

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES

ANALYSIS OF GENETIC DATA VIA DATA
MINING METHODS AND ITS APPLICATIONS

by

Sezin TUNABOYLU

June, 2011

İZMİR

ANALYSIS OF GENETIC DATA VIA DATA MINING METHODS AND ITS APPLICATIONS

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Statistics**

by

Sezin TUNABOYLU

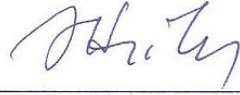
June, 2011

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “ANALYSIS OF GENETIC DATA VIA DATA MINING METHODS AND ITS APPLICATIONS” completed by **SEZİN TUNABOYLU** under supervision of **PROF. DR. EFENDİ NASİBOĞLU** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

PROF. DR. Efendi NASİBOĞLU



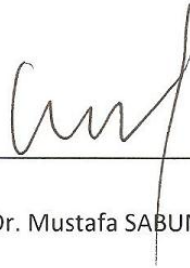
Supervisor

U. Uslu
Doç. Dr. Kemal S. Korkmaz

(Jury Member)

Özlem Ege Çerç
Yrd. Doç. Dr. Özlem EGE ÇERÇ

(Jury Member)



Prof. Dr. Mustafa SABUNCU

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Dr. Efendi NASİBOĞLU for his direction, assistance, and guidance throughout my Master of Science years. I feel very special to have worked with him.

Finally, words alone cannot express the thanks I owe to my husband Çağdaş Tunaboşlu for his help and encouragement. And a special thanks to my parents Leyla and Hasan TANSI for their endless encouragement and support throughout my life.

Sezin TUNABOYLU

ANALYSIS OF GENETIC DATA VIA DATA MINING METHODS AND ITS APPLICATIONS

ABSTRACT

The prediction of the complete structure of genes is one of the important tasks of bioinformatics, especially in eukaryotes. A crucial part in gene structure prediction is to determine the splice sites in the coding region. Identification of splice sites depends on the precise recognition of the boundaries between exons and introns of a given DNA sequence. This problem can be formulated as a classification of sequence elements into ‘exon-intron’ (EI), ‘intron-exon’ (IE) or ‘None’ (N) boundary classes.

In this thesis, we propose a new Weighted Position Specific Scoring Method (WPSSM) to recognize splice sites which uses a position-specific scoring matrix constructed by nucleotide base frequencies. A genetic algorithm is used in order to tune the weight and threshold parameters of the positions on. This method comprises of three phases: learning phase, identification phase and validation phase. In this study, the optimal position weights and threshold parameter are found via genetic algorithm. The proposed WPSS method poses efficient results compared to the performance of various methods proposed in the literature. Computational experiments are conducted on the DNA sequence dataset from ‘UCI Repository of machine learning databases’.

Keywords: Exon-Intron Splice Sites, Weighted Position Specific Scoring Method, Genetic Algorithm.

GENETİK VERİLERİN VERİ MADENCİLİĞİ YÖNTEMLERİ İLE İNCELENMESİ VE UYGULAMALARI

ÖZ

Özellikle ökaryotlarda gen yapılarının tahmin edilmesi biyoformatiğin önemli konularından biridir. Gen yapılarının tahmin edilmesindeki en önemli konu kodlanan bölgelerdeki kesim bölgeleridir. Kesim bölgelerinin belirlenmesi de verilen DNA dizisindeki eksonlar ve intronlar arasındaki bölgelerin doğru tanımlanmasına bağlıdır. Bu problem dizi elementlerinin ‘exon-intron’ (EI), ‘intron-exon’ (IE) or ‘None’ (N) sınıfları olarak sınıflandırılması olarak tanımlanabilir.

Bu tez çalışmasında, kesim bölgelerini belirlemek için geliştirilen yeni bir yöntem olan ve nükleotit baz frekanslarından oluşan spesifik pozisyonel skorlama matrisini kullanan Ağırlıklandırılmış Pozisyonel Skorlama Metodu (WPSSM) önerilmiştir. Ayrıca optimal ağırlıkların ve eşik değerinin belirlenmesinde genetik algoritma kullanılmıştır. Bu metod öğrenme, tanımlama ve geçerlilik aşamaları olmak üzere üç aşamadan oluşmaktadır. Önerilen WPSS metodu literatürdeki pek çok metodun performansı ile karşılaştırıldığında etkili sonuçlar vermiştir. Hesaplamalar, ‘UCI Repository of machine learning databases’ veri tabanından alınan DNA dizileri üzerinde gerçekleştirilmiştir.

Anahtar Kelimeler: Ekson-Intron Kesim Bölgeleri, Ağırlıklandırılmış Pozisyonel Skorlama Metodu, Genetik Algoritma.

CONTENTS

	Page
M.Sc. THESIS EXAMINATION RESULT FORM.. Error! Bookmark not defined.	
ACKNOWLEDGMENTS	ii
ABSTRACT.....	iv
ÖZ	v
CHAPTER ONE - INTRODUCTION	1
1.2 Historical Development of Bioinformatics	3
1.3 Tasks in Bioinformatics.....	4
1.4 Splice Sites	8
CHAPTER TWO - MATERIALS and METHODS	12
2.1 Dataset	12
2.2 Position-Specific Scoring Matrix (PSSM)	12
2.2.1 The computational steps of PSSM.....	13
2.2.2 Prediction of the Sample's Class	14
2.2.3 PSSM Example	15
2.3 Genetic Algorithm (GA)	17
2.3.1 Terminologies of Genetic Algorithm.....	18
2.3.2 Operators of Genetic Algorithm	20
2.4 Cross Validation Method.....	31

CHAPTER THREE - WEIGHTED POSITION-SPECIFIC SCORING METHOD	34
3.1 Learning Phase	34
3.2 Identification Phase	35
3.3 Validation phase	37
CHAPTER FOUR - RESULTS AND DISCUSSION	40
CHAPTER FOUR - CONCLUSION	46
REFERENCES.....	48
APPENDIX.....	53

CHAPTER ONE

INTRODUCTION

Bioinformatics is a multidisciplinary research area at the interface between computer science and biological science. Bioinformatics involves the technology that uses computers for storage, retrieval, manipulation, and distribution of information related to biological macromolecules such as DNA, RNA, and proteins (Xiong, 2006).

The ultimate aim of the bioinformatics is to better understand a living cell and how it functions at the molecular level by using computer technology. Sequence analysis of DNA sequences is one of the important tasks in bioinformatics.

The required information for the smallest living unit cell to regularly perform cellular activities, are hidden in DNA molecules, the kernel known as the brain of the cell. A DNA molecule which is a giant molecular chain contained in all living cells is considered as a "knowledge bank" since it contains genetic information.

Specific regions which encode knowledge of our physical features and physiological activities on the DNA are called "genes". These genes encode the various proteins and provide continuing of our lives. The identification of genes is a major task in bioinformatics. However, gene structure needs to be predicted for the accurate identification of genes. Recognizing splice sites in eukaryotic genes play an important role in identifying gene structure. The accurate identification of the splice sites depends on correct estimation of exon-intron structures. This accurate identification problem can be formulized as a classification problem.

Recently, many splice site classification/prediction methods have been proposed. These methods can be classified as probabilistic methods (Salzberg, 1997; Chen et al., 2005; Pertea et al., 2001; Marashi et al., 2006a; Zhang and Marr, 1993; Castello and Guigo, 2004; Cai et al., 2000), neural networks and support vector machines (Marashi et al., 2006b; Reese et al., 1997; Sun et al., 2003; Reese, 2001; Zhang et al.,

2003; Sonnenburg, 2002; Degroeve et al., 2005; Rajapakse et al., 2005; Baten et al., 2006; Sonnenburg et al., 2007; Ratsch et al., 2006), and methods based on discriminant analysis (Chuang and Roth, 2001; Zhang and Luo, 2003). Many works posed that machine learning methods has better results than canonical pattern matching based classification methods. The main methods which have good results are knowledge based neural network (KBANN) which combines explanation based and empirical learning methods (Noordewier et al., 1991), probabilistic model which estimates position-specific probabilities of splice sites (Pertea et al., 2001), Genomic Splice Site Prediction (GSSP) method which characterizes the interdependency among the nucleotides and base positions based on the eigen-patterns (Tsai et al., 2009), BRAIN algorithm which infers Boolean formulae from examples and considers the splicing rules, and Hierarchical Multi-classifier which use many classification methods (Lumini and Nanni, 2006). Also, there are influential models based on weight matrix model (WMM). The WMM weights can be optimized using a neural network method (Brunak et al., 1991). Another model uses the position-specific compositional biases in splice sites (Staden, 1984).

In this study, Weighted Position-Specific Scoring Method has been proposed as a new method concerning the classification of three classes of splice sites as “EI”, “IE”, and “NONE” classes. In this method, weighted position-specific scores which consist of frequencies of DNA sequence elements are used. The weights of the positions and threshold parameter have been calculated via genetic algorithm. The preliminary results of this study are presented in (Nasibov and Tunaboylu, 2010a). The developed study of preliminary results is presented in (Nasibov and Tunaboylu, 2010b).

This thesis is composed of five chapters. This introductory chapter is an overview of bioinformatics and preliminary works of this classification problem.

In Chapter 2, underlying methods used in the WPSSM as position specific scoring matrix, genetic algorithm and ten-fold cross validation method are discussed in a detail. The definitions and algorithms of these methods are covered.

In Chapter 3, the definitions of WPSSM's phases as learning, identification and validation phases are presented in detail; and their flow-diagrams are presented.

Later, starting from Chapter 4, preliminary WPSSM and improved WPSSM are applied to dataset from 'UCI Repository and Machine Learning' to classify sequences. The results are presented and compared with each other and other works in the literature.

In the conclusion part of this thesis, the results, novelty and advantages of the WPSSM are discussed.

1.2 Historical Development of Bioinformatics

The development of bioinformatics as a field is the result of advances in both molecular biology and computer science over the past 30–40 years. Bioinformatics arose as a fusion of two trends in biology: the application of computer programs to the analysis of protein and nucleotide sequences and the storage of molecular sequences in computer databases. The history of databases started in 1965 with the work of Margaret Dayhoff and her Atlas of Protein Sequences, which became the foundation for the first online database, the Protein Information Resource. Subsequently, in the early 1970s, the Brookhaven National Laboratory established the Protein Data Bank for archiving three-dimensional protein structures.

In the beginning, the database stored less than a dozen protein structures, compared to more than 30,000 structures today. The first sequence alignment algorithm was developed by Needleman and Wunsch in 1970. This was a fundamental step in the development of bioinformatics, which paved the way for the routine sequence comparisons and database searching practiced by modern biologists. The first protein structure prediction algorithm was developed by Chou and Fasman in 1974. Though it is rather unimproved by today's standard, it pioneered a series of developments in protein structure prediction. In the 1980s, GenBank was established and the fast database searching algorithms such as FASTA

by William Pearson and BLAST by Stephen Altschul and coworkers were developed. The start of the human genome project in the late 1980s provided a major boost for the development of bioinformatics. The development and the increasingly widespread use of the internet in the 1990s made instant access, exchange and dissemination of biological data possible. These are only the major milestones in the establishment of this new field. The fundamental reason that bioinformatics gained prominence as a discipline was the advancement of genome studies that produced unprecedented amounts of biological data. The explosion of genomic sequence information generated a sudden demand for efficient computational tools to manage and analyze the data. The development of these computational tools depended on knowledge generated from a wide range of disciplines including mathematics, statistics, computer science, information technology, and molecular biology. The merger of these disciplines created an information oriented field in biology, which is now known as bioinformatics (Xiong, 2006).

1.3 Tasks in Bioinformatics

Bioinformatics comprises of two subfields: the development of computational tools and databases and the application of these tools and databases in generating biological knowledge to better understand living systems. These two subfields are complementary to each other. The tool development includes writing software for sequence, structural, and functional analysis, as well as the construction of biological databases. These tools are used in three areas of genomic and molecular biological research: sequence analysis, structural analysis, and functional analysis. The analyses of biological data often generate new problems and challenges that in turn spur the development of new and better computational tools.

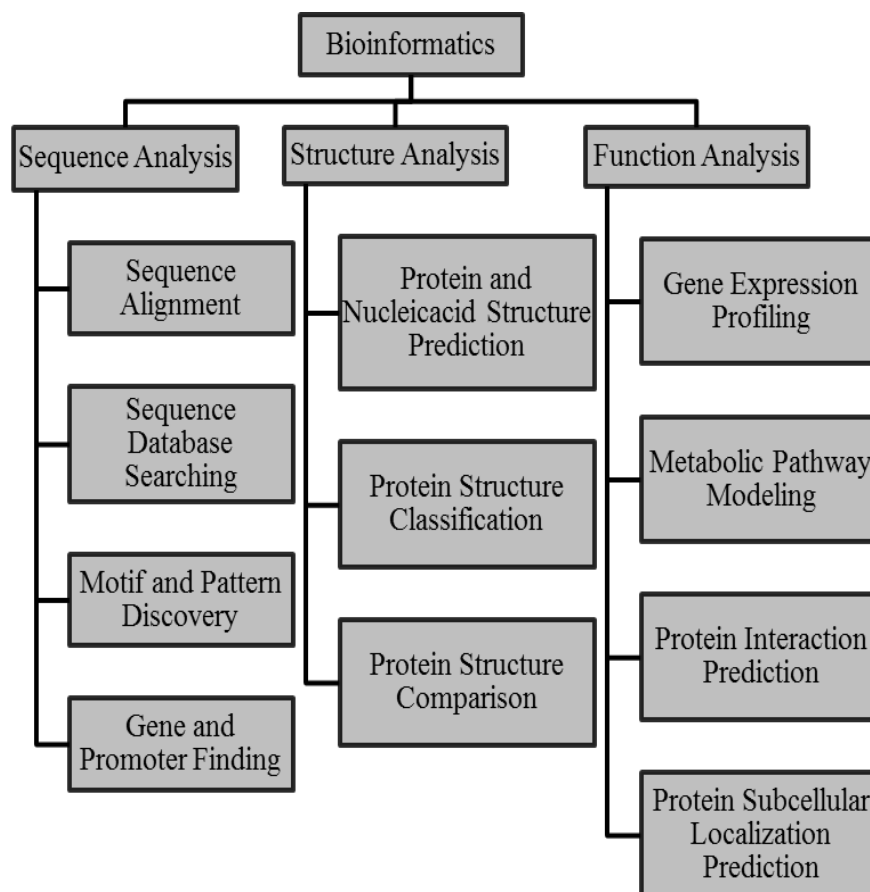


Figure 1.1 Application tasks of bioinformatics.

The areas of sequence analysis include sequence alignment, sequence database searching, motif and pattern discovery, gene and promoter finding, reconstruction of evolutionary relationships, and genome assembly and comparison. Structural analyses include protein and nucleic acid structure analysis, comparison, classification, and prediction. The functional analyses include gene expression profiling, protein–protein interaction prediction, protein subcellular localization prediction, metabolic pathway reconstruction, and simulation (Figure 1.1).

The three aspects of bioinformatics analysis are not isolated but often interact to produce integrated results. For instance, protein structure prediction depends on sequence alignment data; clustering of gene expression profiles requires the use of phylogenetic tree construction methods derived in sequence analysis. Sequence-based promoter prediction is related to functional analysis of co-expressed genes. Gene annotation involves a number of activities, which include distinction between

coding and noncoding sequences, identification of translated protein sequences, and determination of the gene's evolutionary relationship with other known genes; prediction of its cellular functions employs tools from all three groups of the analyses.

Sequence Analysis: Sequence analysis task includes sequence alignment, sequence database searching, motif and pattern discovery, gene and promoter finding tasks. These tasks are handled as follows.

- *Sequence alignment:* Sequence alignment is the procedure of comparing two (pairwise alignment) or more (multiple sequence alignment) sequences by searching for a series of individual characters or character patterns that are in the same order in the sequences (Mount, 2004).

There are two types of sequence alignment, global and local. In global alignment, an attempt is made to align the entire sequence, using as many characters as possible, up to both ends of each sequence. Sequences that are quite similar and approximately the same length are suitable candidates for global alignment. The global alignment is shown in Figure 1.2.

```

L G P S S K Q T G K G S - S R I W D N
|           |   |   |           |   |
L N - I T K S A G K G A I M R L G D A

```

Figure 1.2 Global alignment.

In local alignment, stretches of sequence with the highest density of matches are aligned, thus generating one or more islands of matches or sub alignments in the aligned sequences. Local alignments are more suitable for aligning sequences that are similar along some of their lengths but dissimilar in others, sequences that differ in length or sequences that share a conserved region or domain. The local alignment is shown in Figure 1.3.

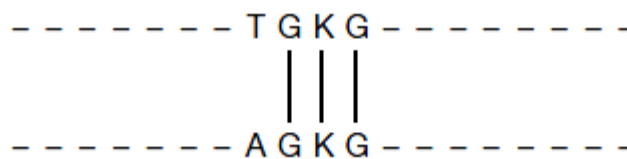


Figure 1.3 Local alignment.

- *Sequence Database Searching:* Sequence database searching can be remarkably useful for finding the function of genes whose sequences have been determined in the laboratory. The sequence of the gene of interest is compared with the best matching sequences are shown and scored. If a query sequence can be readily aligned to database sequence of a known function, structure, or biochemical activity, the query sequence is predicted to have the same function, structure, or biochemical activity. The strength of these predictions depends on the quality of the alignment between the sequences. As a rough rule, if more than one-half of the amino acid sequence of query and database proteins is identical in the sequence alignments, the prediction is very strong. As the degree of similarity decreases, confidence in the prediction also decreases. The programs such as FASTA and BLAST, used for these database searches provide statistical evaluations that serve as a guide for evaluation of the alignment scores.

- *Motif and Pattern Discovery:* A motif is a short conserved sequence pattern associated with distinct functions of a protein or DNA. It is often associated with a distinct structural site performing a particular function. A typical motif, such as a Zn-finger motif, is ten to twenty amino acids long.

Motifs are evolutionarily more conserved than other regions of a protein and tend to evolve as units, which are gained, lost, or shuffled as one module. The identification of motifs in proteins is an important aspect of the classification of protein sequences and functional annotation. Because of evolutionary divergence, functional relationships between proteins often cannot be distinguished through simple BLAST or FASTA database searches. In addition, proteins or enzymes often perform multiple functions that cannot be fully described using a single annotation

through sequence database searching. To resolve these issues, identification of the motifs becomes very useful.

Identification of motifs heavily relies on multiple sequence alignment as well as profile and hidden Markov model (HMM) construction. Motifs are first constructed from multiple alignment of related sequences. Based on the multiple sequence alignment, commonly conserved regions can be identified. The regions considered motifs then serve as diagnostic features for a protein family. The consensus sequence information of motifs can be stored in a database for later searches of the presence of similar sequence patterns from unknown sequences.

- *Gene and Promoter Finding:* Computational gene prediction is a prerequisite for detailed functional annotation of genes and genomes. The process includes detection of the location of open reading frames (ORFs) and delineation of the structures of introns as well as exons if the genes of interest are of eukaryotic origin. The ultimate goal is to describe all the genes computationally with near 100% accuracy. The ability to accurately predict genes can significantly reduce the amount of experimental verification work required.

1.4 Splice Sites

Encoding protein genes consist of coding (exons) and non-coding (introns) in eukaryotes. During the transcription process, introns are removed and mRNA is formed by joining exons, then mRNA generates proteins called the translation process. All of these processes are shown in Figure 1.4.

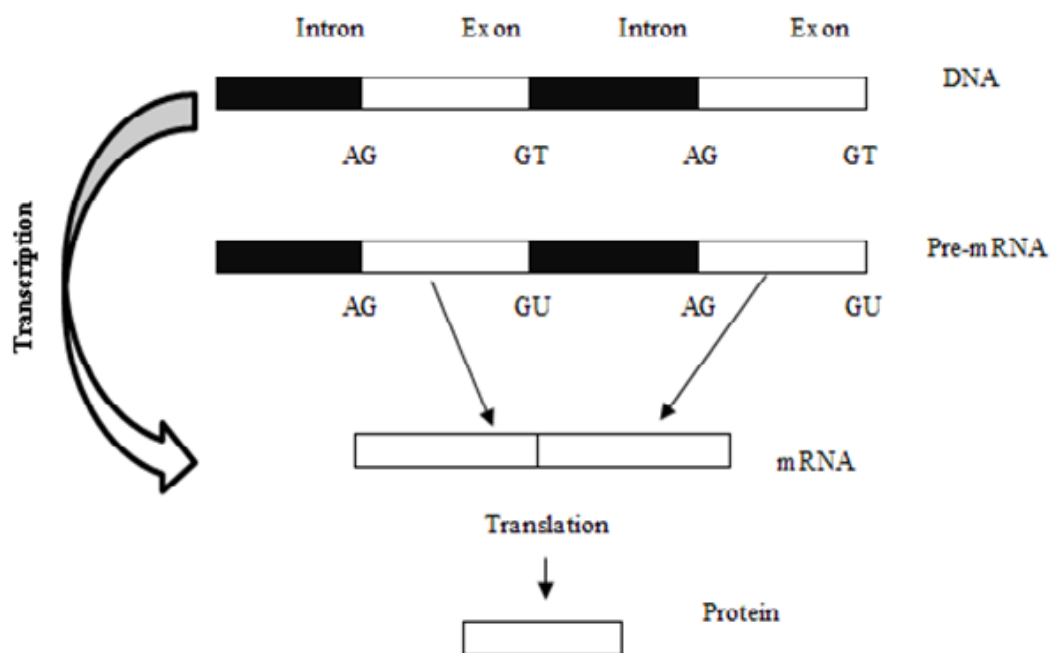


Figure 1.4 Process of generating mRNA and protein.

In DNA sequence, the border between coding region (exon) and noncoding region (intron) is called splice site. The splice site upstream of an intron is called the donor splice site and one that is downstream of an intron is the acceptor splice site (Fig. 1.5). Dinucleotides are frequently observed in the splice sites. In the donor splice sites, "GT" and in the acceptor splice sites "AG" dinucleotide are consensus. The splice sites which consist of certain consensus dinucleotide are known as canonical splice sites (Bursat et al., 2000). Sequences which have "GT" dinucleotide in the splice sites are classified as "EI", which have "AG" dinucleotide in the splice sites are classified as "IE", and which do not have any of these dinucleotide in the splice sites are classified as "None".

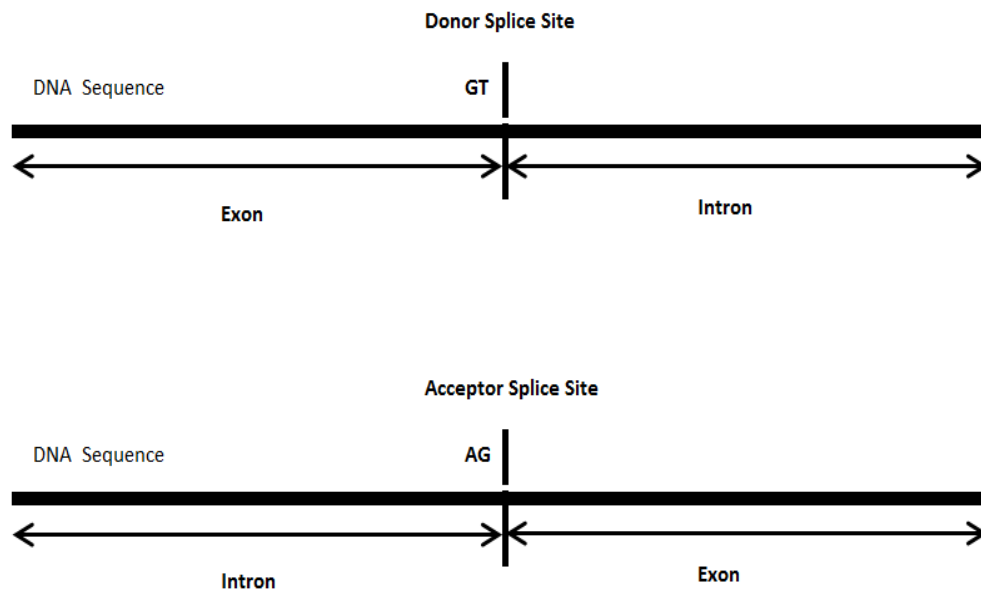


Figure 1.5 Acceptor and donor sites of splice sites.

What is known about splicing:

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

Why splice site detection is important? A complete understanding of splice sites does not only help to correctly predict mRNA and thus proteins from DNA, but can also be of great help in localization of the genes. Actually several other sites, like start and stop codons, branch points, promoters and terminators of transcription and various transcription factor binding sites belonging to the class of local sites can help to detect genes (Haussler, 1998). In computational gene finding, these signals are often contrasted with variable length regions, like exons and introns. While the latter are recognised by methods called content sensors, the former can be recognised by, e.g., weight matrices, decision trees, etc., methods named signal sensors (Haussler, 1998).

CHAPTER TWO

MATERIALS and METHODS

In this thesis, the proposed method is based on the position specific scoring matrix and a genetic algorithm is utilized for optimization of the weight and threshold parameters. Also in the validation phase, cross validation method is used. In this chapter, these methods are explained in a detail and dataset is covered.

2.1 Dataset

Nowadays, high technology eases reaching to electronic version of DNA data. Many valuable researches can be done using these data. In our work, the data is taken from ‘UCI Repository of machine learning databases’ (Asuncion and Newman, 2007). This data set consists of 3190 sequences with 60 nucleotides. It contains 767 ‘EI’, 768 ‘IE’, and 1655 ‘None’ sequences. These sequences have ‘A’, ‘C’, ‘T’, ‘G’ nucleotides and also ‘D’, ‘N’, ‘S’, ‘R’ ambiguity characters. These characters based on the following rules:

- ‘D’ = ‘A’, ‘G’, or ‘T’
- ‘N’ = ‘A’, ‘C’, ‘T’, or ‘G’
- ‘S’ = ‘C’, or ‘G’
- ‘R’ = ‘A’ or ‘G’

2.2 Position-Specific Scoring Matrix (PSSM)

PSSM is a table which covers probability information of amino acids or nucleotides in each position. In this table, positions of the residues are in rows and name of the residues are in columns or vice versa. The values in the table are log odds scores (Xiong, 2006).

Let $X = \{x^k, k = 1, \dots, N\}$, be a learning set of sequences with n ($n = 60$) residues, where $x_i^k \in \{A, C, T, G\}$, $i = 1, \dots, n$. Row frequencies of each residue at each column position are counted first to construct a PSS matrix. The frequencies are normalized via dividing positional frequencies of each residue by overall frequencies. Finally, logarithms (generally to the base of 2) of the values are calculated.

2.2.1 The computational steps of PSSM

The computational steps for construction of the PSSM are given in Algorithm 1.

Algorithm 1 (Construction of PSS Matrix).

Input: $X = \{x^k, k = 1, \dots, N\}$, set of sequences, where $x_i^k \in \{A, C, T, G\}$, $i = 1, \dots, n$;

Output: $PSSM_x$ is positional scoring matrix with $4 \times n$ dimension for class x .

$PSSM_x[j, i]$, is a score of j^{th} residue, $j = \{A, C, T, G\}$ in i^{th} position, $i = 1, \dots, n$.

Step 1. Construct the nucleotide based frequency table by following formula:

$$f_j^i = \frac{c_j^i}{N}, \quad i = 1, \dots, n, \quad j = A, C, T, G$$

where, f_j^i is a frequency of j^{th} residue at i^{th} position; c_j^i is a count of j^{th} residue at i^{th} position; N is a count of sequences;

Step 2. Normalize the values by dividing overall frequencies of each nucleotide base. Overall frequencies and normalization is calculated by following formulas:

$$o_j = \frac{\sum_{i=1}^n c_j^i}{(n \times N)}, \quad j = A, C, T, G$$

$$p_j^i = \frac{f_j^i}{o_j}, \quad i = 1, \dots, n, \quad j = A, C, T, G$$

where, o_j is an overall frequency of j^{th} residue; p_j^i is a normalized frequency of j^{th} residue in i^{th} position;

Step 3. Take the log odds of these values by following formula:

$$PSSM_x[j, i] = \text{Log}_2(p_j^i), \quad i = 1, \dots, n, \quad j = A, C, T, G$$

where, $PSSM_x[j,i]$ is log odd score of j^{th} residue at i^{th} position for given class X ;

End.

Position specific scoring matrix (PSSM) is usually used in multiple sequence alignment (MSA) problems. But in this study, PSSM is used for classification of splice-junction sequences of DNA. After the construction of PSSM, match scores of residues of given sequence are added. The total match score shows how the given sequence is similar to sequence family. Also, this score can be interpreted as the probability of sequence fitting as 2^{score} times more likely than by random chance. As a conclusion, the new sequence with high score can be classified as a member of the sequence family.

2.2.2 Prediction of the Sample's Class

After PSS matrices of each class are constructed by Algorithm 1, another algorithm (Algorithm 2) is used to predict the class of a given sequence.

Algorithm 2.

Input:

a) Score vectors $PSSM_{EI}$, $PSSM_{IE}$ and $PSSM_{None}$, which are calculated for 'EI', 'IE' and 'None' classes respectively by using Algorithm 1.

b) Given $x^* = (x_1^*, x_2^*, \dots, x_n^*)$ sequence which is needed to be classified, where $x_i^* \in \{A, C, T, G\}$, $i = 1, \dots, n$.

Output: Class of the given x^* sequence ($class(x^*)$).

Step 1. Scoring vectors belonging to 'EI', 'IE' and 'None' classes of the x^* sequence are calculated by using $PSSM_{EI}$, $PSSM_{IE}$ and $PSSM_{None}$ matrices, respectively:

$$S_{EI}(x^*) = (PSSM_{EI}[x_1^*, 1], PSSM_{EI}[x_2^*, 2], \dots, PSSM_{EI}[x_n^*, n]),$$

$$S_{IE}(x^*) = (PSSM_{IE}[x_1^*, 1], PSSM_{IE}[x_2^*, 2], \dots, PSSM_{IE}[x_n^*, n]),$$

$$S_{None}(x^*) = (PSSM_{None}[x_1^*, 1], PSSM_{None}[x_2^*, 2], \dots, PSSM_{None}[x_n^*, n]),$$

where $PSSM_{[]}[x_i^*, i]$ is a scoring value of x_i^* residue in i^{th} position for a handled $class[.]$.

Step 2. Scores belonging to ‘EI’, ‘IE’ and ‘None’ classes of given x^* sequence are calculated by using $S_{EI}(x^*)$, $S_{IE}(x^*)$ and $S_{None}(x^*)$ scoring vectors:

$$s_{EI}(x^*) = \sum_{i=1}^n S_{EI}^i(x^*),$$

$$s_{IE}(x^*) = \sum_{i=1}^n S_{IE}^i(x^*),$$

$$s_{None}(x^*) = \sum_{i=1}^n S_{None}^i(x^*),$$

where $S_{[]}^i(x^*)$ is a i^{th} component of $S_{[]}(x^*)$ vector.

Step 3. The class of x^* sequence is identified as follows:

$$class(x^*) = \arg \max_{\{EI, IE, None\}} \{s_{EI}(x^*), s_{IE}(x^*), s_{None}(x^*)\};$$

End.

2.2.3 PSSM Example

The construction of a PSS matrix is shown in the following example. Assume that there is a class of five sequences with ten residues (Table 1).

Table 2.1 Sequence table

Position	1	2	3	4	5	6	7	8	9	10
Sequence1	C	C	A	G	C	T	G	C	A	T
Sequence2	A	G	A	C	C	C	G	C	C	G
Sequence3	G	A	G	G	T	G	A	A	G	G
Sequence4	G	G	G	C	T	G	C	G	T	T
Sequence5	G	C	T	C	A	G	C	C	C	C

This sequence table is converted to frequency matrix with Step 1 of Algorithm 1 (Table 2).

Table 2.2 Frequency matrix

<i>Pos.</i>	1	2	3	4	5	6	7	8	9	10	Overall freq.
<i>A</i>	0.2	0.2	0.4	0	0.2	0	0.2	0.2	0.2	0	0.16
<i>C</i>	0.2	0.4	0	0.6	0.4	0.2	0.2	0.6	0.4	0.2	0.32
<i>T</i>	0	0	0.2	0	0.4	0.2	0	0.2	0.2	0.4	0.16
<i>G</i>	0.6	0.4	0.4	0.4	0	0.6	0.4	0	0.2	0.4	0.34

The values of the above table are normalized with Step 2 of Algorithm 1 (Table 3).

Table 2.3 Normalized values

<i>Pos.</i>	1	2	3	4	5	6	7	8	9	10
<i>A</i>	1.25	1.25	2.5	0	1.25	0	1.25	1.25	1.25	0
<i>C</i>	0.625	1.25	0	1.875	1.25	0.625	0.625	1.875	1.25	0.625
<i>T</i>	0	0	1.25	3.75	2.5	1.25	0	1.25	1.25	2.5
<i>G</i>	1.76	1.18	1.18	1.18	0	1.76	1.18	0	0.59	1.18

Finally, the log odds of these values are calculated (Table 4).

Table 2.4 Log odds

<i>Pos.</i>	1	2	3	4	5	6	7	8	9	10
<i>A</i>	0.3219	0.32	1.32	-	0.32	-	0.322	0.322	0.32	-
<i>C</i>	-0.678	0.32	-	0.907	0.32	-0.68	-0.68	0.907	0.32	-0.68
<i>T</i>	-	-	0.32	1.907	1.32	0.322	-	0.322	0.32	1.322
<i>G</i>	0.8156	0.24	0.24	0.239	-	0.816	0.239	-	-0.8	0.239

Let assume that there is a sequence as $x^* = \{A, C, G, G, T, A, C, C, T, T\}$ and we have to find its fitting score to the class. We can calculate the fitting score of the sequence x^* by using Algorithm 2 as follows:

$$\begin{aligned} s(x^*) &= 0.3219 + 0.32 + 0.24 + 0.239 + 1.32 + 0 + (-0.68) + 0.907 + 0.32 + 1.322 \\ &= 4.3099 \end{aligned}$$

From the above equation, the score of x^* (i.e., $s(x^*)$) is found as 4.3099. Respectively, x^* sequence may be confidently classified as a member of sequence

family and fits the matrix as $2^{4.3099}$ or approximately 20 times more likely than by random chance.

2.3 Genetic Algorithm (GA)

Genetic algorithms in particular became popular through the work of John Holland in the early 1970s. Genetic algorithms (GAs) are the derivative-free stochastic optimization methods based on the terms of natural selection and evolutionary processes. The popularity of GAs, depends on independence from functional derivatives and incorporation of the following characteristics (Jang et al., 1997):

- GAs are parallel-search procedures which can be applied on parallel-processing machines for speeding up their operations.
- GAs are implemented in both continuous and discrete optimization problem.
- GAs are stochastic and less likely to get trapped in local minima, which inevitably are present in any practical optimization application.
- GAs' flexibility simplifies both structure and determination of parameter in complex systems like neural networks and fuzzy systems.

GAs encode each point with binary codes which are called chromosome, and each point related to "fitness" value. Instead of one point, GAs usually keep a set of points as a population or gene pool, which is then evolved repeatedly toward a better overall fitness value. In each generation, the GA constructs a new population using genetic operators such as crossover and mutation; members with higher fitness values are more likely to survive and to participate in crossover operations. After a number of generations, the population contains members with better fitness values; this is analogous to Darwinian models of evolution by random mutation and natural selection. GAs and their variants are sometimes referred to as methods of population based optimization that improves performance by upgrading entire populations rather than individual members.

It is necessary to perform certain operations over these individuals for Genetic Algorithms to find a best optimum solution. This section discusses the basic terminologies and operators used in Genetic Algorithms to achieve a good enough solution for possible terminating conditions (David, 1991).

2.3.1 Terminologies of Genetic Algorithm

Gene: In the biology, specific region which are encoding knowledge of our physical features and physiological activities are called “gene”. In the GA, the genes may describe a possible solution to a problem, without actually being the solution. A gene is a bit string of arbitrary lengths. The bit string is a binary representation of number of intervals from a lower bound. A gene is the GA’s representation of a single factor value for a control factor, where control factor must have an upper bound and lower bound (Fig. 2.1).

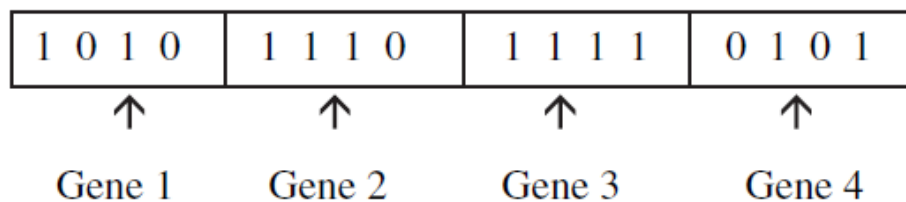


Figure 2.1 Representation of gene.

Allele: Allele, in biology, is the term given to the appropriate range of values for genes. In genetic algorithms, an allele is the value of the gene (or genes).

Chromosome: A chromosome is a sequence of genes (Fig. 2.2).

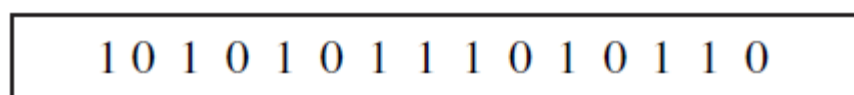


Figure 2.2 Representation of chromosome.

Genotype: The genotype is the structure of the solution produced by the genetic program (Fig. 2.3).

Phenotype: The actual values of a genome (its position in the solution space) are called the phenotype (Fig. 2.3). While the genotype expresses the overall properties of the genetic algorithm by defining the nature of the chromosome, the phenotype represents an individual expression of the genome (or genotype). This is somewhat similar to the relationship between classes and objects in an object-oriented programming language: a class represents the definition of an object, whereas an object represents a concrete instantiation of a class.

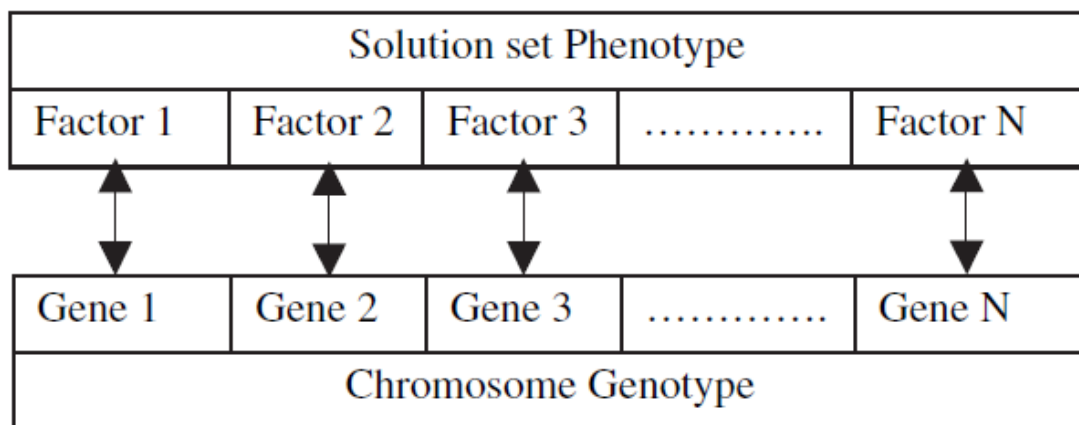


Figure 2.3 Representation of genotype and phenotype.

Population: A population is a collection of individuals. A population consists of a number of individuals being tested, the phenotype parameters defining the individuals and some information about search space (Fig. 2.4).

Population	Chromosome 1	1 1 1 0 0 0 1 0
	Chromosome 2	0 1 1 1 1 0 1 1
	Chromosome 3	1 0 1 0 1 0 1 0
	Chromosome 4	1 1 0 0 1 1 0 0

Figure 2.4 Representation of Population.

2.3.2 Operators of Genetic Algorithm

There are many genetic operators. The major genetic operators are selection, mutation and crossover operators (Pham and Karaboğa, 2000). These operators are covered in detail.

- *Initial Population Selection:* Initial population selection is done randomly with a probability depending on the relative fitness of the individuals so that best ones are often chosen for reproduction than poor ones.

- *Encoding schemes:* Encoding is a process of representing individual genes. The process can be performed using bits, numbers, trees, arrays, lists or any other objects. The encoding depends mainly on solving the problem. For example, one can encode directly real or integer numbers. There are various kinds of encoding schemes.

Binary Encoding: The most common way of encoding is a binary string, which would be represented as in Fig. 2.5. Binary coded strings with 1s and 0s are mostly used. The length of the string depends on the accuracy.

Chromosome 1	1 1 0 1 0 0 0 1 1 0 1 0
Chromosome 2	0 1 1 1 1 1 1 1 1 1 0 0

Figure 2.5 Binary encoding.

Each chromosome encodes a binary (bit) string. Each bit in the string can represent some characteristics of the solution. Every bit string, therefore, is a solution but not necessarily the best solution. Another possibility is that the whole string can represent a number. The way bit strings can code differs from problem to problem.

Binary encoding gives many possible chromosomes with a smaller number of alleles. On the other hand this encoding is not natural for many problems and sometimes corrections must be made after genetic operation is completed.

Octal Encoding: This encoding uses string made up of octal numbers (0–7) as in Fig. 2.6.

Chromosome 1	03467216
Chromosome 2	15723314

Figure 2.6 Octal encoding.

Hexadecimal Encoding: This encoding uses string made up of hexadecimal numbers (0–9, A–F) as in Fig. 2.7.

Chromosome 1	9CE7
Chromosome 2	3DBA

Figure 2.7 Hexadecimal encoding.

Permutation Encoding (Real Number Coding): Every chromosome is a string of numbers, which represents the number in sequence. Sometimes corrections have to be done after genetic operation is completed. In permutation encoding, every chromosome is a string of integer/real values, which represents number in a sequence (Fig. 2.8).

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Figure 2.8 Permutation encoding.

Permutation encoding is only useful for ordering problems. Even for these problems some particular crossover and mutation corrections must be made to leave the chromosome consistent.

Value Encoding: Every chromosome is a string of values and the values can be anything connected to the problem. This encoding produces best results for some special problems. Direct value encoding can be used in problems, where some complicated values, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult. In value encoding, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or characters to some complicated objects (Fig. 2.9).

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

Figure 2.9 Value encoding.

Value encoding is appropriate for some special problems. On the other hand, for this encoding, it is often necessary to develop some new specific crossover and mutation for the problem.

- *Fitness evaluation:* The first step after the creating the generation is to calculate the fitness value of each individual. The fitness of an individual in a genetic algorithm is the value of an objective function for its phenotype. For calculating fitness, the chromosome has to be first decoded and the objective function has to be evaluated. The fitness not only indicates how good the solution is, but also corresponds to how close the chromosome is to the optimal one.

- *Selection:* After the evaluation, we have to generate a new population from the current generation. Selection operator determines which parents participate in

producing offspring for the next generation, and it is analogous to survival of the fittest in natural selection (Baker, 1985). The selection can be made according to various criteria such as Roulette Wheel Selection, Rank Selection, Random Selection, Tournament Selection, and Elitism Selection.

Roulette-wheel selection: Roulette selection is one of the traditional GA selection techniques. The commonly used reproduction operator is the proportionate reproductive operator where a string is selected from the mating pool with a probability proportional to the fitness. The principle of roulette selection is a linear search through a roulette wheel with the slots in the wheel weighted in proportion to the individual's fitness values. A target value which is a random proportion of the sum of the fitnesses in the population is set.

The Roulette process can also be explained as follows: The expected value of an individual is fitness divided by the actual fitness of the population. Each individual is assigned to slice of the roulette wheel, the size of the slice being proportional to the individual's fitness. The wheel is spun N times, where N is the number of individuals in the population. On each spin, the individual under the wheel's marker is selected to be in the pool of parents for the next generation (Fig. 2.10).

Steps of the Roulette Wheel Selection is as follows:

Step1. Sum the fitness of each member of the population.

Step2. Determine the relative fitness of each member of the population.

Step3. Generate a random number (SPIN) between zero and some predefined maximum value (MAX).

Step4. Select next individual.

Step5. From SPIN, subtract the individual's relative proportion of MAX (i.e., relative fitness times MAX).

Step6. Repeat steps 4 and 5 until SPIN is less than or equal to zero.

Step7. Repeat steps 3 to 6 until mating pool is full.

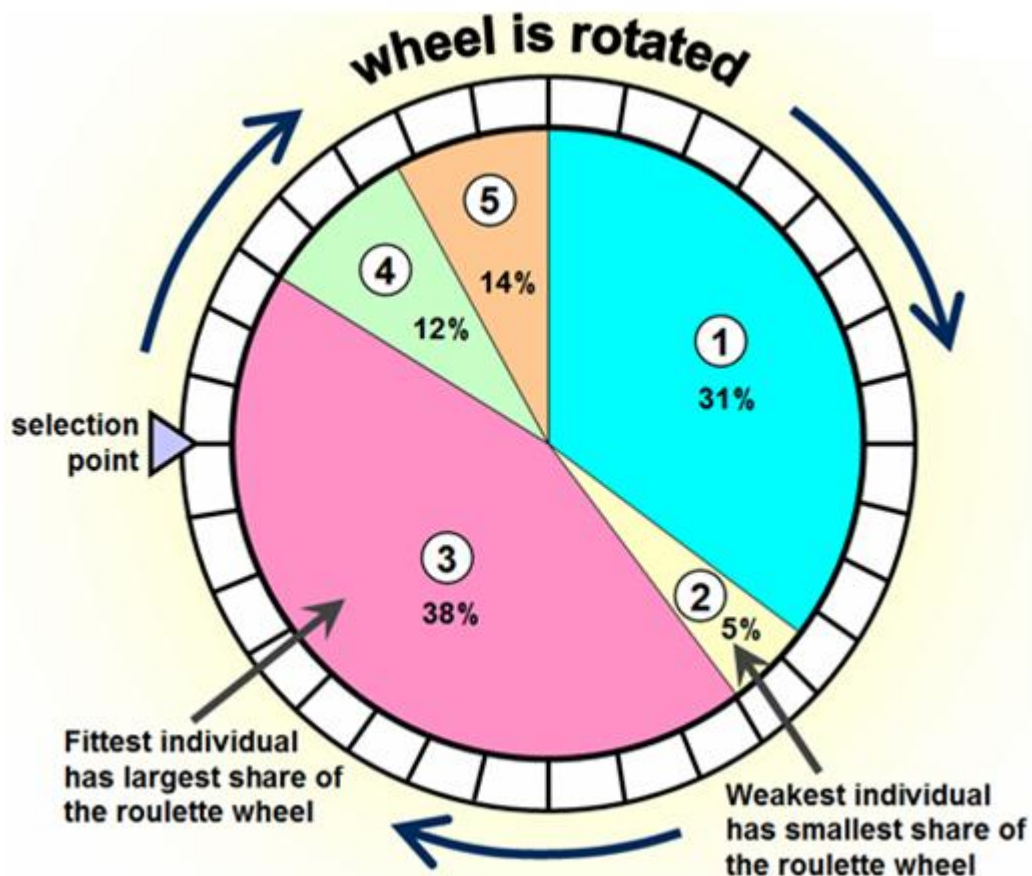


Figure 2.10 Roulette wheel selection.

Random Selection: This technique randomly selects a parent from the population. In terms of disruption of genetic codes, random selection is a little more disruptive than roulette wheel selection.

Rank Selection: The Roulette wheel will have a problem when the fitness values differ very much. If the best chromosome fitness is 90%, its circumference occupies 90% of Roulette wheel, and then other chromosomes have too few chances to be selected. Rank Selection ranks the population and every chromosome receives fitness from the ranking. The worst has a fitness of 1 and the best has a fitness of N . It results in slow convergence but prevents too quick convergence. It also keeps up selection pressure when the fitness variance is low. It preserves diversity and hence leads to a successful search. In effect, potential parents are selected and a tournament is held to decide which of the individuals will be the parent.

Tournament Selection: An ideal selection strategy should be such that it is able to adjust its selective pressure and population diversity so as to fine-tune GA search performance. Unlike the Roulette wheel selection, the tournament selection strategy provides selective pressure by holding a tournament competition among N_u individuals.

The best individual from the tournament is the one with the highest fitness, which is the winner of N_u . Tournament competitions and the winner are then inserted into the mating pool. The tournament competition is repeated until the mating pool for generating new offspring is filled. The mating pool comprising of the tournament winner has higher average population fitness. The fitness difference provides the selection pressure, which drives GA to improve the fitness of the succeeding genes. This method is more efficient and leads to an optimal solution.

Elitism Selection: The first best chromosome or the few best chromosomes are copied to the new population. The rest is done in a classical way. Such individuals can be lost if they are not selected to reproduce or if crossover or mutation destroys them. This significantly improves the GA's performance.

- *Reproduction:* Reproduction in genetic programming is asexual, thus imitating the process of budding in biology. Through reproduction, an identical copy of the individual selected is carried over into the next generation: survival of the fittest. Fitness proportionate reproduction is the asexual reproduction of chromosomes selected stochastically from the population. According to Koza, "the operation of fitness proportionate reproduction for the genetic programming paradigm is the basic engine of Darwinian reproduction and survival of the fittest" [Koza, 1990; Grant, 1995]. In other words, the selection of individual chromosome is based on a probability that is relative to that chromosome's relative fitness within its population.

- *Crossover*: To use the potential of the current gene pool, we use crossover operators to generate new chromosomes that will hopefully retain good features from the previous generation (Sivanandam and Deepa, 2007).

Single Point Crossover: The traditional genetic algorithm uses single point crossover, where the two mating chromosomes are cut once at corresponding points and the sections after the cuts exchanged. Here, a cross-site or crossover point is selected randomly along the length of the mated strings and bits next to the cross-sites are exchanged. If appropriate site is chosen, better children can be obtained by combining good parents else it severely hampers string quality. Single point crossover is illustrated as in Fig. 2.11 and it can be observed that the bits next to the crossover point are exchanged to produce children. The crossover point can be chosen randomly.

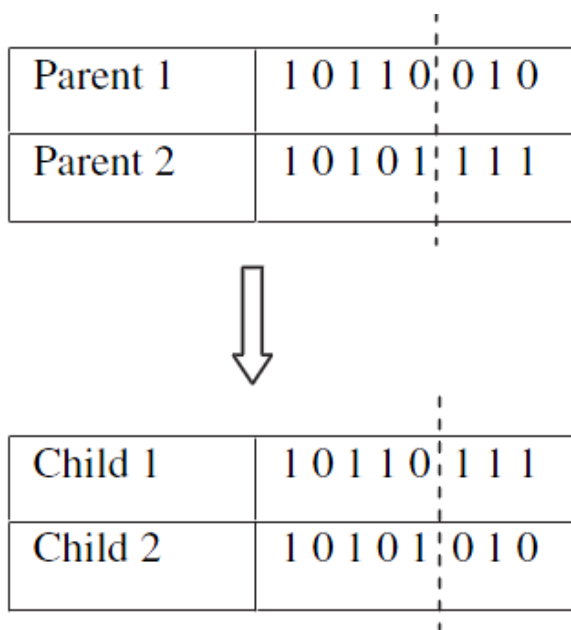


Figure 2.11 Single Point Crossover.

Two Point Crossover: In two-point crossover, two crossover points are chosen and the contents between these points are exchanged between two mated parents. Apart from single point crossover, many different crossover algorithms have been devised, often involving more than one cut point. It should be noted that adding further crossover points reduces the performance of the GA. The problem with

adding additional crossover points is that building blocks are more likely to be disrupted. However, an advantage of having more crossover points is that the problem space may be searched more thoroughly.

In Figure 2.12 the dotted points indicate the crossover points. Thus the contents between these points are exchanged between the parents to produce new children for mating in the next generation.

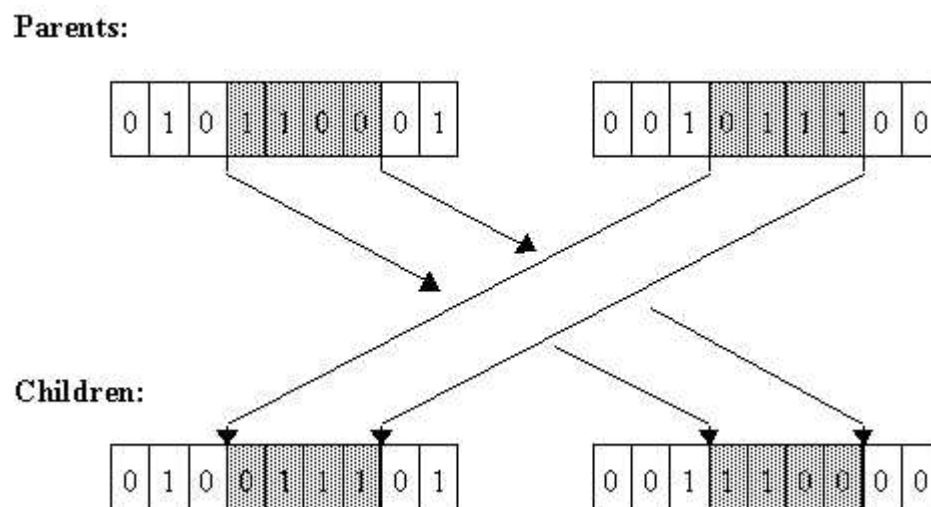


Figure 2.12 Two Point Crossover.

Uniform Crossover: In the uniform crossover each gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to a random generated binary crossover mask of the same length as the chromosomes. Where there is a 1 in the crossover mask, the gene is copied from the first parent, and where there is a 0 in the mask the gene is copied from the second parent. A new crossover mask is randomly generated for each pair of parents. Offsprings, therefore, contain a mixture of genes from each parent. The number of effective crossing point is not fixed, but will average $L/2$ (where L is the chromosome length).

In Figure 2.13, new children are produced using uniform crossover approach. It can be noticed, that while producing child 1, when there is a 1 in the mask, the gene

is copied from the parent 1 else from the parent 2. On producing child 2, when there is a 1 in the mask, the gene is copied from parent 2, when there is a 0 in the mask; the gene is copied from the parent 1.

Parent 1	1 0 1 1 0 0 1 1
Parent 2	0 0 0 1 1 0 1 0
Mask	1 1 0 1 0 1 1 0
Child 1	1 0 0 1 1 0 1 0
Child 2	0 0 1 1 0 0 1 1

Figure 2.13 Uniform Crossover

Crossover Probability: The basic parameter in crossover technique is the crossover probability (P_c). Crossover probability is a parameter to describe how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's chromosome. If crossover probability is 100%, then all offspring are made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!). Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old population survives to next generation.

- *Mutation:* Crossover makes use of current gene potentials, but if the population does not include all the encoded information needed to solve a particular problem, no amount of gene mixing can produce a satisfactory solution. For this reason, a mutation operator capable of spontaneously generating new chromosomes is contained (Fig. 2.14). The most common way of applying mutation is to flip a bit with a probability equal to a very low given mutation rate. A mutation operator can prevent any single bit from converging to a value throughout the entire population

and, more important, it can prevent the population from converging and stagnating at any local optima.

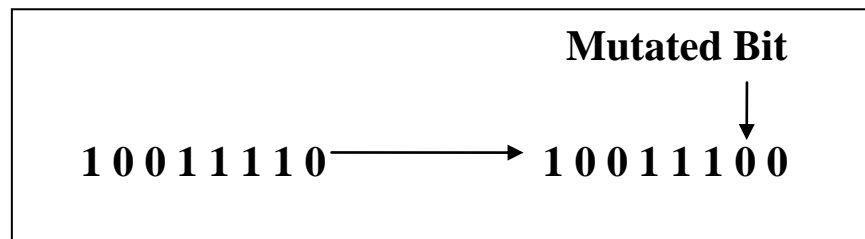


Figure 2.14 Mutation Operator.

The types of mutation are as follows:

Flip Bit Mutation: A mutation operator that simply inverts the value of the chosen gene (0 goes to 1 and 1 goes to 0). This mutation operator can only be used for binary genes.

Boundary Mutation: A mutation operator that replaces the value of the chosen gene with either the upper or lower bound for that gene (chosen randomly). This mutation operator can only be used for integer and float genes.

Non-Uniform Mutation: A mutation operator that increases the probability that the amount of the mutation will be close to 0 as the generation number increases. This mutation operator keeps the population from stagnating in the early stages of the evolution then allows the genetic algorithm to fine tune the solution in the later stages of evolution. This mutation operator can only be used for integer and float genes.

Uniform Mutation: A mutation operator that replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. This mutation operator can only be used for integer and float genes.

Gaussian Mutation: A mutation operator that adds a unit Gaussian distributed random value to the chosen gene. The new gene value is clipped if it falls outside of the user-specified lower or upper bounds for that gene. This mutation operator can only be used for integer and float genes.

Mutation Probability: An important parameter in the mutation technique is the mutation probability (P_m). The mutation probability decides how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed. Mutation generally prevents the GA from falling into local extremes. Mutation should not occur very often, because then GA will in fact change to random search.

How the Genetic Algorithm Works: The algorithm begins by creating a random initial population. Then the algorithm creates a sequence of new populations. At each step, the algorithm uses the individuals in the current generation to create the next population. To create the new population, each member of the current population is scored by computing its fitness value. Then, the raw fitness scores are scaled to convert them into a more usable range of values. It selects members, called parents, based on their fitness. Some of the individuals in the current population that have lower fitness are chosen as elite. These elite individuals are passed to the next population. Children are produced from the parents. Children are produced either by making random changes to a single parent mutation or by combining the vector entries of a pair of parents crossover. The current population is replaced with the children to form the next generation. The algorithm stops when one of the stopping criteria is met. The outline above can be summarized by the following steps (Goldberg, 1989):

Step 1: Determine the initial population with randomly generated individuals and compute fitness value.

Step 2: Select two individuals from the population with probabilities proportional to their fitness values.

Step 3: Apply crossover with the probabilities equals to crossover rate.

Step 4: Apply mutation with the probabilities equals to mutation rate.

Step 5: Repeat the steps 2 - 4 until sufficient member is generated for the next generation.

Step 6: Repeat the steps 2-5 until a stopping criteria is satisfied.

Step 7: Output the best solution of the last generation as an approximate optimal solution.

End.

Figure 2.15 is the figure illustrating how to produce the next generation from the current one.

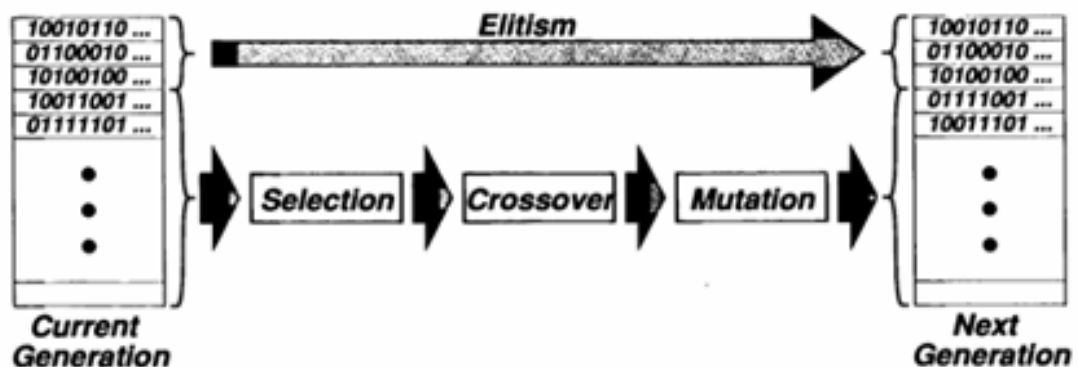


Figure 2.15 Producing the Next Generation in GAs.

2.4 Cross Validation Method

Cross-Validation is a statistical method of evaluating and comparing learning algorithms by dividing data into two segments: one used to learn or train a model and the other used to validate the model. In typical cross-validation, the training and validation sets must cross-over in successive rounds such that each data point has a chance of being validated against. The basic form of cross-validation is k-fold cross-validation. Other forms of cross-validation are special cases of k-fold cross-

validation or involve repeated rounds of k -fold cross-validation such as leave-one-out cross-validation and $k \times 2$ cross-validation.

In K -fold cross-validation, the original sample is randomly partitioned into K subsamples. Of the K subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $K - 1$ subsamples are used as training data. The cross-validation process is then repeated K times (the folds), with each of the K subsamples used exactly once as the validation data. The K results from the folds then can be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used (McLachlan et al., 2004). Ten fold cross-validation is the standard way of measuring the error rate of a learning scheme on a particular dataset; for reliable results, 10 times 10-fold cross-validation. To show the working mechanism of k -fold cross validation, three-fold cross validation is illustrated in Figure 2.16.

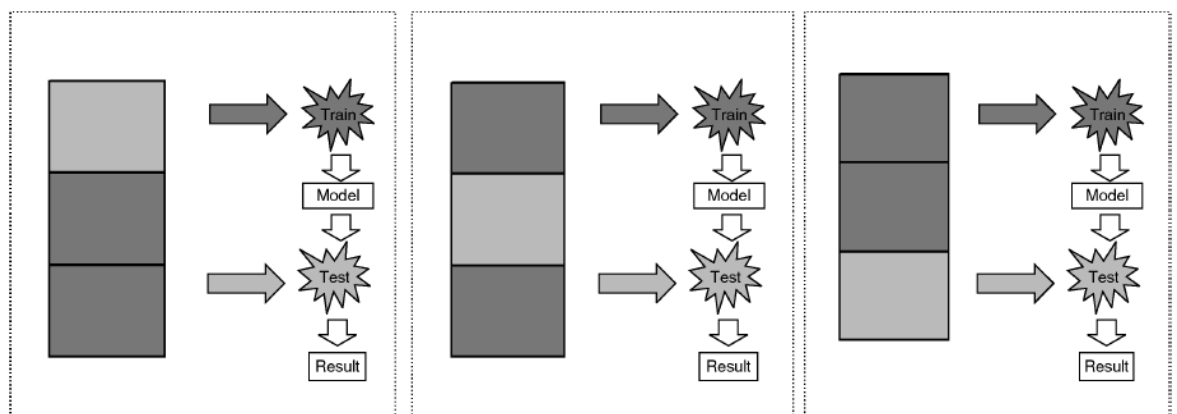


Figure 2.16 Three-fold Cross Validation.

In stratified K -fold cross-validation, the folds are selected so that the mean response value is approximately equal in all the folds. In the case of a dichotomous classification, this means that each fold contains roughly the same proportions of the two types of class labels.

There are two possible goals in cross-validation:

- 1) To estimate performance of the learned model from available data using one algorithm. In other words, to gauge the generalizability of an algorithm.
- 2) To compare the performance of two or more different algorithms and find out the best algorithm for the available data, or alternatively to compare the performance of two or more variants of a parameterized model.

These two goals are highly related, since the second goal is automatically achieved if one knows the accurate estimates of performance.

CHAPTER THREE

WEIGHTED POSITION-SPECIFIC SCORING METHOD

We developed a weighted position-specific scoring method by using position specific scoring matrix, genetic algorithm and 10-fold cross validation. In this method, the process consists of learning, identification and validation phases. These phases will be covered in detail.

3.1 Learning Phase

In the learning phase, we calculate the position frequencies and then construct the position-specific scoring matrices for these classes by using logarithms of the position specific nucleotide probabilities for each learning class ('EI', 'IE' and 'None'). To compare the efficiency of our method, a position specific scoring matrix is constructed by each variant as normalization step and without normalization step and the results are compared.

A threshold parameter (t) is used to consider deterministic odds-impact of observed frequencies with respect to the distance from the natural frequency. The optimal position weights (\bar{w}) and optimal threshold (\bar{t}) are computed by using genetic algorithm for the learning set. Flow chart of the learning phase is shown in Figure 2.17.

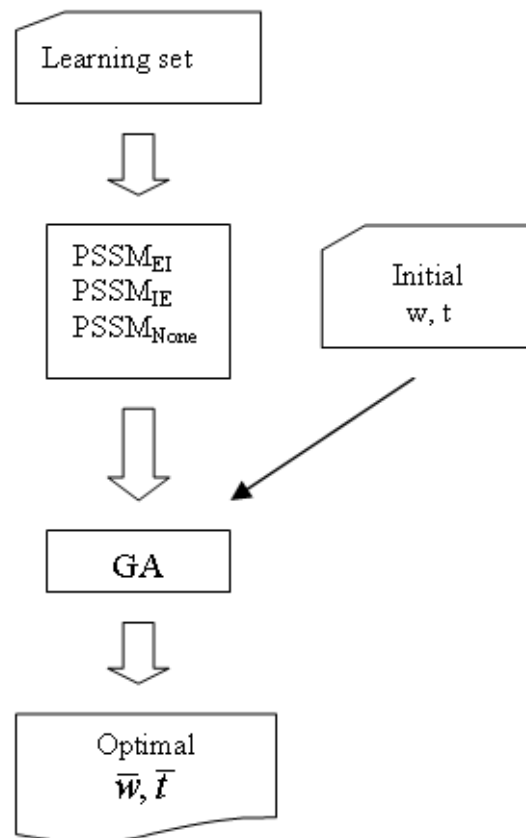


Figure 2.17 Learning phase flow chart.

3.2 Identification Phase

In the identification phase, the optimal weights and threshold calculated from learning phase are applied to each sequence in the test set. Then each sequence classified into 'EI', 'IE', 'None' classes. The flow chart of identification phase is illustrated in Figure 2.18.

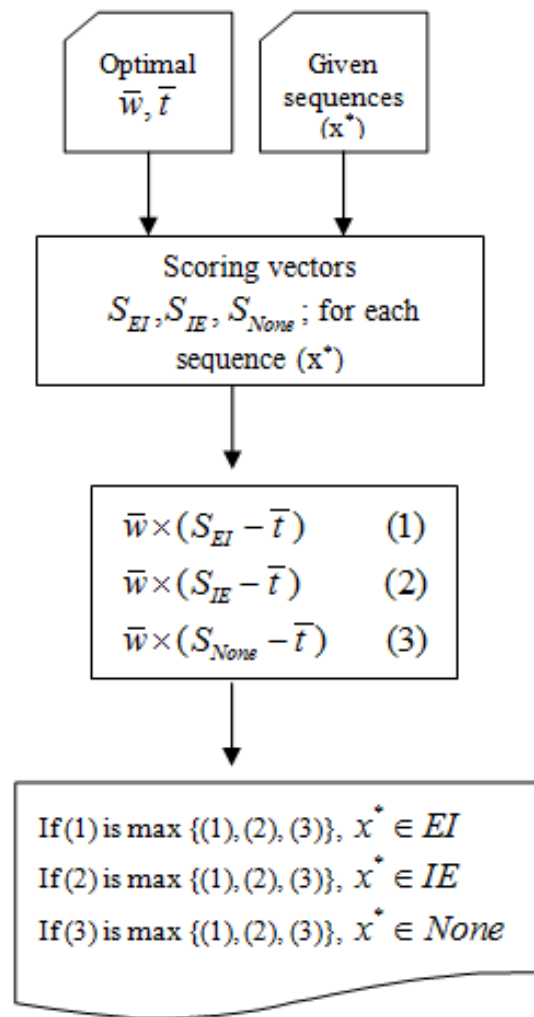


Figure 2.18 Identification phase flow chart.

Algorithm of the identification phase is as follows:

Algorithm 3 (WPSSM).

Input:

a) $PSSM_{EI}$, $PSSM_{IE}$ and $PSSM_{None}$ which are calculated for ‘EI’, ‘IE’ and ‘None’ classes by using Algorithm 1.

b) $x^* = (x_1^*, x_2^*, \dots, x_n^*)$ sequence which is needed to be classified, where $x_i^* \in \{A, C, T, G\}$, $i = 1, \dots, n$.

Output: Class of x^* sequence ($class(x^*)$).

Step 1. Scoring vectors belonging to ‘EI’, ‘IE’ and ‘None’ classes of x^* sequence are calculated by using $PSSM_{EI}$, $PSSM_{IE}$ and $PSSM_{None}$ matrices, respectively:

$$S_{EI}(x^*) = (PSSM_{EI}[x_1^*, 1], PSSM_{EI}[x_2^*, 2], \dots, PSSM_{EI}[x_n^*, n]),$$

$$S_{IE}(x^*) = (PSSM_{IE}[x_1^*, 1], PSSM_{IE}[x_2^*, 2], \dots, PSSM_{IE}[x_n^*, n]),$$

$$S_{None}(x^*) = (PSSM_{None}[x_1^*, 1], PSSM_{None}[x_2^*, 2], \dots, PSSM_{None}[x_n^*, n]),$$

where $PSSM_{[.]}[x_i^*, i]$ is a scoring value of x_i^* residue in i^{th} position for the $class[.]$.

Step 2. Scores of the given sequence x^* , $S_{EI}(x^*)$, $S_{IE}(x^*)$ and $S_{None}(x^*)$ belonging to ‘EI’, ‘IE’ and ‘None’ classes are calculated by using the scoring vectors:

$$s_{EI}(x^*) = \sum_{i=1}^n w_i \cdot (S_{EI}^i(x^*) - t),$$

$$s_{IE}(x^*) = \sum_{i=1}^n w_i \cdot (S_{IE}^i(x^*) - t),$$

$$s_{None}(x^*) = \sum_{i=1}^n w_i \cdot (S_{None}^i(x^*) - t),$$

where w_i is a weight of i^{th} position; $S_{[.]}^i(x^*)$ is a i^{th} component of $S_{[.]}(x^*)$ vector; t is threshold value.

Step 3. The class of x^* sequence is identified as follows:

$$class(x^*) = \arg \max_{\{EI, IE, None\}} \{s_{EI}(x^*), s_{IE}(x^*), s_{None}(x^*)\};$$

End.

3.3 Validation phase

Now, let $X = \{x^k, k=1, \dots, N\}$, be a learning set of sequences with n ($n=60$) residue where $x_i^k \in \{A, C, T, G\}$, $i=1, \dots, n$. The classes of test sequences are

predicted as explained in the identification phase. The aim of this phase is to measure the accuracy.

In the literature, True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) values, and different indexes which are calculated from these values, are generally used to measure the validation of a method. Here, FP is a count of sequences which are incorrectly assigned to predicted class; FN is a count of sequences which are not assigned to predicted class as incorrect; TP is a count of sequences which are accurately assigned to predicted class; TN is a count of sequences which are not assigned to predicted class as accurate.

In our study, three classes are used as ‘EI’, ‘IE’, ‘None’ (Table 3.1). TP, TN, FP and FN values are calculated from the formulas in Table 3.2, where a_{ij} indicates number of predicted sequences.

Table 3.1. Confusion matrix for each class

Actual Class	Predicted Class		
	EI	IE	None
EI	a_{11}	a_{12}	a_{13}
IE	a_{21}	a_{22}	a_{23}
None	a_{31}	a_{32}	a_{33}

Table 3.2. TN, TP, FN and FP rates formulas for each class

EI	IE	None
$TP = a_{11}$	$TP = a_{22}$	$TP = a_{33}$
$TN = a_{22} + a_{23} + a_{32} + a_{33}$	$TN = a_{11} + a_{13} + a_{31} + a_{33}$	$TN = a_{11} + a_{12} + a_{21} + a_{22}$
$FP = a_{21} + a_{31}$	$FP = a_{12} + a_{32}$	$FP = a_{31} + a_{32}$
$FN = a_{12} + a_{13}$	$FN = a_{21} + a_{23}$	$FN = a_{13} + a_{23}$

Error rates (ERs) are computed according to true/false classification results with the following formula:

$$ER = \frac{FP + FN}{TP + TN + FP + FN} \quad (3.1)$$

Also overall error rate (OER) is computed with the below formula:

$$OER = 1 - \frac{\sum_{i=\{EI,IE,N\}} (TP_i + TN_i)}{\sum_{i=\{EI,IE,N\}} (TP_i + TN_i + FP_i + FN_i)} \quad (3.2)$$

CHAPTER FOUR

RESULTS AND DISCUSSION

In this work, our proposed method has been applied to data set from the ‘UCI Repository of machine learning databases’ which contains 3190 samples and each sample is a sequence of 60 nucleotides (Asuncion and Newman, 2007). This data set includes 767 ‘EI’, 768 ‘IE’ and 1655 ‘None’ sequences.

In the preliminary works of our study, 2140 sequences which include 517 EI, 518 IE and 1105 N, are taken as learning set. The rest of this data set which contains 1050 sequences with 250 EI, 250 IE and 550 N, are taken as test set; and this test set is classified by using weights and threshold computed from learning set. The optimal position weights and threshold in the learning phase of our study found by genetic algorithm are as in Table 4.1. The classification results are shown in Table 4.2.

Table 4.1 The optimal position weights and threshold computed by GA

	1	2	3	4	5	6	7	8	9	10
W=	1.00	0.16	0.90	0.82	0.44	1.00	0.60	0.27	0.76	0.64
	11	12	13	14	15	16	17	18	19	20
	0.21	0.27	0.80	0.40	0.13	0.14	0.55	0.80	0.82	0.61
	21	22	23	24	25	26	27	28	29	30
	0.96	0.83	0.83	1.00	1.00	0.26	1.00	0.95	0.85	0.82
	31	32	33	34	35	36	37	38	39	40
	0.72	1.00	0.77	1.00	1.00	0.62	0.35	0.53	0.57	0.23
	41	42	43	44	45	46	47	48	49	50
	0.36	0.86	0.56	0.72	0.41	0.99	0.23	0.59	0.31	0.29
	51	52	53	54	55	56	57	58	59	60
	0.53	0.97	0.65	0.22	0.21	0.26	0.98	0.47	1.00	1.00
t=	0.6945									

Table 4.2 Classification Results for the Test Set

<i>Results</i>		<i>Predicted</i>		
		<i>EI</i>	<i>IE</i>	<i>N</i>
<i>Actual</i>	<i>EI</i>	244	4	2
	<i>IE</i>	5	238	7
	<i>N</i>	9	16	525

Error rates are computed from the table above for each class with equation (3.1). The error rates are: ‘EI’ %1.9, ‘IE’ %3.0, and ‘NONE’ %3.32.

We repeated the experiments without using threshold parameter to investigate efficiency of this parameter. The classification results are found as in Table 4.3.

Table 4.3 Classification results for test set without using the threshold parameter

<i>Results</i>		<i>Predicted</i>		
		<i>EI</i>	<i>IE</i>	<i>N</i>
<i>Actual</i>	<i>EI</i>	242	5	3
	<i>IE</i>	5	238	7
	<i>N</i>	11	18	521

The error rates without using threshold parameter are: ‘EI’ %2.28, ‘IE’ %3.33, and ‘NONE’ %3.71. It is seen that using threshold parameter decreased error rates and improved the effectiveness of the method. Error rates of the WPSSM are shown in the Table 4.4.

Table 4.4. Error rates of WPSSM.

<i>Methods</i>	‘EI’ %	‘IE’ %	‘N’ %	Overall Error Rates (%)
WPSSM	1,9	3	3,32	2,74
WPSSM without Threshold	2,28	3,33	3,71	3.107

In the developed work of our study, 10-fold cross-validation method is applied to this data set. In this method, the dataset is broken into 10 sets of size $n/10$ of the each fold containing 319 sequences. One fold is selected as test set and the rest folds are selected as learning set. This process is repeated 10 times and then the mean

accuracy is taken. The test sets are classified by using the optimal weights and threshold computed from learning sets.

The optimal position weights and threshold in the learning phase of our study found by genetic algorithm are as follows (Table 4.5).

Table 4.5 The optimal position weights and threshold computed by GA

	1	2	3	4	5	6	7	8	9	10
w=	0,6126	0,6384	0,5973	0,5590	0,8086	0,6466	0,6623	0,6351	0,5674	0,6707
	11	12	13	14	15	16	17	18	19	20
	0,6648	0,8888	0,4702	0,6048	0,7096	0,4815	0,5349	0,7917	0,8125	0,6664
	21	22	23	24	25	26	27	28	29	30
	1,0106	0,6225	0,6790	0,5602	0,8951	0,6916	0,4668	0,9281	0,9118	0,8180
	31	32	33	34	35	36	37	38	39	40
	0,7638	0,9731	0,6681	0,8493	1,0684	0,9346	0,4778	0,5386	0,6489	0,4447
	41	42	43	44	45	46	47	48	49	50
	0,5757	0,8416	0,5857	0,7866	0,6046	0,6176	0,5746	0,7389	0,5389	0,7966
	51	52	53	54	55	56	57	58	59	60
	0,5335	0,3935	0,5626	0,6969	0,6597	0,6517	0,5992	0,5939	0,6153	0,7209
	61									
t=	0,3176									

In the learning set, weights and threshold parameters are calculated by genetic algorithm with tuned options as *Generations*, *Initial Population*, *Population Size*, *Stall Gen Limit*, *Stall Time Limit*, *Elite Count*. In the computations, maximum iteration number is taken as 30 (*Generations=30*), initial population is taken as 'w' vector with 1×61 dimension ones vector which includes 1×60 ones for weights and a one for threshold in the last position (*Initial Population*), population size is taken as 30 (*Population Size=30*). The algorithm is stopped if the weighted average change in the fitness function value over 5000 stall generations (*Stall Gen Limit=5000*) is less than function tolerance. Also, the algorithm is stopped if there is no improvement in the best fitness value for an interval of time in seconds specified by 2000 (*Stall Time Limit=2000*). Five individuals are taken as elite count (*Elite Count=5*) for each generation.

The classification results are shown in Table 4.6. Each cell in Table 4.6 consists of average classification count and average classification rate. Moreover, the error rates and overall error rates for each class are computed for each fold by the equation (3.1-3.2). The mean of error rates for classes ‘EI’, ‘IE’ and ‘None’ are found as %2.73, %2.70, and %3.17, respectively.

Table 4.6 Classification results for the test set

Results		Predicted		
		EI	IE	None
Actual	EI	73.4 %95.697	2.1 %2.738	1.2 %1.564
	IE	1.5 %1.953	73.5 %95.703	1.8 %2.344
	None	3.9 %2.356	3.2 %1.933	158.4 %95.71

We repeated the experiments with normalization step and without threshold parameter to investigate superiority of this parameter to normalization step. The classification results are found as in Table 4.7. This table consists of average classification counts and rates.

Table 4.7 Classification results for test set with normalization step and without threshold parameter

Results		Predicted		
		EI	IE	None
Actual	EI	73.3 %95.567	2.4 %3.129	1 %1.304
	IE	1.2 %1.562	73.8 %96.094	1.8 %2.344
	None	4 %2.417	3.9 %2.356	157.6 %95.347

The error rates of WPSSM with normalization step and without threshold parameter are %2.70, %2.92, and %3.35 for the classes ‘EI’, ‘IE’ and ‘None’, respectively. It is seen that using threshold parameter decreases error rates and

improves the effectiveness of the method than normalization step. Error rates of the WPSSM are shown in Table 4.8.

Table 4.8 Error rates of WPSSM

Methods	'EI' %	'IE' %	'None' %	Overall %
WPSSM	2.73	2.70	3.17	2.86
WPSSM with normalization and without threshold	2.70	2.92	3.35	2.99

Also performance measurements such as specificity (Sp), sensitivity (Sn), Mathew's correlation coefficient (MCC) and accuracy rate are computed via following formulas:

$$Sn = \frac{TP}{TP + FN} \quad (4.1)$$

$$Sp = \frac{TN}{TN + FP} \quad (4.2)$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.3)$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \quad (4.4)$$

Performance parameters are found as in Table 4.9.

Table 4.9 Performance parameters

Methods	Sn	Sp	MCC	Accuracy
WPSSM	0.9634	0.9753	0.9356	0.9714
WPSSM with normalization and without threshold	0.9606	0.9748	0.9328	0.9701

Sensitivity measures the proportion of actual positives which are correctly identified and specificity measures the proportion of predicted positives which are correctly identified. The success of a method depends on high value of both specificity and sensitivity.

WPSSM shows significant improvements in the sensitivity and specificity of splice sites identification. Its accuracy rate is higher than many methods in the literature (Grau et al., 1999; Li et al., 2007). The Matthew's correlation coefficient (MCC) is generally regarded as being one of the best such measures. The MCC is in essence a correlation coefficient between the observed and predicted binary classifications; it takes a value between -1 and $+1$. A coefficient of $+1$ represents a perfect prediction, 0 an average random prediction and -1 an inverse prediction. Also, WPSSM has a MCC approximately 1. It shows that proposed method is a good classification method.

CHAPTER FOUR

CONCLUSION

The handled problem in this thesis is to identify, given a sequence of DNA letters, splice sites. In this work, we proposed a novel weighted position-specific scoring method (WPSSM). We show that our method is able to improve classification accuracy and performance parameters in accordance with other methods.

The WPSS method uses position weights and threshold parameter instead of a normalization step. The optimal position weights and threshold parameter are found via genetic algorithm. The threshold is used instead of normalization step in the construction of the position specific scoring matrix. The normalization step of the PSSM uses a fixed value, but the threshold in our proposed method is a tuned value calculated by optimization procedure. As it is seen from the experimental results this approach gives better results than classical PSSM approach. The mentioned results poses that WPSSM can be efficient in solution of splice site recognition problem.

In the preliminary works, learning and test sets are selected randomly. This method is applied to learning set and the weights calculated according to learning set are tested with the test set. However, in the developed work, learning and test sets are selected according to ten-fold cross validation method. This selection method is more logical than random selection.

The proposed method (WPSSM) shows significant improvements in the sensitivity and specificity of splice sites identification. Accuracy rate of this method is higher than many other methods in the literature. Also the method has significant Mathew's correlation coefficient rate.

When the classification error rates of our WPSSM are compared with many methods proposed in the literature, the results of WPSSM have showed lower error rates than many of the known methods in the literature. The methods which have the higher results than our proposed method WPSSM are Hierarchical Multi Classifier Method (HM), Linear Support Vector Machine (Rank SVM) and Support Vector Machine (SVM). The comparison results are given in Table 5.1.

Table 5.1 Error rates of different methods

<i>Methods</i>	<i>'EI' %</i>	<i>'IE' %</i>	<i>'N' %</i>	<i>Overall Error Rates %</i>
HM	0.76	1.59	1.6	1.317
Rank SVM	0.86	1.86	1.71	1.477
SVM	1.65	1.99	1.9	1.847
WPSSM	1.9	3	3.32	2.740
SUBSPACE	1.68	6.25	1.61	3.180
NN BRAIN	2.6	4.3	n.d.	3.450
BRAIN	5	4	4	4.333
KBANN	7.6	8.5	4.6	6.900
MLP	5.7	10.7	5.3	7.233
BACKPROPAGATION	5.74	10.75	5.29	7.260
PEBLS	8.18	7.55	6.86	7.530
ID3	10.6	14	8.8	11.133
COBWEB	15.04	9.46	11.8	12.100
PERCEPTRON	16.32	17.41	3.99	12.573
NEAREST NEIGHBOR	11.6	9.1	31.1	17.267

REFERENCES

- Asuncion, A. & Newman, DJ. (2007) *UCI Machine Learning Repository*. Irvine, CA: University of California, School of Information and Computer Science, Retrieved March 2010 from (<http://www.ics.uci.edu/~mllearn/MLRepository.html>).
- Baker, J.E. (1985) Adaptive Selection Methods for Genetic Algorithms. *Proc. 1st Int. Conf. Genetic Algorithms and Their Applications*, Lawrence Erlbaum Associates, Hillsdale, NJ, 100-101.
- Baten, A., Chang, BCH., Halgamuge SK., & Li J. (2006) Splice site identification using probabilistic parameters and SVM classification. *BMC Bioinformatics*, 7(5),15.
- Brunak, S., Engelbrecht, J., & Knudsen, S. (1991) Prediction of mRNA donor and acceptor sites from the DNA sequence. *Journal of Molecular Biology*, 220, 49-65.
- Burset, M., Seledtsov, A., & Solovyeva, V.V. (2000) Analysis of canonical and non-canonical splice sites in mammalian genomes. *Nucleic Acids Research*, 28(21), 4364-4375.
- Cai, D., Delcher, A., Kao, B., & Kasif, S. (2000) Modeling splice sites with Bayes networks. *Bioinformatics*, 16(2), 152-158.
- Castelo, R., & Guigo, R. (2004) Splice site identification by idlBNs. *Bioinformatics*, 20(1), 69-76.
- Chen, TM., Lu, CC., & Li, WH. (2005) Prediction of splice sites with dependency graphs and their expanded Bayesian networks. *Bioinformatics*, 21(4), 471-482.
- Chuang, JS., & Roth, D. (2001) Splice site prediction using a sparse network of winnows. *In Technical Report University of Illinois*, Urbana-Champaign, Champaign, IL, USA.
- David, L. (1991). *Handbook of Genetic Algorithms* (1st ed.). Van Nonstrand Reinhold, New York, NY.

- Degroeve, S., Saeys, Y., Baets, B.D., Rouze, P., & Peer, Y.V.D. (2005). SpliceMachine: predicting splice sites from high-dimensional local context representations. *Bioinformatics*, 21(8), 1332-1338.
- Goldberg, D.E. (1989) *Genetic Algorithms in search, optimization and machine learning* (1st ed.). Addison – Wesley, Reading, MA.
- Grant, K. (1995). An introduction to genetic algorithms. *C/C++ Users Journal*, 13(3), 45 - 58.
- Grau, A., Mar, M., Molinero, H., & Daniel, L. (1999). Feature selection in codebook based methods provides high accuracy. *IJCNN'99*, 3, 1856-1860.
- Haussler, D.. (1998) Computational gene finding. *Trends Biochem. Sci. Suppl. Guide Bioinformatics*, 12–15.
- Holland, J.H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI.
- Jang, JSR., Sun, CT., & Mizutani, E. (1997). *Neuro-Fuzzy and soft computing* (1st ed.). Prentice-Hall, Inc. Simon & Schuster/A Viacom Company Upper Saddle River, NJ 07458.
- Koza, J. R. (1990). Genetic Programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical Report No. STAN-CS-90-314, *Computer Sciences Department*, Stanford University.
- Li, K., Chang, D., Rouchka, E., & Chen, YY. (2007). Biological sequence mining using Plausible Neural Network and its application to exon/intron boundaries prediction. *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, 165-169.
- Lumini, A., & Nanni, L. (2006). Identifying splice-junction sequences by hierarchical multiclassifier. *Pattern Recognition Letters*, 27, 1390–1396.

- Marashi, SA., Eslahchi, C., Pezeshk, H., & Sadeghi, M. (2006a). Impact of RNA structure on the prediction of donor and acceptor splice sites. *BMC Bioinformatics*, 7, 297.
- Marashi, SA., Goordarzi, H., Sadeghi, M., Eslahchi, C., Pezeshk, H. (2006b) Importance of RNA secondary structure information for yeast donor and acceptor splice site predictions by neural networks. *Computational Biology and Chemistry*, 30, 50-57.
- McLachlan GJ, Do K.A., & Ambrose C. (2004). *Analyzing microarray gene expression data* (1st ed.). Wiley.
- Mount D. W. (2004). *Bioinformatics – sequence and genome analysis* (2nd ed.). Cold Spring Harbor Laboratory Press, New York, 283-334.
- Nasibov E.N., & Tunaboylu S. (2010a). A Novel Weighted Position-Specific Scoring Method for Splice Site Recognition in DNA Sequences. *The 1.st International Symposium on Computing in Science & Engineering*, 225-230.
- Nasibov E.N., & Tunaboylu S., (2010b). Classification of splice-junction sequences via weighted position specific scoring approach. *Computational Biology and Chemistry*, 34, Issues 5-6, 293-299.
- Needleman, S.B. & Wunsch, C. D. (1970). A General method applicable to the search for similarities in the amino acid sequence of two proteins. *J.Mol. Biol.*, 48, 443-453.
- Noordewier, MO., Towell, GG., & Shavlik, JW. (1991). Training Knowledge-Based Neural Networks to Recognize Genes in DNA Sequences. *Advances in Neural Information Processing Systems*, 3, 530-536.
- Pertea, M., Lin, X., & Salzberg, SL. (2001). GeneSplicer: a new computational method for splice site detection. *Nucleic Acids Research*, 29(5), 1185-1190.

- Pham, D.T. & Karabođa, D. (2000). *Intelligent Optimization Techniques* (1st ed.). Springer – Verlag, London, UK.
- Rajapakse, J.C., & Ho, L.S. (2005). Markov encoding for detecting signals in genomic sequences. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(2), 131-142.
- Ratsch, G., Sonnenburg, S., & Schafer, C. (2006). Learning Interpretable SVMs for Biological Sequence Classification. *BMC Bioinformatics*, 7(1), S9.
- Reese, M.G. (2001). Application of a time-delay neural network to promoter annotation in the *Drosophila melanogaster*. *Computer chem.*, 26(1), 51-56.
- Reese, M.G., Eeckman, F., Kupl, D., & Haussler, D. (1997). Improved splice site detection in Genie. *Journal of Computational Biology*, 4(3), 311-324.
- Salzberg, S.L. (1997). A method for identifying splice sites and translation start site in eucaryotic mRNA. *Computer Applications in the Biosciences*, 13(4), 384-390.
- Sivanandam S.N. & Deepa S. N..(2007). *Introduction to Genetic Algorithms*.Springer.
- Sonnenburg, S. (2002). New methods for detecting splice junction sites in DNA sequence. *In Master's Thesis Humboldt University, Germany*.
- Sonnenburg, S., Schweikert, G., Philips, P., Behr, J., & Ratsch, G. (2007). Accurate splice site prediction using support vector machines. *BMC Bioinformatics*, 8(10), S7.
- Staden, R. (1984). Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Res.*, 12, 505–519.
- Sun, Y.F., Fan, X.D., & Li, Y.D. (2003). Identifying splicing sites in eukaryotic RNA: Support vector machine approach. *Computers in biology and medicine*, 33, 17-29.

- Tsai, KN., Lin, SH., Shih, SR., Lai, JS., & Chen, CM. (2009). Genomic splice site prediction algorithm based on nucleotide sequence pattern for RNA viruses. *Computational Biology and Chemistry*, 33, 171-175.
- Xiong, J. (2006). *Essential Bioinformatics* (1st ed.). Cambridge University Press, 75-84.
- Zhang, L., & Luo, L. (2003). Splice site prediction with quadratic discriminant analysis using diversity measure. *Nucleic Acids Research*, 31(21), 6214-6220.
- Zhang, M., & Marr, TG. (1993). A weight array method for splicing signal analysis. *Bioinformatics*, 9, 499-509.
- Zhang, XHF., Heller, K.A., Hefter, I., Leslie, C.S., & Chasin, L.A. (2003). Sequence information for the splicing of human pre-mRNA identified by support vector machine classification. *Genome Research*, 13, 2637-2650.

APPENDIX

Construction of the learning dataset and test dataset via ten-fold cross validation

method:

```

indices = crossvalind('Kfold',3190,10);
i=1:3190;
for j=1:10
    TS{j} = i(indices == j);
    LS{j} = i(indices ~=j);

    TSEI{j}=TS{j}(find(TS{j}<=767));
    TSIE{j}=TS{j}(find((TS{j}>=768)&(TS{j}<=1535)));
    TSN{j}=TS{j}(find(TS{j}>=1536));

    LSEI{j}=LS{j}(find(LS{j}<=767));
    LSIE{j}=LS{j}(find((LS{j}>=768)&(LS{j}<=1535)));
    LSN{j}=LS{j}(find(LS{j}>=1536));
end

save('DSets','TS','LS','TSEI','TSIE','TSN','LSEI','LSIE',
'LSN');

```

Program:

```

clear all
clc
global dizi EI IE N sinif pred_sinif index numEI numIE
numN w TP
global LSEInum LSIEnum LSNnum
global EItestindex EIlearnindex IETestindex IElearnindex
Ntestindex Nlearnindex

sinif=importdata('sinif.txt',' ');
konum=importdata('konum.txt',' ');
dizi=importdata('dizi.txt',' ');

load DSets;

error_rates=[];

for j=1:10
    EItestindex=TSEI{j};
    EIlearnindex=LSEI{j};
    IETestindex=TSIE{j};
    IElearnindex=LSIE{j};
    Ntestindex=TSN{j};
    Nlearnindex=LSN{j};

```

```

LSEInum=size(LSEI{j},2);
LSIEnum=size(LSIE{j},2);
LSNnum=size(LSN{j},2);
disp(' LEARNING PROCESS ')
spliceLearn;

% sonucLearn=['SonucLearn' 47+j];
% save (sonucLearn,'w', 'TP', 'numEI','numIE', 'numN');

disp(' TEST PROCESS ')
spliceTest;
sonucTest=['SonucTest' 47+j];
save (sonucTest,'TP', 'numEI','numIE', 'numN');
Pred_sinif=['pred_sinif' 47+j];
save(Pred_sinif,'pred_sinif');

% Test seti sonucları

nnTSEI(j)=0;
nnTSIE(j)=0;
nnTSN(j)=0;
for k=TSEI{j}
    nnTSEI(j)=nnTSEI(j)+strcmp(pred_sinif{k},'EI');
    nnTSIE(j)=nnTSIE(j)+strcmp(pred_sinif{k},'IE');
    nnTSN(j)=nnTSN(j)+strcmp(pred_sinif{k},'NN');
end

TestMatrix_EI=[nnTSEI(j) nnTSIE(j) nnTSN(j)];
nnTSEI(j)=0;
nnTSIE(j)=0;
nnTSN(j)=0;
for k=TSIE{j}
    nnTSEI(j)=nnTSEI(j)+strcmp(pred_sinif{k},'EI');
    nnTSIE(j)=nnTSIE(j)+strcmp(pred_sinif{k},'IE');
    nnTSN(j)=nnTSN(j)+strcmp(pred_sinif{k},'NN');
end

TestMatrix_IE=[nnTSEI(j) nnTSIE(j) nnTSN(j)];
nnTSEI(j)=0;
nnTSIE(j)=0;
nnTSN(j)=0;
for k=TSN{j}
    nnTSEI(j)=nnTSEI(j)+strcmp(pred_sinif{k},'EI');
    nnTSIE(j)=nnTSIE(j)+strcmp(pred_sinif{k},'IE');
    nnTSN(j)=nnTSN(j)+strcmp(pred_sinif{k},'NN');
end

TestMatrix_N=[nnTSEI(j) nnTSIE(j) nnTSN(j)];

TestRes{j}=[TestMatrix_EI;TestMatrix_IE;TestMatrix_N];

```

```

TestRes{j};
Test_Res=['TestRes' 47+j];
save(Test_Res, 'TestRes')

% Error Rates for each class

ER_EI(j)=(sum(TestRes{j}(2:3,1))+sum(TestRes{j}(1,2:3)))/
sum(sum(TestRes{j}));
ER_IE(j)=(sum(TestRes{j}(2,1))+sum(TestRes{j}(2,3))+
sum(TestRes{j}(1,2))+sum(TestRes{j}(3,2)))/sum(sum(
TestRes{j}));

ER_N(j)=(sum(TestRes{j}(3,1:2))+sum(TestRes{j}(1:2,3)))/
sum(sum(TestRes{j}));

error_rates=[error_rates [ER_EI(j) ;ER_IE(j) ;ER_N(j)]];
disp({'error_rates' j});
disp(error_rates);
Error_Rates=['error_rates' 47+j];
save(Error_Rates, 'error_rates');
end
sum(error_rates')/10

```

Performance Parameters:

```

% Calculation of TN, TP, FN, FP for each fold.for j=1:10

for j=1:10
FP(j)=sum(TestRes{j}(2:3,1))+ sum(TestRes{j}(2,1))+
sum(TestRes{j}(2,3))+sum(TestRes{j}(3,1:2));
FN(j)=sum(TestRes{j}(1,2:3))+sum(TestRes{j}(1,2))+
sum(TestRes{j}(3,2))+sum(TestRes{j}(1:2,3));
TP(j)=sum(diag(TestRes{j}));
TN(j)=sum(sum(TestRes{j}(2:3,2:3)))+
sum(sum(TestRes{j}(1,1)))+sum(sum(TestRes{j}(3,1)))
+sum(sum(TestRes{j}(3,3)))+sum(sum(TestRes{j}(1,3)))
+sum(sum(TestRes{j}(1:2,1:2)));
end

data=[TP' TN' FP' FN']

TP=mean(TP);
TN=mean(TN);
FP=mean(FP);
FN=mean(FN);
Sn=TP/((TP)+(FN))
Sp=TN/((TN)+(FP))

```

```
CC= ( (TP*TN) -  
      (FP*FN) ) / (sqrt ( (TP+FP) * (TP+FN) * (TN+FP) * (TN+FN) ) )  
Accuracy= (TP+TN) / (TP+TN+FP+FN)  
F= (2*Sn*Sp) / (Sn+Sp)  
save ('Measurements', 'Sn', 'Sp', 'CC', 'Accuracy', 'F')
```