one to eight. For each increase in n, the number of data points stored is multiplied by four. A method had to be devised to keep track of each n-tuple, and where it should be stored in the array. To keep working with various length text strings directly is too complex, so a unique way to assign a number to each n-tuple was developed. The numbering is done in a base-4 mathematics system, directly using the four base symbols A, C, G and T with a value assigned to each of them. A = 0, C =1, G =2 and T = 3. The method for converting strings to a value is the same as converting a decimal value to a binary value. As the base-4 number for each string is unique, and ranges from 0 to n^4-1, the number was also used as the index value for all the arrays. This reduced complexity and keeps track of where n-tuple strings are stored. The function *seq2comb*() on the CD accompanying this thesis performs this mapping of a string of bases to a unique index, whereas the converse function is achieved with the function *comb2seq()*.

4.4 Statistical outputs and results

The programs described in the previous section take the six class text files, created during the extraction process, then do a base-by-base (or n-tuple by n-tuple) count and create large arrays containing percentage values of how often each n-tuple occurs. For the 1-tuple, for example, the following values were obtained:

Table 4: Single base statistics

Variable name:	%A	%C	%G	%T
countIntron =	28.1184	21.9026	21.6785	28.3005
countExon =	24.0237	25.7332	25.1529	25.0901
countI2E =	24.9636	24.7963	23.5679	26.6722
countE2I =	22.6953	26.7275	28.8017	21.7755
countPromoter =	18.2521	30.6166	32.0649	18.7389
countMRNA =	25.6163	24.7641	25.2560	24.3636

From these values one sees that bases C and G seem to be more likely to occur in promoter samples, whereas all four bases are almost equally likely in Exons. However, these differences are not large enough to form the basis for reliable promoter detection. In order to extract the meaningful statistics, further processing is required.

The Matlab function funcDraw.m was written to represent this information graphically. This helped to get a better view of the magnitude of the differences that occur within the data. The full function is given in Addendum 7, and the input arguments and function options are described in Addendum 8.

The main function of the graphs was simply to see where (which position) in the sequence there were significant differences between the classes that could be used as distinguishing features. Each of the lines in the graph represents one n-tuple (or base). The X-axis for the graphs contains the position in the samples, while the Y-axis contains the percentage occurrence for the selected n-tuple. The graphs for n-tuples 1 to 5 are given below for the equal sized training set (see Table 3, Section 3.4).

From all the n-tuples comparable information can be extracted. All n-tuples are more or less equally likely to occur, no matter the position in the sample. For promoters there seem to be strong features at position 200, as well as something between position 150 and 200 (approximately 175). For the mRNA start class the same positions seem to have features (175 and 200) but the features are less prominent than for the promoters. The I2E splice site shows strong features from 175 to 200, and E2I splices show strong features from 200 to 250. Introns and exons show no significant features whatsoever, and each n-tuple seems to be as likely to occur as any other. The occurrence percentage for intron and exon samples seems to be approximately $\frac{100}{4^n}$% for all positions, and all n-tuples.

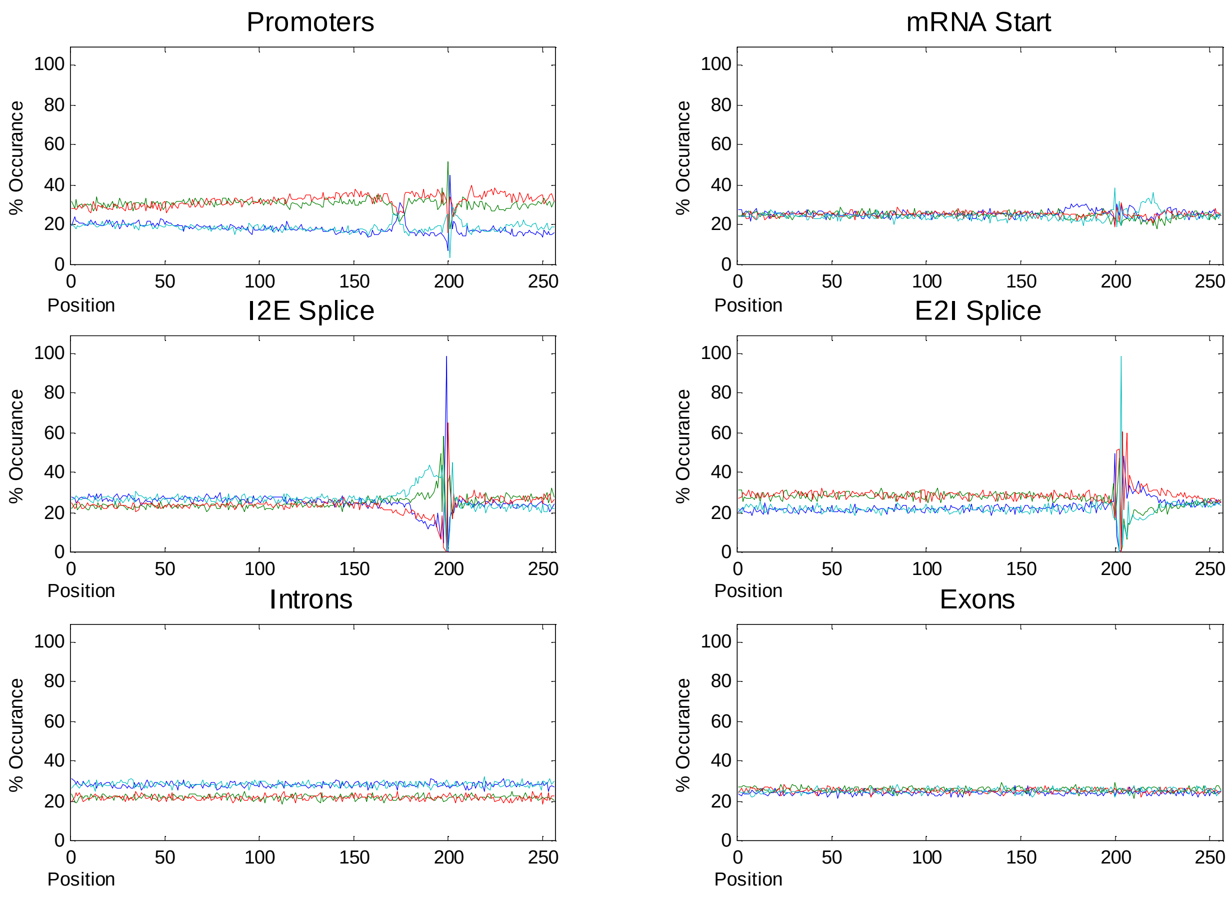

Figure 9: Matlab statistics 1-tuple,

showing the %occurrence of each of the 4 1-tuples position 1 to 256.

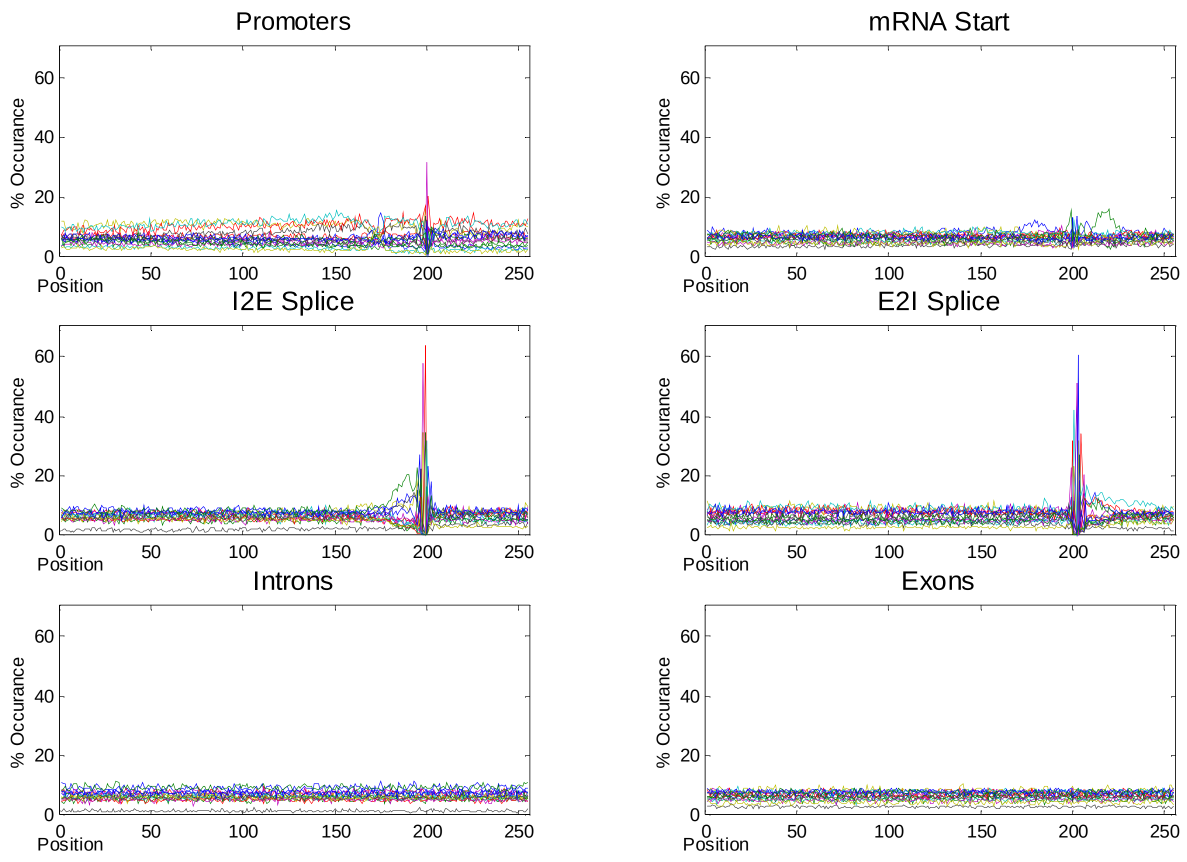

Figure 10: Matlab statistics 2-tuple,

showing the %occurrence of each of the 16 2-tuples position 1 to 255.

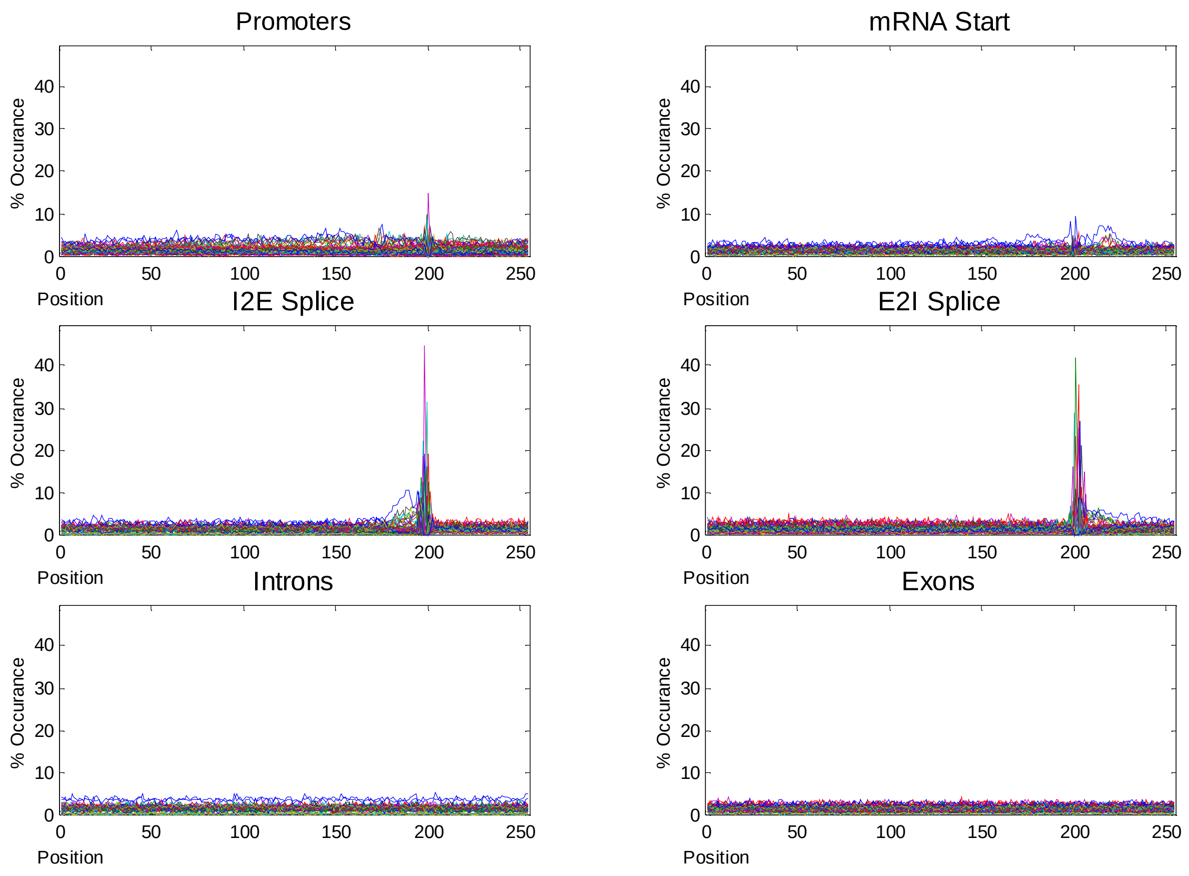

Figure 11: Matlab statistics 3-tuple,

showing the %occurrence of each of the 64 3-tuples position 1 to 254.

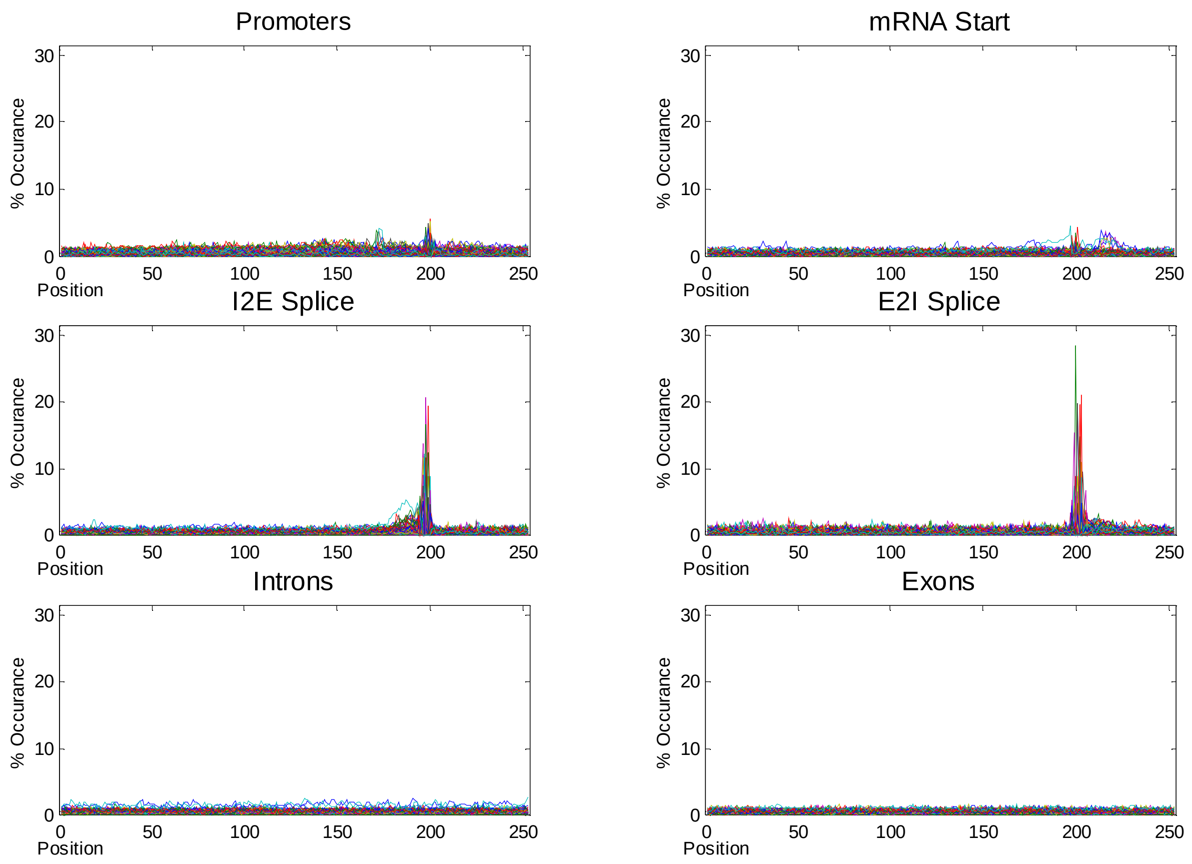

Figure 12: Matlab statistics 4-tuple,

showing the %occurrence of each of the 256 4-tuples position 1 to 253.

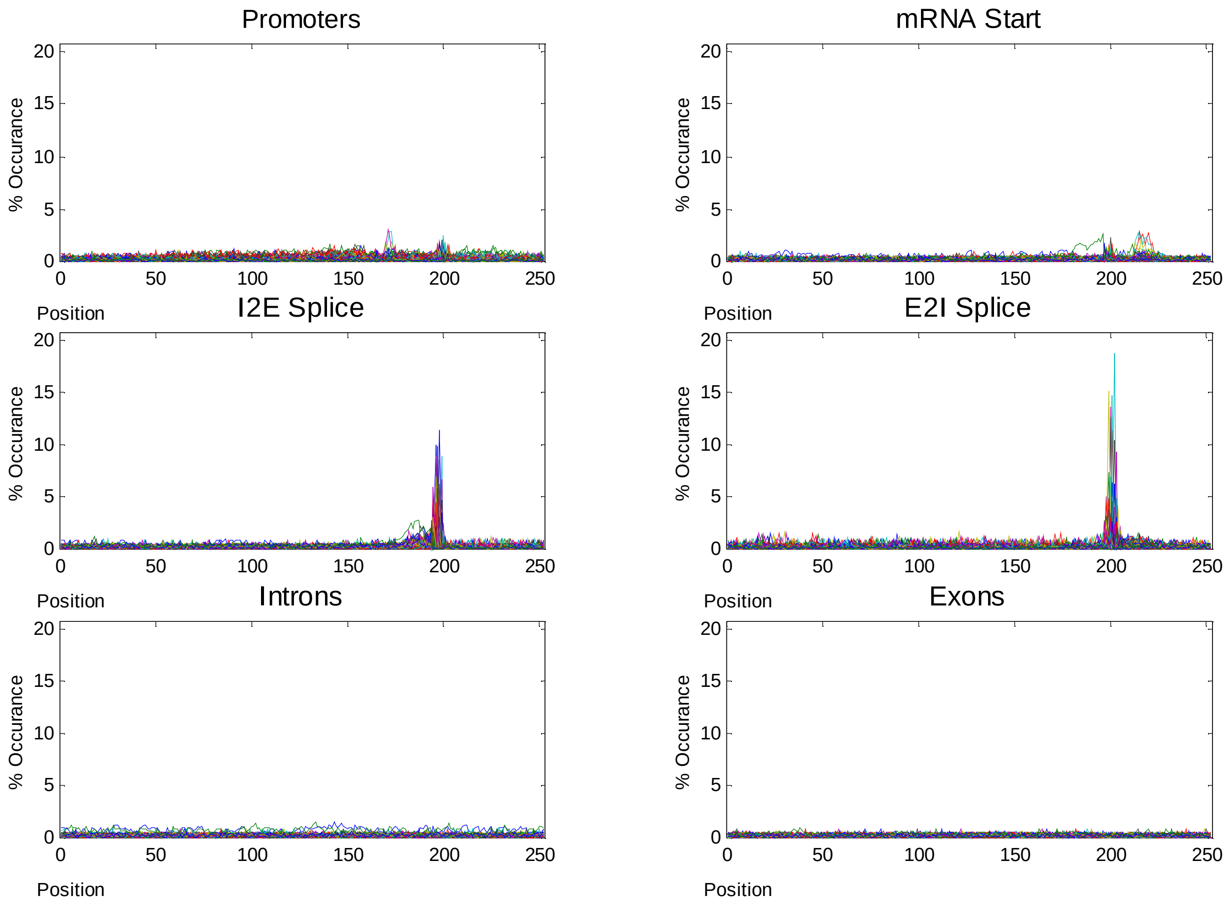

Figure 13: Matlab statistics 5-tuple,

showing the %occurrence of each of the 1024 5-tuples position 1 to 252.

The graphs for the other training sets (see Table 3, Section 3.4) are not given here; they contain very similar results.

The conclusions that can be drawn from these graphs are:

Some features (a feature is a specific n-tuple occurring at a specific position) exist that might help distinguish between the different classes.

Most features seem to be within 50 bases from position 200. Remember that the original samples were selected in such a manner that the transcription start site, or the exact splice site is at position 200.

The exact n-tuple and position can be determined with this function using the full function capabilities (see Addendum 7.1), but this process is far too complex and slow to determine features one by one.

A better, more reliable method is required to extract the useful features from the statistical data.

4.5 Using entropy as feature significance indicator:

A quick review on the process so far:

Samples were generated from EPD and NCBI data.

The samples were separated into six different classes and into three different sets.

n-Tuple occurrence statistics were generated on a position specific basis.

A function was written that displays the position specific statistics in graphs.

More refinement is required to get useful indicator features from the statistics.

The next section addresses point 5 given above. A well-established method to extract meaningful data from a large number of statistical values, based on the entropy of the partition induced by a particular feature, was used. We used the definition

$$Ent(X,Y) = -\left((\frac{A_{xy}}{z}) * \ln(\frac{A_{xy}}{z}) + (\frac{B_{xy}}{z}) * \ln(\frac{B_{xy}}{z}) + (\frac{C_{xy}}{z}) * \ln(\frac{C_{xy}}{z}) + \ldots \right), \qquad \text{(eq 1)}$$

where Ent(X,Y) is the entropy of n-tuple X at position Y, and A_{xy}, B_{xy} and C_{xy} are the number of times n-tuple X occurs at position Y for sample type A, B and C., (In the rest of this document, such an XY combination will be called a bin). A, B and C can be any of the classes, and the equation can be extended to include all the class types by simply adding terms. Z is the sum of A_{xy}, B_{xy} and C_{xy}. The most obvious way to use the equation is to use the classes Promoter, Intron and Exon for A, B, and C respectively. The lower the entropy, the more likely it is for a specific feature to be a useful class indicator, since a low entropy corresponds to one of the classes being preponderant. Using entropy is more reliable than simply doing a direct comparison between classes, because entropy is based on the relationships between all class occurrences. For example, compare the relationship A:B:C = 10:30:10 to A:B:C = 2:45:3. For both the cases it is obvious that the n-tuple in class B occurs much more frequently than both classes A and C. But for the first case the n-tuple occurs only three times as much in class B as in class A and C, while the relationship is more significant for the second example. The entropy calculation bears out this expectation.

Ent(Example 1) = –10/50* ln(10/50) – 30/50*ln(30/50) – 10/50*ln(10/50)
= 0.2*1.609 + 0.6*0.511 + 0.2*1.609
= 0.9502

Ent (Example 2) = –ln(2/50) – ln(45/50) – ln(3/50)
= 0.04*3.219 + 0.9*0.105 + 0.06*2.813
= 0.39204

Thus, entropy gives a straightforward indication of the better discriminatory power of example 2. A Matlab program getEnt5.m (Addendum 8) was written to automatically calculate the entropy for each n-tuple at each position for a given length n. This program takes in several parameters explained in Addendum 8.3 - amongst others, the n-tuple length, decision boundary parameters and, the class for which one is trying to extract features.

When calculating entropy there is a very important thing to keep in mind, and that is that the logarithm of 0 cannot be calculated. That means for each bin a check has to be made to make sure that it is not zero, and when it is zero the term should just be ignored. If all the terms are zero the entropy cannot be calculated, but obviously if a certain n-tuple does not occur in any of the classes it cannot be used as a useful indicator. However it cannot be simply set to zero, since entropy works on an inversed scale, where 0 is the best entropy. To overcome this the program was written to simply set the entropy to 1 if there is no class that contains at least one sample.

A second problem was that if the total number of occurrences summing the bins for a given feature (Z in Equation 1) is not high enough, the feature is not a good indicator even if it shows low entropy. For example if A_{xy} = 1, B_{xy} = 0 and C_{xy} = 0 then the total entropy would be zero, because the B_{xy} and C_{xy} terms would be ignored, and the entropy of 1/1*ln(1/1) = 1*0 = 0. So for all the samples only one contained the feature, but as none of the other classes contained it a relationship of A:B:C = (100%) : (0%) : (0%). This seems to be a good feature, but, to the contrary, it is not because if there were, say, 1 300 training samples and only one contained the feature it is obviously not a good feature to use for classification. This results in the problem that low entropy is used to indicate good features, but if the bins do not contain enough values a low entropy is automatically found. To overcome this problem, a minimum threshold for bin content was enforced, enforced by the "significance" parameter. Hence, all bins containing fewer than a prescribed number of samples were disregarded, even if the entropy seems to be low enough to indicate a significant feature.

Note that a low entropy value indicates a good feature, but does not directly indicate which class is suggested by the feature. Fortunately, one can simply check which of the bins contained the most samples; the class that contains the most samples for the feature is suggested by the presence of the feature.

Another thing to keep in mind is that the entropy, when comparing all classes, differs from the entropy when comparing only say promoters with introns and exons. Currently, the program has only two modes of operation (as seen in Addendum 8.2) for selecting classes. In the one mode, it uses the three classes (Promoter, Intron and Exon), and in the second mode the I2E and E2I splice classes are added. The mRNA class is never used for entropy calculation, but as explained in Chapter 3 this class was only extracted to compare with the promoters downloaded from the EPD, and due to automated generation cannot be trusted in any case.

The last phase of the entropy calculation showed the results graphically. Different graphical formats were examined to aid in the search for good features. One option was to make a 3-dimensional plot, with the position on the X-axis, the n-tuple number on the Y-axis and the entropy on the Z-axis. Using this one can easily see where the features with low entropy are located. In Figure 14 below, the entropy for the 2-tuple training data is given. It can be seen that 2-tuple 7 (meaning string "CT" using function comb2seq from Section 4.3.2) has a lower entropy, and also that there are several 2-tuples near position 200 that show either very low or very high entropy. Rotating the figure so that it is seen directly from above (from the positive Z-axis direction), as shown in Figure 15, the same conclusions as above can be drawn by using the change in colour as indication. This is useful but not 100% accurate, as the change in colour indicates a change in entropy values, but does not give a clear indication of where exactly the high and low entropy values are.

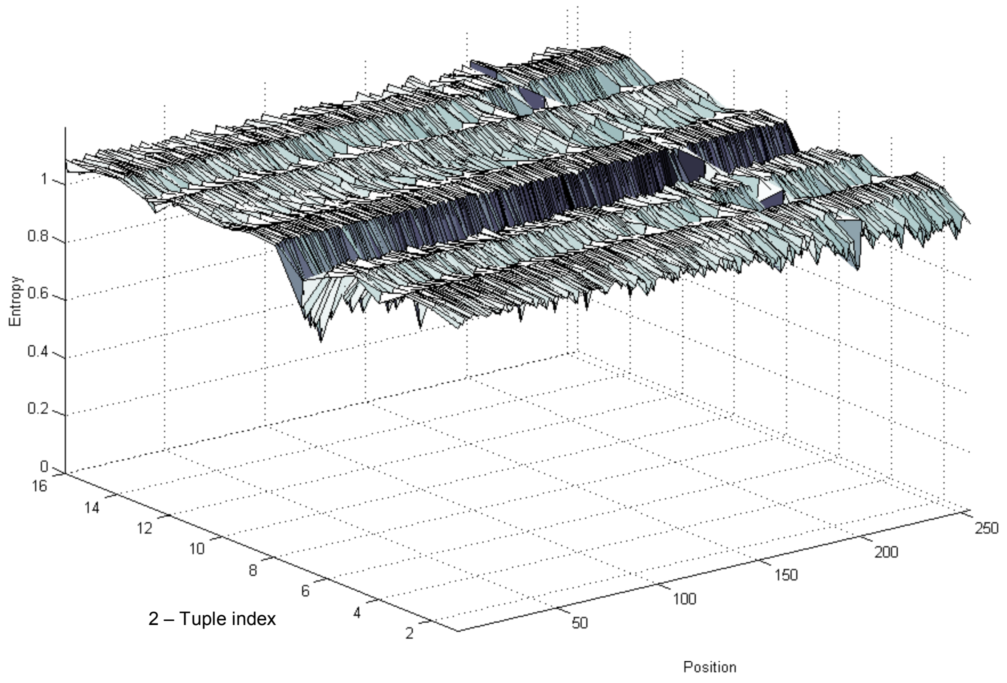

Figure 14: Sample of 3D entropy plot.

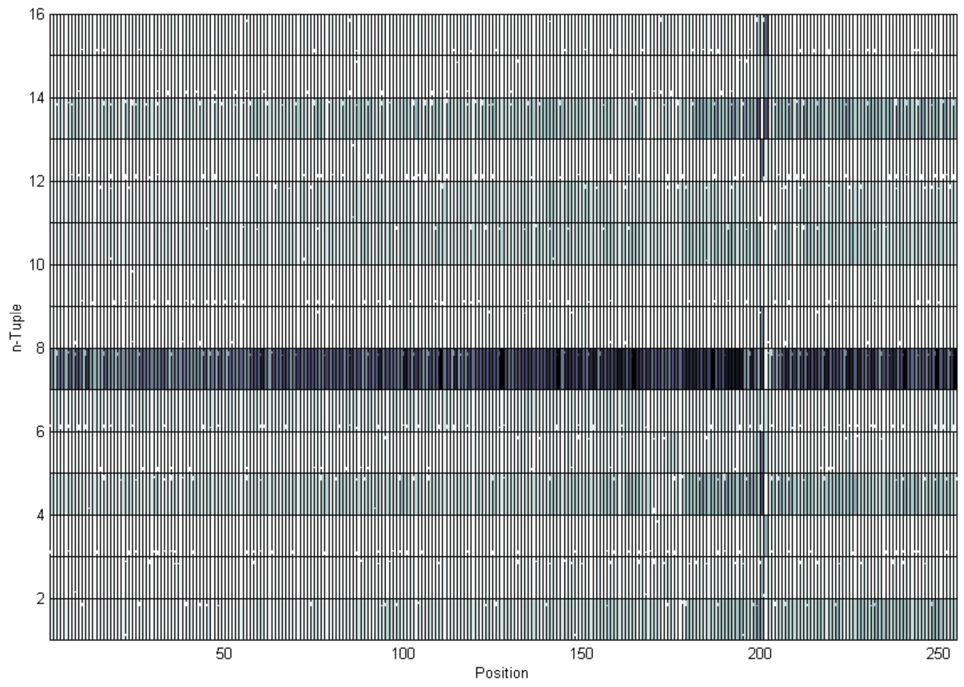

Figure 15: Sample of 3D entropy plot, rotated for clearer view.

The next graph was developed (Figure 16) to give the entropy on the Y-axis and the total number of occurrences of the tuple (in other words Z in Equation 1) on the X-axis. On this graph it is not possible to directly identify features, but it is possible to see what the general trend and relationship between feature and their corresponding bin values were. Any point on the bottom right of this graph is a good feature, while any point on the top left is a bad feature, since data points with low entropy that occur frequently are useful for discrimination.

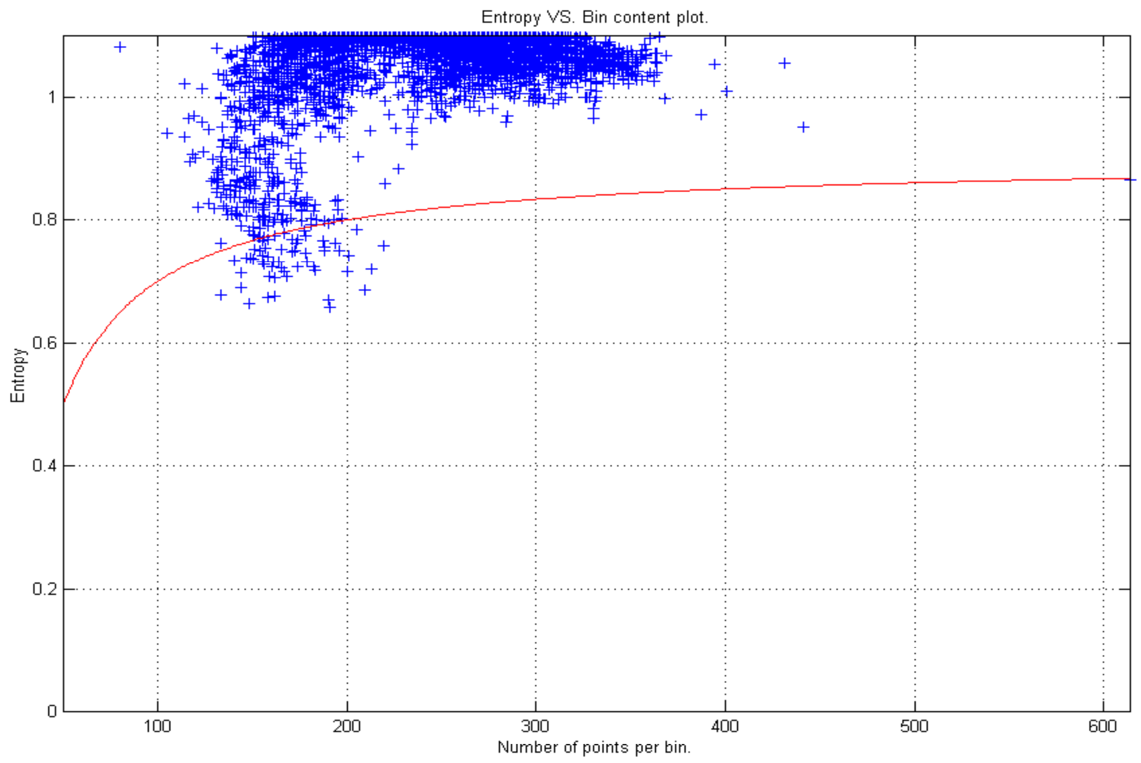

Figure 16: Sample of 2D entropy plot, including threshold line (parameters –20, 0.9).

The line on the figure indicates a threshold boundary. Every feature below, and to the right of the line is identified as a useful feature, while everything above and to the left of the line is ignored. The line is mathematically calculated using parameters of the entropy function (see Addendum 8.3 for details) and compared with the entropy values calculated. If the entropy value is less than the line function value the feature is a good class identifier. Matlab automatically generates a text output as shown below.

```
Mean entropy: 1.05433      Min entropy: 0.65833      Max entropy: 1.09861

[199,4,2,0.13552]          0
[10,6,2,0.23135]           1
[59,6,2,0.26826]           2
[92,6,2,0.26674]           3
[99,6,2,0.28710]           4
[109,6,2,0.32581]          5
[113,6,2,0.26112]          6
.. .. ..
```

Extract 8: Matlab entropy outputs.

The text output of Matlab gives the entropy mean, maximum and minimum values for the n-tuple used. It also provides feature outputs in a format that will later be used directly by the artificial neural network. These text outputs are available in Matlab and are also written to a text file “features.txt”. The text file also includes all the parameters used to extract the features so that it can easily be reproduced at a later stage if required.

The format of the features is:

[Position, n-tuple #, n-tuple length, 1-Entropy]   Sample number.

The value 1-Entropy is used to get an indication of the strength of the feature. The higher the number, the better the feature. No feature with an entropy of more than one will be selected by the threshold function, so 1 - Entropy will always be positive. The sample number given after the brackets is just to get an indication of how many features for a given n-tuple were selected in total.

4.6 Results: The selected features.

After doing entropy calculations for n-tuples of length one to five, two different sets of features were identified that could be used as detection features. Only features that can be used to identify promoters were selected at first. Different threshold detection boundaries were used for the different n-tuple lengths. In each case an attempt was made to select the optimal boundary where all the features have a large number of samples per bin, but still have a low entropy value.

The following two tables give the two feature sets that were selected for promoter detection. All the sets shown below were generated based on the equal sized set, Table 3, using only the intron, exon and promoter data sets. Some of the extraction entropy graphs are shown in Section 6, but were excluded here to save space. The values in the table are: Firstly, the unique n-tuple identification number. Secondly the actual sequence, followed by the position and length; then the probability is given that it is a promoter given the n-tuple and position combination. The last column is simply the number of the feature in the list.

Table 5: 54 Features extracted with entropy calculation.

n-Tuple #	String	Position	Length	Probability	Feature number
18	CAG	199	3	0.7433	1
19	CAT	199	3	0.6614	2
22	CCG	216	3	0.7286	3
25	CGC	147	3	0.7568	4
25	CGC	166	3	0.7174	5
25	CGC	184	3	0.7424	6
25	CGC	199	3	0.8169	7
26	CGG	148	3	0.8056	8
26	CGG	164	3	0.8158	9
36	GCA	198	3	0.6978	10
38	GCG	138	3	0.8158	11
38	GCG	147	3	0.8312	12
38	GCG	155	3	0.7722	13
38	GCG	157	3	0.8485	14
38	GCG	162	3	0.9079	15
38	GCG	178	3	0.8539	16
38	GCG	190	3	0.8243	17
38	GCG	191	3	0.8462	18
38	GCG	195	3	0.8272	19
38	GCG	216	3	0.8378	20
52	TCA	198	3	0.7030	21
48	ATAA	172	4	0.7887	22
72	CAGA	199	4	0.8272	23
75	CAGT	199	4	0.8636	24
89	CCGC	156	4	0.8919	25
89	CCGC	216	4	0.9600	26
100	CGCA	197	4	0.8889	27
101	CGCC	185	4	0.9143	28
101	CGCC	196	4	0.8780	29
105	CGGC	211	4	0.8611	30
106	CGGG	141	4	0.9615	31
116	CTCA	197	4	0.8116	32
146	GCAG	198	4	0.8209	33
148	GCCA	197	4	0.8281	34
153	GCGC	166	4	0.9310	35
153	GCGC	178	4	0.9643	36
153	GCGC	190	4	0.9600	37

154	GCGG	147	4	0.9355	38
154	GCGG	156	4	0.9333	39
154	GCGG	157	4	0.9630	40
154	GCGG	210	4	0.9091	41
166	GGCG	134	4	0.9286	42
166	GGCG	146	4	0.9143	43
166	GGCG	152	4	0.8919	44
166	GGCG	154	4	0.9118	45
166	GGCG	156	4	0.9032	46
166	GGCG	159	4	0.9615	47
166	GGCG	211	4	0.9118	48
169	GGGC	110	4	0.9615	49
170	GGGG	142	4	0.8542	50
210	TCAG	198	4	0.8209	51
192	ATAAA	172	5	0.8511	52
816	TATAA	170	5	0.9512	53
816	TATAA	171	5	0.9000	54

Table 6: 133 Features extracted with entropy calculation.

n-Tuple #	String	Position	Length	Probability	Total features
11	AGT	200	3	0.7213	1
17	CAC	199	3	0.6148	2
18	CAG	199	3	0.7433	3
19	CAT	199	3	0.6614	4
20	CCA	198	3	0.6736	5
21	CCC	62	3	0.5956	6
21	CCC	158	3	0.5288	7
22	CCG	145	3	0.6829	8
22	CCG	182	3	0.7667	9
22	CCG	216	3	0.7286	10
24	CGA	199	3	0.7000	11
25	CGC	10	3	0.8491	12
25	CGC	43	3	0.6935	13
25	CGC	82	3	0.6721	14
25	CGC	123	3	0.7966	15
25	CGC	139	3	0.8308	16
25	CGC	146	3	0.7407	17
25	CGC	147	3	0.7568	18
25	CGC	157	3	0.7975	19
25	CGC	166	3	0.7174	20
25	CGC	184	3	0.7424	21
25	CGC	185	3	0.7727	22
25	CGC	190	3	0.8364	23
25	CGC	196	3	0.8382	24
25	CGC	199	3	0.8169	25
25	CGC	210	3	0.7647	26
26	CGG	141	3	0.6721	27
26	CGG	148	3	0.8056	28
26	CGG	164	3	0.8158	29
26	CGG	183	3	0.8333	30
26	CGG	196	3	0.8026	31
26	CGG	219	3	0.7671	32
29	CTC	197	3	0.5790	33

36	GCA	198	3	0.6978	34
37	GCC	186	3	0.6036	35
37	GCC	197	3	0.7000	36
38	GCG	55	3	0.7647	37
38	GCG	105	3	0.6765	38
38	GCG	113	3	0.6406	39
38	GCG	120	3	0.8182	40
38	GCG	125	3	0.8980	41
38	GCG	135	3	0.7313	42
38	GCG	138	3	0.8158	43
38	GCG	145	3	0.8182	44
38	GCG	152	3	0.7037	45
38	GCG	155	3	0.7722	46
38	GCG	157	3	0.8485	47
38	GCG	162	3	0.9079	48
38	GCG	177	3	0.8219	49
38	GCG	178	3	0.8539	50
38	GCG	186	3	0.8312	51
38	GCG	190	3	0.8243	52
38	GCG	191	3	0.8462	53
38	GCG	195	3	0.8272	54
38	GCG	214	3	0.7692	55
38	GCG	216	3	0.8378	56
41	GGC	71	3	0.6263	57
41	GGC	101	3	0.5932	58
41	GGC	159	3	0.5798	59
41	GGC	176	3	0.5776	60
41	GGC	211	3	0.6250	61
41	GGC	212	3	0.5913	62
41	GGC	227	3	0.6224	63
42	GGG	70	3	0.5946	64
42	GGG	91	3	0.5776	65
42	GGG	92	3	0.5437	66
42	GGG	114	3	0.5660	67
42	GGG	116	3	0.5161	68
42	GGG	123	3	0.5323	69
42	GGG	124	3	0.5893	70
42	GGG	142	3	0.5963	71

42	GGG	143	3	0.6855	72
42	GGG	145	3	0.6330	73
42	GGG	149	3	0.5455	74
42	GGG	155	3	0.5714	75
42	GGG	178	3	0.5167	76
52	TCA	198	3	0.7030	77
54	TCG	244	3	0.6129	78
63	TTT	147	3	0.5375	79
48	ATAA	172	4	0.7887	80
72	CAGA	199	4	0.8272	81
75	CAGT	199	4	0.8636	82
89	CCGC	102	4	0.8966	83
89	CCGC	138	4	0.9231	84
89	CCGC	156	4	0.8919	85
89	CCGC	216	4	0.9600	86
89	CCGC	246	4	0.9231	87
100	CGCA	197	4	0.8889	88
101	CGCC	185	4	0.9143	89
101	CGCC	196	4	0.8780	90
102	CGCG	136	4	0.9259	91
102	CGCG	177	4	0.9286	92
105	CGGC	211	4	0.8611	93
106	CGGG	141	4	0.9615	94
106	CGGG	157	4	0.9231	95
116	CTCA	197	4	0.8116	96
146	GCAG	198	4	0.8209	97
148	GCCA	197	4	0.8281	98
153	GCGC	166	4	0.9310	99
153	GCGC	178	4	0.9643	100
153	GCGC	190	4	0.9600	101
154	GCGG	122	4	0.8378	102
154	GCGG	141	4	0.8571	103
154	GCGG	147	4	0.9355	104
154	GCGG	153	4	0.9310	105
154	GCGG	155	4	0.9231	106
154	GCGG	156	4	0.9333	107
154	GCGG	157	4	0.9630	108
154	GCGG	181	4	0.8966	109

154	GCGG	195	4	0.8966	110
154	GCGG	210	4	0.9091	111
154	GCGG	220	4	0.9286	112
166	GGCG	71	4	0.8966	113
166	GGCG	124	4	0.9000	114
166	GGCG	134	4	0.9286	115
166	GGCG	146	4	0.9143	116
166	GGCG	152	4	0.8919	117
166	GGCG	154	4	0.9118	118
166	GGCG	156	4	0.9032	119
166	GGCG	159	4	0.9615	120
166	GGCG	211	4	0.9118	121
169	GGGC	110	4	0.9615	122
169	GGGC	143	4	0.8571	123
170	GGGG	142	4	0.8542	124
170	GGGG	155	4	0.8571	125
210	TCAG	198	4	0.8209	126
192	ATAAA	171	5	0.8750	127
192	ATAAA	172	5	0.8511	128
290	CAGAG	199	5	0.8710	129
331	CCAGT	198	5	0.8621	130
816	TATAA	170	5	0.9512	131
816	TATAA	171	5	0.9000	132
840	TCAGA	198	5	0.8846	133

From these features it can be seen that most promoters seem to come from regions with higher GC values. This corresponds to the theoretical evidence that one might have to look for GC-rich areas. The second thing to notice is that there does seem to be some form of TAT box at position 170 or 171. This was also predicted from earlier experimental work.

5 CHAPTER 5: THE ARTIFICIAL NEURAL NETWORK

5.1 Selecting a neural network.

After creation of the features, a suitable classifier was selected for classification of the samples. The main reason for selecting an artificial neural network was based on the fact that the underlying relationship between the features is unknown, as mentioned in the discussion of the relevant literature. Statistically, the selected features occur more frequently, but we do not know anything about their interrelationships. An ANN is easy to implement, and can create a model of arbitrarily complex interrelationships [Bishop]. ANNs are commonly used and have well-understood training algorithms. As the aim of this project is not to develop new classification algorithms, but rather to classify accurately, using an ANN was a logical choice.

Using ANNs does require a certain amount of care. They can suffer from over-training, which leads to bad generalisation. This happens when a training set is not very large compared to the number of parameters in the network. A network trained on a small training set can learn the exact set instead of learning the underlying features. An over-trained network can be seen as something equivalent to a lookup table, where each input simply corresponds to a pre-defined output. When the network is presented with previously unseen samples, however, it will then not necessarily make a good classification.

A second problem, illustrated below with the case of a simple 2 input XOR system is that an incomplete set of training inputs can give unwanted results. If, for example, only cases (1), (2) and (3) below were used to train a classification network the network will learn that if the first value is a 1 the input is always 1, and if the first input is 0 the output is equal to the second input.

This will lead to an incorrect conclusion that 1 XOR 1 should be 1.

(1) 0 XOR 0 = 0

(2) 0 XOR 1 =1

(3) 1 XOR 0 = 1

(4) 1 XOR 1 = 0

This illustrates the problem in that a large enough training set is a requirement to create a reliable classification system when using a neural network (or any other trainable classifier).

5.2 The network structure.

5.2.1 Normal ANN structure.

A detailed description of artificial neural networks and the relevant mathematical derivations are given in Bishop [13], Negnevistky [14] and Russell [15] and will only be summarized here. An artificial neural network consists of several layers of neurons, with all the neurons in a layer typically having the same transfer function. A neuron is the basic component of the neural network, and each layer in the network contains at least one neuron. These neurons are abstractions of the neurons found in the human brain in both structure and functionality. Figure 17, below, shows the structure of a single neuron.

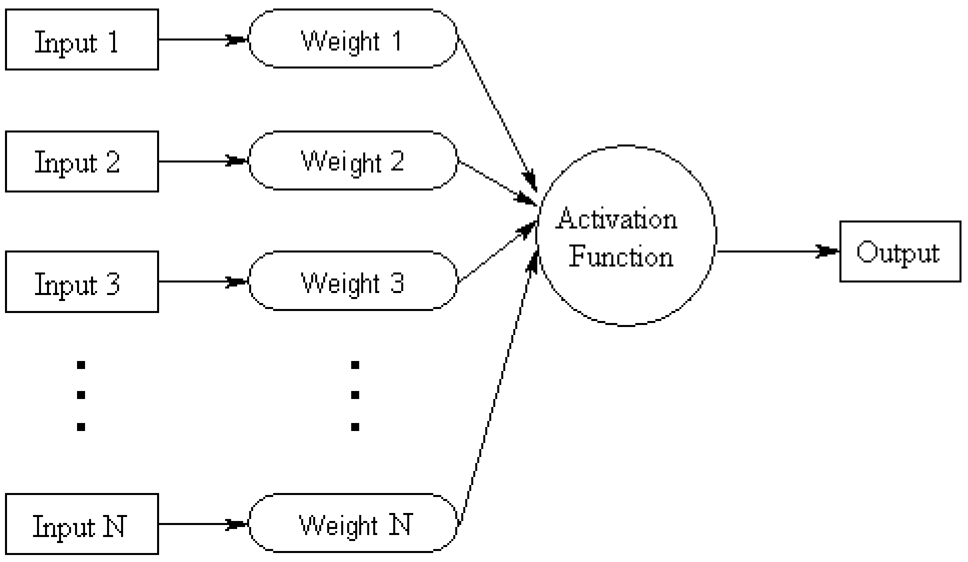

Figure 17: A single neuron structure as used in an ANN.

Briefly, it takes several inputs, multiplies each input with an independent weight factor, calculates the sum of these products, and then passes the answer through an activation (or transfer) function. The output of the neuron is the value of this activation function.

For example, the Sigmoid activation function is

$$Ysig = \frac{1}{1+e^{-Z}} \quad \text{(eq 2)}$$

with

$$Z = \sum_{i=1}^{n} x_i w_i \qquad \text{(eq 3)}$$

the weighted sum of the inputs.

Other activation functions exist, for instance the step function, sign function and linear activation function [13-15].

These neurons are combined in layers to form the total neural network structure as shown in Figure 18 below.

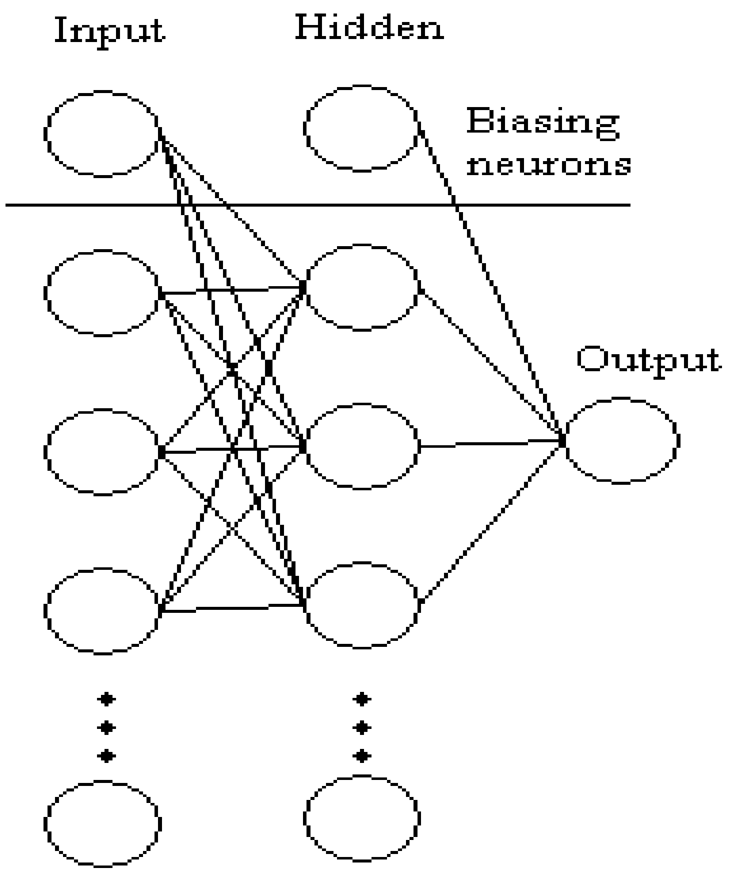

Figure 18: Total artificial neural network structure.

5.2.2 The three layers.

The three layers of the ANN are the input-, hidden- and output layers. The input layer simply takes a single feature vector from each sample and propagates it to all the hidden neurons via the input-to-hidden weights. Note that there is no activation function in the input layer – each input neuron equals the value of one of the features in the feature vector.

The hidden layer is the first actual calculation layer. It takes all the weighted inputs, and performs the sigmoid activation function on the weighted sum. Note that each line in Figure 18 can be seen as both a path of information travel as well as a unique weight. When working with ANNs it is common practice to use the subscript numbers to address each of the weights. That is, we refer to $\text{weight}_{\text{Source Destination}}$ with both the source and the destination being neuron numbers; the source is in the input layer and the destination in the hidden layer The function of the hidden layer is to mathematically combine all the input features. Each unit derives a different, unique combination of inputs. This information is then transferred to the output neuron.

The output layer can, in general, consist of several neurons, but for the classification network used by NNPromoterFind1.0 a single output neuron is used. This neuron takes the combinations from the hidden neuron, once again applying weighting and an activation function and comes up with a single value. This is the output of the ANN.

The input and hidden layers should also contain one biasing neuron. While each neuron in the network can be seen as a "soft" threshold function between different inputs the biasing neuron is used to move around the threshold to ensure that the threshold is not simply a line through the origin, but a line that can be located appropriately in n-dimensional space. The value for the bias neuron is a constant –1, but the weights are used exactly as for normal inputs.

5.2.3 Layer implementation for NNPromoterFind1.0

The implementation of the input layer is done based directly on the features derived in Section 4.6. on a one-to-one basis. For each feature a single input neuron is used plus one additional input for biasing. That means that two different ANNs were implemented, one with 55 inputs and one with 134 inputs.

There is no mathematically proven way to select the number of hidden neurons. For each hidden neuron a unique combination of inputs is calculated, and there can be an infinite number of combinations. The assumption was made that the number of combinations should be about one third of the number of inputs. This assumption was made after several different numbers of hidden neurons were used and tested.

A single output is used and the output simply states whether a sample belongs to the promoter class or not. Ideally the output should be 1 for promoters and 0 for non-promoters.

5.2.4 The network training process.

Much research has been performed on the training of neural networks. The simple back-propagation training method was selected for NNPromoterFind1.0 because it is easy to implement and proven to be effective. The flow diagram shown below can be used to get an abbreviated idea of how the training process works. The samples are selected one by one from the training set, and the features are then extracted. The features selected by the entropy measure are kept in a list so that they can easily be found in each sample. If a feature is found in a sample the corresponding input is set to 1, otherwise it is set to 0. The features are then propagated to the hidden layer. The hidden neuron applies the appropriate weight multiplier to each input, and sums the totals. The Sigmoid activation function is then applied to the sum and the output is propagated further to the output layer. This is done for each hidden neuron in turn, each applying a unique set of weights. The output neuron repeats the weighting, summing and activation to get the final output. The final output is then compared to the desired output. For each sample in the training set the output is known beforehand, as the class is known. If the sample is from the promoter class the desired output is 1, otherwise it is 0.

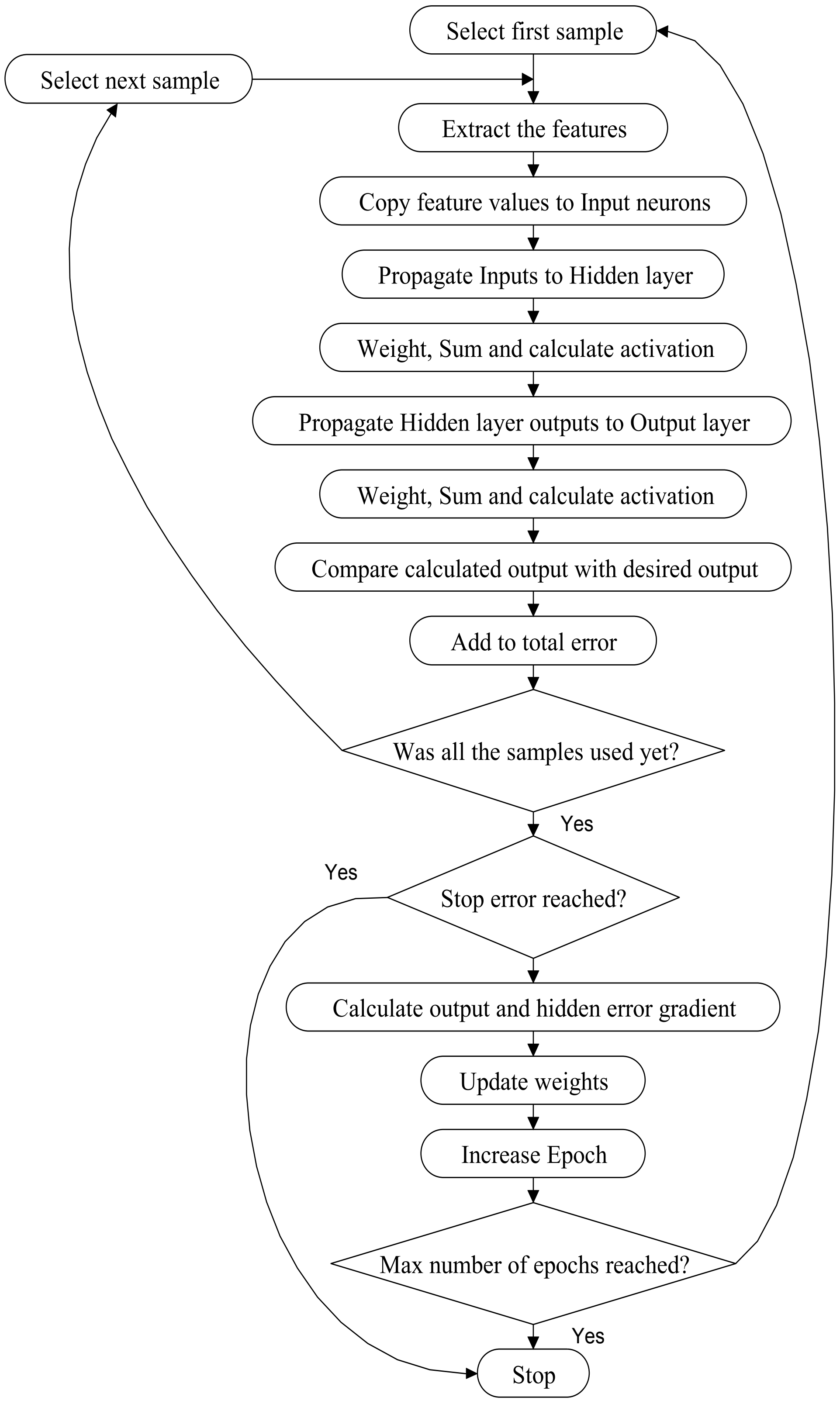
Select first sample
Select next sample
Extract the features
Copy feature values to Input neurons
Propagate Inputs to Hidden layer
Weight, Sum and calculate activation
Propagate Hidden layer outputs to Output layer
Weight, Sum and calculate activation
Compare calculated output with desired output
Add to total error
Was all the samples used yet?
Yes
Yes
Stop error reached?
Calculate output and hidden error gradient
Update weights
Increase Epoch
Max number of epochs reached?
Yes
Stop

Figure 19: Network training process.

The difference between the calculated output and desired output is then calculated by subtraction and squaring. This error is calculated for each of the samples in the training set, and added together. This is called the sum of squared errors, SSE[1]. The desired SSE should be close to 0 if the network has been trained well, with a fully trained, 100% effective network resulting in an SSE of exactly 0. The scale of the error is set by the fact that there are 1300 * 5 = 6500 samples, the worst-case SSE therefore being 6500. (If this arises every sample is classified incorrectly.) A stopping SSE of about 220 was reached, meaning a total error of about 3.5%. This error is propagated backwards from the output layer to the input layer, hence the algorithm name: back-propagation. This is done by first calculating the output layer error gradient, then the hidden layer error gradient.

Because the SSE can possibly never reach the minimum stopping requirement [Bishop] a second method is required to stop the training process. This is done by simply setting a maximum number to the epochs after which the training is stopped. (An epoch is one presentation of each sample in the training set,). Each weight in the system is changed once during each epoch to attempt to get a smaller SSE. If the network topology is known the network can easily be reproduced if all the weights in the network are stored.

5.3 Software implementation.

5.3.1 The training program.

A program "workANN.c" given in Addendum 9 was written to implement the training process as described in the flowchart in Figure 19. Before the training process can begin an additional step is required: combining the different classes into one single file. For this purpose the program "ClassCombine.c" given in Addendum 10 was written. This program is very similar to the program "ClassSets.c" given in Addendum 3. ClassSets.c was used to split a single class into the three different sets. ClassCombine.c is used to take all the class files for a single set and combine it into one sample file. The different classes in the output file, samples.txt are randomly distributed in the output file for training purposes. Note that this program should be run once for each of the three sets. All three times the output filename will be the same:

samples.txt. The test set file should then be manually renamed to testsamp.txt for use in the ANN training and testing programs.

This text file that contains all the samples for a certain class is one of the inputs required to perform the training process. The other required inputs are shown in Figure 20 below. Changing these inputs allows one to work with various options and for various training depths. One important shortcoming of this version of the training program is that the size of the network, in other words the number of input and hidden neurons, cannot be changed directly by the user. The only way to change these is to manually change them in the program code and then recompile the program. This obvious limitation is addressed in Chapter 7.

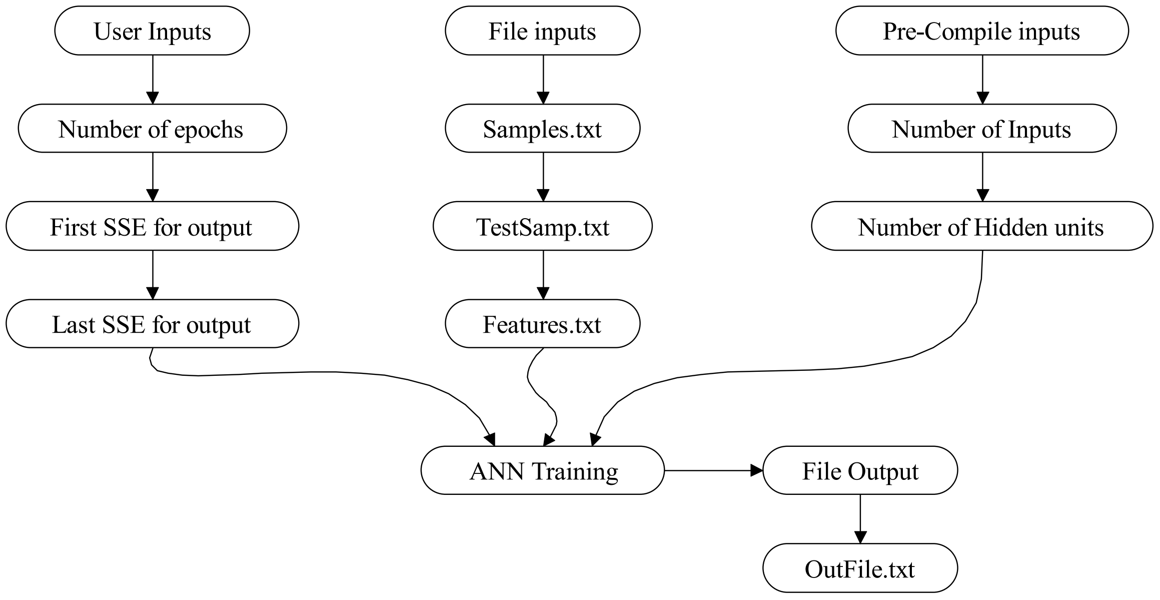

Figure 20: ANN file and user I/O.

The program opens the feature.txt file to extract and store a list of the features that will be used as network inputs. Next, the samples.txt file containing all the samples from the complete training set is opened, and the features from the feature list are extracted from the samples. A list of features for each sample is then stored in memory. This uses a large amount of memory but drastically decreases the training time. The training process is then done as described earlier, training until either the maximum selected number of epochs is reached, or the minimum required SSE is obtained. The training process is then stopped and the final output file is generated.

The output file generated during the training step is a text list of all the weights generated by the ANN configuration. For each weight a single number is stored, one per line. The file also contains some information of when the weights were stored during the training process, as well as the number of input and hidden units used. The output file is rewritten multiple times during the training process. This is done to ensure that if a run is prematurely terminated (e.g. due to a power failure) , the entire training process is not lost. Starting with the initial SSE, and then for every SSE reduction of 10 units, all system weights are written to the output fie. Text files were used rather than binary files to enable the user to inspect and manually change the weights. Inspection is useful, since it allows the user to see how the different features interact by looking at where the weights are combined to make a neuron fire.

The following extract shows the text output that is produced when the training program is run. The current SSE is given along with the epoch number. The last value displayed is the learning rate of the training algorithm (alpha). This value is adapted after each epoch so that that the training rate is increased when going in the right direction, and decreased when negative changes are made. Each time the weights are written to the output file a prompt is displayed showing the user what the exact SSE was for that specific case.

```
Training started: .
SSE: 403.6532   Epoch: 0 of 100 Alpha 0.105000.
SSE: 429.2585   Epoch: 1 of 100 Alpha 0.073500.
SSE: 432.0001   Epoch: 2 of 100 Alpha 0.077175.
SSE: 403.4374   Epoch: 3 of 100 Alpha 0.081034.
SSE: 371.8850   Epoch: 4 of 100 Alpha 0.085085.
SSE: 338.7242   Epoch: 5 of 100 Alpha 0.089340.
SSE: 316.4705   Epoch: 6 of 100 Alpha 0.093807.
SSE: 304.8020   Epoch: 7 of 100 Alpha 0.098497.
SSE: 298.4278   Epoch: 8 of 100 Alpha 0.103422
Train output file opened on error 298.428(300.000).
SSE: 294.5162   Epoch: 9 of 100 Alpha 0.108593.
SSE: 291.9166   Epoch: 10 of 100        Alpha 0.114023.
```

Extract 9: Training process output.

5.3.2 The percentage of samples used.

When the program workANN.c is run only some of the promoter training samples are actually used. This is a direct result from the way the features were selected based on the entropy. Some of the actual promoters do not contain any of the classification features. In the example shown in Extract 12, below, there were 341 of the 1300 promoters that did not contain a single feature. This is very bad for the training process as 26% of the promoters contain inputs of 0 for all the features, which is the same inputs found in the typical non-promoter. This means negative training will take place for these features. A check is also performed to see how many of the non-promoter samples contain non-zero inputs. In the example shown only 13.13 % of the non-promoters have features that are the same as the actual promoters. The other 86.87% have all-zero inputs.

```
Undetectable samples: 341, train on 73.77 percent of the samples.
Perfect negative samples: 4517, (13.13 percent needs training)
```

Extract 10: Training percentages.

These numbers were taken from the ANN with 55 inputs and 20 hidden units. When the larger system was implemented and 134 inputs were used, both these numbers were increased. That means that more promoters were actually used for the training (which is good) but also that more non-promoters contained positive features (which is bad). In Chapter 6, where the final results are given, this will be addressed again. The percentage of promoter samples used were increased from 73.77% to 93.23%, which is useful, but suggests that there is still more room for improvement.

5.3.3 The testing program.

The final step in our development was to test the trained network using the test and validation set.s A last program, LoadANN.c, given in Addendum 11, was written to test the network performance. The user must supply the number of input and hidden units used in the network. This program uses the weight output file that was generated during the training process as input. An ANN is configured, and the weights are read from the file outFile.txt. The feature file with the detection features selected by the entropy process is then opened and the desired features are loaded into the program. The file testSamp.txt, which contains the randomly distributed test set samples, is then opened and a feature list is again generated in memory.

The final step is for the user to input a threshold value used for the final classification. The aim of the program is to generate an output of 1 for all promoter classes and 0 for all non-promoters, but the final system supplies a number between 0 and 1 for all samples introduced, and unfortunately most of these values are very similar for samples from both classes. All values higher than the user-selected threshold are classified as promoters and all values below the threshold are classified as non-promoters.

The samples are then introduced to the ANN one by one for classification. The number of false detections (FD[1]) and false rejections (FR[2]), as well as the number of true detections (TD[3]) and true rejections (TR[4]) are counted and saved. These values are required to make a final assessment of how accurate the classifier is. The performance measure is computed based on the equations used by Bajic et al. [7] that define sensitivity and positive prediction values (PPV[5]). This is done to ensure that different detection systems can reliably be compared with one another. The sensitivity is a measure of how many actual promoters are detected and correctly classified. The sensitivity value is calculated by

$$Sen = 100 * \frac{TD}{TD + FR} \qquad \text{(eq 4)}$$

Higher sensitivity values mean that more promoters are correctly classified.
The other measure of accuracy is the positive prediction value calculated by

$$PPV = 100 * \frac{TD}{TD + FD}, \qquad \text{(eq 5)}$$

which is a measure of how many non-promoters were incorrectly classified as promoters.
All of these values are supplied directly to the user in the form of text output as shown in the example below.

```
True detections:  113   True rejections:  686
False detections: 14    False rejections: 237
PPV: 88.976378
Se: 32.285714
Detection threshold: 0.950000
```

Extract 11: Performance output.

[1] FD – False detections
[2] FR – False rejections
[3] TD – True detections
[4] TR – True rejections
[5] PPV – Positive prediction value

There is usually a very strong trade-off between sensitivity and PPV. Lowering the detection threshold increases the sensitivity because more promoters are correctly identified, but it lowers the PPV since more non-promoters are also identified as promoters.

6 CHAPTER 6: SYSTEM TESTS AND RESULTS

6.1 Data flow through entire process.

The following three figures show the entire data lifetime from raw data through to the final sensitivity and PPV calculations for NNPromoterFind1.0.

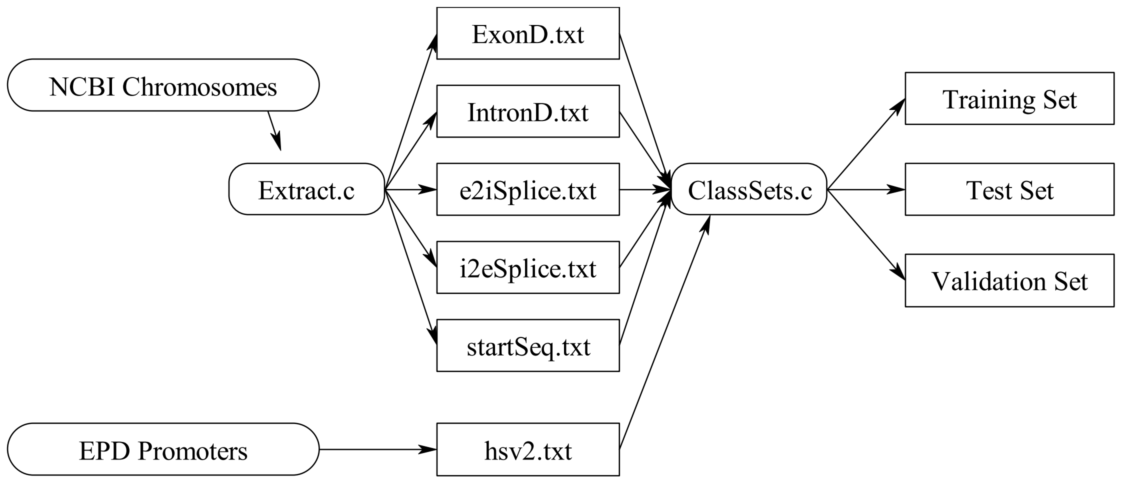

Figure 21: Raw data files to sets.

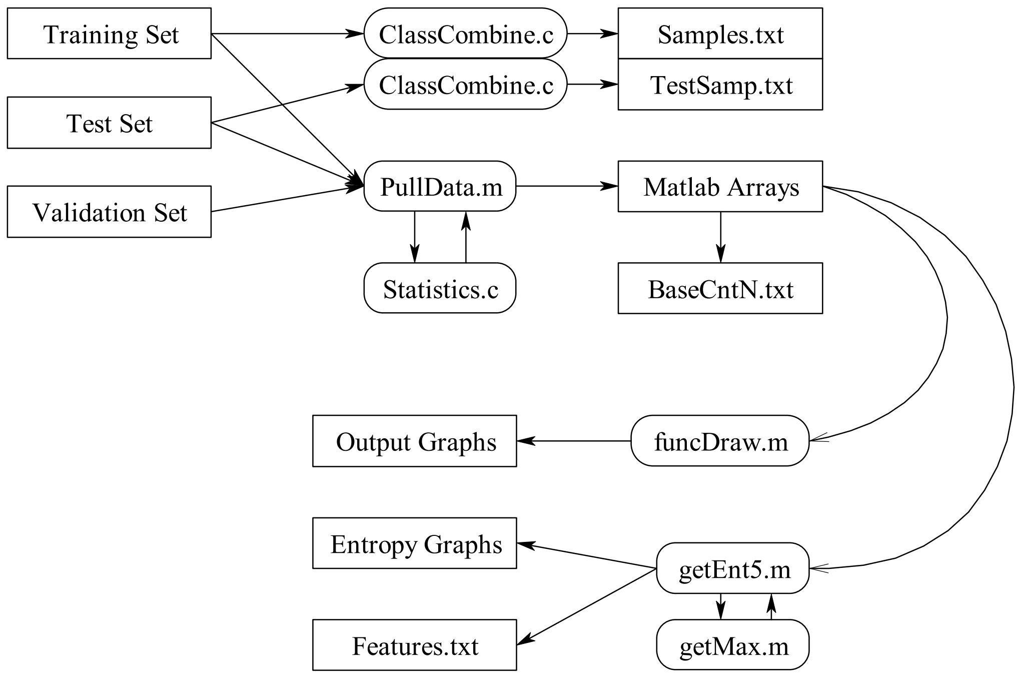

Figure 22: Sets to statistic graphs and entropy features.

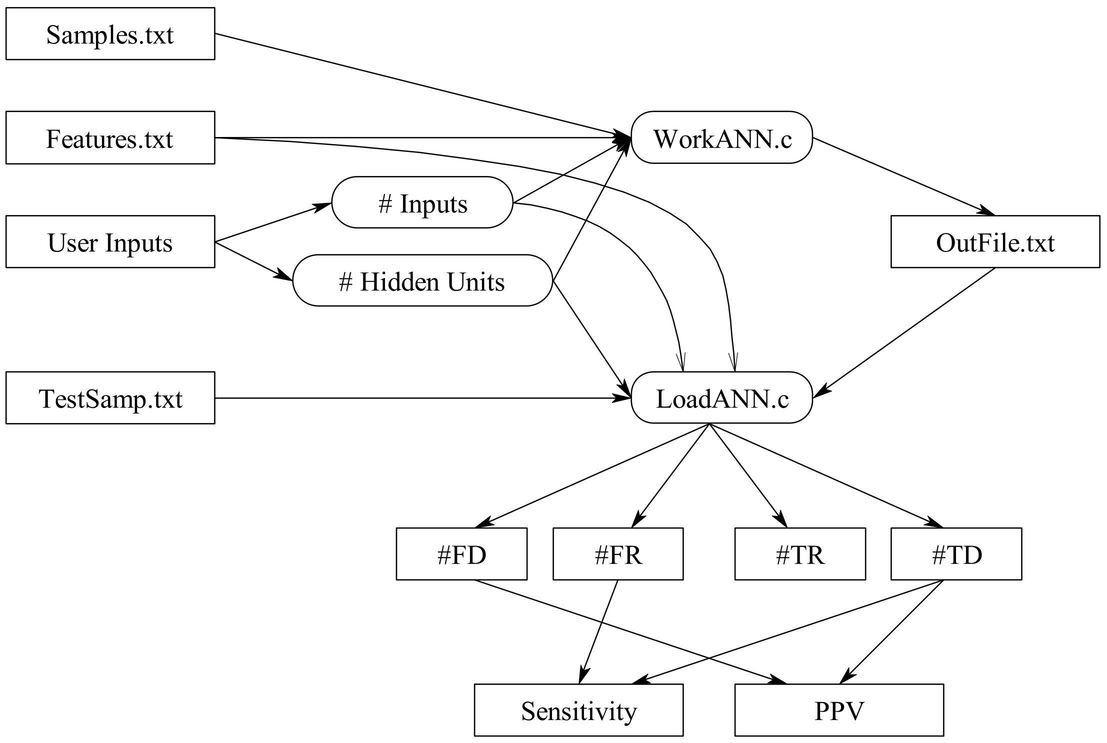

Figure 23: Features and sample data to sensitivity and PPV.

6.2 The first outputs.

6.2.1 Data files.

The first outputs of the system are the six text files containing the samples for each class. These results are not shown here - they are long lists of nucleotide sequences with no obvious meaning. These were then split into three sets with the numbers as shown in Figure 3. Most of the work was done on the equal- sized, since the prior probabilities are not known for real genetic DNA.

6.2.2 Statistics graphs.

The next outputs obtained were the statistical graphs that were presented in Section 4.4. These graphs can be directly used to establish a good idea of where the actual classification features are located in the samples. Although of little direct use, they are useful to guide the search for features, and to confirm the average frequency of the different n-tuples.

6.3 Entropy outputs.

The first directly useful outputs were the entropy outputs given in the following few pages. These graphs display the entropy outputs used to extract the 54 input features of the ANN. The 54 features are given in the exact format used in features.txt as given by the Matlab program getEnt5.m. Each graph is the entropy for a single selected n-tuple with the threshold function parameters given as well. Only the first 3D-plot is given since the second graph along with the text output is more significant. They were extracted from the equal sized data set (See Table 3) using only the intron, exon and promoter samples. When the I2E and E2I-splices were included for entropy calculations it was found that they have very strong features, so strong that the promoter features are totally dwarfed. Because the main aim is to extract promoters not splices, the splice features were excluded. A better system, which incorporates the splice features, is suggested in Chapter 7.

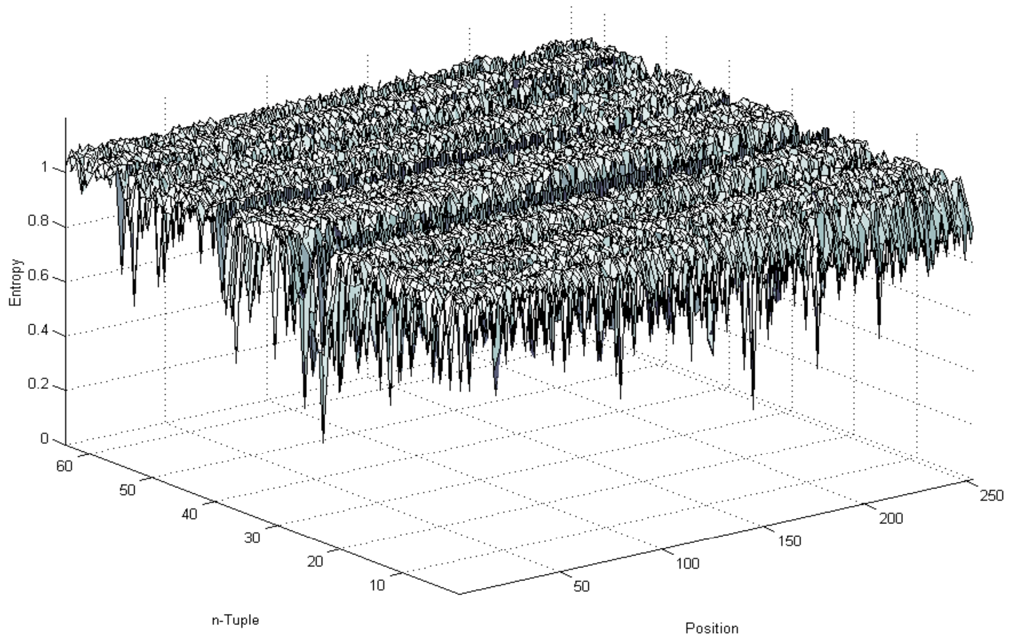

Figure 24: 3D- plot of the 3-tuple entropy data.

Entropy selection parameters:

3-tuple

Threshold parameters –16; 0.7

Equal sized training set.

Search for promoter.

Note that the strongest features seem to be at n-tuple 25 and 38 judging by the dip in the graph. This is confirmed by the outputs of the second graph, Figure 25.

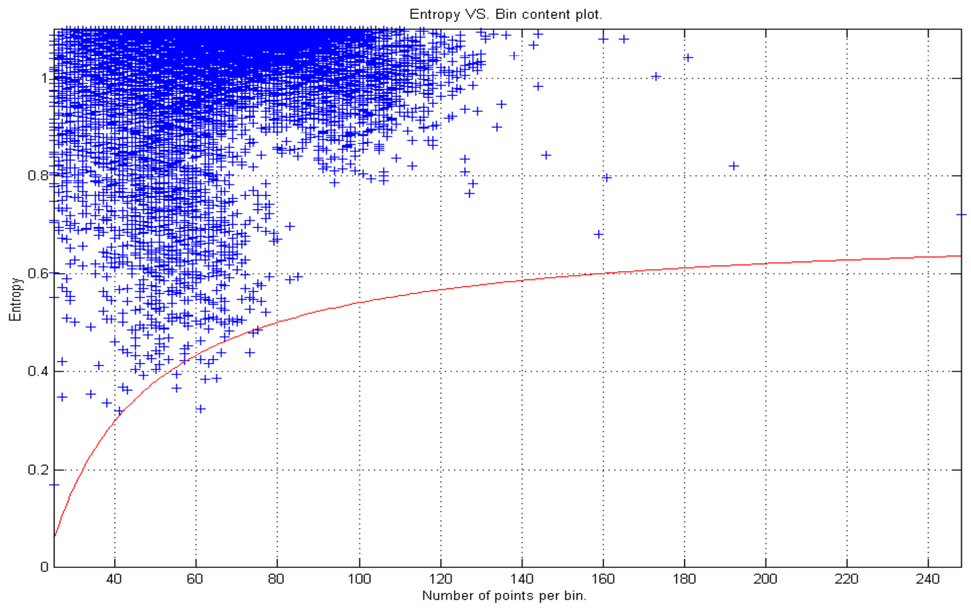

Figure 25: Entropy plot of 3-tuple data. Selected samples under the threshold line.

Entropy selection parameters:

3-tuple

Threshold parameters –16; 0.7

Equal sized training set.

Search for promoter.

Entropy, significance 25

```
[216,22,3,0.57501]     0
[147,25,3,0.60624]     1
[199,25,3,0.61417]     2
[148,26,3,0.61546]     3
[138,38,3,0.56251]     4
[147,38,3,0.56127]     5
[155,38,3,0.67544]     6
[157,38,3,0.57613]     7
[162,38,3,0.52142]     8
[178,38,3,0.52142]     9
[190,38,3,0.58469]     10
[191,38,3,0.63418]     11
[195,38,3,0.56251]     12
[216,38,3,0.51444]     13
Useable points: 14
```

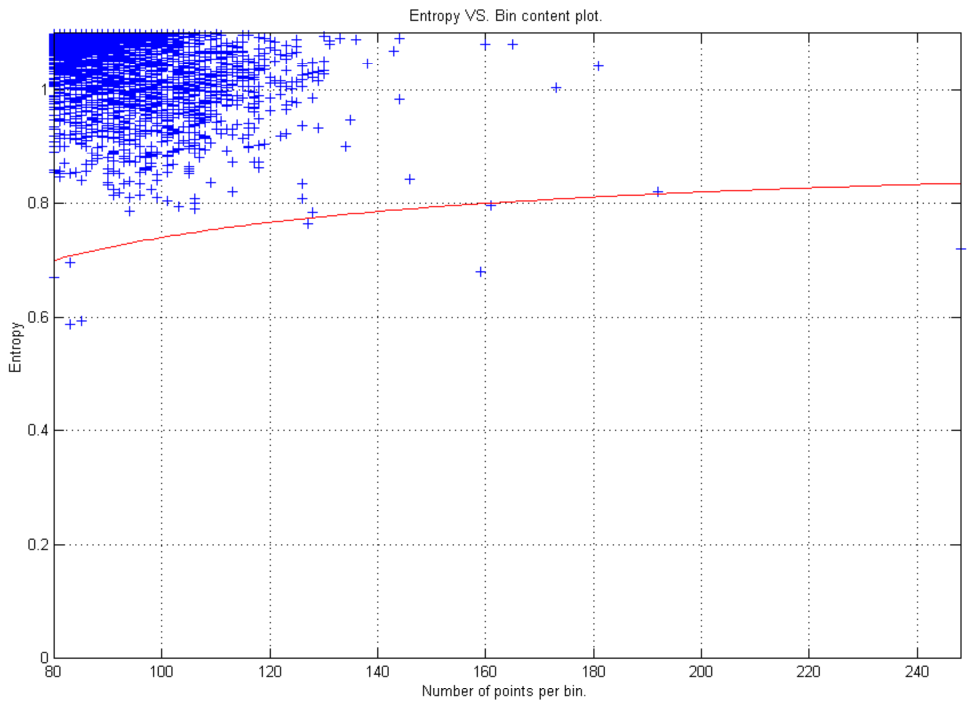

Figure 26: Entropy plot of 3-tuple data, different threshold settings.

Entropy selection parameters:

3-tuple

Threshold parameters –16; 0.9

Equal sized training set.

Search for promoter.

Entropy, significance 80

```
[199,18,3,0.28021]     0
[199,19,3,0.23532]     1
[166,25,3,0.40770]     2
[184,25,3,0.30368]     3
[164,26,3,0.41248]     4
[198,36,3,0.32044]     5
[198,52,3,0.20369]     6
Useable points: 7
```

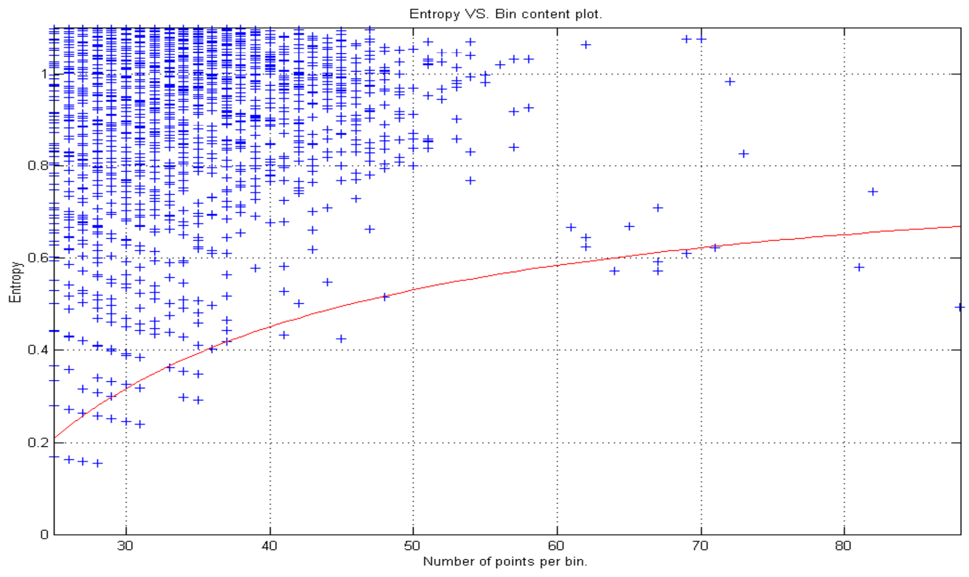

Figure 27: Entropy plot of 4-tuple data.

Entropy selection parameters:

4-tuple

Threshold parameters –16; 0.85

Entropy, significance 25

```
[172,48,4,0.37865]     0
[199,72,4,0.41984]     1
[199,75,4,0.50717]     2
[156,89,4,0.58252]     3
[216,89,4,0.83206]     4
[197,100,4,0.57639]     5
[185,101,4,0.65293]     6
[196,101,4,0.56818]     7
[211,105,4,0.59706]     8
[141,106,4,0.83698]     9
[197,116,4,0.39057]     10
[198,146,4,0.42925]     11
[197,148,4,0.42849]     12
[166,153,4,0.74905]     13
[178,153,4,0.84592]     14
[190,153,4,0.83206]     15
[147,154,4,0.76078]     16
[156,154,4,0.75507]     17
[157,154,4,0.84159]     18
[210,154,4,0.63750]     19
[134,166,4,0.74268]     20
[146,166,4,0.70749]     21
[152,166,4,0.58252]     22
[154,166,4,0.70156]     23
[156,166,4,0.68206]     24
[159,166,4,0.83698]     25
[211,166,4,0.64540]     26
[110,169,4,0.83698]     27
[142,170,4,0.48500]     28
[198,210,4,0.40832]     29
Useable points: 30
```

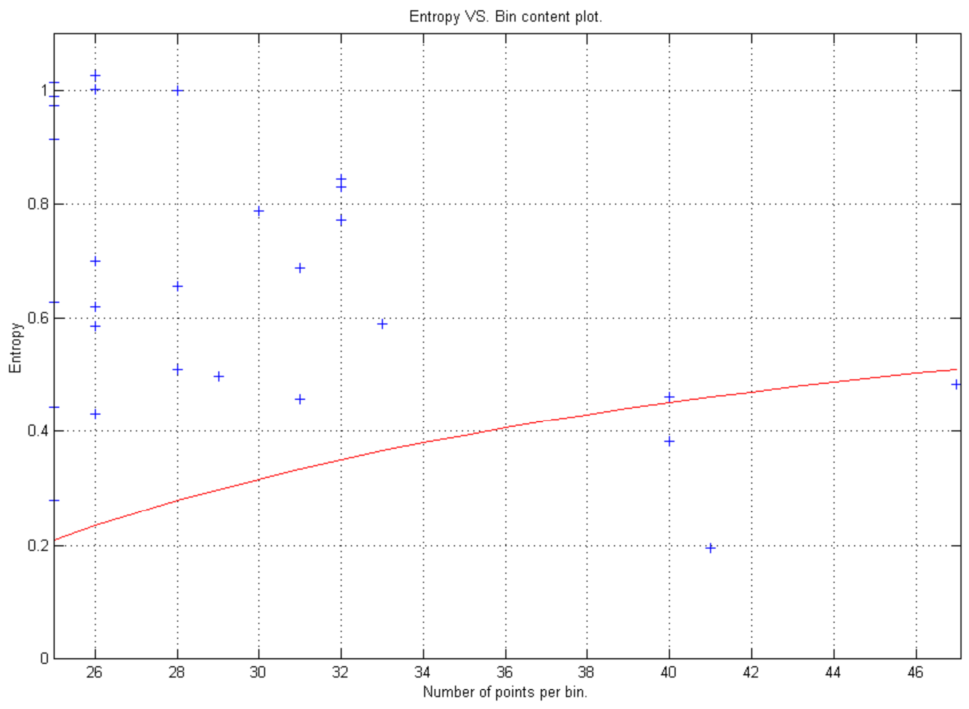

Figure 28: Entropy plot of 5-tuple data.

Entropy selection parameters:

5-tuple

Threshold parameters –16; 0.85

Equal sized training set.

Search for promoter.

Entropy, significance 25

```
[172,192,5,0.51806]     0
[170,816,5,0.80509]     1
[171,816,5,0.61868]     2
Useable points: 3
```

The features shown above are those that were finally selected for the detection ANN. By changing the parameters of the threshold function the number of features can be increased or decreased. For example, in Figure 28 a feature is shown for a bin containing 43 points, whose entropy falls just above the threshold line. However the feature selected from the bin containing 47 points (on the edge of the graph) has the same entropy, but falls below the selection threshold. By raising the threshold slightly, both these points can be included. A few different feature sets were chosen, but only the 54-feature (55 input) and 133-feature (134 input) ANNs were fully trained and tested.

This is one of the areas where further work can still be done as discussed in Chapter 7. Another thing to note is that if the larger training set, which contains more than 37 000 samples, is used, a different feature set might arise. This is due to the fact that the number of promoters is kept more or less the same in the region of 1 300, while the non-promoters are raised from 5 200 to about five times that number.

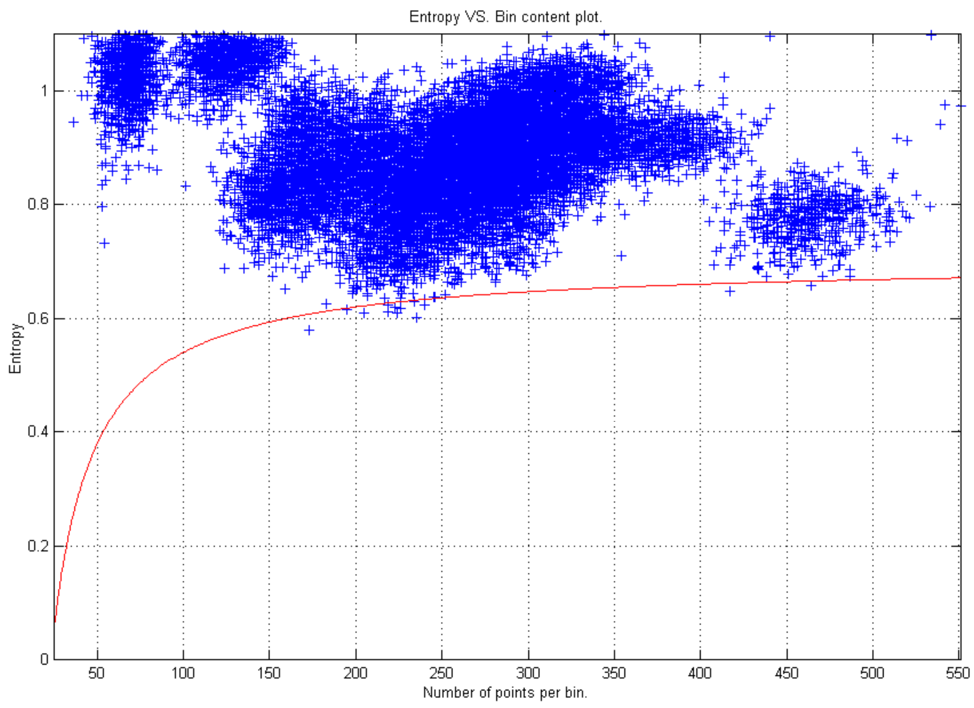

Figure 29: Entropy plot of 3-tuple data, large data set.

Entropy selection parameters:

3-tuple

Threshold parameters –16; 0.7

Equal sized training set.

Search for promoter.

Entropy, significance 25

0	-Promoter points
14	-Intron points
0	-Exon points

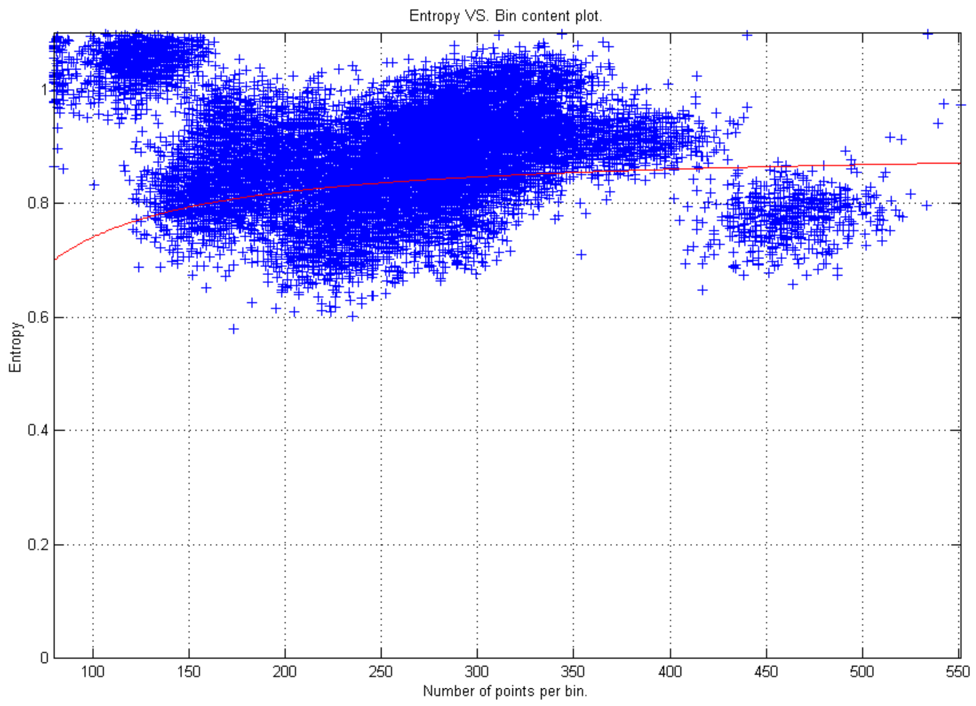

Figure 30: Entropy plot of 3-tuple data, large data set, shifted threshold

Entropy selection parameters:

3-tuple

Threshold parameters –16; 0.9

Equal sized training set.

Search for promoter.

Entropy, significance 80

0	-Promoter points
3782	-Intron points
398	-Exon points

This is also a clear example of a bad threshold. The aim of the threshold in the entropy calculations is to isolate the most significant data points. In the case shown above it simply cuts the main cluster of data points in two.

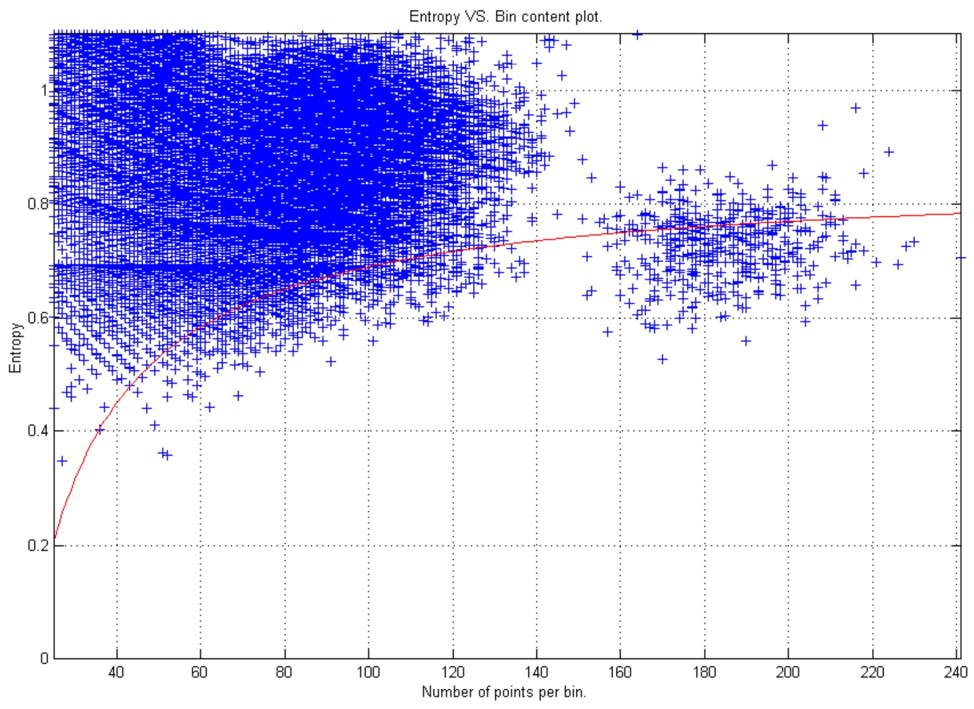

Figure 31: Entropy plot of 4-tuple data, large data set.

Entropy selection parameters:

4-tuple

Threshold parameters –16; 0.85

Equal sized training set.

Search for promoter.

Entropy, significance 25

0	-Promoter points
1053	-Intron points
4	-Exon points

These three graphs show exactly why the smaller set was selected to get the classification features and not the larger ones. In the larger sets the non-promoter samples outnumber the promoter samples by six to one and five to one, respectively. This means that the entropy equation is dominated by the contribution of the non-promoters. To get a more useful calculation of entropy, the number of intron samples in each position-n-tuple bin should be divided by six, and the number of exon samples should be divided by five before they are inserted into the entropy calculation. This can be done in the current case, since the relationships are known, but if other data sets are used then the program getEnt5.m has to be manually changed to adapt to the prior probabilities of the classes. In this fashion, the 3-tuple case was recalculated with a modified version of getEnt5.m (with the prior probabilities included).

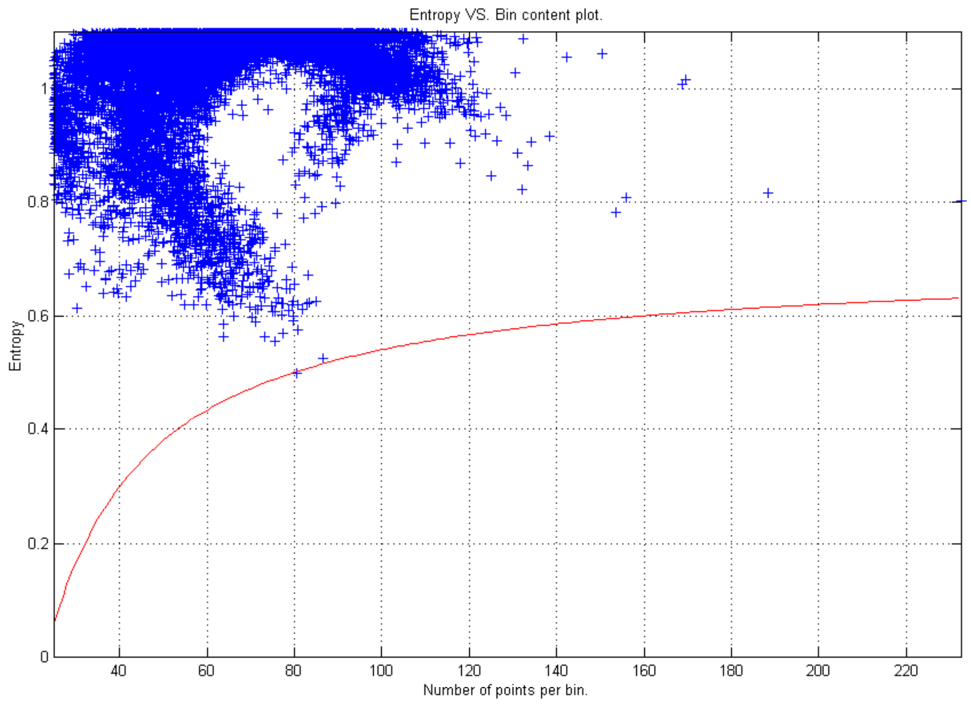

Figure 32: Entropy plot of 3-tuple data, large data set with prior-probabilities.

Entropy selection parameters:

3-tuple

Threshold parameters –16; 0.7

Equal sized training set.

Search for promoter.

Entropy, significance 25

1 -Promoter points

[162,38,3,0.50049] 0

Although this graph only selected a single feature as a good promoter feature the entire data clustering looks a lot more like it had when the equal sized set was used. The point extracted: 3-tuple number 38 at position 162 is one of the features that had been selected by the equal sized training set as well, the point marked 8 below Figure 25.

In the end, however, only the features given in Table 5 were used to train the 55-input ANN, and the features given in Table 6 were used to train the 134-input ANN.

6.4 ANN Results.

This section contains the final result if our research. The performance of the entire classification system is reflected here, in terms of the sensitivity and PPV measures obtained. Many different combinations of inputs and hidden units were tried, and the results of all the various trials will not be given here. The 55 input-, 20 hidden ANN configuration gave the best final results but used only 73% of the training promoter samples. The 134 input-, 49 hidden ANN configurations gave similar results but used 20% more of the input training samples, so in the end the second configuration should give a better generalization but with much longer training periods.

The first results were generated using the ANN with 55 Inputs, 20 Hidden Units, 1 Output, trained with the equal-sized training set. It trained for 164 epochs, resulting in a final SSE of 250. Table 6 and Figure 33 below give the results using different detection thresholds. The first group of results used only the promoter, intron and exon data of the test set. The second group used the same test set again, but with the splice samples included. The third set used the equal-sized validation set with the splices excluded once again, and lastly all the samples in the validation set were used. This format is used for the next three results as well.

The second set of results are from the same ANN, 55 Inputs, 20 Hidden units, 1 Output but trained on the large training set. It trained for 115 epochs to a final SSE of 640. The results are shown in Table 7 and Figure 34.

The third was taken from the large ANN, 134 Inputs, 49 Hidden units, 1 Output, trained on the equal sized training data. This time the training lasted for 84 epochs resulting in a final SSE of 180. Table 8 and Figure 35 below give the final output results.

Lastly, the large ANN was trained using the large training data. The ANN was trained for only 19 epochs, to a SSE of 590. This looks far worse than the first small ANN, but each epoch involves far more weight training than the small ANN and small data set. The results are given in Table 9 and Figure 36.

Table 7: Final results on equal sized training set, 50-20-1 ANN.

Prom	**Intron**	**Exon**	**I2E**	**E2I**	**# Promoters**	**# Non-Promoters**	**Total**	**Threshold**	**TD**	**FD**	**TR**	**FR**	**Sens%**	**PPV%**
Yes	Yes	Yes	No	No	350	700	1050	0.98	339	689	11	11	96.86	32.98
					350	700	1050	0.985	239	80	620	111	68.29	74.92
					350	700	1050	0.99	233	74	626	117	66.57	75.90
					350	700	1050	0.999	137	20	680	213	39.14	87.26
					350	700	1050	0.9999	91	6	694	259	26.00	93.81
Prom	**Intron**	**Exon**	**I2E**	**E2I**	**# Promoters**	**# Non-Promoters**	**Total**	**Threshold**	**TD**	**FD**	**TR**	**FR**	**Sens%**	**PPV%**
Yes	Yes	Yes	Yes	Yes	350	1400	1750	0.98	339	1383	17	11	96.86	19.69
					350	1400	1750	0.985	239	177	1223	111	68.29	57.45
					350	1400	1750	0.99	233	164	1236	117	66.57	58.69
					350	1400	1750	0.999	137	44	1356	213	39.14	75.69
					350	1400	1750	0.9999	91	18	1382	259	26.00	83.49
Prom	**Intron**	**Exon**	**I2E**	**E2I**	**# Promoters**	**# Non-Promoters**	**Total**	**Threshold**	**TD**	**FD**	**TR**	**FR**	**Sens%**	**PPV%**
Yes	Yes	Yes	No	No	221	442	663	0.98	211	437	5	10	95.48	32.56
					221	442	663	0.985	136	72	370	85	61.54	65.38
					221	442	663	0.99	134	71	371	87	60.63	65.37
					221	442	663	0.999	65	13	429	156	29.41	83.33
					221	442	663	0.9999	47	7	435	174	21.27	87.04
Prom	**Intron**	**Exon**	**I2E**	**E2I**	**# Promoters**	**# Non-Promoters**	**Total**	**Threshold**	**TD**	**FD**	**TR**	**FR**	**Sens%**	**PPV%**
Yes	Yes	Yes	Yes	Yes	221	884	1105	0.98	211	868	16	10	95.48	19.56
					221	884	1105	0.985	136	129	755	85	61.54	51.32
					221	884	1105	0.99	134	124	760	87	60.63	51.94
					221	884	1105	0.999	65	32	852	156	29.41	67.01
					221	884	1105	0.9999	47	17	867	174	21.27	73.44

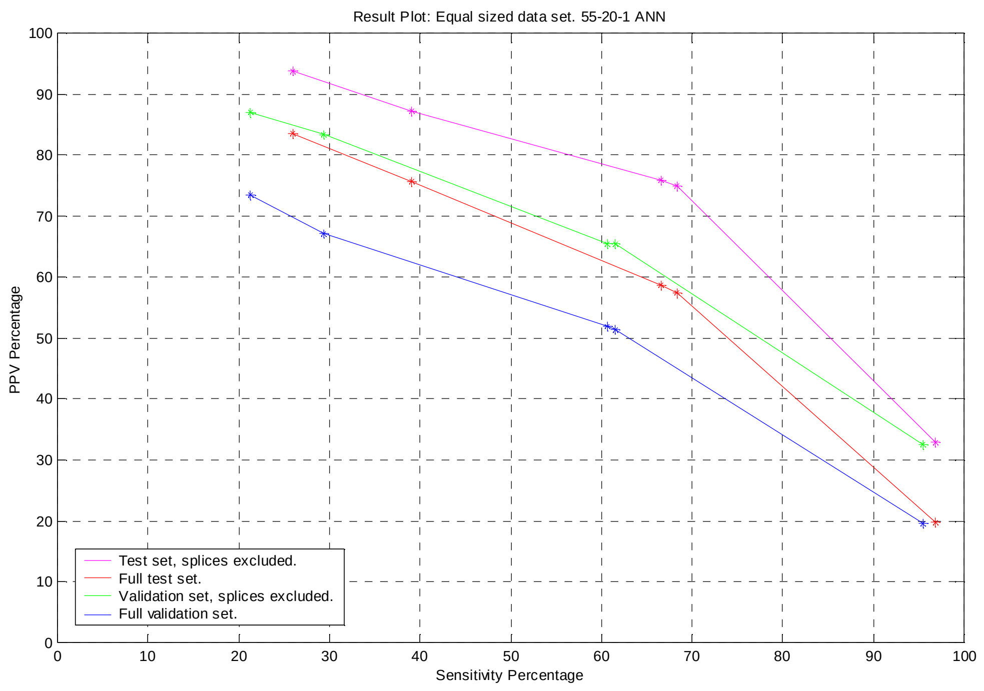

Figure 33: Sensitivity VS. PPV of equal size training set, small ANN.

Table 8: Final results on large training set, 50-20-1 ANN.

Prom	Intron	Exon	I2E	E2I	# Promoters	# Non-Promoters	Total	Threshold	TD	FD	TR	FR	Sens%	PPV%
Yes	Yes	Yes	No	No	383	3396	3779	0.00001	380	3385	11	3	99.22	10.09
					383	3396	3779	0.000015	253	512	2884	130	66.06	33.07
					383	3396	3779	0.00004	239	498	2898	144	62.40	32.43
					383	3396	3779	0.0001	204	338	3058	179	53.26	37.64
					383	3396	3779	0.0004	155	238	3158	228	40.47	39.44
					383	3396	3779	0.001	130	145	3251	253	33.94	47.27
					383	3396	3779	0.01	44	11	3385	339	11.49	80.00
Yes	Yes	Yes	Yes	Yes	383	7437	7820	0.00001	380	7417	20	3	99.22	4.87
					383	7437	7820	0.000015	253	1075	6362	130	66.06	19.05
					383	7437	7820	0.00004	239	1025	6412	144	62.40	18.91
					383	7437	7820	0.0001	204	691	6746	179	53.26	22.79
					383	7437	7820	0.0004	155	447	6990	228	40.47	25.75
					383	7437	7820	0.001	130	278	7159	253	33.94	31.86
					383	7437	7820	0.01	44	28	7409	339	11.49	61.11
Yes	Yes	Yes	No	No	194	1448	1642	0.00001	189	1441	7	5	97.42	11.60
					194	1448	1642	0.000015	131	214	1234	63	67.53	37.97
					194	1448	1642	0.00004	125	201	1247	69	64.43	38.34
					194	1448	1642	0.0001	108	148	1300	86	55.67	42.19
					194	1448	1642	0.0004	83	113	1335	111	42.78	42.35
					194	1448	1642	0.001	69	69	1379	125	35.57	50.00
					194	1448	1642	0.01	31	6	1442	163	15.98	83.78
Yes	Yes	Yes	Yes	Yes	194	3238	3432	0.00001	189	3229	9	5	97.42	5.53
					194	3238	3432	0.000015	131	475	2763	63	67.53	21.62
					194	3238	3432	0.00004	125	444	2794	69	64.43	21.97
					194	3238	3432	0.0001	108	317	2921	86	55.67	25.41
					194	3238	3432	0.0004	83	213	3025	111	42.78	28.04
					194	3238	3432	0.001	69	124	3114	125	35.57	35.75
					194	3238	3432	0.01	31	10	3228	163	15.98	75.61

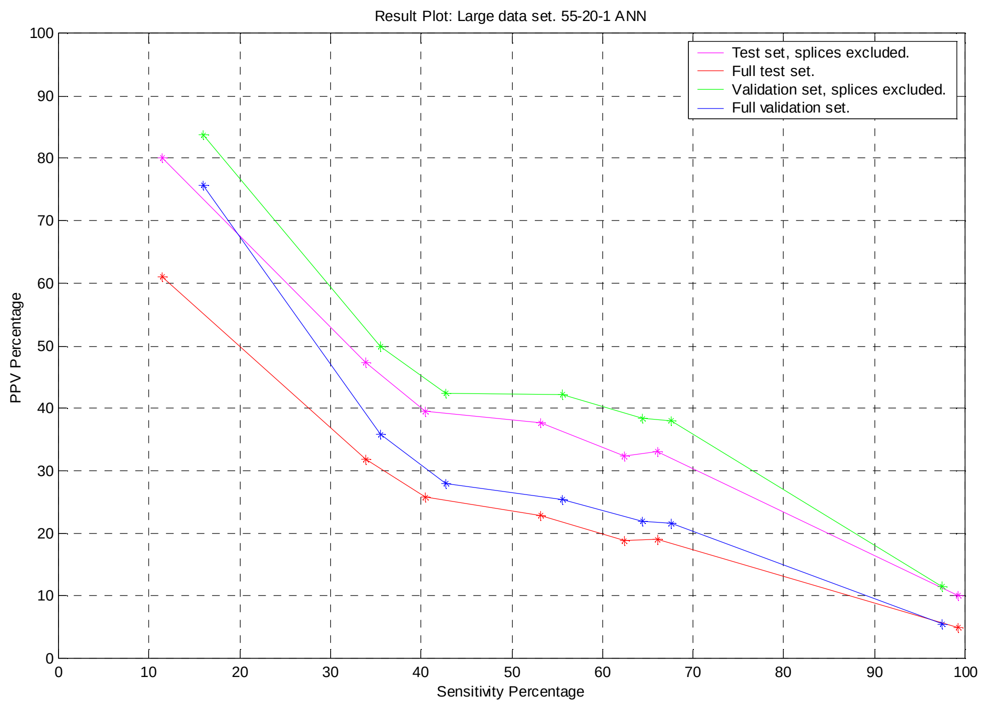

Figure 34: Sensitivity vs. PPV of large training set, small ANN.

Table 9: Final results on equal sized training set, 134-49-1 ANN.

Prom	Intron	Exon	I2E	E2I	# Promoters	# Non-Promoters	Total	Threshold	TD	FD	TR	FR	Sens%	PPV%
Yes	Yes	Yes	No	No	350	700	1050	0.98	318	275	425	32	90.86	53.63
					350	700	1050	0.985	316	268	432	34	90.29	54.11
					350	700	1050	0.99	307	220	480	43	87.71	58.25
					350	700	1050	0.999	272	127	573	78	77.71	68.17
					350	700	1050	0.9999	222	65	635	128	63.43	77.35
					350	700	1050	0.99999	184	40	660	166	52.57	82.14
					350	700	1050	0.999999	159	25	675	191	45.43	86.41
					350	700	1050	1	6	0	700	344	1.71	100.00
Yes	Yes	Yes	Yes	Yes	350	1400	1750	0.98	318	536	864	32	90.86	37.24
					350	1400	1750	0.985	316	524	876	34	90.29	37.62
					350	1400	1750	0.99	307	438	962	43	87.71	41.21
					350	1400	1750	0.999	272	246	1154	78	77.71	52.51
					350	1400	1750	0.9999	222	132	1268	128	63.43	62.71
					350	1400	1750	0.99999	184	79	1321	166	52.57	69.96
					350	1400	1750	0.999999	159	45	1355	191	45.43	77.94
					350	1400	1750	1	6	0	1400	344	1.71	100.00
Yes	Yes	Yes	No	No	221	442	663	0.98	196	206	236	25	88.69	48.76
					221	442	663	0.985	192	199	243	29	86.88	49.10
					221	442	663	0.99	181	173	269	40	81.90	51.13
					221	442	663	0.999	156	100	342	65	70.59	60.94
					221	442	663	0.9999	134	65	377	87	60.63	67.34
					221	442	663	0.99999	103	48	394	118	46.61	68.21
					221	442	663	0.999999	88	32	410	133	39.82	73.33
					221	442	663	1	7	0	442	214	3.17	100.00
Yes	Yes	Yes	Yes	Yes	221	884	1105	0.98	196	392	492	25	88.69	33.33
					221	884	1105	0.985	192	379	505	29	86.88	33.63
					221	884	1105	0.99	181	320	564	40	81.90	36.13
					221	884	1105	0.999	156	177	707	65	70.59	46.85
					221	884	1105	0.9999	134	111	773	87	60.63	54.69
					221	884	1105	0.99999	103	73	811	118	46.61	58.52
					221	884	1105	0.999999	88	46	838	133	39.82	65.67
					221	884	1105	1	7	1	883	214	3.17	87.50

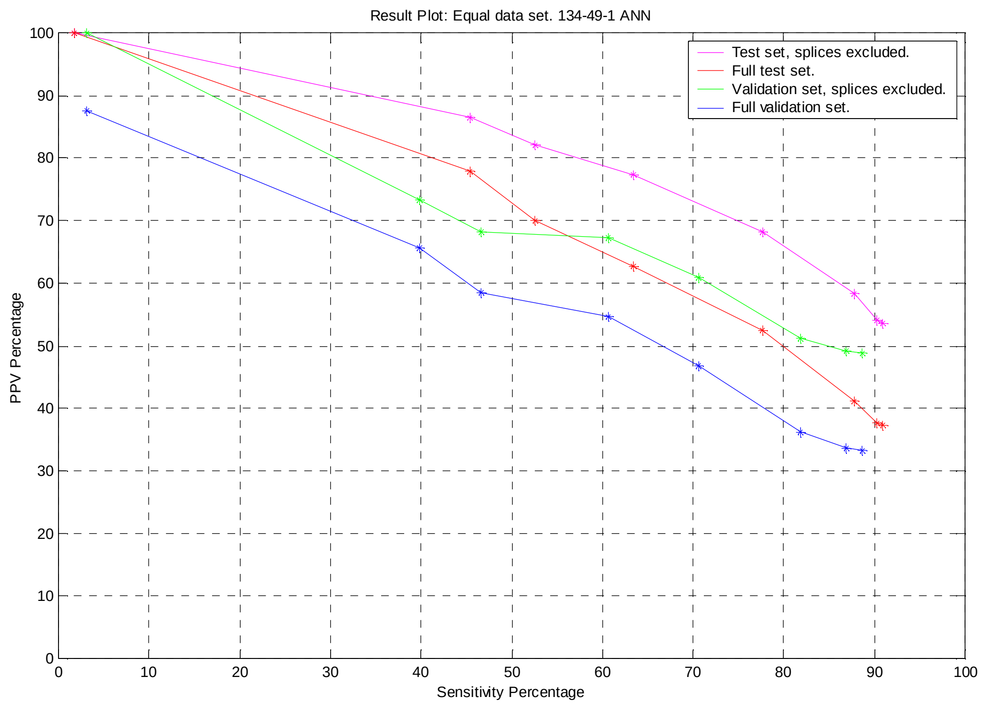

Figure 35: Sensitivity vs. PPV of equal sized training set, large ANN.

Table 10: Final results on large training set, 134-49-1 ANN.

Prom	Intron	Exon	I2E	E2I	# Promoters	# Non-Promoters	Total	Threshold	TD	FD	TR	FR	Sens%	PPV%
Yes	Yes	Yes	No	No	383	3396	3779	0.000005	312	886	2510	71	81.46	26.04
					383	3396	3779	0.00001	286	559	2837	97	74.67	33.85
					383	3396	3779	0.00005	221	217	3179	162	57.70	50.46
					383	3396	3779	0.0001	194	147	3249	189	50.65	56.89
					383	3396	3779	0.0005	133	60	3336	250	34.73	68.91
					383	3396	3779	0.001	103	36	3360	280	26.89	74.10
					383	3396	3779	0.005	60	15	3381	323	15.67	80.00
					383	3396	3779	0.01	52	6	3390	331	13.58	89.66
					383	3396	3779	0.05	31	1	3395	352	8.09	96.88
Prom	**Intron**	**Exon**	**I2E**	**E2I**	**# Promoters**	**# Non-Promoters**	**Total**	**Threshold**	**TD**	**FD**	**TR**	**FR**	**Sens%**	**PPV%**
Yes	Yes	Yes	Yes	Yes	383	7437	7820	0.000005	312	1769	5668	71	81.46	14.99
					383	7437	7820	0.00001	286	1166	6271	97	74.67	19.70
					383	7437	7820	0.00005	221	459	6978	162	57.70	32.50
					383	7437	7820	0.0001	194	298	7139	189	50.65	39.43
					383	7437	7820	0.0005	133	122	7315	250	34.73	52.16
					383	7437	7820	0.001	103	72	7365	280	26.89	58.86
					383	7437	7820	0.005	60	30	7407	323	15.67	66.67
					383	7437	7820	0.01	52	15	7422	331	13.58	77.61
					383	7437	7820	0.05	31	4	7433	352	8.09	88.57
Prom	**Intron**	**Exon**	**I2E**	**E2I**	**# Promoters**	**# Non-Promoters**	**Total**	**Threshold**	**TD**	**FD**	**TR**	**FR**	**Sens%**	**PPV%**
Yes	Yes	Yes	No	No	194	1448	1642	0.000005	165	391	1057	29	85.05	29.68
					194	1448	1642	0.00001	159	269	1179	35	81.96	37.15
					194	1448	1642	0.00005	125	108	1340	69	64.43	53.65
					194	1448	1642	0.0001	106	69	1379	88	54.64	60.57
					194	1448	1642	0.0005	65	25	1423	129	33.51	72.22
					194	1448	1642	0.001	52	16	1432	142	26.80	76.47
					194	1448	1642	0.005	27	8	1440	167	13.92	77.14
					194	1448	1642	0.01	20	2	1446	174	10.31	90.91
					194	1448	1642	0.05	7	1	1447	187	3.61	87.50
Prom	**Intron**	**Exon**	**I2E**	**E2I**	**# Promoters**	**# Non-Promoters**	**Total**	**Threshold**	**TD**	**FD**	**TR**	**FR**	**Sens%**	**PPV%**
Yes	Yes	Yes	Yes	Yes	194	3238	3432	0.000005	165	812	2426	29	85.05	16.89
					194	3238	3432	0.00001	159	544	2694	35	81.96	22.62
					194	3238	3432	0.00005	125	203	3035	69	64.43	38.11
					194	3238	3432	0.0001	106	133	3105	88	54.64	44.35
					194	3238	3432	0.0005	65	47	3191	129	33.51	58.04
					194	3238	3432	0.001	52	30	3208	142	26.80	63.41
					194	3238	3432	0.005	27	11	3227	167	13.92	71.05
					194	3238	3432	0.01	20	3	3235	174	10.31	86.96
					194	3238	3432	0.05	7	2	3236	187	3.61	77.78

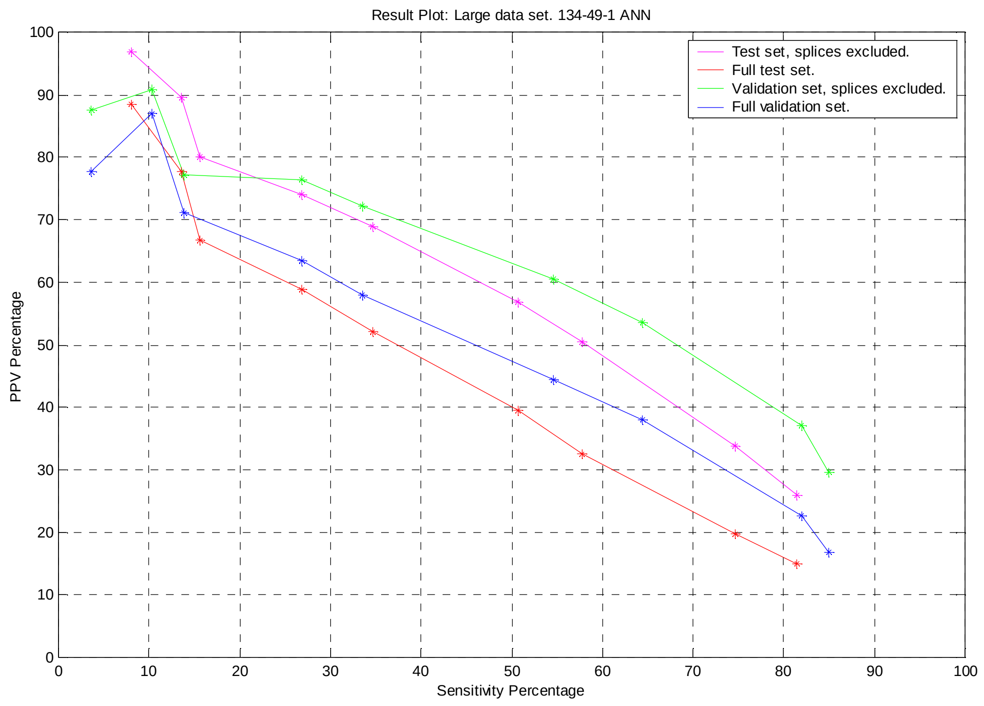

Figure 36: Sensitivity VS. PPV of large training set, large ANN.

For both the ANNs it was found that the final validation set results were slightly worse than the test set outputs. This suggests that the system still does not generalise very well, but overall the results follow similar trends.

6.5 NNPromoterFind1.0 compared to other systems.

To evaluate the quality of NNPromoterFind1.0, it will now be compared with other similar promoter finding algorithms and programs. Unfortunately the exact validation set used by other authors could not be obtained, so the comparison method is not 100% reliable. To ensure accurate comparisons can be made the result graph used by Bajic et al. [7] in their published paper is used directly, given as Figure 37 below. Although a few different systems are given on the graph, Dragon Promoter Finder (DPF[1]) is clearly the best of these systems (and is also claimed to be the state of the art in [7]). Hence, our comparison will focus on the results obtained with this system.

[1] DPF – Dragon Promoter Finder

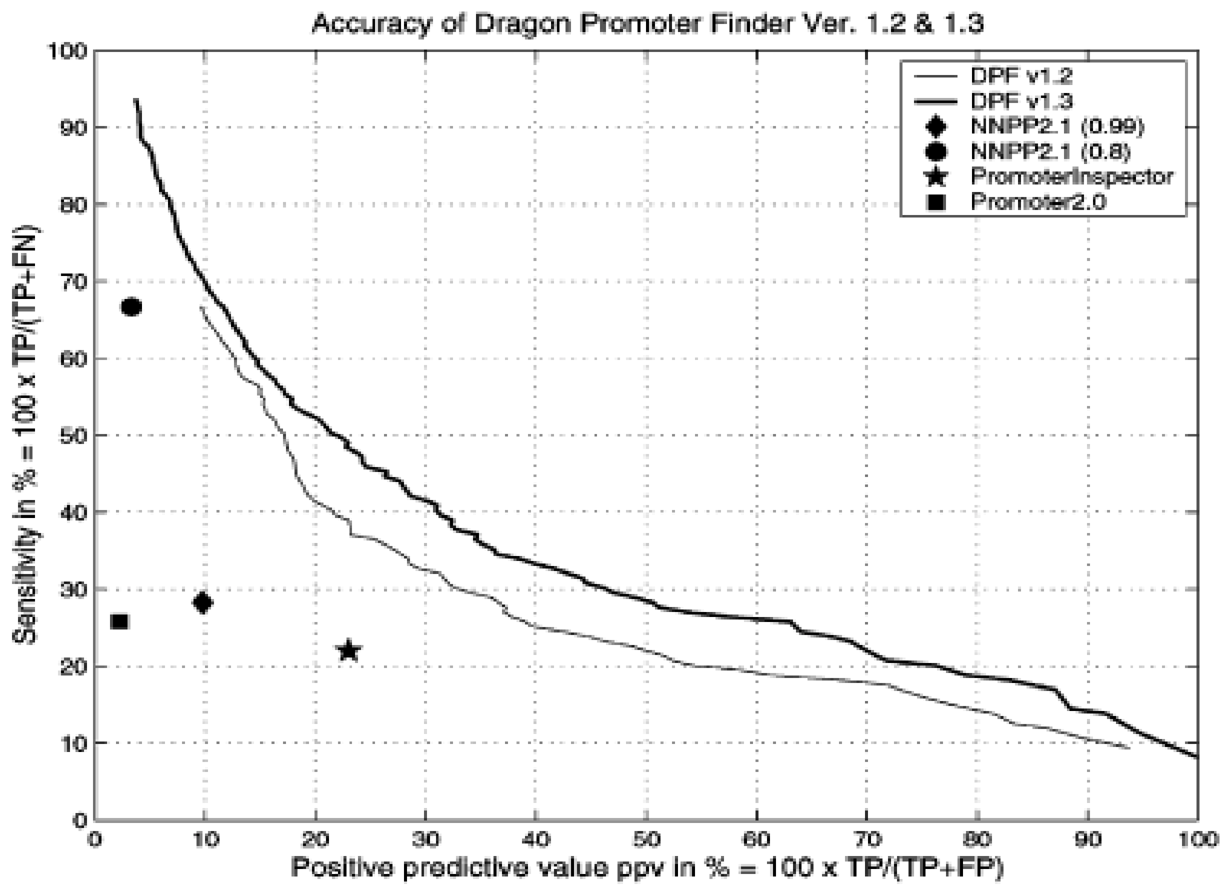

Figure 37: Other promoter detection systems.

Figure taken directly from the paper published by Bajic et al. [7]. p329

Table 11: DPV v1.3 VS. NNPromoterFind1.0.

	DPV v1.3 PPV	**NNPromoterFind1.0 PPV**			
		50-20-1 ANN		134-49-1 ANN	
Sensitivity		Equal set	Large set	Equal set	Large set
20	53	74	68	77	68
30	43	68	47	72	61
40	34	63	32	65	54
50	28	58	27	57	47
60	26	53	24	55	41
70	23	44	20	48	34
80	19	24	15	38	25

By looking at the final results shown in Table 10, above, it can be seen that NNPromoterFind1.0 gave superior results. Only one of the tested ANN combinations proved to be less accurate: the small ANN tested on the large data set. This result is not unexpected as the small ANN had too few features to discriminate against the large number of non-promoters (it was trained using only 73% of the available promoters). Even with all of this it performed only slightly worse (maximum of 4% worse) than DPF. The large ANN (134 input features) displays the most realistic results, because of the larger number of non-promoters used. The large ANN performed better than DPF with performance increases of between 6% and 20% found at different sensitivity levels.

7 CHAPTER 7: FURTHER ENHANCEMENTS AND CONCLUSION

7.1 Suggested system improvements.

7.1.1 Feature selection 1 or 2 bases upstream and downstream.

Selecting features based on entropy introduces a new problem to the classification system, namely the positioning of the features. As mentioned in the beginning of this document, DNA features are separated by DNA segments of random length. This means that the way features were selected in NNPromterFind1.0 can be improved by not only checking features at a specified position, but also including checks for the same feature at the positions one or two bases upstream as well as downstream. By simply looking at the surrounding positions in close vicinity more actual promoters might be correctly classified.

This is only suggested, and not implemented because it is the opinion of the author that this suggestion by itself will have limited impact. The reason for this is that the features were carefully selected based on entropy, and this process did actually look at ALL the positions, already including the ones 1 and 2 bases up-/downstream. So although it might help to correctly identify one or two samples the overall effect will probably be limited, or even negative(because many non-promoters may also be incorrectly identified as promoters). To make this idea practical, additional refinements will be required.

7.1.2 Detecting splices first.

While doing entropy calculations it was found that the splice sites, both I2E and E2I actually had stronger features than the promoter samples. This was the main reason for not including the splice sites in the entropy calculations in the first place. A possible method to improve the entire classification system is to first extract the splice site features using the entropy calculations, then to develop an entire splice detecting ANN, one for each splice. Unknown samples can then first be checked using the splice detecting ANNs. If a sample is not classified by these ANNs as being either of the splices it is forwarded further to the main promoter detecting ANN. As an experiment two feature sets were extracted for the two splice networks. 77 features were selected for the I2E ANN and 71 features for the E2I ANN. The 55 input small promoter ANN was used.

Using the 55-20-1 promoter ANN a sensitivity of 70.86% was used to get a PPV of 57.54% when used on its own, when the splice detecting ANNs were not used. When the test samples were first checked by the splice detecting ANNs, the sensitivity dropped slightly to 70.00% but the PPV was increased to 67.31%. This is a PPV increase of about 10% by simply filtering out the splice sites first. This is due to the fact that some of the splice sites contain some features that were selected as promoter classifier features. By detecting and eliminating these samples first a much better promoter PPV can be obtained.

7.1.3 Using different feature sets.

Another way to improve the overall system performance is to spend more time selecting the features. The entropy method works fairly well, as seen in the results given, but the threshold line parameters in the entropy program were selected heuristically. Selecting a different threshold function will result in different features. The features selected are good entropy features according to the training set, but might not necessarily be good generalisation features. There seems to be much scope for developing a more sophisticated way of trading off entropy against the number of occurrences observed.

7.1.4 Using independent ANNs.

One last possibility to improve overall performance is to have a few independent ANNs. The unknown samples are then introduced to all of the ANNs, and only if all of them classify an unknown feature as being a promoter it is classified as such. The independent ANNs could all have a different feature set, resulting in smaller, faster trainable ANNs. Alternatively each ANN could have the same feature set, but trained for different periods, or on different training data. This results in better generalisation, as shown by Wolpert [].

7.1.5 Conclusion on improvement.

In conclusion, although the current system works fairly well, there is reason to believe that it can be improved more sophisticated feature-selection schemes. If good feature selection is done a better classifier can be trained. Better features can also be obtained if more promoter and better promoter samples are available. Currently the number of promoter samples is a constant of 1 871. When compared to the thousands of possible introns, exons and splice samples it is obvious that this is not sufficient, especially when one considers the millions of actual bases in the real chromosome data.

7.2 Final Conclusion.

The overall problem of automated promoter detection was solved with comparative success. An automated system was developed that could be implemented to train a classifier system with ease. The feature selection and training process is quite time-consuming, but the final classifier is very fast. The system also showed good improvement when compared to its current state of the art competitors.

The most significant hurdle towards writing a reliable automated promoter detection algorithm is to obtain a sufficient number of promoter samples. More samples, and more diverse samples, will improve the overall performance of the classifier system. Beyond that, improved understanding of the biochemistry of promoters is required for further improvements.

Our novel approach, which uses only certain features in the sample instead of the entire 256 base length sequence proved to be very useful, but much further work can be done to obtain better and more reliable features. When analysing the results, it was shown that feature selection can be improved substantially to make sure that the ANN is trained correctly on all of the data, not only on some of the samples as was the case with NNPromoterFind1.0.

Overall the project was a success, but with numerous suggestions for future work. We have made some progress in terms of performance and understanding of automated promoter detection, but much remains to be done.

REFERENCES

[1] HF Lodish, JE Darnell and D Baltimore, *Molecular Cell Biology*, 3rd ed, Scientific American Books, New York. 1995.

[2] RH Tamarin, *Principles of Genetics*, 6th ed, McGraw-Hill Boston. pp. 243-279, 1999.

[3] D Latchman, *Gene Regulation – A Eukaryotic perspecrive*, 3rd ed, Stanley Thornes, Cheltenham. 1998.

[4] T Beebee and J Burke, *Gene structure and transcription*, 2nd ed, IRL Press, New York. 1992.

[5] RP Wagner, MP Maguire and RL Stallings, *Chromosomes – A Synthesis*, Wiley-Liss, New York. 1993.

[6] A Kornberg and TA Baker, *DNA Replication*, 2nd ed, WH. Freeman, New York. 1992.

[7] VB Bajic, SH Seah, A Chong, SPT Krishnan, JLY Koh and V Brusic, Computer model for recognition of functional transcription start sites in RNA polymerase II promoters of vertebrates, Journal of Molecular Graphics and Modelling, Vol 21, pp 323–332, 2003.

[8] S Knudsen, Promoter2.0: for the recognition of PolII promoter sequences, Bioinformatics, Vol 15. no. 5, pp 356-361, 1999.

[9] SH Cross and AP Bird, CpG islands and genes, Curr. Opin. Genet. Dev., 309-314, 1995.

[10] F Larsen, G Gundersen, R Lopez and H Prydz, CpG islands as gene markers in the human genome, Genomics Vol 13, pp 1095-1107, 1992.

[11] M Scherf, A Klingenhoff and T Werner, Highly specific localisation of promoter regions in large genomic sequences by PromoterInspector: a novel context analysis approach, J. Mol. Boil. Vol 297, pp 599-606, 2000.

[12] U Ohler, H Niemman, GC Liao and GM Rubin, Joint modelling of DNA sequence and physical properties to improve eukaryotic promoter recognition, Bioinformatics, Vol 17 (Suppl. 1), pp 199-206, 2001.

[13] CM Bishop, *Neural Networks for Pattern Recognition*, Oxford university press, New York. 2003.

[14] M Negnevistky, Artificial Intelligence – A Guide to Intelligent Systems, Addison-Wesley, pp 163-216, 2002.

[15] S Russell and P Norvig, *Artificial Intelligence – A Modern Approach*, 2nd ed, Prentice Hall, Upper Saddle River, USA. pp. 736-748, 2003.

[16] RC Périer, V Praz, T Junier, C Bonnard and P Bucher, The Eukaryotic Promoter Database (EPD), Nucl. Acid Res. Vol 28, pp 302-303, 2000.

[17] ftp://ftp.ncbi.nih.gov/genomes/H_Sapiens, 24 Augustus 2003.

[18] G Myburgh, *Report companion CD-Rom*, 26-February 2005.

ADDENDUMS

1 Extract.c → Program used to extract non-promoter sequences from GBK files.

See companion cd-rom. (cd-drive:\Programs\Extract\Extract.c)

2 Length Calculation for sample extraction

To ensure that there is no overlap in the data some careful considerations had to be made during the extraction process. In the first run through the GBK file the program extract.c (Addendum 1) marks the start of all the available splice borders in different files. The splice borders occur in the original GBK file in the format:
((mRNA..E2I),(I2E..E2I)…(I2E..E2I))

These splice borders were extracted, checked for validation and then written to files, one file for each different sample type.

exo2int.txt	=> Stores all the E2I splice sample start positions.
int2exo.txt	=> Stores all the I2E splice sample start positions.
start	=> Stores all the mRNA start site positions.
Intron	=> Stores all the Intron sample start positions.
Exon	=> Stores all the Exon sample start positions.

Each sample contains the data –200 to +55 with reference to the actual splice. This means that the value read in the GBK file should firstly be decreased by 200 to get the position in the sequence where the sample starts. To make sure that each sample is a **valid** 256-bases long without overlapping with neighbouring samples a few different scenarios should be considered. As the splice sites and mRNA start position samples are more specific than intron or exon samples these three classes were extracted and validated first. The values stored in the abovementioned text files are the actual start positions of the sample, and thus not the value read directly from the GBK file, but rather the value read minus 200.

A2.1 Valid length calculations

Consider the gene sequence in Figure A1 with markings (A), (B), (C), (D) and (E) extracted from the GBK file.

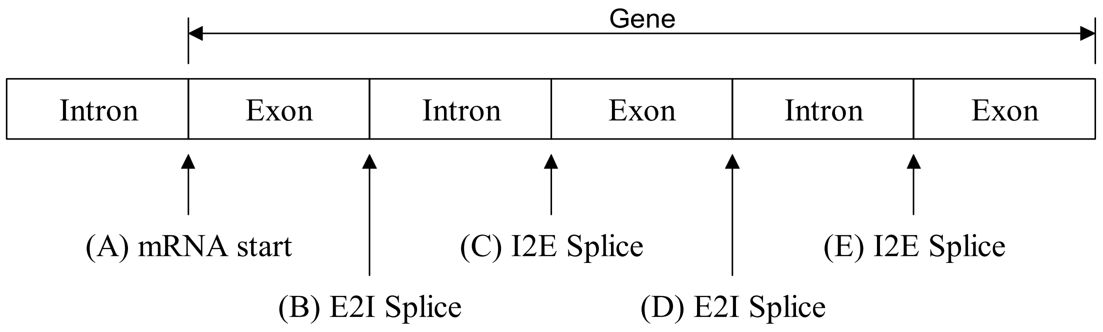

Figure A1: Calculation of valid sample length.

A2.1.1 All valid.

For a sample to be valid it must contain 256 bases without overlapping into the previous or the next sample. The easiest way to make certain that all samples are valid is to use only splice markings that are at least 2*256 = 512 bases from each other as seen in Figure A2. This unfortunately does not leave many valid samples, so the process has to be refined by looking at different scenarios.

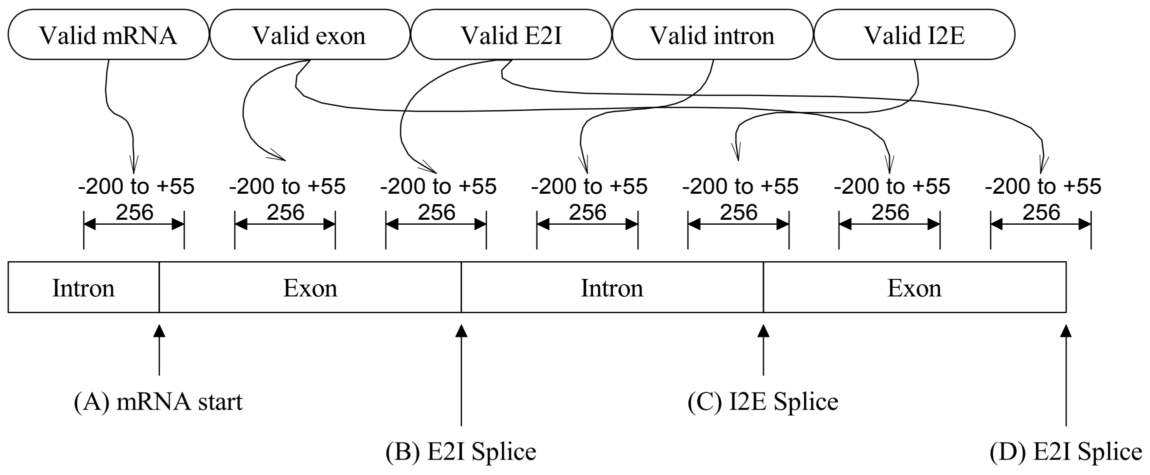

Figure A2: All valid length samples. Splices are more than 512 bases from each other.

In cases like this care should be taken to ensure that each sample is non-overlapping by doing boundary checking. For example, if one wanted to extract the exon sample between points (A) and (B) one has to make sure that the exon sample starts at least 55 bases after point (A), and ends at least 200 bases away from point (B).

Exon = Valid if and only if Exon-Start $>$ (A)+55 and Exon-End $<$ (B)-200.

A2.1.2 Only splices valid.

The first other possibility is that both the intron and the exon are more than 256 bases, but less than 512 bases. In these cases valid splice site (and mRNA site) samples can be extracted, but intron and exon samples cannot be extracted. This is shown in Figure A3.

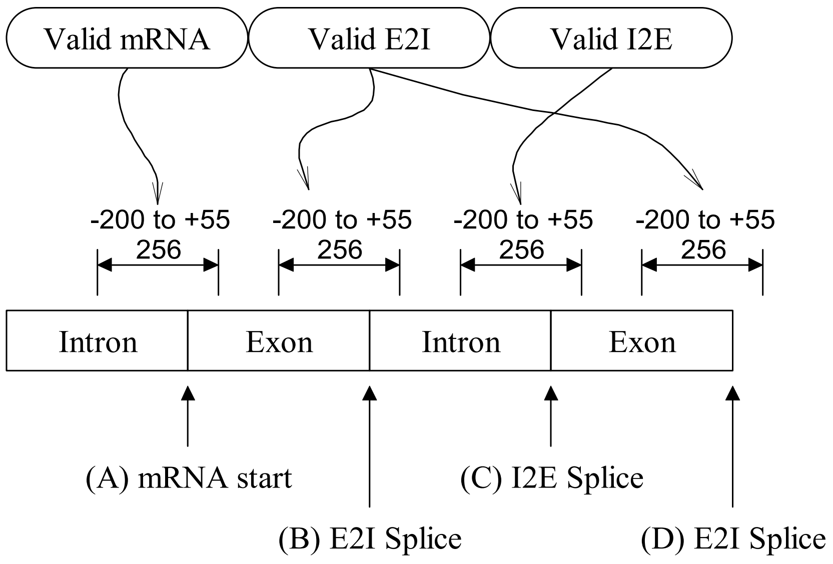

Figure A3: Valid splices. 256 bases < Splice separation < 512 bases.

There will be cases where, say, the intron before (C) is more than 512 bases long, but the exon after (C) is only 256 bases long. In such a case (C) is still a valid I2E splice sample, the intron before it contains a valid intron sample but the exon area will not contain a valid sample. The same is true for the other way around where the intron is too short and but the exon is long enough to contain a sample. For splices (B) and (D) the same can happen, with the positions of the intron and exon simply reversed.

A2.1.3 None valid.

The last case to consider is when splice sites are closer than 256 bases from each other. In such cases it is obviously not possible to extract intron or exon data since a 256 sample cannot fit between the splice. It is also not possible to extract splice samples, as the end of the first sample will overlap with the start of the next one. For example if one takes Figure A1 again and let (B) and (C) be only, say, 210 (anything less than 256) bases from each other. Now the E2I sample centred at (B) ends at (B)+55, while the I2E sample centred around (C) starts at (C)-200.

But (C) – (B) = 180 < 256, and in other words (C) = 180 + (B), and (B) = (C) – 180.

Start position of the I2E sample:

Start = (C) – 200
= [210 + (B)]- 200
= (B) + 10.

End position of the E2I sample:

End = (B) + 55
= [(C) – 210] + 55
= (C) – 155.

So if the intron between (B) and (C) is only 210 bases from each other the I2E sample overlaps the E2I sample with 45 (210 – 10 – 155 = 45) bases.

256 – 210 = 46. Thus the minimum distance between two splices is exactly the same length as the samples, 256 bases long.

3 ClassSets.c → Program used to split each class into 3 different sets.

See companion cd-rom. (cd-drive:\Programs\Sets\ClassSets.c)

4 PullData.m → Matlab program used to extract position statistics.

See companion cd-rom. (cd-drive:\Programs\Statistics\PullData.m)

5 PullHalf.m → Second Matlab program used to extract position statistics.

See companion cd-rom. (cd-drive:\Programs\Statistics\PullHalf.m)

6 Statistics.c → Program used to extract position statistics.

See companion cd-rom. (cd-drive:\Programs\Statistics\Statistics.c)

7 Funcdraw.m → Grahical output of statistics.

See companion cd-rom. (cd-drive:\Programs\Statistics\Funcdraw.m)

A7.1 Funcdraw.m → Explanation of the function and its input arguments.

```
function
funcDraw=funcDraw(n,what,start,stop,percentage,percentage2,positionDataPromoter,posi
tionDataMRNA,positionDataI2E,positionDataE2I,positionDataIntron,positionDataExon,cou
ntPromoter,countMRNA,countI2E,countE2I,countIntron,countExon)
```

Input arguments:

n: integer with the n-tuple length

What: Integer value that determines the graph type, explained in more detail later since it is one of the most important parameters.

Start: start n-tuple (See what = 2,3 and 4)

Stop: stop n-tuple (See what = 2,3 and 4)

Percentage: minimum plot percentage (See what = 9)

Percentage2: maximum plot percentage (See what = 10)

The next six parameters are the arrays containing the position data for each class as generated by the programs pullData.m or pullHalf.m

The dimensions of these arrays are (4^n+1) rows and (256-n+1) columns.

PositionDataPromoter:

PositionDataMRNA:

PositionDataI2E:

PositionDataE2I:

PositionDataIntron:

PositionDataExon:

The next six parameters are the vectors containing the percentage times each n-tuple occurred in each of the classes, once again generated by the programs pullData.m or pullHalf.m

They contain (4^n) entries each, corresponding to the 4^n possible n-tuples of length n.

CountPromoter:

CountMRNA:

CountI2E:

CountE2I:

CountIntron:

CountExon:

The possible options of the what parameter

What == 1.

Plot 6 graphs, each containing on of the class types. This was the option used to generate the graphs shown in Section 4.4.

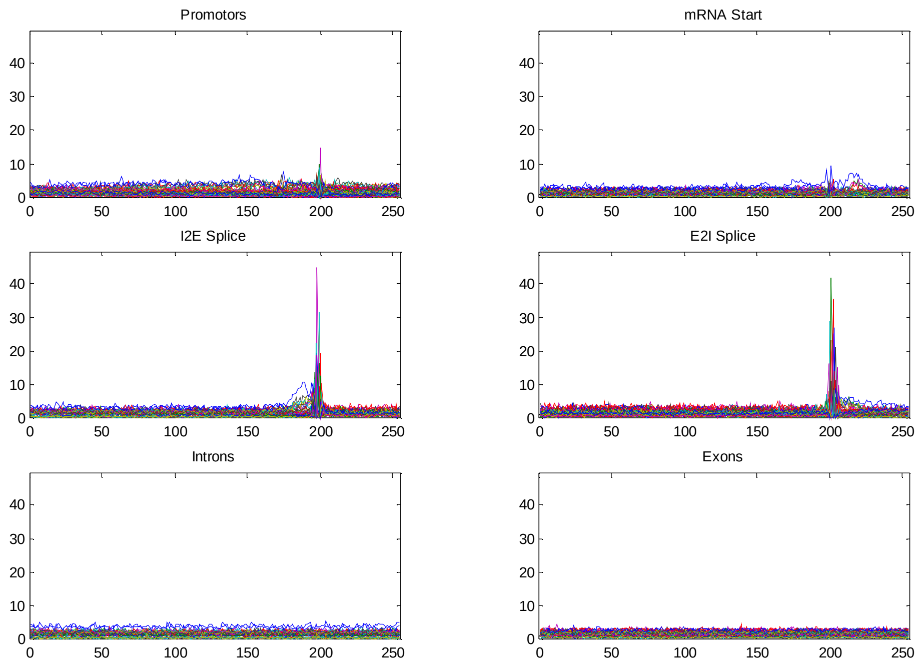

Figure A4: FuncDraw with parameter what == 1.

What == 2.

Uses the parameters Start and Stop.

Plots the total occurrence of each n-tuple between Start and Stop for promoters and splices.

Note that the parameter n is ignored for bounds calculation.

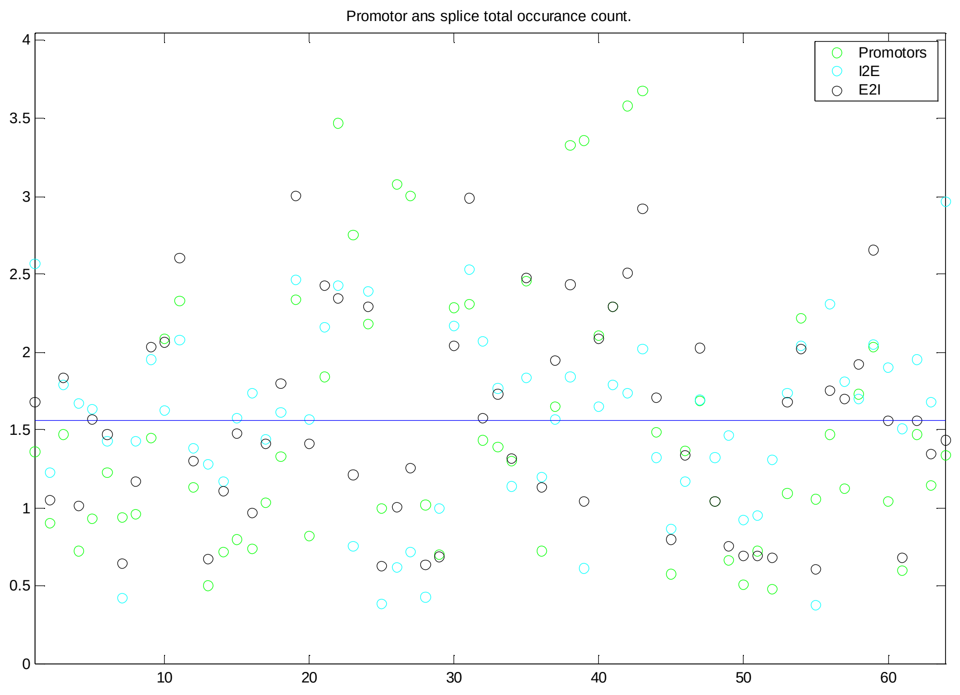

Figure A5: FuncDraw with parameter what == 2.

What == 3.

The same as What==2, but the Intron and Exon classes are included.

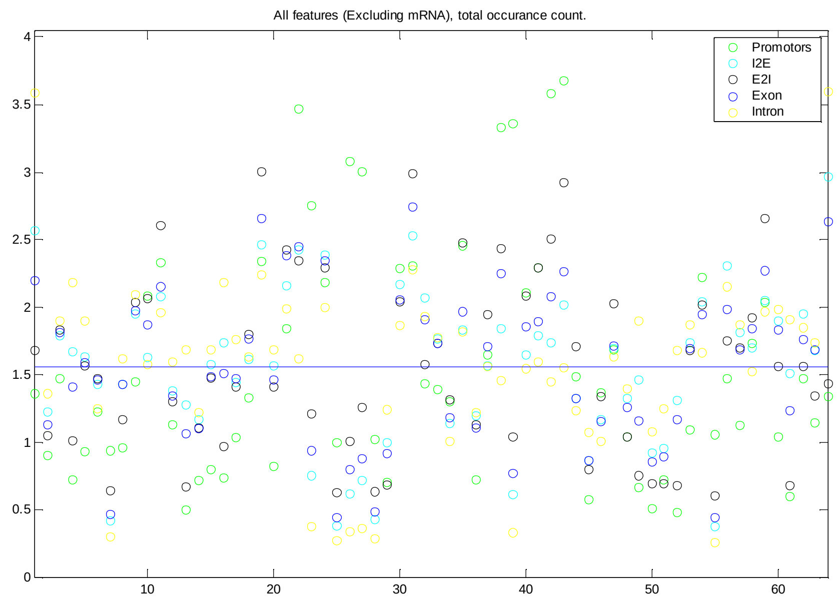

Figure A6: FuncDraw with parameter what == 3.

What == 4.

The same as What==3, with the inclusion of the mRNA class. Thus all 6 classes are drawn when the What parameter is set to 4.

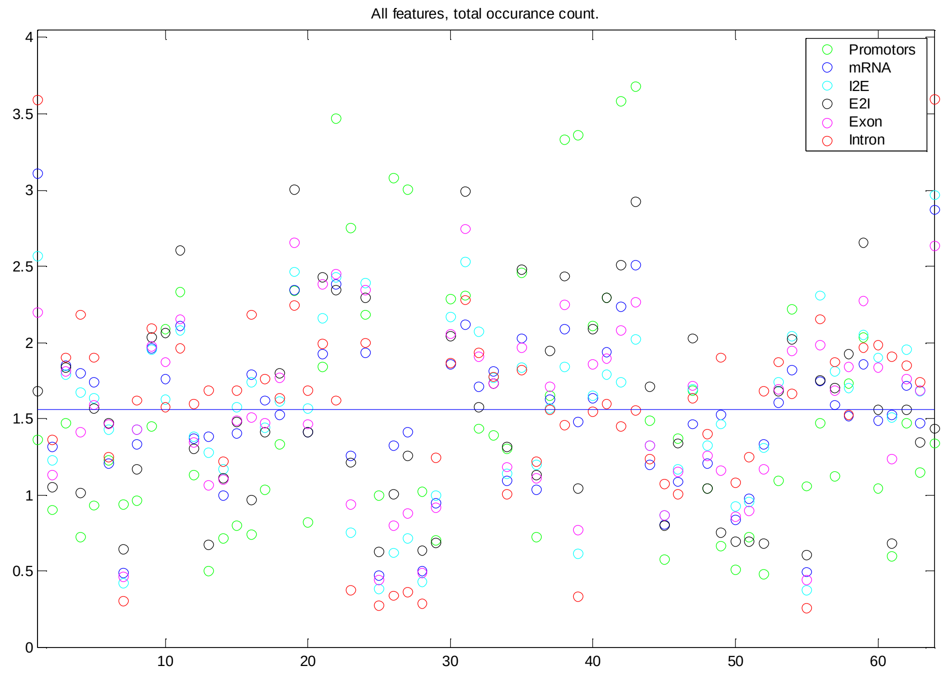

Figure A7: FuncDraw with parameter what == 4.

What == 5.

This gives a similar graph to 2-4, but with the difference that only the promoters and the mRNA samples are used. This was done to see the correlation between the actual downloaded promoters and the ones extracted from the computer marked files.

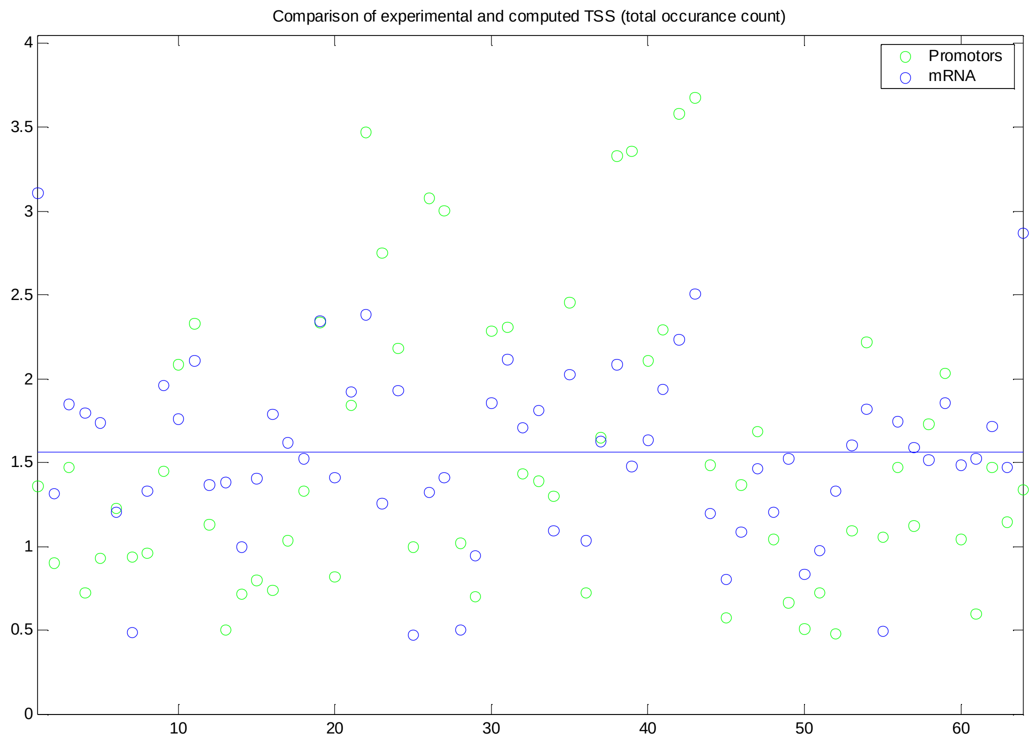

Figure A8: FuncDraw with parameter what == 5.

From this figure it can be seen that the actual promoters differ quite a lot from the mRNA samples.

What == 6.

This option provides an output similar to What==4, but with two differences. Firstly the value of n is actually used to calculate borders. Secondly, and much more importantly is that all the values are normalised to be between 0 and 1. This option also provides borders that determine where possible features could be.

For example, in the figure there is an Intron at n-tuple 55 that is below the 0.25 line, with all the other classes marked above the line. This could mean that the absence of this feature is an indicator of Introns. This was a very crude method used to extract features before the Entropy calculations were used.

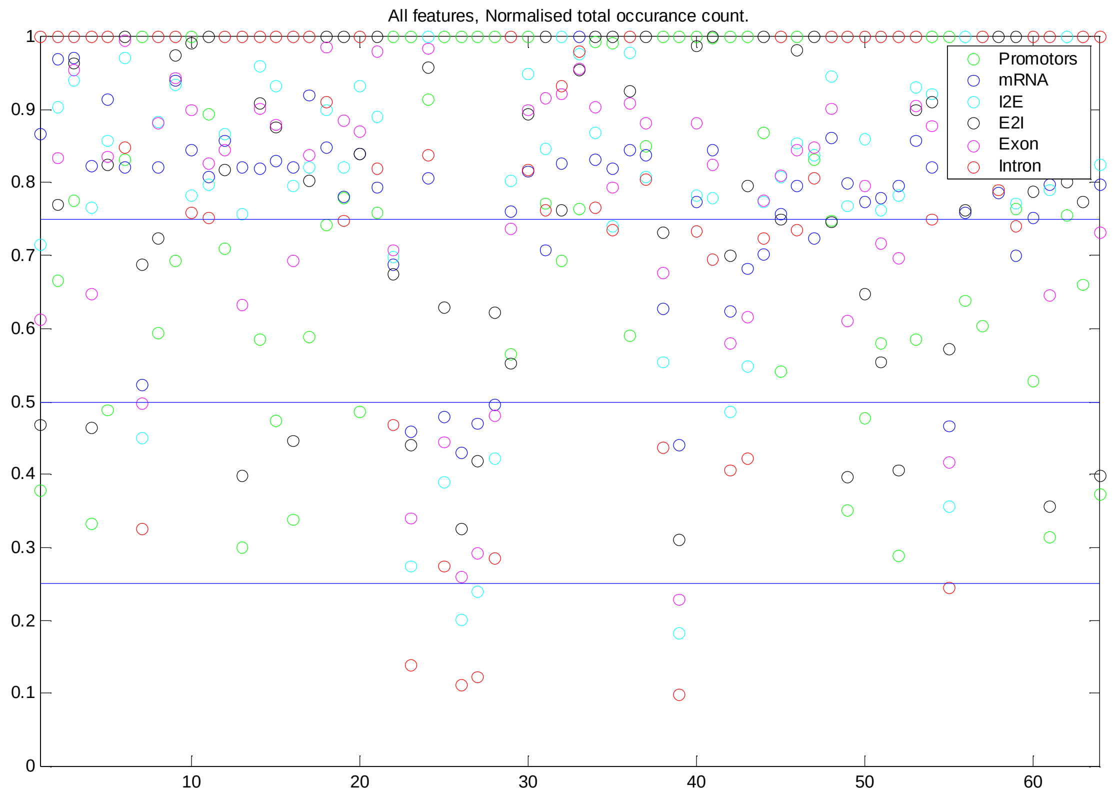

Figure A9: FuncDraw with parameter what == 6.

What == 7.

This option draws a graph similar to What==1, but where What==1 draws all the n-tuples of length n, this option only draws the n-tuples between the Start and Stop values. A single n-tuple can be isolated and plotted down by setting the Start and Stop parameters to the same value.

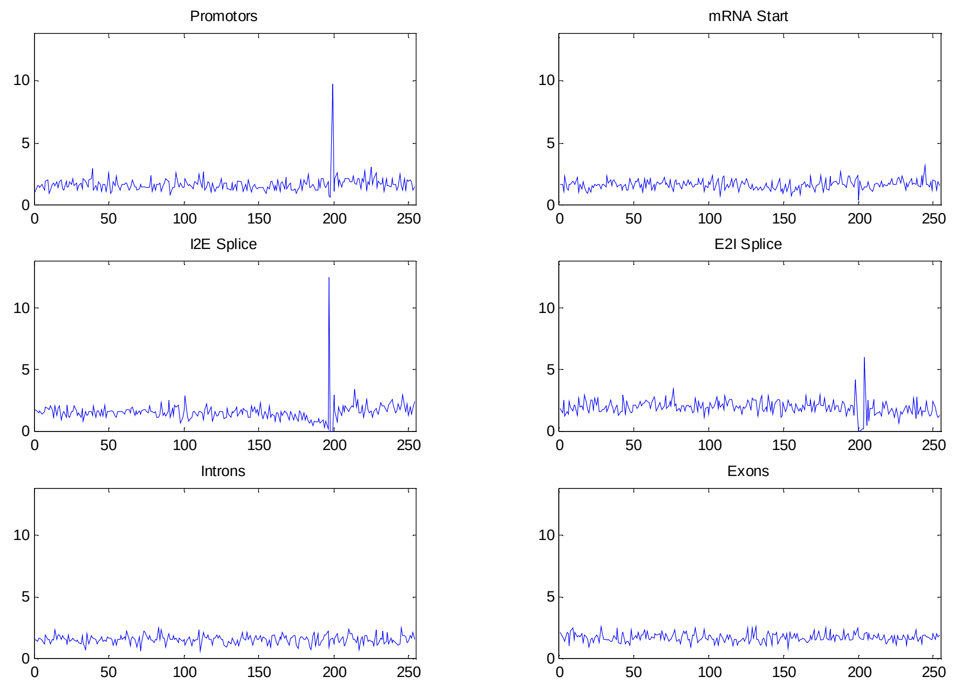

Figure A10: FuncDraw with parameter what == 7.

For example the 3-tuple number 37 was isolated and plotted. This 3-tuple is a possible feature for extraction when one looks at position 200 on the promoter graph.

What == 8.

It is the same as What==1, except that only n-tuples that have a percentage value greater than the Percentage parameter are shown on the plot.

What == 9.

It is also the same as What==1, except that only n-tuples that have a maximum percentage value greater than the Percentage parameter, AND smaller than the Percentage2 parameter are shown on the plot.

8 Entropy calculation.

A8.1 GetMax.m → Function used by GetEnt5.m

See companion cd-rom. (cd-drive:\Programs\Entropy\GetMax.m)

```
%getMax(6,199,positionDataPromoter,positionDataIntron,positionDataExon,posi
tionDataI2E,positionDataE2I)

function
[maxType]=getMax2(nTuple,Pos,positionDataPromoter,positionDataIntron,positi
onDataExon)

maxType = 0;
max = -1;

if (positionDataPromoter(nTuple,Pos) > max)
   max = positionDataPromoter(nTuple,Pos);
   maxType = 1;
end;

if (positionDataIntron(nTuple,Pos) > max)
   max = positionDataIntron(nTuple,Pos);
   maxType = 2;
end;

if (positionDataExon(nTuple,Pos) > max)
   max = positionDataExon(nTuple,Pos);
   maxType = 3;
end;
```

A8.2 GetEnt5.m Entropy extraction function.

See companion cd-rom. (cd-drive:\Programs\Entropy\GetEnt5.m)

A8.3 Entropy function parameters.

```
function
[realEnt2,value]=getEnt5(dataType,useAll5,Var1,Var2,n,start,stop,nStart,nSt
op,significance,positionDataPromoter,positionDataMRNA,positionDataI2E,posit
ionDataE2I,positionDataIntron,positionDataExon,countPromoter,countMRNA,coun
tI2E,countE2I,countIntron,countExon,a,b,c,d,e,f);
```

Input Parameters:

DataType: Integer value that selects the class for which the entropy is calculated. 1=Promoter, 2=Intron, 3=Exon, 4=I2E, 5=E2I

useAll5: Flag that selects to use all five the classes for entropy calculation or only the promoters, introns and exons. 1 = All 5, 0 = Exclude splices.

Var1: Used to determine the threshold border.

Var2: Used to determine the threshold border.

Border value at n-tuple “i” = Var1/I + Var2;

N: Like all other functions the n-tuple length.

Start: The start postion for which entropy is calculated.

Stop: The stop position for which entropy is calculated.

NStart: The start n-tuple for which entropy is calculated.

NStop: The stop n-tuple for which entropy is calculated.

The entropy is calculated only for the position beginning at Start and ending at Stop and for the n-tuples between nStart and nStop.

Significance: The minimum number of samples per bin before entropy is considered good.

The next six parameters are the arrays containing the position data for each class as generated by the programs pullData.m or pullHalf.m
The dimentions of these arrays are (4^n+1) rows and (256-n+1) columns.

PositionDataPromoter:

PositionDataMRNA:

PositionDataI2E:

PositionDataE2I:

PositionDataIntron:

PositionDataExon:

The next six parameters are the vectors containing the percentage times each n-tuple occurred in each of the classes, once again generated by the programs pullData.m or pullHalf.m
They contain (4^n) entries each, corresponding to the 4^n possible n-tuples of length n.
CountPromoter:
CountMRNA:
CountI2E:
CountE2I:
CountIntron:
CountExon:

And the last six parameters are integer values containing the number of samples available for each class.

A:	Number of Introns
B:	Number of Exons
C:	Number of I2E
D:	Number of E2I
E:	Number of Promoters
F:	Number of mRNA

Output Parameters:
RealEnt2: Contains an array with the entropy values for the area between the border parameters Start, Stop, nStart and nStop.

Value: Contains an array with the same dimensions as the RealEnt2 array, but with flag values of 0 or 1. 0 indicating that the entropy is not valid because the position – n-tuple bin did not contain enough samples (Enough being the value given in the parameter "significance"), and a 1 indicating that there was enough samples.

9 WorkANN.c → ANN training program.

See companion cd-rom. (cd-drive:\Programs\ANN\WorkANN.c)

10 ClassCombine.c → Program that combines classes to one file.

See companion cd-rom. (cd-drive:\Programs\ANN\ClassCombine.c)

11 LoadANN.c → ANN testing program and final system output.

See companion cd-rom. (cd-drive:\Programs\ANN\LoadANN.c)

Inventory of the companion CD-Rom.

Extract.c	and	Extract.exe
ClassSets.c	and	ClassSets.exe
Pulldata.m		
Pullhalf.m		
Statistics.c	and	Statistics.dll
Funcdraw.m		
GetMax.m		
GetEnt5.m		
WorkANN.c	and	WorkANN.exe
ClassCombine.c		
LoadANN.c		