Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

NJIT
New Jersey's Science &
Technology University

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

KNOWLEDGE DISCOVERY IN BIOLOGICAL DATABASES: A NEURAL NETWORK APPROACH

by
Qicheng Ma

Knowledge discovery in databases, also known as data mining, is aimed to find significant information from a set of data. The knowledge to be mined from the dataset may refer to patterns, association rules, classification and clustering rules, and so forth. In this dissertation, we present a neural network approach to finding knowledge in biological databases. Specifically, we propose new methods to process biological sequences in two case studies: the classification of protein sequences and the prediction of E. Coli promoters in DNA sequences. Our proposed methods, based on neural network architectures, combine techniques ranging from Bayesian inference, coding theory, feature selection, dimensionality reduction, to dynamic programming and machine learning algorithms. Empirical studies show that the proposed methods outperform previously published methods and have excellent performance on the latest dataset. We have implemented the proposed algorithms into an infrastructure, called Genome Mining, developed for biosequence classification and recognition.

KNOWLEDGE DISCOVERY IN BIOLOGICAL DATABASES: A NEURAL NETWORK APPROACH

by
Qicheng Ma

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer and Information Science

Department of Computer and Information Science

August 2000

Copyright © 2000 by Qicheng Ma

ALL RIGHTS RESERVED

APPROVAL PAGE

Knowledge Discovery in Biological Databases: A Neural Network Approach

Qicheng Ma

Dr. Jason T. L. Wang, Dissertation Advisor Professor of Computer and Information Science Department, NJIT	Date
Dr. James A. McHugh, Committee Member Professor of Computer and Information Science Department, NJIT	Date
Dr. Frank Y. Shih, Committee Member Professor of Computer and Information Science Department, NJIT	Date
Dr. D. C. Douglas Hung, Committee Member Associate Professor of Computer and Information Science Department, NJIT	Date
Dr. Michael Halper, Committee Member Associate Professor of Computer Science Department, Kean University	Date

BIOGRAPHICAL SKETCH

Author:	Qicheng Ma
Degree:	Doctor of Philosophy in Computer and Information Science
Date:	August 2000

Education:

- Doctor of Philosophy in Computer and Information Science
  New Jersey Institute of Technology
  Newark, New Jersey USA, August 2000.
- Master of Engineering in Computer Science
  Beijing Research Institute of Computer Application and
  Simulation Technology
  Beijing, China, May 1995.
- Bachelor of Engineering in Computer Science and Engineering
  Beijing University of Aerospace and Aeronautics
  Beijing, China, July 1992.

Publications:

Jason T. L. Wang, Qicheng Ma, and Katherine Herbert. "Software Engineering and Knowledge Engineering Issues in Bioinformatics", *Handbook of Software Engineering and Knowledge Engineering*, edited by Shi-Kuo Chang, World Scientific Publishing Company, to appear.

Qicheng Ma, Jason T. L. Wang, and James R. Gattiker. "Mining Biomolecular Data Using Background Knowledge and Artificial Neural Networks", *Handbook of Massive Data Sets*, edited by James Abello, Panos M. Pardalos, and Mauricio G. C. Resende, Kluwer Academic Publishers, to appear.

Jason T. L. Wang, Qicheng Ma, Dennis Shasha, and Cathy H. Wu. "An Empirical Study of Bioinformatics Tools for Protein Sequence Classification", *IBM Systems Journal*, Special Issue on Deep Computing in the Life Sciences, to appear.

Jason T. L. Wang, Qicheng Ma, Dennis Shasha, and Cathy H. Wu. "Application of Neural Networks to Biological Data Mining: A Case Study in Protein Sequence Classification", *Proceedings of the Sixth ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, to appear, August 20-23, 2000 Boston, MA, USA.

Qicheng Ma, Jason T. L. Wang and Cathy H. Wu. "Application of Bayesian Neural Networks to Biological Data Mining: A Case Study in DNA Sequence Classification", *Proceedings of the Twelfth International Conference on Software Engineering and Knowledge Engineering*, pp. 23-30, July 6-8, 2000 IL, USA.

Jason T. L. Wang, Xiong Wang, Dennis Shasha, Bruce A. Shapiro, Kaizhong Zhang, Qicheng Ma, and Zasha Weinberg. "An Approximate Search Engine for Structural Databases", *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pp. 584, May 14-19, Dallas, TX, USA.

Qicheng Ma and Jason T. L. Wang. "Evaluating the Significance of Motifs by Minimum Description Length Principle", *Proceedings of the Fifth Joint Conference on Information Sciences*, pp. 798-801, February 27 – March 3, 2000, Atlantic City, NJ, USA.

Qicheng Ma and Jason T. L. Wang. "Biological Data Mining Using Bayesian Neural Networks: A Case Study", *International Journal on Artificial Intelligence Tools*, Special Issue on Biocomputing, pp. 433–452, Volume 8, Number 4, December 1999, World Scientific Publishing Company.

Qicheng Ma and Jason T. L. Wang. "Recognizing Promoters in DNA using Bayesian Neural Networks", *Proceeding of the IASTED International Conference of Artificial Intelligence and Soft Computing*, pp. 301–305, August, 9-12, 1999, Honolulu, HI USA.

Qicheng Ma and Jason T. L. Wang. "Application of Bayesian Neural Networks to Protein Sequence Classification", *Proceedings of the 1999 International Conference on Artificial Intelligence*, (IC-AI'99), pp. 530–535, June 28 - July 1, 1999, Monte Carlo Resort, Las Vegas, NV, USA.

Qicheng Ma, Jason T. L. Wang and Cathy H. Wu. "Detection of Alu Sequences in DNA: A Neural Network Approach", *Proceedings of the Fourth Joint Conference on Information Sciences*, volume 1, pp. 392–395, October 24-28, 1998, Research Triangle Park, NC USA.

This work is dedicated to my family

ACKNOWLEDGMENT

I would like to take this opportunity to thank my wife, Xinhuan Zheng, for her continuing support. I also thank my advisor, Dr. Jason T. L. Wang, for his guidance on my Ph. D. dissertation writing. I want to thank Dr. James A. McHugh, Dr. Frank Y. Shih, Dr. D. C. Douglas Hung and Dr. Michael Halper for being my committee members. I would like to thank my colleagues, Xiong Wang, George J. S. Chang, and Zhiyuan Wang for their technical help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

As the result of the Human Genome Project [26] and related efforts, DNA, RNA and protein data are accumulated at a speed growing at an exponential rate. Mining these biological data to extract significant information becomes extremely important in accelerating genome processing. The significant information may refer to genes, protein sequence patterns and 3D protein structural motifs [67]. Classification is one of the major data mining processes. This process is to classify a set of data into two or more categories. When there are only two categories, it is called *binary classification*. Here we focus on binary classification of biosequences. In binary classification, we are given some training data including both positive and negative examples. The positive data belong to a target class, whereas the negative data belong to the non-target class. The goal is to assign unlabeled test data to either the target class or the non-target class.

Currently, techniques used for biological sequence classification roughly fall into two categories:

(1) *Similarity search* – This approach is to classify unlabeled test sequences by searching for either the global similarity or the local similarity in the sequences. Global similarity search involves either pairwise sequence comparison [2, 51] or multiple sequence alignment [3]. Local similarity search is to find patterns in the sequences; see [12] for an excellent survey.

(2) *Machine learning* – This approach was surveyed in [33]. Various machine learning techniques have been applied to biological sequence classification. For example, hidden Markov models have been used in gene identification [40] as well as protein family modeling [39]. Neural networks have been applied to the

analysis of biomolecular sequences; see [70] for a survey. In [57], a decision tree was employed to find genes in DNA.

In this dissertation we present new methods for biosequence classification. Specially, we present two case studies: the classification of protein sequences and the prediction of promoters in DNA sequences. Our proposed methods combine techniques ranging from neural networks, Bayesian inference, coding theory, feature selection, dimensionality reduction, to dynamic programming and machine learning algorithms. Empirical studies show that our proposed methods outperform previously published methods and have excellent performance on the latest dataset.

The rest of the dissertation is organized as follows. Chapter 2 discusses some background knowledge. Chapter 3 discusses our proposed method on protein sequence classification. Chapter 4 demonstrates our new techniques on promoter sequence recognition. Chapter 5 describes a Web based genome mining tool. Chapter 6 discusses future work and concludes the dissertation.

CHAPTER 2

BACKGROUND

In this chapter, we will discuss some background knowledge on molecular biology.

2.1 DNA

In every nucleus of a person's cell, human genome consists of tightly coiled threads of deoxyribonucleic acid (DNA), and the associated protein molecules, organized into structures called chromosomes. A DNA molecule consists of two strands that wrap around each other. Two strands of a DNA form a highly regular double-stranded helix. Each strand of a DNA consists of repeating nucleotide units composed of a phosphate group, a sugar (deoxyribose), and a base (`A, C, G, or T`). Two strands of a DNA are linked by hydrogen bonds between `G` and `C` and between `A` and `T`. From computer science point of view, a DNA strand is viewed as a string over alphabet $\mathcal{D}=$ {`A, C, G, or T`}. The human genome contains roughly 3 billion base pairs (bp). The Human Genome Project [26] is to sequence all of the 3 billion bp and interpret sequenced data.

Certain subsequences of a DNA strand, called genes, serve as blueprints for proteins. The transcription process synthesizes the RNA molecule using genes as a template.

2.2 Protein

An RNA is a one strand molecule. An RNA can leave the nucleus and enter the cytoplasm, where the translation process synthesizes a protein molecule using the RNA as a template. Each string of three consecutive nucleotides in an RNA encodes a single amino acid. There are 20 amino acids.

Let $\mathcal{A}$= {`A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y`}. The primary structure (sequence) of a protein is a string of amino acids over the alphabet $\mathcal{A}$. A protein folds into a unique 3D structure, which determines its function. Figure 2.1 shows a protein structure.

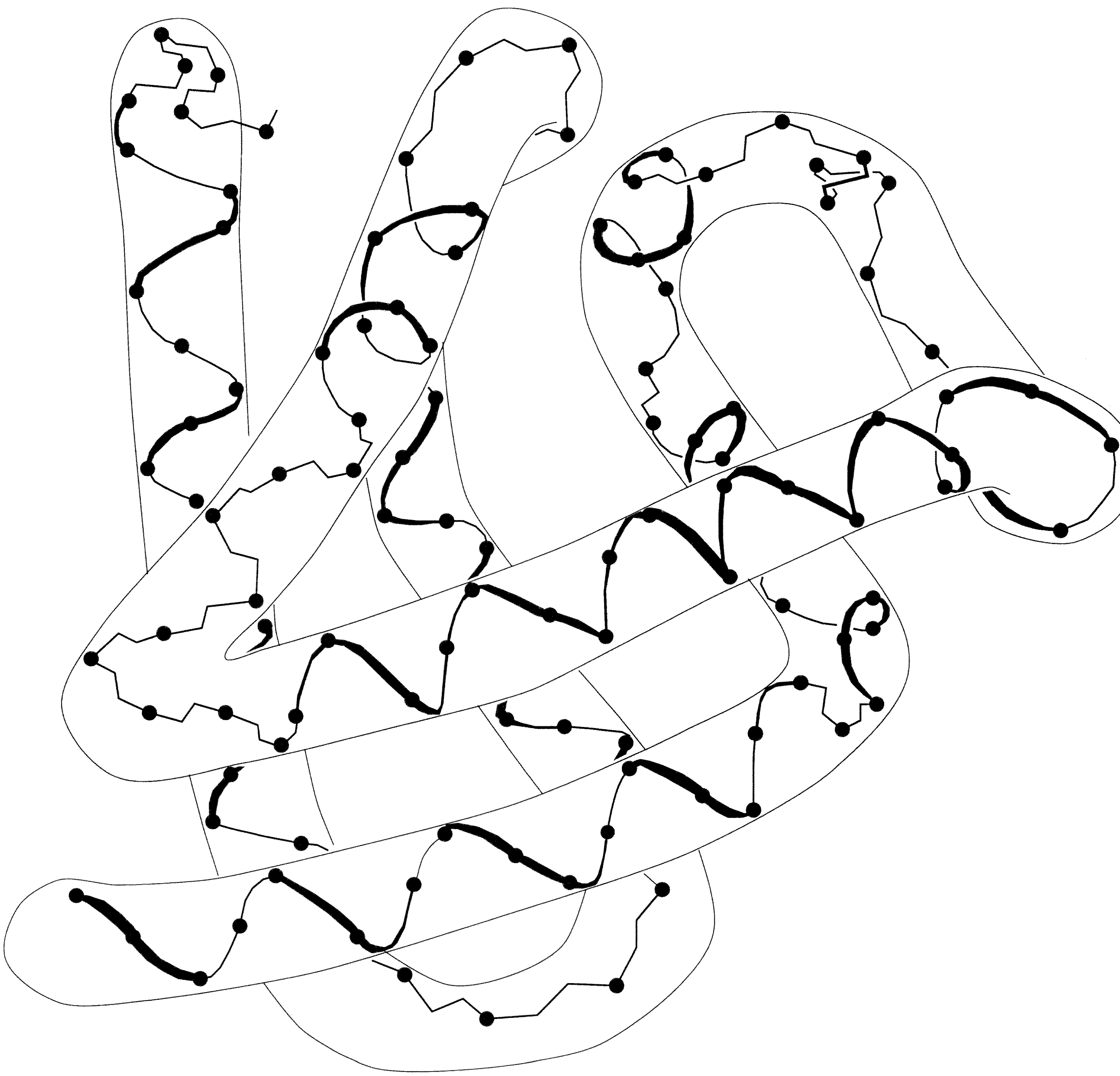


Figure 2.1 A protein structure.

CHAPTER 3

PROTEIN SEQUENCE CLASSIFICATION

3.1 Introduction

In this chapter, we discuss the classification of unlabeled protein sequences into existing, known superfamilies. The problem studied here can be stated formally as follows. Given are an unlabeled protein sequence S and a known superfamily $\mathcal{F}$, we want to determine whether or not S belongs to $\mathcal{F}$. (We refer to $\mathcal{F}$ as the *target class* and the set of sequences not in $\mathcal{F}$ as the non-target class.) In general, a superfamily is a group of proteins that share similarity in structure and function. If the unlabeled sequence S is detected to belong to $\mathcal{F}$, then one can infer the structure and function of S. This process is important in many aspects of bioinformatics and computational biology. For example, in drug discovery, if sequence S is obtained from some disease X and it is determined that S belongs to the superfamily $\mathcal{F}$, then one may try a combination of the existing drugs for $\mathcal{F}$ to treat the disease X.

There are several approaches available for protein sequence classification. One approach is based on hidden Markov models (HMMs) [37]. HMMs are a machine learning algorithm, which uses probabilistic graphical models to model time series and sequence data. It was originally applied to speech recognition [55], and now also is applied to modeling and analyzing protein superfamilies. When applying HMMs to protein sequence classification, one uses the log-odds scores produced by the models to discriminate between sequences in the target class $\mathcal{F}$ and the sequences in the non-target class.

Another approach for protein sequence classification is to compare the unlabeled sequence S with the sequences in the target class $\mathcal{F}$ and the sequences in the non-target class using an alignment tool such as BLAST [2]. One then assigns S to the class containing the sequence best matching S. This linear search approach is unsatisfactory, however, when the dataset is large. For example, consider the globin superfamily in the International Protein Sequence Database [8] maintained at the Protein Information Resource (PIR) of the National Biomedical Research Foundation at the Georgetown University Medical Center. This superfamily has 831 sequences. There are roughly 1.7×10^5 non-globin sequences in the database. Using BLAST, it would take about 40 seconds to classify an unlabeled sequence S by aligning S with all the sequences. On the other hand, using a classifier built based on machine learning algorithms may require less time in performing the classification. This is important if many classifications must be performed.

More important is that different classification approaches often complement each other; combining them yields higher precision than using them individually, as our experimental results will show later.

3.1.1 Feature Extraction from Protein Data

From a one-dimensional point of view, a protein sequence contains characters from the 20-letter amino acid alphabet $\mathcal{A}$ = {A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}. An important issue in applying neural networks to protein sequence classification is how to encode protein sequences, i.e., how to represent the protein sequences as the input of the neural networks. Indeed, sequences may not be the best representation at all. Good input representations make it easier for the neural networks to recognize underlying regularities. Thus, good input representations are crucial to the success of neural network learning [36].

We propose here new encoding techniques that entail the extraction of high-level features from protein sequences. The best high level features should be "relevant". By "relevant," we mean that there should be high mutual information between the features and the output of the neural networks, where the mutual information measures the average reduction in uncertainty about the output of the neural networks given the values of the features.

Another way to look at these features is that they capture both the global similarity and the local similarity of protein sequences. The global similarity refers to the overall similarity among multiple sequences whereas the local similarity refers to motifs (or frequently occurring substrings) in the sequences. Sections 3.2 and 3.3 elaborate on how to find the global and local similarity of the protein sequences. Section 3.4 presents our classification algorithm, which employs the Bayesian neural network originated from Mackay [43]. Section 3.5 evaluates the performance of the proposed classifier. Section 3.6 compares our approach with other protein classifiers. Section 3.7 concludes the chapter.

3.2 Global Similarity of Protein Sequences

To calculate the global similarity of protein sequences, we adopt the 2-gram encoding method originally proposed in [70, 72, 73]. The 2-gram encoding method extracts and counts the occurrences of patterns of two consecutive amino acids (residues) in a protein sequence. For instance, given a protein sequence `PVKTNVK`, the 2-gram amino acid encoding method gives the following result: 1 for `PV` (indicating `PV` occurs once), 2 for `VK` (indicating `VK` occurs twice), 1 for `KT`, 1 for `TN`, 1 for `NV`.

We also adopt the 6-letter exchange group $\{e_1, e_2, e_3, e_4, e_5, e_6\}$ to represent a protein sequence [71], where $e_1 \in \{\mathtt{H}, \mathtt{R}, \mathtt{K}\}$, $e_2 \in \{\mathtt{D}, \mathtt{E}, \mathtt{N}, \mathtt{Q}\}$, $e_3 \in \{\mathtt{C}\}$, $e_4 \in \{\mathtt{S}, \mathtt{T}, \mathtt{P}, \mathtt{A}, \mathtt{G}\}$, $e_5 \in \{\mathtt{M}, \mathtt{I}, \mathtt{L}, \mathtt{V}\}$, $e_6 \in \{\mathtt{F}, \mathtt{Y}, \mathtt{W}\}$. Exchange groups represent conservative replacements through evolution. These exchange groups are effectively equivalence classes of amino acids and are derived from PAM [21].[1] For example, the above protein sequence `PVKTNVK` can be represented as $e_4e_5e_1e_4e_2e_5e_1$. The 2-gram exchange group encoding for this sequence is: 1 for e_4e_5, 2 for e_5e_1, 1 for e_1e_4, 1 for e_4e_2, 1 for e_2e_5.

[1]Both PAM and BLOSUM [35] are amino acid substitution matrices; the latter is derived from the BLOCKS database [34].

For each protein sequence, we apply both the 2-gram amino acid encoding and the 2-gram exchange group encoding to the sequence. Thus, there are $20 \times 20 + 6 \times 6 = 436$ possible 2-gram patterns in total. If all the 436 2-gram patterns are chosen as the neural network input features, it would require many weight parameters and training data. This makes it difficult to train the neural network—a phenomenon called "curse of dimensionality." Different methods have been proposed to solve the problem by careful feature selection and by scaling of the input dimensionality [17, 71]. Below we propose to select relevant features (i.e. 2-grams) by employing a distance measure to calculate the relevance of each feature.[2]

Let X be a feature and let x be its value. Let $P(x|Class = 1)$ and $P(x|Class = 0)$ denote the class conditional density functions for feature X, where $Class\_1$ represents the target class and $Class\_0$ is the non-target class. Let $D(X)$ denote the distance function between $P(x|Class = 1)$ and $P(x|Class = 0)$, defined as [9]

$$D(X) = \int |P(x|Class = 1) - P(x|Class = 0)| dx$$

The distance measure prefers feature X to feature Y if $D(X) > D(Y)$. Intuitively, this means it is easier to distinguish between $Class\_1$ and $Class\_0$ by observing feature X than feature Y. That is, X appears often in $Class\_1$ and seldom in $Class\_0$ or vice versa. In our work, each feature X is a 2-gram pattern. Let c denote the occurrence number of the feature X in a sequence S. Let l denote the total number of 2-gram patterns in S and let $len(S)$ represent the length of S. We have $l = len(S) - 1$. Define the feature value x for the 2-gram pattern X with respect to the sequence S as

$$x = \frac{c}{len(S) - 1} \tag{3.1}$$

For example, suppose S = `PVKTNVK`. Then the value of the feature `VK` with respect to S is 2/(7-1) = 0.33.

[2]The term "distance" is from [20] which addresses feature selection for classification.

Because a protein sequence may be short, random pairings can have a large effect on the result. $D(X)$ can be approximated by the Mahalonobis distance [61] as

$$D(X) = \frac{(m_1 - m_0)^2}{d_1^2 + d_0^2} \tag{3.2}$$

where m_1 and d_1 (m_0 and d_0, respectively) are the mean value and the standard deviation of the feature X in the positive (negative, respectively) training dataset. The mean value m and the standard deviation d of the feature X in a set $\mathcal{S}$ of sequences are defined as

$$m = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3.3}$$

$$d = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - m)^2} \tag{3.4}$$

where x_i is the value of the feature X with respect to sequence $S_i \in \mathcal{S}$, and N is the total number of sequences in $\mathcal{S}$.

Let $X_1, X_2, \ldots, X_{N_g}$, $N_g \ll 436$, be the top N_g features (2-gram patterns) with the largest $D(X)$ values.[3] Intuitively, these N_g features occur more frequently in the positive training dataset and less frequently in the negative training dataset. For each protein sequence S (whether it is a training or a test sequence), we examine the N_g features in S, calculate their values as defined in Equation (3.1), and use the N_g feature values as input feature values to the Bayesian neural network for the sequence S.

To compensate for the possible loss of information due to ignoring the other 2-gram patterns, a linear correlation coefficient (LCC) between the values of the 436 2-gram patterns with respect to the protein sequence S and the mean value of the 436 2-gram patterns in the positive training dataset is calculated and used as another input feature value for S. Specifically, the LCC of S is defined as:

$$LCC(S) = \frac{436\sum_{j=1}^{436} x_j\overline{x_j} - \sum_{j=1}^{436} x_j \sum_{j=1}^{436}\overline{x_j}}{\sqrt{436\sum_{j=1}^{436} x_j^2 - (\sum_{j=1}^{436} x_j)^2}\sqrt{436\sum_{j=1}^{436}\overline{x_j}^2 - (\sum_{j=1}^{436}\overline{x_j})^2}} \tag{3.5}$$

[3]Our experimental results show that choosing $N_g \geq 30$ can yield a reasonably good performance provided one has sufficient (e.g. > 200) training sequences.

where $\overline{x_j}$ is the mean value of the jth 2-gram pattern, $1 \leq j \leq 436$, in the positive training dataset and x_j is the feature value of the jth 2-gram pattern with respect to S as defined in Equation (3.1).

3.3 Local Similarity of Protein Sequences

In contrast to the 2-gram patterns that occur from the beginning to the end of a sequence (thus referred to as global similarities), the local similarity of protein sequences refers to frequently occurring motifs where a motif is composed of substrings occurring in local regions of a sequence. Let $\mathcal{T}_p = \{S_1, \ldots, S_k\}$ be the positive training dataset. We use a previously developed sequence mining tool Sdiscover [65, 66] to find the regular expression motifs of the forms $*X*$ and $*X*Y*$ where each motif has length $\geq$ *Len* and approximately matches, within *Mut* mutations, at least *Occur* sequences in $\mathcal{T}_p$. Here, a mutation could be a mismatch, an insertion, or a deletion of a letter (residue); *Len*, *Mut*, and *Occur* are user-specified parameters. X and Y are segments of a sequence, i.e., substrings made up of consecutive letters, and $*$ is a variable length don't care (VLDC) symbol. The length of a motif is the number of the non-VLDC letters in the motif. When matching a motif with a sequence S_i, a VLDC symbol in the motif is instantiated into an arbitrary number of residues in S_i at no cost. For example, when matching a motif $*$`VLHGKKVL`$*$ with a sequence `MNVLAHGKKVLKWK`, the first $*$ is instantiated into `MN` and the second $*$ is instantiated into `KWK`. The number of mutations between the motif and the sequence is 1, representing the cost of inserting an `A` in the motif.

Often, the number of motifs returned by Sdiscover is enormous. It's useful to develop a measure to evaluate the significance of these motifs. We propose here to use the minimum description length (MDL) principle [13, 56, 68] to calculate the significance of a motif. The MDL principle states that the best model (a motif in our case) is the one that minimizes the sum of the length, in bits, of the description of the model and the length, in bits, of the description of the data (the positive training sequences in $\mathcal{T}_p$ in our case) encoded by the model.

3.3.1 Evaluating the Significance of Motifs

We adopt information theory in its fundamental form [13, 60] to measure the significance of different motifs. The theory takes into account the probability of an amino acid in a motif (or sequence) when calculating the description length of the motif (or sequence). Specifically, Shannon [60] showed that the length in bits to transmit a symbol b via a channel in some optimal coding is $-log_2 P_x(b)$, where $P_x(b)$ is the probability with which the symbol b occurs. Given the probability distribution P_x over an alphabet $\Sigma_x = \{b_1, b_2, \ldots, b_n\}$, we can calculate the description length of any string $b_{k_1} b_{k_2} \ldots b_{k_l}$ over the alphabet Σ_x by

$$-\sum_{i=1}^{l} log_2 P_x(b_{k_i}) \tag{3.6}$$

In our case, the alphabet Σ_x is the protein alphabet $\mathcal{A}$ containing 20 amino acids. The probability distribution P can be calculated by examining the occurrence frequencies of amino acids in the positive training dataset $\mathcal{T}_p$. One straightforward way to describe (or encode) the sequences in $\mathcal{T}_p$, referred to as Scheme 1, is to encode sequence by sequence, separated by a delimiter \$. Let $dlen(S_i)$ denote the description length of sequence $S_i \in \mathcal{T}_p$. Then

$$dlen(S_i) = -\sum_{j=1}^{20} n_{a_j} log_2 P(a_j) \tag{3.7}$$

where $a_j \in \mathcal{A}$, $1 \leq j \leq 20$; n_{a_j} is the number of occurrences of a_j in S_i. For example, suppose $S_i =$ `MNVLAHGKKVLKWK` is a sequence in $\mathcal{T}_p$. Then

$$dlen(S_i) = -(log_2 P(\texttt{M}) + log_2 P(\texttt{N}) + 2log_2 P(\texttt{V}) + 2log_2 P(\texttt{L}) + log_2 P(\texttt{A}) + log_2 P(\texttt{H}) +$$

$$log_2 P(\texttt{G}) + 4log_2 P(\texttt{K}) + log_2 P(\texttt{W})) \tag{3.8}$$

Let $dlen(\mathcal{T}_p)$ denote the description length of $\mathcal{T}_p = \{S_1, \ldots, S_k\}$. If we ignore the description length of the delimiter \$, then the description length of $\mathcal{T}_p$ is given by

$$dlen(\mathcal{T}_p) = \sum_{i=1}^{k} dlen(S_i) \tag{3.9}$$

Another method to encode the sequences in $\mathcal{T}_p$, referred to as Scheme 2, is to encode a regular expression motif, say M_j, and then encode the sequences in $\mathcal{T}_p$ based on M_j. Specifically, if a sequence $S_i \in \mathcal{T}_p$ can approximately match M_j, then we encode S_i based on M_j. Otherwise we encode S_i using Scheme 1.[4] Let us use an example to illustrate Scheme 2. Consider, for example, $M_j = *\texttt{VLHGKKVL}*$. We encode M_j as 1, $*$, V, L, H, G, K, K, V, L, $*$, \$0 where 1 indicates one mutation is allowed in matching M_j with S_i and \$0 is a delimiter to signal the end of the motif. Let $\sum_1$ denote the alphabet $\{a_1, a_2, \ldots, a_{20}, *, \$0\}$, where $a_1, a_2, \ldots, a_{20}$ are the 20 amino acids. Let P_1 denote the probability distribution over the alphabet $\sum_1$. $P_1(\$0)$ can be approximated by the reciprocal of the average length of motifs. $P_1(*) = n(P_1(\$0))$, $P_1(a_i) = (1 - (n+1)P_1(\$0))P(a_i)$, where n denotes the number of VLDCs in the motif M_j. For a motif of the form $*X*$, n is 2; for a motif of the form $*X*Y*$, n is 3.

Given P_1, we can calculate the description length of a motif by substituting the probability distribution P_1 for the probability distribution P_x in Equation (3.6). Specifically, let $M_j = *a_{j_1}a_{j_2}, \ldots, a_{j_k}*$. Let $dlen(M_j)$ denote the description length, in bits, of the motif M_j. Then

$$dlen(M_j) = -(2log_2P_1(*) + log_2P_1(\$0) + \sum_{i=1}^{k} log_2P_1(a_{j_i})) \tag{3.10}$$

For instance, consider again $M_j = *\texttt{VLHGKKVL}*$. We have

$$dlen(M_j) = -(2log_2P_1(*) + log_2P_1(\$0) + 2log_2P_1(\texttt{V}) + 2log_2P_1(\texttt{L}) + log_2P_1(\texttt{H}) +$$

[4]The actual number of sequences in $\mathcal{T}_p$ that are encoded by Scheme 2 is dependent on motif. For each motif used in the study presented here, more than 1/10 and less than 1/3 of the sequences are encoded based on the motif using Scheme 2.

$$log_2 P_1(\texttt{G}) + 2log_2 P_1(\texttt{K})) \tag{3.11}$$

Sequences that are approximately matched by the motif M_j can be encoded with the aid of the motif. For example, consider again $M_j = *\texttt{VLHGKKVL}*$ and $S_i = \texttt{MNVLAHGKKVLKWK}$. M_j matches S_i with one mutation, representing the cost of inserting an A in the third position of M_j. The first VLDC symbol is instantiated into MN and the second VLDC symbol is instantiated into KWK. We can thus rewrite S_i as MN $\bullet$ SS_i $\bullet$ KWK where SS_i is VLAHGKKVL and $\bullet$ denotes the concatenation of strings. Therefore we can encode S_i as M, N, \$1; 1, (O_I, 3, A); K, W, K, \$1. Here \$1 is a delimiter, 1 indicates that one mutation occurs when matching M_j with S_i and (O_I, 3, A) indicates that the mutation is an insertion that adds the letter A to the third position of M_j. In general, the mutation operations involved and their positions can be observed using approximate string matching algorithms [74]. The description length of the encoded S_i based on M_j, denoted $dlen(S_i, M_j)$, can be calculated easily as in Equation (3.10).

Suppose there are h sequences $S_{p_1} \ldots S_{p_h}$ in the positive training dataset $\mathcal{T}_p$ that can approximately match the motif M_j. The weight (or significance) of M_j, denoted $w(M_j)$, is defined as

$$w(M_j) = \sum_{i=1}^{h} dlen(S_{p_i}) - (dlen(M_j) + \sum_{i=1}^{h} dlen(S_{p_i}, M_j)) \tag{3.12}$$

Intuitively, the more sequences in $\mathcal{T}_p$ approximately matching M_j and the less bits we use to encode M_j and to encode those sequences based on M_j, the larger weight M_j has.

Using Sdiscover, one can find a set $\mathcal{S}$ of regular expression motifs of the forms $*X*$ and $*X*Y*$ from the positive training dataset $\mathcal{T}_p$ where the motifs satisfy the user-specified parameter values Len, Mut and $Occur$. We choose the top N_l motifs with the largest weight. Let $\mathcal{R}$ denote this set of motifs. Suppose a protein sequence S (whether it is a training sequence or a test sequence) can approximately match, within Mut mutations, m motifs in $\mathcal{R}$. Let these motifs be $M_1, \ldots, M_m$. The local similarity (LS) value of S, denoted $LS(S)$, is defined as

$$LS(S) = \begin{cases} \max_{1 \leq i \leq m}\{w(M_i)\} & \text{if } m \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

This LS value is used as an input feature value of the Bayesian neural network for the sequence S. Note that we use the max operator here to maximize discrimination. In general, positive sequences will have large LS values with high probabilities and have small LS values with low probabilities. On the other hand, negative sequences will have small LS values with high probabilities and have large LS values with low probabilities.

Remark. Essentially, the proposed scheme is to count amino acids in a sequence (or motif). This scheme is not complete in the sense that different sequences may have the same description length when they have the same number of the same amino acids. Second, there may be multiple ways to align a motif M with a sequence S and hence the description length of the encoded sequence S based on M may not be unique. As a consequence, the weight of a motif defined in Equation (3.12) may not be unique (in which case the proposed heuristic randomly picks one). There are several other approaches for finding motifs of different forms and for calculating their significance values (see, e.g. [13, 15, 32, 68]). However, motifs have relatively little effect on PIR sequence classification and a combination of the proposed techniques already yields a very high precision, as our experimental results show later.

3.4 The Bayesian Neural Network Classifier

We adopt the Bayesian neural network (BNN) originated from Mackay [43] to classify protein sequences.[5] There are $N_g + 2$ input features, including N_g 2-gram patterns,

[5]Software available at `http://wol.ra.phy.cam.ac.uk/pub/mackay/README.html`.

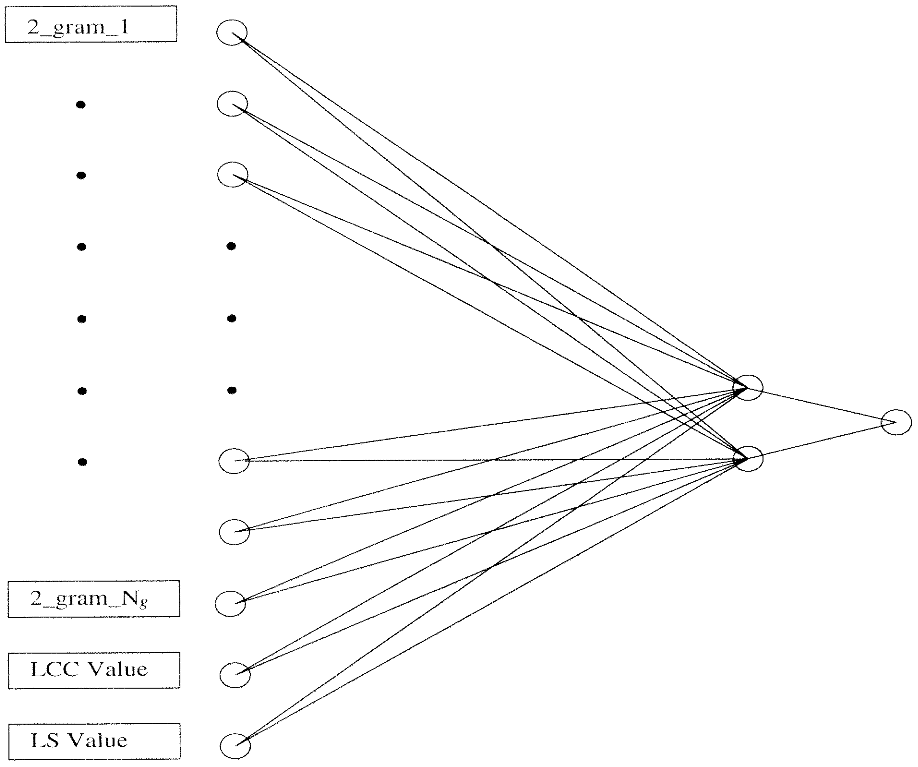


Figure 3.1 The Bayesian neural network architecture.

the *LCC* feature described in Section 2 and the *LS* feature described in Section 3. Thus, a protein sequence is represented as a vector of $N_g + 2$ real numbers. The BNN has one hidden layer containing multiple hidden units. The output layer has one output unit, which is based on the logistic activation function $f(a) = \frac{1}{1+e^{-a}}$. The BNN is fully connected between the adjacent layers. Figure 3.1 illustrates an example BNN model with 2 hidden units.

Let $\mathcal{D} = \{\mathbf{x}^{(m)}, t_m\}, 1 \leq m \leq N$, denote the training dataset including both positive and negative training sequences. $\mathbf{x}^{(m)}$ is an input feature vector including the $N_g + 2$ input feature values, and t_m is the binary (0/1) target value for the output unit. That is, t_m equals 1 if $\mathbf{x}^{(m)}$ represents a protein sequence in the target class, and 0 otherwise.

Let $\mathbf{x}$ denote the input feature vector for a protein sequence, which could be a training sequence or a test sequence. Given the architecture $\mathbf{A}$ and the weights $\mathbf{w}$

of the BNN, the output value y can be uniquely determined from the input vector $\mathbf{x}$. Because of the logistic activation function $f(a)$ of the output unit, the output value $y(\mathbf{x}; \mathbf{w}, \mathbf{A})$ can be interpreted as $P(t = 1|\mathbf{x}, \mathbf{w}, \mathbf{A})$, i.e., the probability that $\mathbf{x}$ represents a protein sequence in the target class given $\mathbf{w}$, $\mathbf{A}$. The likelihood function of the data $\mathcal{D}$ given the model is calculated by

$$P(\mathcal{D}|\mathbf{w}, \mathbf{A}) = \Pi_{m=1}^{N} y^{t_m}(1-y)^{1-t_m} = exp(-G(\mathcal{D}|\mathbf{w}, \mathbf{A})) \tag{3.14}$$

where $G(\mathcal{D}|\mathbf{w}, \mathbf{A})$ is the cross-entropy error function,

$$G(\mathcal{D}|\mathbf{w}, \mathbf{A}) = -\sum_{m=1}^{N} t_m log(y) + (1-t_m) log(1-y) \tag{3.15}$$

The $G(\mathcal{D}|\mathbf{w}, \mathbf{A})$ is the objective function in a non-Bayesian neural network training process and is minimized. This process assumes all possible weights are equally likely. The weight decay is often used to avoid overfitting on the training data and poor generalization on the test data by adding a term $\frac{\alpha}{2}\sum_{i=1}^{q} w_i^2$ to the objective function, where α is the weight decay parameter (hyperparameter), $\sum_{i=1}^{q} w_i^2$ is the sum of the square of all the weights of the neural network, and q is the number of weights. This objective function is minimized to penalize the neural network with weights of large magnitudes. Thus, it penalizes an over-complex model and favors a simple model. However, there is no precise way to specify the appropriate value of α, which is often tuned offline.

In contrast, in the Bayesian neural network, the hyperparameter α is interpreted as the parameter of a model, and is optimized online during the Bayesian learning process. We adopt the Bayesian training of neural networks described in [43] to calculate and maximize the evidence of α, namely $P(\mathcal{D}|\alpha, \mathbf{A})$. The training process employs an iterative procedure; each iteration involves three levels of inference. Figure 3.2 illustrates the training process of the BNN.

In classifying an unlabeled test sequence S represented by its input feature vector $\mathbf{x}$, the output of the BNN, $P(t = 1|\mathbf{x}, \mathbf{w}, \mathbf{A})$, is the probability that S belongs

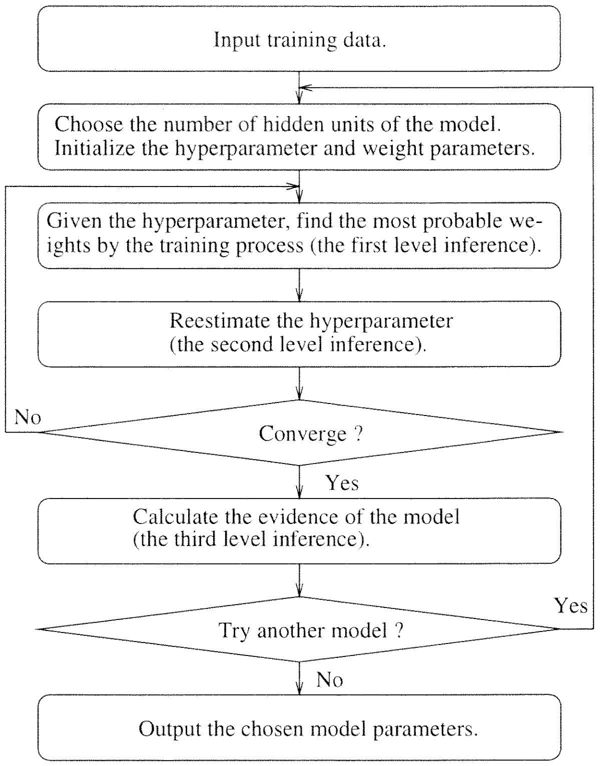

Input training data.
Choose the number of hidden units of the model. Initialize the hyperparameter and weight parameters.
Given the hyperparameter, find the most probable weights by the training process (the first level inference).
Reestimate the hyperparameter (the second level inference).
Converge ?
No
Yes
Calculate the evidence of the model (the third level inference).
Try another model ?
Yes
No
Output the chosen model parameters.

Figure 3.2 The training process of the Bayesian neural network.

to the target class. If the probability is greater than the decision boundary 0.5, S is assigned to the target class; otherwise S is assigned to the non-target class.

3.5 Performance of the BNN Classifier

3.5.1 Data

We carried out a series of experiments to evaluate the performance of the proposed BNN classifier on a Pentium II PC running the Linux operating

Dataset	N	L_m	L_x
Globin	831	115	173
Kinase-related transforming protein	350	151	502
Ras transforming protein	386	106	322
Ribitol dehydrogenase	319	129	335
Negative sequences	1,650	100	200

Table 3.1 Data used in the experiments. N is the number of sequences, L_m is the minimal length of the sequences, and L_x is the maximal length of the sequences.

system. The data used in the experiments were obtained from the International Protein Sequence Database [8], release 62, in the Protein Information Resource (PIR) maintained by the National Biomedical Research Foundation (NBRF-PIR) at the Georgetown University Medical Center. This database, accessible at `http://pir.georgetown.edu`, currently has 172,684 sequences. Table 3.1 summarizes the data used in the experiments.

Four positive datasets were considered; they were globin, kinase, ras, and ribitol superfamilies, respectively, in the PIR protein database. The negative dataset contained 1,650 protein sequences, also taken from the PIR protein database, with lengths ranging from 100 residues to 200 residues; these negative sequences did not belong to any of the four positive superfamilies. Both the positive and negative sequences were randomly divided into training sequences and test sequences, where the size of the training dataset equaled the size of the test dataset multiplied by an integer r. With the same training data, we tested several BNN models with different numbers of hidden units. When there were 2 hidden units, the evidence obtained was the largest (cf. Figure 3.2), so we fixed the number of hidden units at 2. Models with more hidden units would require more training time while achieving roughly the same performance.

Table 3.2 summarizes the parameters and base values used in the experiments. The measure used to evaluate the performance of the BNN classifier is *precision*, PR, which is defined as

$$PR = \frac{NumCorrect}{NumTotal} \times 100\% \tag{3.16}$$

where $NumCorrect$ is the number of test sequences classified correctly and $NumTotal$ is the total number of test sequences. We present the results for the globin superfamily only; the results for the other three superfamilies were similar.

Parameter	Meaning	Value
N_g	Number of 2-gram patterns used by BNN	60
N_l	Number of motifs used by BNN	20
Len	Length of motifs for Sdiscover	6
Mut	Mutation number for Sdiscover	2
$Occur$	Occurrence frequency of motifs for Sdiscover	1/10
r	size ratio	2

Table 3.2 Parameters and their base values for the proposed BNN classifier.

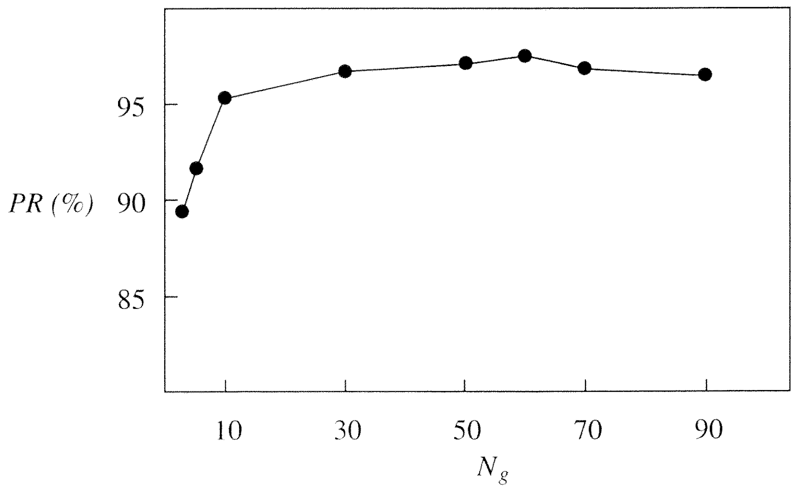


Figure 3.3 Impact of N_g in the BNN classifier.

3.5.2 Results

In the first experiment, we considered only 2-gram patterns and evaluated their effect on the performance of the proposed BNN classifier. Figure 3.3 graphs PR as a function of N_g. It can be seen that the performance improves initially as N_g increases. The reason is that the more 2-gram patterns we use, the more precisely we represent the protein sequences. However, when N_g is too large (e.g. > 90), the training data is insufficient and the performance degrades. In general, the larger N_g, the more input features the BNN classifier has, and thus the larger training dataset BNN requires. In our case, there are 561 positive training sequences and 1,089 negative training sequences. When $N_g > 90$, these data become too few to yield reasonably good performance. Figuring out how big the parameter N_g should be requires some tuning. We have not yet worked out a theory for it.

In the second experiment, we considered only motifs found by Sdiscover and studied their effect on the performance of the classifier. 1,597 motifs were found, with lengths ranging from 6 residues to 34 residues. Figure 3.4 graphs PR as a function of N_l. It can be seen that the more motifs one uses, the better performance one achieves. However, that would also require more time in matching a test sequence with the motifs. We experimented with other parameter values for Len, Mut and $Occur$ used in Sdiscover. The results didn't change as these parameters changed.

Figure 3.5 compares the effects of the various types of features introduced in the chapter. To isolate the effects of these features, we began by using only one type of features and then using their combinations. It can be seen that features generated from global similarities yield better results than those generated from local similarities. This happens because PIR superfamilies are categorized based on the global similarities of sequences. Note also that the best performance is achieved when all the features are used.

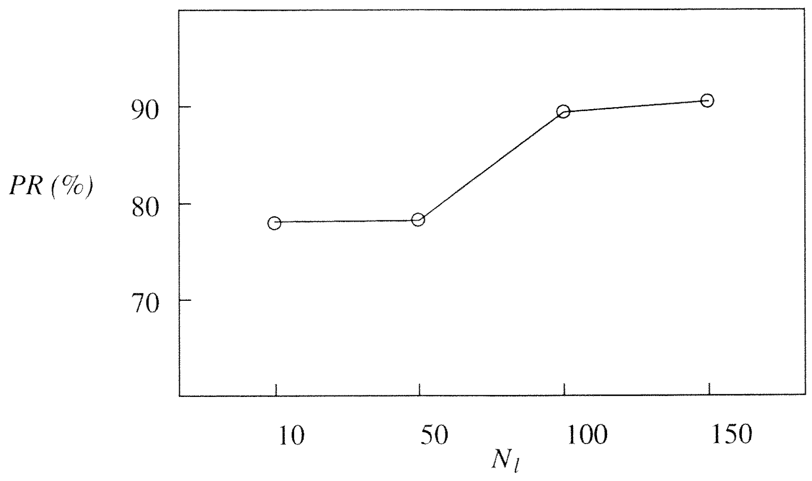


Figure 3.4 Effect of N_l in the BNN classifier.

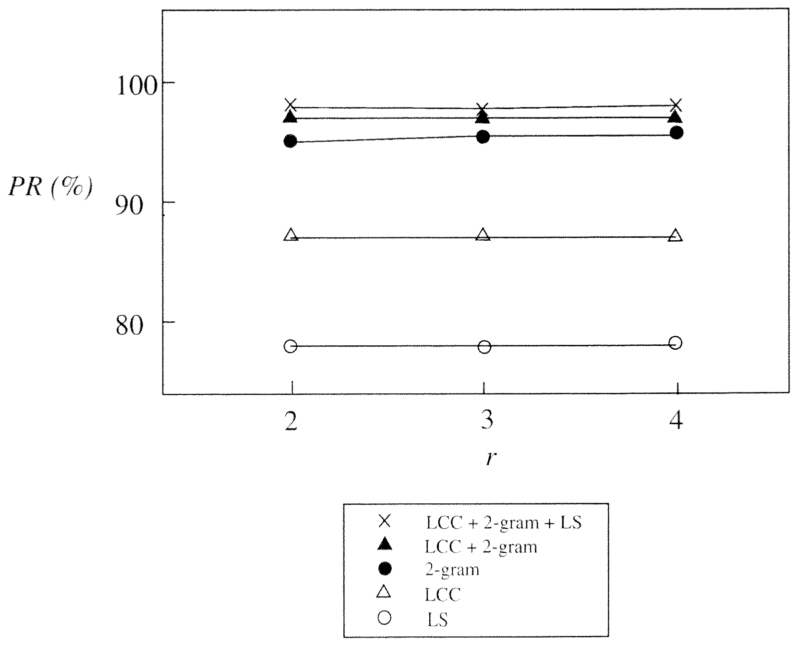


Figure 3.5 Comparison of the various types of features used by the BNN classifier.

Tool	Underlying techniques
BNN	Bayesian neural networks
BLAST	Similarity search and pairwise alignment
SAM	Hidden Markov models

Table 3.3 The bioinformatics tools studied in the chapter.

3.6 Comparison of Three Protein Classifiers

The purpose of this section is to compare the proposed BNN classifier with the BLAST classifier [2] built based on sequence alignment and the SAM classifier [37] built based on hidden Markov models. Table 3.3 summarizes the studied tools. The parameter values for the BNN classifier were as shown in Table 3.2. The BNN classifier used both 2-gram patterns and regular expression motifs. The BLAST version number was 2.0.10. We used default values for the parameters in BLAST. The SAM version number was 3.0; we used internal weighting method 2 as suggested in [37]. We chose SAM because it was shown [37] that this tool outperforms other related tools, such as HMMer [25] and Meta-MEME [29] built based on machine learning algorithms in protein sequence classification.

For BLAST, we aligned an unlabeled test sequence S with the positive training sequences (i.e. those in the target class, e.g., the globin superfamily) and the negative training sequences in the non-target class shown in Table 3.1 using the tool. If S's score was below the threshold of the expectation (e) value of BLAST, S was undetermined or unclassified. Otherwise, we assigned S to the class containing the sequence best matching S.

For SAM, we employed the program buildmodel to build the HMM model by using only the positive training sequences. We then calculated the log-odds scores for all the training sequences using the program hmmscore. The log-odds scores were all negative real numbers; the scores (e.g. -100.3) for the positive training sequences were generally smaller than the scores (e.g. -4.5) for the negative training sequences.

The largest score S_p for the positive training sequences and the smallest score S_n for the negative training sequences were recorded. Let $B_{high} = \max\{S_p, S_n\}$ and $B_{low} = \min\{S_p, S_n\}$. We then calculated the log-odds scores for all the unlabeled test sequences using the program hmmscore. If the score of an unlabeled test sequence S was smaller than B_{low}, S was classified as a member of the target class, e.g., a globin sequence. If the score of S was larger than B_{high}, S was classified as a member of the non-target class. If the score of S was between B_{low} and B_{high}, S was unclassified or undetermined.

In comparing the relative performance of these tools, we use four more measures in addition to the precision PR defined in the previous section: $specificity$, $sensitivity$, $unclassified_p$ and $unclassified_n$ where

$$\text{specificity} = (1 - \frac{N_{fp}}{N_{ng}}) \times 100\% \tag{3.17}$$

$$\text{sensitivity} = (1 - \frac{N_{fn}}{N_{po}}) \times 100\% \tag{3.18}$$

$$\text{unclassified}_p = \frac{N_{up}}{N_{po}} \times 100\% \tag{3.19}$$

$$\text{unclassified}_n = \frac{N_{un}}{N_{ng}} \times 100\% \tag{3.20}$$

N_{fp} is the number of false positives, N_{fn} is the number of false negatives, N_{up} is the number of undetermined positive test sequences, N_{un} is the number of undetermined negative test sequences, N_{ng} is the total number of negative test sequences, and N_{po} is the total number of positive test sequences. A false positive is a non-target member sequence that was misclassified as a target member sequence. A false negative refers to a sequence in the target class (e.g. the globin superfamily) that was misclassified as a non-target member.

In the first experiment, we studied the effect of the threshold of the e value in BLAST. Figure 3.6 shows the impact of e values on the performance of BLAST. It can be seen that with e = 10, BLAST performs well. With smaller e values (e.g.

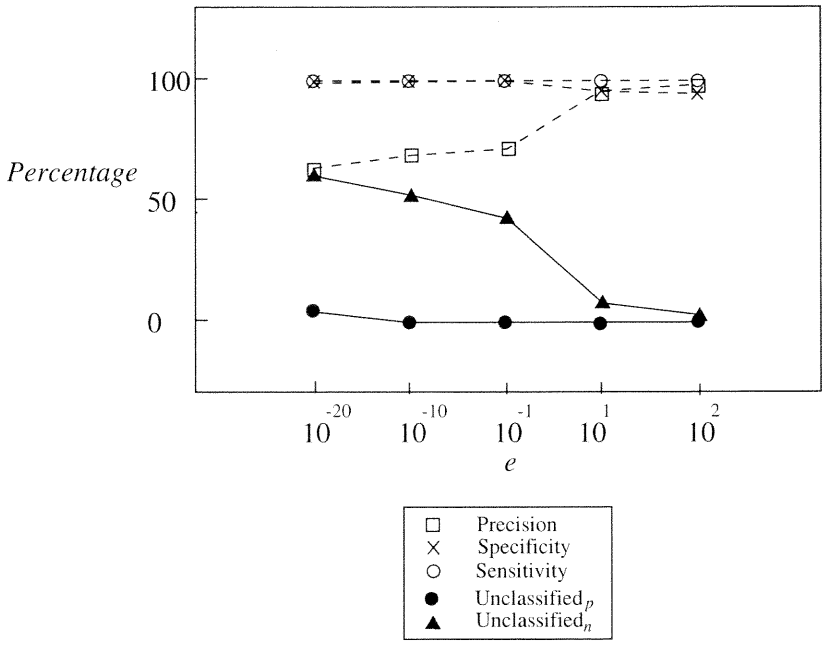


Figure 3.6 Impact of e values for BLAST.

0.1), the specificity of BLAST can approach 100% with very few false positives while the number of unclassified sequences is enormous. Thus, we fixed the threshold of the e value at 10 in subsequent experiments.

Tables 3.4, 3.5, 3.6, and 3.7 summarize the results and classification times, in seconds, of the three studied tools, referred to as basic classifiers, on the four superfamilies in Table 3.1. In addition to the basic classifiers, we developed an ensemble of classifiers, called COMBINER, that employs an unweighted voter and works as follows. If the BNN, BLAST, and SAM agree on the classification results, the result of COMBINER is the same as the results of the three classifiers; if two classifiers agree on the classification results, the result of COMBINER is the same as the results of these two classifiers; if none of the classifiers agrees on the classification results, the result of COMBINER is unclassified or undetermined. It can be seen that in comparison with BLAST and SAM, the BNN classifier is faster, yielding fewer unclassified sequences. COMBINER achieves the highest precision among all the classifiers.

	BNN	BLAST	SAM	COMBINER
Precision	98.0%	92.7%	95.3%	99.1%
Specificity	98.0%	95.7%	99.8%	98.8%
Sensitivity	98.0%	100.0%	99.6%	99.6%
Unclassified$_p$	0.0%	0.0%	1.1%	0.4%
Unclassified$_n$	0.0%	6.7%	6.2%	1.2%
CPU time	36	1,515	80	—

Table 3.4 Comparison of the studied classifiers on the globin superfamily.

	BNN	BLAST	SAM	COMBINER
Precision	99.0%	86.2%	99.4%	99.6%
Specificity	98.8%	87.8%	99.5%	99.5%
Sensitivity	100.0%	100.0%	100.0%	100.0%
Unclassified$_p$	0.0%	0.0%	0.0%	0.0%
Unclassified$_n$	0.0%	4.4%	0.2%	0.0%
CPU time	30	1,214	63	—

Table 3.5 Comparison of the studied classifiers on the kinase superfamily.

Table 3.6 shows the complementarity of the three studied tools BNN, SAM and BLAST. We see that when all the three classifiers agree, the result is correct with probability 85.69%/(85.69%+0.07%) = 99.92%.

3.7 Conclusion

In this chapter, we have presented a Bayesian neural network approach to classifying protein sequences. The main contributions of our work include

- the development of new algorithms for extracting the global similarity and the local similarity from the sequences that are used as input features of the Bayesian neural network;

	BNN	BLAST	SAM	COMBINER
Precision	98.7%	91.0%	95.5%	99.6%
Specificity	99.3%	95.0%	99.8%	99.6%
Sensitivity	96.1%	100.0%	100.0%	99.2%
Unclassified$_p$	0.0%	0.0%	3.1%	0.8%
Unclassified$_n$	0.0%	6.0%	4.6%	0.4%
CPU time	29	1,232	64	—

Table 3.6 Comparison of the studied classifiers on the ras superfamily.

	BNN	BLAST	SAM	COMBINER
Precision	96.6%	88.5%	99.4%	99.9%
Specificity	97.0%	92.6%	100.0%	99.9%
Sensitivity	94.3%	100.0%	100.0%	100.0%
Unclassified$_p$	0.0%	0.0%	2.0%	0.0%
Unclassified$_n$	0.0%	6.2%	0.3%	0.0%
CPU time	27	1,212	62	—

Table 3.7 Comparison of the studied classifiers on the ribitol superfamily.

Classification results	Percentage of the test sequences
All classifiers agreed and all were correct	85.69%
All classifiers agreed and all were wrong	0.07%
The classifiers disagreed and one of them was correct	14.24%
The classifiers disagreed and all were wrong	0.00%

Table 3.8 Complementarity between the three studied tools BNN, SAM and BLAST. The percentages in the table add up to 100%.

- the development of new measures for evaluating the significance of 2-gram patterns and frequently occurring motifs used in classifying the sequences;
- experimental studies in which we compare the performance of the proposed BNN classifier with two other classifiers, namely BLAST and SAM, on four different superfamilies of sequences in the PIR protein database.

The main findings of our work include the following.

- The three studied classifiers, BNN, SAM and BLAST, complement each other; combining them yields better results than using the classifiers individually.
- The training phase, which is done only once, of the two learning-based classifiers BNN and SAM may take some time. After the classifiers are trained, they run significantly faster than the alignment tool BLAST in sequence classification.

CHAPTER 4

PROMOTER RECOGNITION IN DNA SEQUENCES

4.1 Introduction

In this chapter we focus on the recognition of promoter sequences. Promoters are transcription signals which regulate gene expression. Characterization and recognition of Eukaryotic promoters were studied recently [19, 38, 54]. In this chapter, we propose a new method to recognize whether a DNA sequence is an E. Coli promoter sequence or not. The recognition of E. Coli promoters has been studied by Towell *et al.* [64], Mahadevan *et al.* [44], and Opitz *et al.* [47]. In KBANN [64], the topology and weights of the neural network were initialized according to the domain knowledge of E. Coli promoters. Subsequently in REGENT[47], the genetic algorithm was employed to search through the topology space of neural networks. The initial population of neural networks was created by KBANN. The fitness of each neural network was measured on a separate validation dataset. The prediction was made from an ensemble of neural networks. In KBANN and REGENT, a promoter sequence was regarded as a 57 attribute tuple, where 57 is the length of a promoter sequence in their dataset. Promoter sequences were not aligned with respect to the two binding sites. The orthogonal encoding method was employed to directly encode the promoter sequences. In Mahadevan *et al.* [44], promoter sequences were classified in a three phase process. First, the two binding sites for each promoter were located by a neural network. Then, Promoter sequences were aligned with respect to their binding sites. Finally, promoter sequences were classified by another neural network. In contrast to the previous work, we employ the expectation-maximization (EM) algorithm [22] to locate the binding sites. Then, the promoters are aligned with respect to the two binding sites. Significant features within the promoters are chosen according to their information contents. These features are then represented by the orthogonal encoding method and fed to a neural network. We also compare the proposed method with the previous work. Experimental results show that the proposed method achieves better results.

The rest of the chapter is organized as follows. Section 4.2 describes the characteristics of E. Coli promoters. Section 4.3 presents the EM algorithm for locating the binding sites. Section 4.4 shows methods of choosing significant features according to the information contents and the neural network for classifying promoter sequences. We present experimental results and conclude the chapter in Section 4.5.

4.2 Characteristics of E. Coli Promoters

An E. Coli promoter is located immediately before the E. Coli gene. Thus, successfully locating the E. Coli promoter conduces to identifying the E. Coli gene. The uncertain characteristics of the E. Coli promoters contribute to the difficulty in the promoter recognition. The E. Coli promoters contain two binding sites to which the E. Coli RNA polymerase, a kind of protein, binds [42]. The two binding sites are the -35 hexamer box and the -10 hexamer box, respectively. Each binding site consists of 6 bases (nucleotides). The central nucleotides of the two binding sites are roughly 35 bases and 10 bases, respectively, upstream of the transcriptional start site. The transcriptional start site is the first nucleotide of a codon where the transcription begins; it serves as a reference point (position +1). The consensus sequences, i.e., the prototype sequences composed of the most frequently occurring nucleotide at each position, for the -35 binding site and the -10 binding site are `TTGACA` and `TATAAT`, respectively. But none of the promoters can exactly match the two consensus sequences. The average conservation is about 8 nucleotides, meaning that a promoter sequence can match, on average, 8 out of the 12 nucleotides in the two consensus sequences. Figure 4.1 shows an example promoter sequence with the -35 binding site being `TAGCGA` and the -10 binding site being `AAAGAT`. The conservation here includes only 6 nucleotides.

```
ctttgtagcactttcacggTAGCGAaacgttagtttgaatggAAAGATgcctgCAgacacataa

                  -35 region                  -10 region      +1 region
```

Figure 4.1 An example promoter sequence. The regions are highlighted by upper case letters. The -35 region, -10 region, and +1 region are `TAGCGA`, `AAAGAT` and `CA`, respectively.

The two binding sites are separated by a spacer. The length of the spacer has an effect on the relative orientation between the -35 region and the -10 region. A spacer of 17 nucleotides is most probable. The promoter sequence in Figure 4.1 has a spacer of 17 nucleotides. Another spacer between the -10 region and the transcriptional start site also has a variable length. The most probable length of this spacer is 7 nucleotides. The promoter sequence in Figure 4.1 has a spacer of 6 nucleotides.[6] Because of the variable spacing, it is not appropriate to use the orthogonal encoding directly to encode or view a promoter sequence as an n attribute tuple, where n is the length of the promoter sequence. Many promoter sequences have the pyrimidine (`C` or `T`) at the position -1 (one nucleotide upstream of the transcriptional start site), and the purine (`A` or `G`) at the transcriptional start site (position +1). The +1 region includes the nucleotides at the position -1 and the transcriptional start site. The promoter sequence in Figure 4.1 has a nucleotide `C` at the position -1 and a nucleotide `A` at the transcriptional start site.

In addition to these salient characteristics in the two binding sites and the transcriptional start site, there are some non-salient characteristics in other regions. In Galas *et al.* [27] and Mengeritsky *et al.* [46], a pattern matching method was applied to the characterization of E. Coli promoters. Some weak motifs were found around the -44 and the -22 regions. A weak motif is a subsequence, which occurs

[6]In general, the distance between the -10 binding site and the transcriptional start site varies from 3 to 11 bases. The distance between the -35 binding site and the -10 binding site varies from 15 to 21 bases. These varying distances render promoter recognition difficult, as both the contents and positions of the binding sites are uncertain.

frequently in a region. We use the term "weak", since the frequency of a base of the motif is not as significant as the frequency of a base of the consensus sequences occurring in the binding sites. In Cardon *et al.* [16], as many as 8 nucleotides (weak motifs) within the spacer region between the two binding sites were found to have contributions to the specificity of the promoter sequences. Recently, Pedersen and Engelbrecht [53] adopted a neural network to characterize E. Coli promoters. The significance of a weak motif was measured by the decrease in the maximum correlation coefficient when all motifs except that weak motif were fed into the neural network. By using this method, the authors found some weak motifs in the +1, -22, and -44 regions. It is interesting to observe that these weak motifs are spaced regularly with a period of 10–11 nucleotides corresponding to one helical turn. This phenomenon suggests that the RNA polymerase makes contact with the promoter on one face of the DNA. Subsequently, the characterization of E. Coli promoter sequences was carried out by the hidden Markov model [52]. It was observed that the position of the -35 binding site relative to the transcriptional start site is very flexible. More recently, the periodic occurrence was confirmed [48, 49].

These weak motifs can also be revealed by the sequence logos described in Schneider and Stephens [59]. Figure 4.2 displays the sequence logos of 438 E. Coli promoters aligned according to the transcriptional start site.[7] Given a set of aligned sequences, the sequence logos measure the non-randomness of each position l independently by the Shannon entropy for that position:

$$R(l) = log_2(|\mathcal{D}|) - (-\sum_{b\in\mathcal{D}} f(b,l)log_2 f(b,l))), \tag{4.1}$$

where $|\mathcal{D}|$ is the cardinality of the 4-letter DNA alphabet $\mathcal{D} = \{\texttt{A, T, G, C}\}$, $log_2(|\mathcal{D}|) = 2$ is the maximum uncertainty at any given position, $-\sum_{b\in\mathcal{D}} f(b,l)log_2 f(b,l)$

[7]The sequence logos were produced by using the software available at http://www-lecb.ncifcrf.gov/~toms/delila.html.

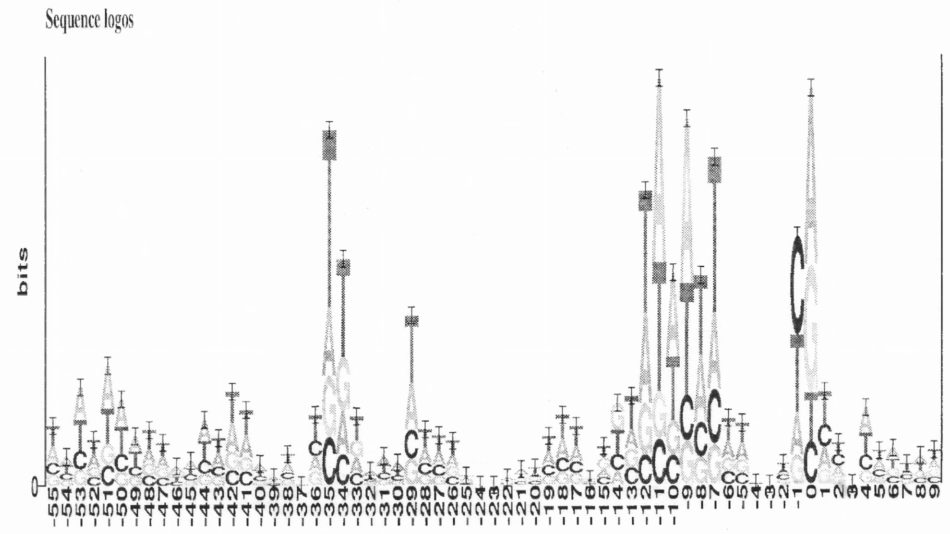


Figure 4.2 The sequence logos of 438 E. Coli promoter sequences. Position 0 in the figure is the transcriptional start site, which is equivalent to position +1 described in the text. The negative positions in the figure are consistent with those described in the text.

is the Shannon entropy of position l, and $f(b,l)$ is the frequency of base b at position l.

The height at each position represents the information content of that position. The higher the information content, the less random that position is. The size of each base at each position of the logos is proportional to the frequency of the base. Recall that a weak motif is a frequently occurring subsequence in a region. In the sequence logos, a weak motif consists of positions (bases) with non-zero information content. From Figure 4.2, it can be seen that some weak motifs exist in the +1, -22, -29, and -44 regions. In the following section, we present an EM algorithm for locating the binding sites of promoter sequences.

4.3 Locating Binding Sites by the EM Algorithm

To align subsequences in the -44 region, the -35 region, the -29 region, the -22 region and the -10 region, we need to locate the two binding sites in the E. Coli promoters. Locating the -35 region and the -10 region may be done by the EM algorithm. In general, the EM algorithm can be applied for the maximum likelihood estimation problem when data are incomplete. Locating the binding sites by the EM algorithm was proposed by Lawrence *et al.* [41] and Bailey [6]. It was then generalized by Cardon *et al.* [16] to allow for different spacers between the two binding sites. By contrast, our method uses the Bayesian Maximum *A Posteriori* (MAP) EM algorithm and considers the binding sites separately from the spacer. Secondly, our method does not assume the spacer length to be uniformly distributed.

Let T represent the set of promoter sequences in the training set, i.e., T contains all positive training sequences. Let K denote the cardinality of T. For a promoter sequence $S_i \in T$, the length of the spacer between the -10 region and the transcriptional start site, denoted sp_{10}, and the length of the spacer between the -35 region and the -10 region, denoted sp_{35}, are unobserved, though S_i is observed. Specifically, we refer to the positive training sequences as "observed" data since they are given. These observed data are incomplete, because the lengths of the two spacers are not given (the lengths are referred to as "unobserved" or "missing" data). In general, sp_{10} varies from 3 to 11 and sp_{35} varies from 15 to 21. For each S_i, the missing data sp_{10} and sp_{35} are represented by a vector $z_i = (z_{i,1}, \ldots, z_{i,63})$, where

$$z_{i,f(m,n)} = \begin{cases} 1 & \text{if } m = sp_{10}, \text{ and } n = sp_{35} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

where $f(m,n) = (m-3) * 7 + n - 15$. Each binding site consists of 6 bases. Assume that the nucleotides at the two binding sites are independent. Then one can use the Position Weight Matrix (PWM) described in Staden [62] to model nucleotides at each position of the two binding sites. Let $P_{10,j}(x), j = 1, \ldots, 6$, denote the probability of

```
Input: the training and the test DNA sequences.
Output: the position weight matrices $\mathbf{P}_{10}$, $\mathbf{P}_{35}$, and the putative $sp_{10}$, $sp_{35}$ of each DNA sequence. Initialize probability distributions $\mathbf{P}_{10}^0$, $\mathbf{P}_{35}^0$, and $P(z_{i,f(sp_{10},sp_{35})})^0$;
do /* iterate to convergence */
    begin
        /* E step. */
        for each promoter sequence $S_i \in T$ do
            begin
                for each possible values of $sp_{10}$, $sp_{35}$ do
                    calculate $P(S_i|z_{i,f(sp_{10},sp_{35})} = 1, \theta^t)$ according to Equation (4.9);
                for each possible values of $sp_{10}$, $sp_{35}$ do
                    calculate $P(z_{i,f(sp_{10},sp_{35})} = 1|S_i, \theta^t)$ according to Equation (4.10);
            end;
        calculate $f_{10,j}$, $f_{35,j}$, and $f_s$ according to Equation (4.13);
        /* M step. */
        calculate $\mathbf{P}_{10}^{t+1}$, $\mathbf{P}_{35}^{t+1}$, and $P^{t+1}(z_{i,f(sp_{10},sp_{35})})$ according to Equation (4.14);
    end;
while the changes of the values of $\mathbf{P}_{10}^{t+1}$, $\mathbf{P}_{35}^{t+1}$, and $P^{t+1}(z_{i,f(sp_{10},sp_{35})})$ > a predefined threshold;
for each DNA sequence $S_i \in$ the training and test set do
    choose the values of $sp_{10}$, $sp_{35}$ which maximizes $P(S_i, z_{i,f(sp_{10},sp_{35})} = 1|\theta)$;
```

Figure 4.3 The proposed EM algorithm.

x, $x \in \mathcal{D}$, occurring at position j in the -10 region. Let $\mathbf{P_{10}}$ denote $(P_{10,1}, \ldots, P_{10,6})$. Let $P_{35,j}(x), j = 1, \ldots, 6$, denote the probability of x occurring at position j in the -35 region. Let $\mathbf{P_{35}}$ denote $(P_{35,1}, \ldots, P_{35,6})$. Thus, $P_{10,j}$ and $P_{35,j}$, $1 \leq j \leq 6$ (from upstream nucleotides to downstream nucleotides) are in the multinomial distribution. Let θ denote the PWM model parameter ($\mathbf{P_{10}}$, $\mathbf{P_{35}}$). For each promoter sequence, had we known the lengths of the two spacers, it would be easy to calculate the model parameter θ. The proposed EM algorithm can estimate the model parameter θ from the incomplete data. Based on the estimates of the model parameter, it is possible to determine the locations of the two putative binding sites for any DNA sequence. Figure 4.3 shows the algorithm.

The EM algorithm proceeds iteratively to converge. Each iteration consists of two steps: the Expectation step (E step) and the Maximization step (M step). Unfortunately, the EM algorithm can not guarantee to reach the global maxima. It may be trapped in the local maxima. Thus, we use a MAP EM algorithm to make the objective function more concave [45]. The prior probabilities of $P_{10,j}$ and $P_{35,j}$, $j = 1, \ldots, 6$, are in the Dirichlet distributions, conjugate to the multinomial distributions, which means the posterior probabilities are also in the Dirichlet distributions [10, 58]. The Dirichlet distribution on the probability vector $P = (p(\mathtt{A}), p(\mathtt{C}), p(\mathtt{G}), p(\mathtt{T}))$ (P could be $P_{10,j}$ or $P_{35,j}$, $j = 1, \ldots, 6$) has the form:

$$P_D(p(\mathtt{A}), p(\mathtt{C}), p(\mathtt{G}), p(\mathtt{T}) | \alpha_{\mathtt{A}}, \alpha_{\mathtt{C}}, \alpha_{\mathtt{G}}, \alpha_{\mathtt{T}}) = \frac{\Gamma(\alpha_0)}{\Pi_{x=\mathtt{A}}^{\mathtt{T}} \Gamma(\alpha_x)} \Pi_{x=\mathtt{A}}^{\mathtt{T}} p(x)^{\alpha_x - 1} \tag{4.3}$$

where $\alpha_0 = \sum_{x=\mathtt{A}}^{\mathtt{T}} \alpha_x$, $0 \leq p(x) \leq 1$, $\sum_{x=\mathtt{A}}^{\mathtt{T}} p(x) = 1$, and $\alpha_x > 0$, $\Gamma()$ is a Gamma function. The mean of $p(x)$, $x \in \mathcal{D}$, is

$$E(p(x)) = \frac{\alpha_x}{\alpha_0} \tag{4.4}$$

The Variance of $p(x)$, $x \in \mathcal{D}$, is

$$Var(p(x)) = \frac{(\alpha_0 - \alpha_x)\alpha_x}{\alpha_0^2(\alpha_0 + 1)} \tag{4.5}$$

The mean values of the Dirichlet distribution on the probability vector $P_{10,j}$ and $P_{35,j}$, $1 \leq j \leq 6$ are taken from [31]. Thus, α_x, $x \in \mathcal{D}$, of the Dirichlet distribution can be calculated from (4.4) given α_0 of the Dirichlet distribution, which is regarded as a parameter.[8]

The E step calculates the sum of log of the prior probability of θ, Pr_θ, and the expected complete-data log likelihood, where the expectation is over the distribution of the missing data given the observed data and current estimates of θ. Thus, the E step calculates

$$E_{Z|T,\theta^t} log P(T, Z|\theta) + log Pr_\theta. \tag{4.6}$$

Assume all S_i, $1 \leq i \leq K$ are independent, and $P(Z|\theta) = P(Z)$, i.e., the probability distribution of unobserved data, is independent of θ.

$$E_{Z|T,\theta^t} log P(T, Z|\theta)$$

[8]Our experiments show the performance is not very sensitive to the value of α_0. Consequently we choose $\alpha_0 = 20$.

$$= E_{Z|T,\theta^t} log(P(T|Z,\theta)P(Z))$$

$$= \sum_{i=1}^{K}\sum_{m=3}^{11}\sum_{n=15}^{21} P(z_{i,f(m,n)}=1|S_i,\theta^t) log(P(S_i|z_{i,f(m,n)}=1,\theta)P(z_{i,f(m,n)}=1)) \quad (4.7)$$

Suppose that all promoter sequences in the training set T are 65 nucleotides long (the position 1 is now at the upstream end and the position 65 now is at the downstream end) and are aligned with respect to the transcriptional start site, which is at position 56. Let $S_{i,j}$ denote the nucleotide at position j of the promoter sequence S_i. Define

$$I_{i,j,x} = \begin{cases} 1 & \text{if } S_{i,j} = x \\ 0 & \text{otherwise} \end{cases} \quad (4.8)$$

For each S_i, given θ^t and $z_{i,f(m,n)} = 1$, the likelihood of S_i is

$$P(S_i|z_{i,f(m,n)} = 1, \theta^t) = \Pi_{j=1}^{6} P_{10,j}^t(S_{i,49-m+j}) \Pi_{j=1}^{6} P_{35,j}^t(S_{i,43-m-n+j}) \quad (4.9)$$

From the Bayes' law, we have

$$P(z_{i,f(m,n)=1}|S_i,\theta^t)$$
$$= \frac{P(S_i|z_{i,f(m,n)}=1,\theta^t)P^t(z_{i,f(m,n)}=1)}{P(S_i|\theta^t)}$$
$$= \frac{P(S_i|z_{i,f(m,n)}=1,\theta^t)P^t(z_{i,f(m,n)}=1)}{\sum_{m=3}^{11}\sum_{n=15}^{21} P(S_i|z_{i,f(m,n)}=1,\theta^t)P^t(z_{i,f(m,n)}=1)} \quad (4.10)$$

Leaving out the terms not involving θ, we have log of the prior of θ, Pr_θ

$$logPr_\theta = \sum_{j=1}^{6}\sum_{x=\mathtt{A}}^{\mathtt{T}} (\alpha_x^{10,j} - 1) logP_{10,j}(x) + \sum_{j=1}^{6}\sum_{x=\mathtt{A}}^{\mathtt{T}} (\alpha_x^{35,j} - 1) logP_{35,j}(x) \quad (4.11)$$

Substituting (4.9) and (4.10) into (4.7), we have

$$E_{Z|T,\theta^t} logP(T,Z|\theta) + logPr_\theta$$
$$= \sum_{j=1}^{6}(K + \alpha_0^{10,j} - 4)\sum_{x=\mathtt{A}}^{\mathtt{T}} f_{10,j}(x) logP_{10,j}(x) + \sum_{j=1}^{6}(K + \alpha_0^{35,j} - 4)\sum_{x=\mathtt{A}}^{\mathtt{T}}$$
$$f_{35,j}(x) logP_{35,j}(x) + K\sum_{m=3}^{11}\sum_{n=15}^{21} f_s(m,n) logP(z_{i,f(m,n)} = 1) \quad (4.12)$$

where

$$f_{10,j}(x) = \frac{1}{K + \alpha_0^{10,j} - 4}(\alpha_x^{10,j} - 1 + \sum_{i=1}^{K}\sum_{m=3}^{11}\sum_{n=15}^{21} I_{i,49-m+j,x}P(z_{i,f(m,n)} = 1|S_i, \theta^t))$$

$$f_{35,j}(x) = \frac{1}{K + \alpha_0^{35,j} - 4}(\alpha_x^{35,j} - 1 + \sum_{i=1}^{K}\sum_{m=3}^{11}\sum_{n=15}^{21} I_{i,43-m-n+j,x}P(z_{i,f(m,n)} = 1|S_i, \theta^t))$$

$$f_s(m,n) = \frac{1}{K}\sum_{i=1}^{K}\sum_{m=3}^{11}\sum_{n=15}^{21} P(z_{i,f(m,n)} = 1|S_i, \theta^t) \qquad (4.13)$$

Let θ^0 denote the value of θ at the beginning of the first iteration. θ^0 was initialized with random values so that the E step can proceed. In each iteration, we use the current estimate θ^t to calculate the sum of log of the prior probability of θ and the expected complete data log likelihood.

The M step maximizes (4.12) with respect to θ. According to the information theory (Lemma 1.4.1 of [4]), $\sum_{x=\mathtt{A}}^{\mathtt{T}} f_{10,1}(x)logP_{10,1}(x)$ is maximized when $P_{10,1}(x)$ equals $f_{10,1}(x)$, where $f_{10,1}(x)$ is a constant. For instance, when $f_{10,1}(\mathtt{A})$, $f_{10,1}(\mathtt{C})$, $f_{10,1}(\mathtt{G})$, and $f_{10,1}(\mathtt{T})$ are 0.4, 0.3, 0.2, and 0.1 respectively, $\sum_{x=\mathtt{A}}^{\mathtt{T}} f_{10,1}(x)logP_{10,1}(x)$ can be maximized when $P_{10,1}(\mathtt{A})$, $P_{10,1}(\mathtt{C})$, $P_{10,1}(\mathtt{G})$, and $P_{10,1}(\mathtt{T})$ are 0.4, 0.3, 0.2, and 0.1 respectively. Thus the maximum likelihood estimate of θ is just sample frequencies $f_{10,j}$, $f_{35,j}$, and f_s, $j = 1, \ldots, 6$. That is,

$$P_{10,j}^{t+1}(x) = f_{10,j}(x), x \in \mathcal{D}$$

$$P_{35,j}^{t+1}(x) = f_{35,j}(x), x \in \mathcal{D}$$

$$P^{t+1}(z_{i,f(m,n)} = 1) = f_s(m,n) \qquad (4.14)$$

The new value of θ can be used in the next iteration. The process iterates to convergence. Given the model parameters calculated from the positive training sequences (i.e., the promoter sequences in the training data set T), we can determine the locations of the two putative binding sites of any DNA sequence S_i, which could be a training sequence or a test sequence, a positive sequence or a negative sequence, by choosing the two spacer lengths sp_{10} and sp_{35} which are calculated by $\max_{3\leq m\leq 11, 15\leq n\leq 21}\{P(S_i, z_{i,f(m,n)} = 1|\theta)\}$

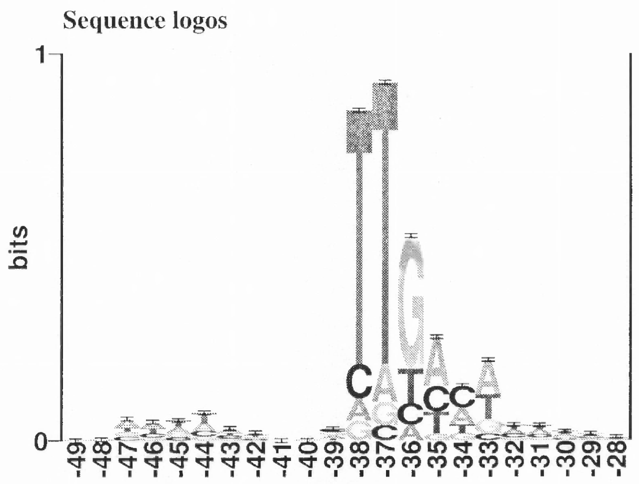


Figure 4.4 The sequence logos around the -35 binding site.

4.4 Feature Extraction and Classification

After locating the two binding sites, we can align all training promoter sequences with respect to the two binding sites as well as the transcriptional start site. Figure 4.4, 4.5, and 4.6 show the sequence logos of regions around the -35 binding site, the -10 binding site, and the transcriptional start site respectively for all the promoter sequences. Compared to Figure 4.2, it is easy to see the consensus sequences, indicating that the EM algorithm can precisely locate the binding sites for each promoter without the prior knowledge of the contents of two consensus sequences. For the training promoter sequences, positions with high information contents are chosen as features for classification. Thus, 17 positions around the -35 binding site, 11 positions around the -10 binding site, and 7 positions around the transcriptional start site respectively are chosen as features. The 35 nucleotides for each training sequence and test sequence are encoded by the *orthogonal encoding*.

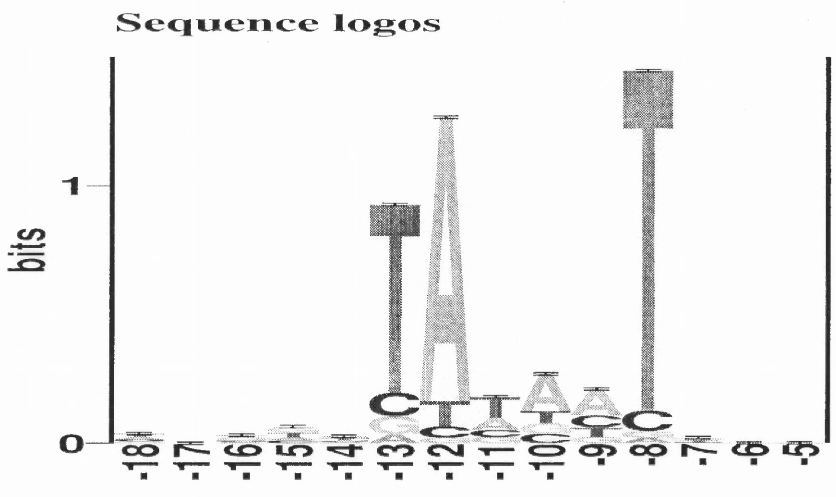


Figure 4.5 The sequence logos around the -10 binding site.

In orthogonal encoding, nucleotides in a DNA sequence are viewed as unordered categorical values, and are represented by 4 dimensional orthogonal binary vectors, where 4 is the cardinality of the 4-letter DNA alphabet $\mathcal{D}$. That is, we use 4 binary (0/1) variables, among which only one binary variable is set to 1 to represent one of the 4 possible categorical values and the rest are all set to 0. For instance, we represent the nucleotide `A` by "1000". Figure 4.7 shows an example of the orthogonal encoding. When there is an uncertain nucleotide denoted by '`X`', we use "1111" to represent it. Besides these 35 positions, the two spacer lengths are also chosen as features. Thus, there are 142 input units in the input layer of the neural network of a DNA sequence.

The neural network we use has one hidden layer with sigmoid activation functions. The output layer of the neural network has one output unit. The output value is bounded between 0 and 1. The neural network is fully connected. The network is trained with scaled conjugate gradient algorithm [11]. We tested the

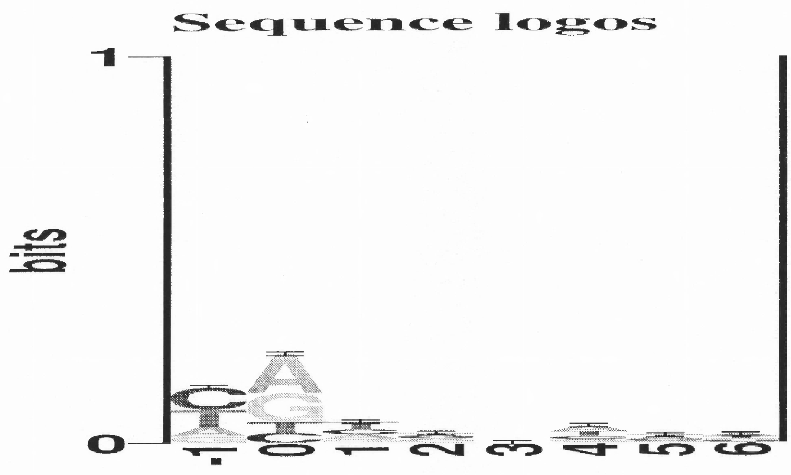


Figure 4.6 The sequence logos around the transcriptional start site.

neural networks with different numbers of hidden units; the system has the best performance with 20 hidden units.

4.5 Results

We conducted three experiments. Table 4.1 shows the datasets used in the experiments. We use *precision* to measure the performance of the proposed method. The precision is defined as

$$\frac{C}{N} * 100\% \tag{4.15}$$

where C is the number of test sequences classified correctly and N is the total number of test sequences. A *false positive* is a non-promoter test sequence that was misclassified as a promoter sequence. A *true positive* is a promoter test sequence that was also classified as a promoter sequence. The *specificity* is defined as

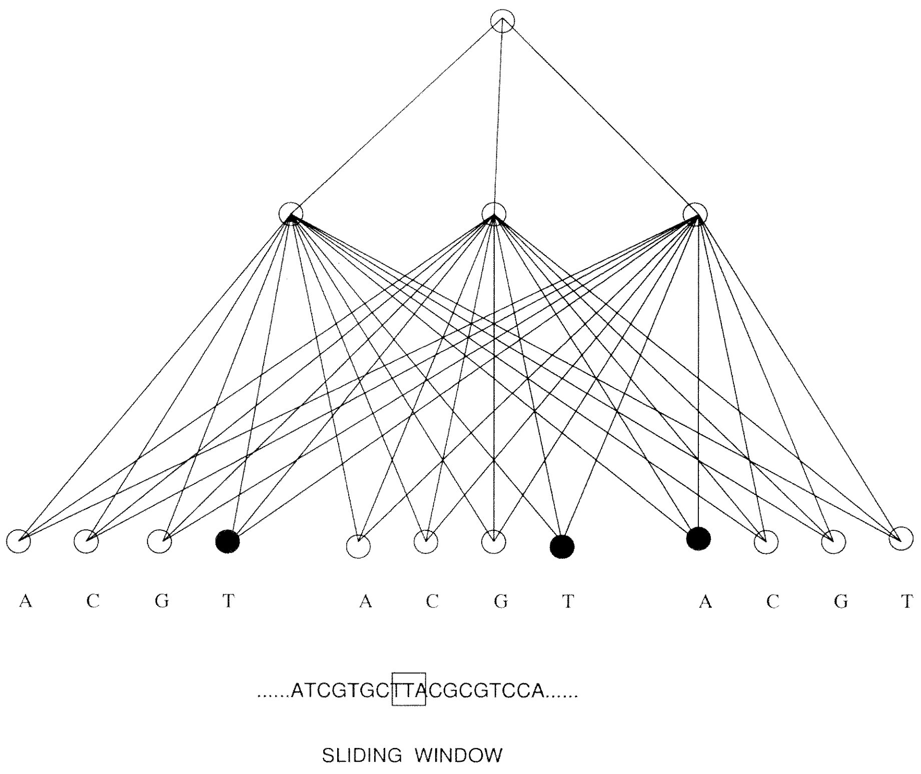


Figure 4.7 An example of the orthogonal encoding of a DNA sequence.

$$(1 - \frac{N_{fp}}{N_{ng}}) * 100\% \tag{4.16}$$

where N_{fp} is the number of false positives and N_{ng} is the total number of negative test sequences. The *sensitivity* is defined as

$$\frac{N_{tp}}{N_{po}} * 100\% \tag{4.17}$$

where N_{tp} is the number of true positives and N_{po} is the total number of positive test sequences.

In the first two experiments, we compare our system with three recently published approaches: REGENT [47], KBANN [64], and Mahadevan *et al.* [44]. Table 4.2 compares our system with REGENT and KBANN using a ten-fold cross validation test on the same dataset as used in [47, 64] which contains 234 promoter sequences and 702 negative sequences. In ten-fold cross validation, the dataset containing both

Test number	1	2	3
P_{tr}	$\frac{9}{10}$ of 234	362	$\frac{9}{10}$ of 438
N_{tr}	$\frac{9}{10}$ of 702	4500	$\frac{9}{10}$ of 5000
P_{ts}	$\frac{1}{10}$ of 234	126	$\frac{1}{10}$ of 438
N_{ts}	$\frac{1}{10}$ of 702	5000	$\frac{9}{10}$ of 5000
Tot_p	234	488	438
Tot_n	702	9500	5000

Table 4.1 The datasets used in the three experiments. P_{tr}, N_{tr}, P_{ts}, and N_{ts} represent the numbers of positive training, negative training, positive test, and negative test sequences respectively. Tot_p and Tot_n represent the total number of positive and negative sequences respectively.

	Our System	REGENT	KBANN
Precision	97.22%	95.83%	93.70%
Errors	26	39	59

Table 4.2 Comparison of our system with REGENT and KBANN.

the positive data (promoters) and the negative data (non-promoters) was randomly split into ten mutually exclusive folds $D_1, D_2, \ldots, D_{10}$ of approximately equal size. The neural network was trained and tested ten times. During the ith time, it was trained on $D - D_i$, and tested on D_i. We allocated the data in such a way that the training dataset $D - D_i$ (the test dataset D_i respectively) has approximately $\frac{9}{10}$ ($\frac{1}{10}$, respectively) positive data and $\frac{9}{10}$ ($\frac{1}{10}$, respectively) negative data. The average over the ten tests was taken. Table 4.3 compares our system with Mahadevan *et al.* [44]. The training set we used includes 362 promoter sequences, and 4500 random sequences with 60% `AT` composition, which means the sum of probabilities of `A` and `T` is 0.6. The system is tested on a test dataset containing 126 promoter sequences and 5000 random sequences with the same `AT` composition as those used in Mahadevan *et al.* [44].

	Our System	Mahadevan *et al.*
Precision	91.94%	90.40%
Specificity	91.76%	90.20%
Sensitivity	99.20%	98.00%

Table 4.3 Comparison of our system with Mahadevan *et al.*

Precision Rate	96.29%
Specificity	96.68%
Sensitivity	91.78%

Table 4.4 Performance of the neural network on the latest dataset.

In the third experiment, we adopted E. Coli promoter sequences taken from the latest E. Coli promoter compilation [49]. There were 441 E. Coli promoters aligned with respect to the transcriptional start site. We trimmed each promoter sequence to a sequence of 65 nucleotides including nucleotides from -55 (55 nucleotides upstream of the transcriptional start site) to +10 (10 nucleotides downstream of the transcriptional start site). This gave us 438 promoter sequences. The negative data (i.e., non-promoter sequences) was randomly generated with 60% `AT` composition. Each negative sequence is also 65 nucleotides long. There were 5000 negative sequences. Table 4.4 gives the ten-fold cross validation results. As shown in these tables, our system achieves better performance, which is due to precisely locating the binding sites by the EM algorithm.

4.6 Conclusion

In this chapter, we have proposed a new technique to recognize E. Coli promoter sequences. We first use a Bayesian MAP EM algorithm to locate the binding sites of the promoter sequences. We then align promoters with respect to the two binding sites as well as the transcriptional start site. Significant features within promoters are extracted according to their information contents. These features are then represented by the orthogonal encoding method and fed to neural networks. Empirical study shows that the proposed approach achieves better results when comparing with existing methods on the same dataset. This happens because our EM algorithm is able to precisely locate the binding sites of the promoter sequences.

CHAPTER 5

GENOME MINING

5.1 Introduction

We developed a web-based genome mining tool which allows a user to run our genome mining software from the web. The genome mining tool includes two components. The protein classification component takes as the input a protein sequence in the FASTA format, extracts the global similarity information (2-gram encoding) and the local similarity information (motifs), and feeds these values to the trained neural networks. The neural networks can classify the input protein sequence into globin, kinase, ras, and ribitol superfamilies in the Protein Information Resources (PIR) at the National Biomedical Research Foundation.

The promoter recognition component takes as the input a 65 nucleotide long DNA sequence in the FASTA format, locates the two putative binding sites within the DNA sequence, extracts the DNA segments using high information contents, represents the DNA segments using the orthogonal encoding method, and feeds these values to the trained neural network. The neural network can recognize whether the input DNA sequence is a E. Coli promoter or not.

5.2 Architecture of the Genome Mining tool

Figure 5.1 illustrates the system architecture of the genome mining tool. A web user can access the main page of the genome mining tool from http://www.cis.njit.edu/~eservice (see Figure 5.2). The web user can run either the protein classification component or the promoter recognition component. The web user can submit a query sequence or use the sample sequence provided by the web server (see Figure 5.3 and 5.4). The web server validates the query sequence and passes the query sequence to either the protein classification component or the promoter recognition component depending on the web user's choice. Given the query sequence, the

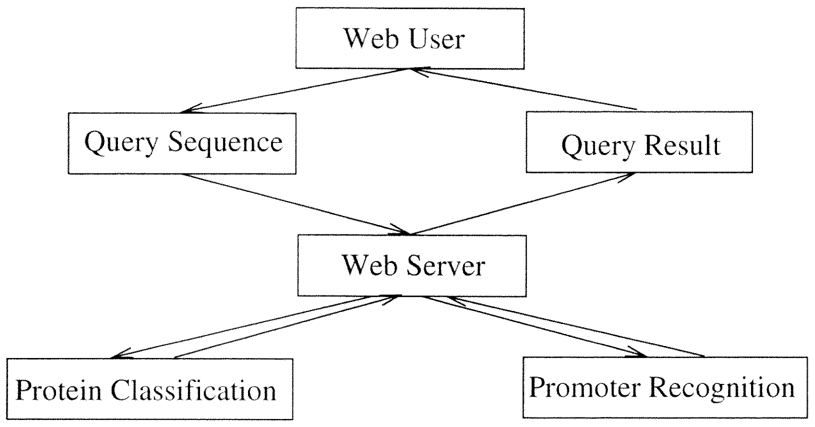
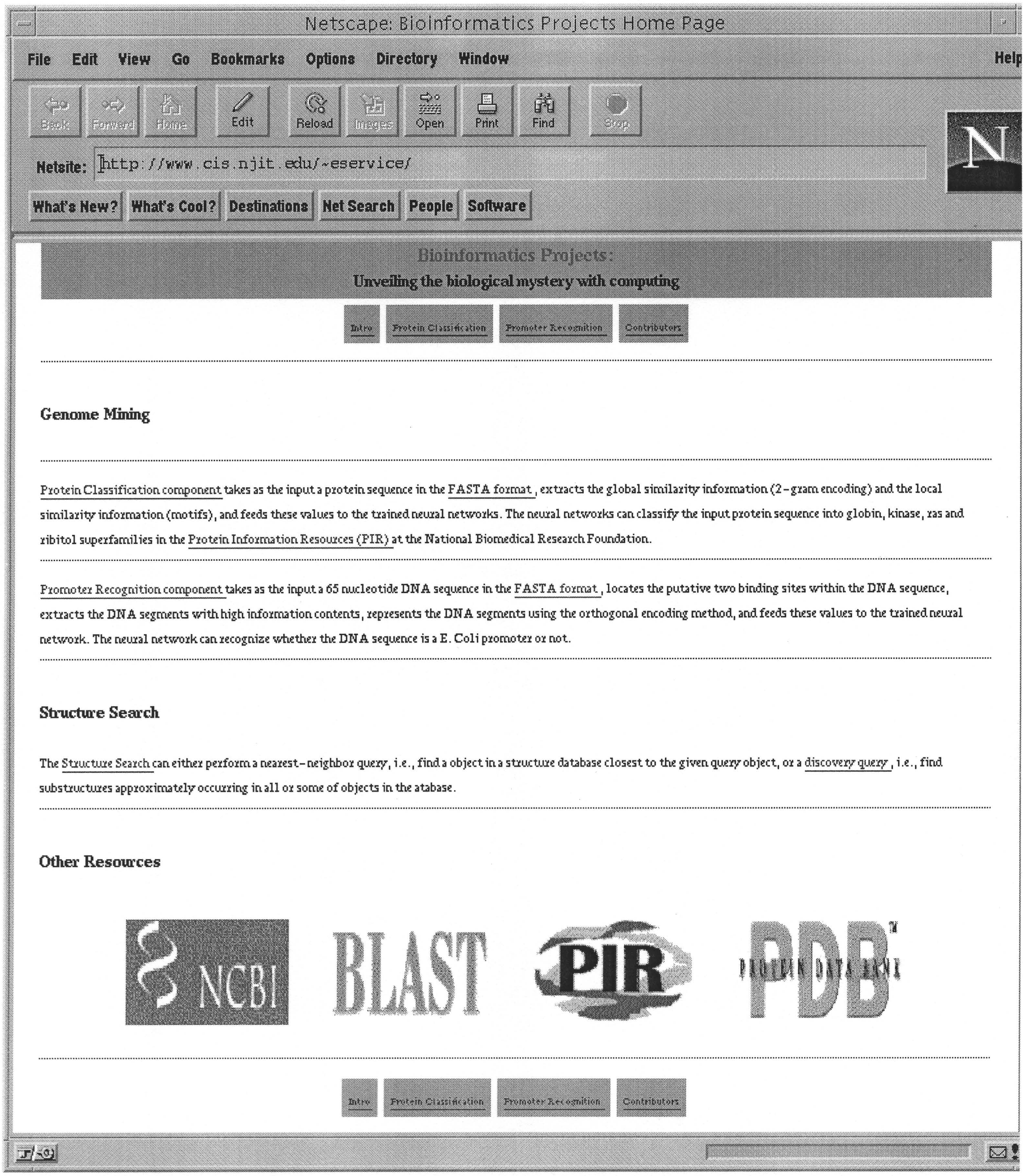


Figure 5.1 The architecture of the genome mining tool.

appropriate software module is called. The result is returned to the web server. The web server eventually sends the query result back to the web user (see Figure 5.5 and 5.6).



http://www.cis.njit.edu/~eservice/

Bioinformatics Projects:

Unveiling the biological mystery with computing

Genome Mining

Protein Classification component takes as the input a protein sequence in the FASTA format, extracts the global similarity information (2-gram encoding) and the local similarity information (motifs), and feeds these values to the trained neural networks. The neural networks can classify the input protein sequence into globin, kinase, ras and ribitol superfamilies in the Protein Information Resources (PIR) at the National Biomedical Research Foundation.

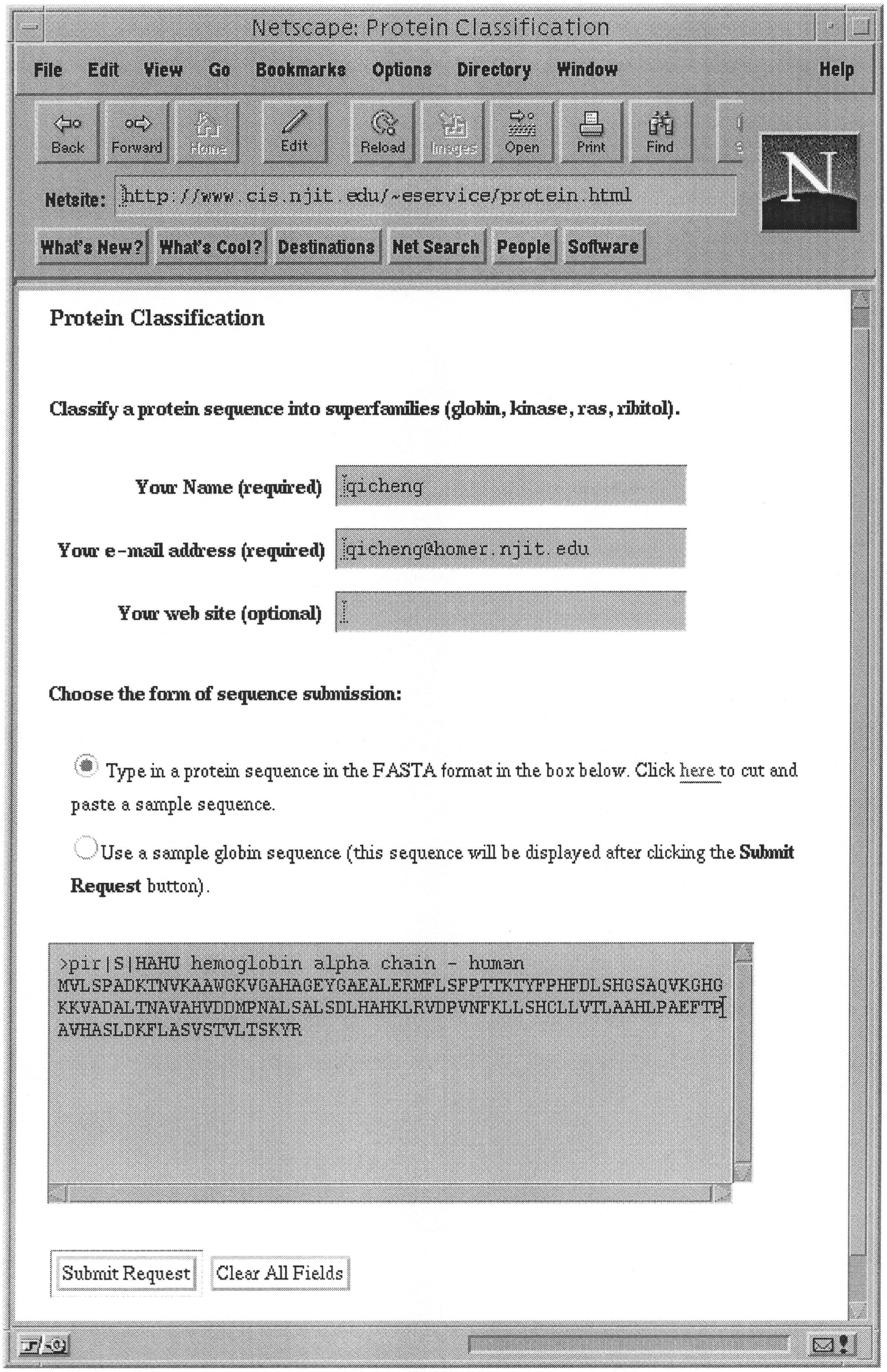
Promoter Recognition component takes as the input a 65 nucleotide DNA sequence in the FASTA format, locates the putative two binding sites within the DNA sequence, extracts the DNA segments with high information contents, represents the DNA segments using the orthogonal encoding method, and feeds these values to the trained neural network. The neural network can recognize whether the DNA sequence is a E. Coli promoter or not.

Structure Search

The Structure Search can either perform a nearest-neighbor query, i.e., find a object in a structure database closest to the given query object, or a discovery query, i.e., find substructures approximately occurring in all or some of objects in the atabase.

Other Resources

Figure 5.2 The main page of the genome mining tool.


Netscape: Protein Classification
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find
Netsite:
What's New? What's Cool? Destinations Net Search People Software

http://www.cis.njit.edu/~eservice/protein.html

Protein Classification

Classify a protein sequence into superfamilies (globin, kinase, ras, ribitol).

Your Name (required)	qicheng
Your e-mail address (required)	qicheng@homer.njit.edu
Your web site (optional)	

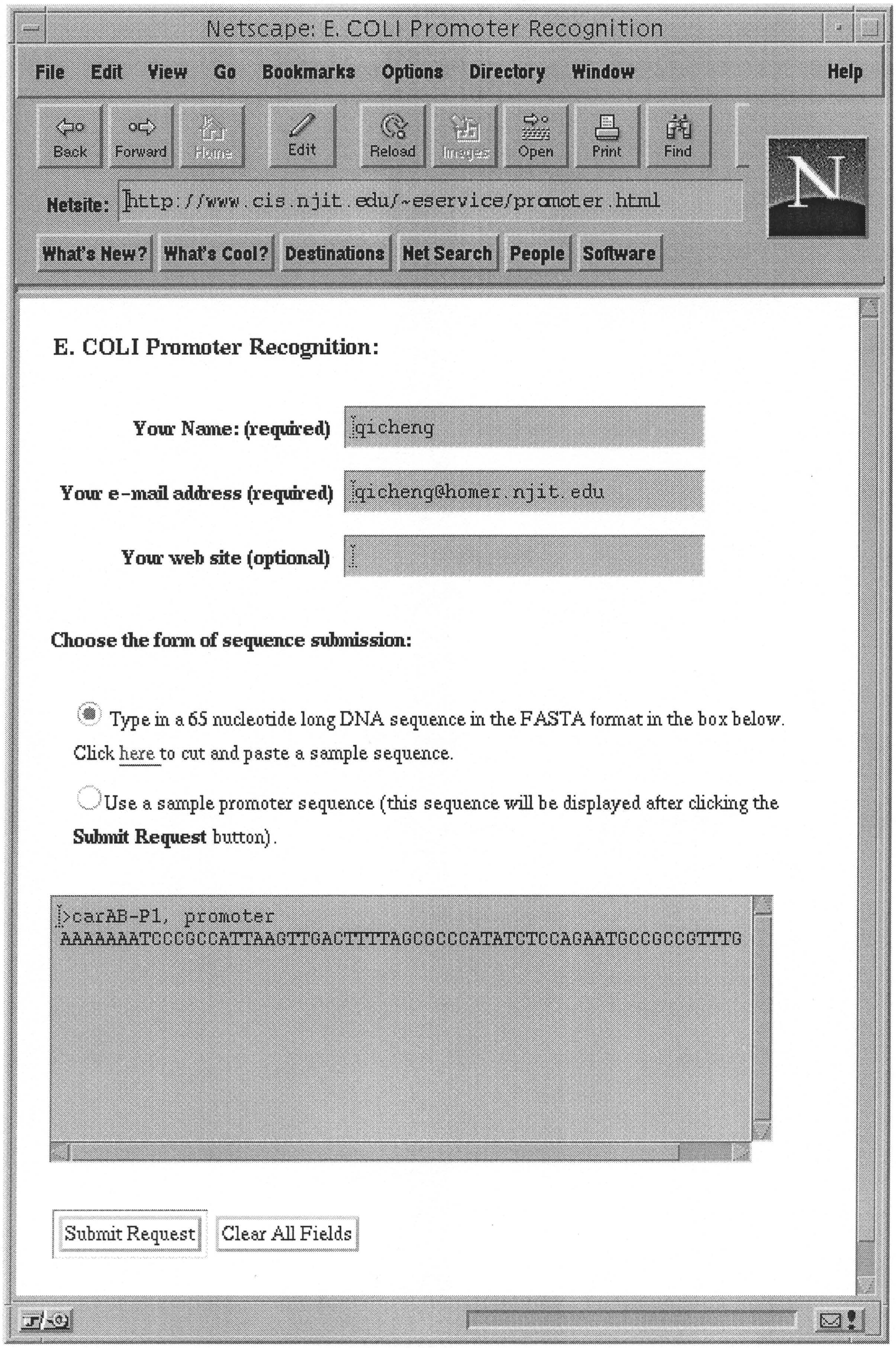
Choose the form of sequence submission:

[✓] Type in a protein sequence in the FASTA format in the box below. Click here to cut and paste a sample sequence.

Use a sample globin sequence (this sequence will be displayed after clicking the **Submit Request** button).

```
>pir|S|HAHU hemoglobin alpha chain - human
MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHG
KKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTP
AVHASLDKFLASVSTVLTSKYR
```

Figure 5.3 The interface for submitting a protein query sequence.


Netscape: E. COLI Promoter Recognition
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find
What's New? What's Cool? Destinations Net Search People Software

Netsite: http://www.cis.njit.edu/~eservice/promoter.html

E. COLI Promoter Recognition:

Your Name: (required) qicheng

Your e-mail address (required) qicheng@homer.njit.edu

Your web site (optional)

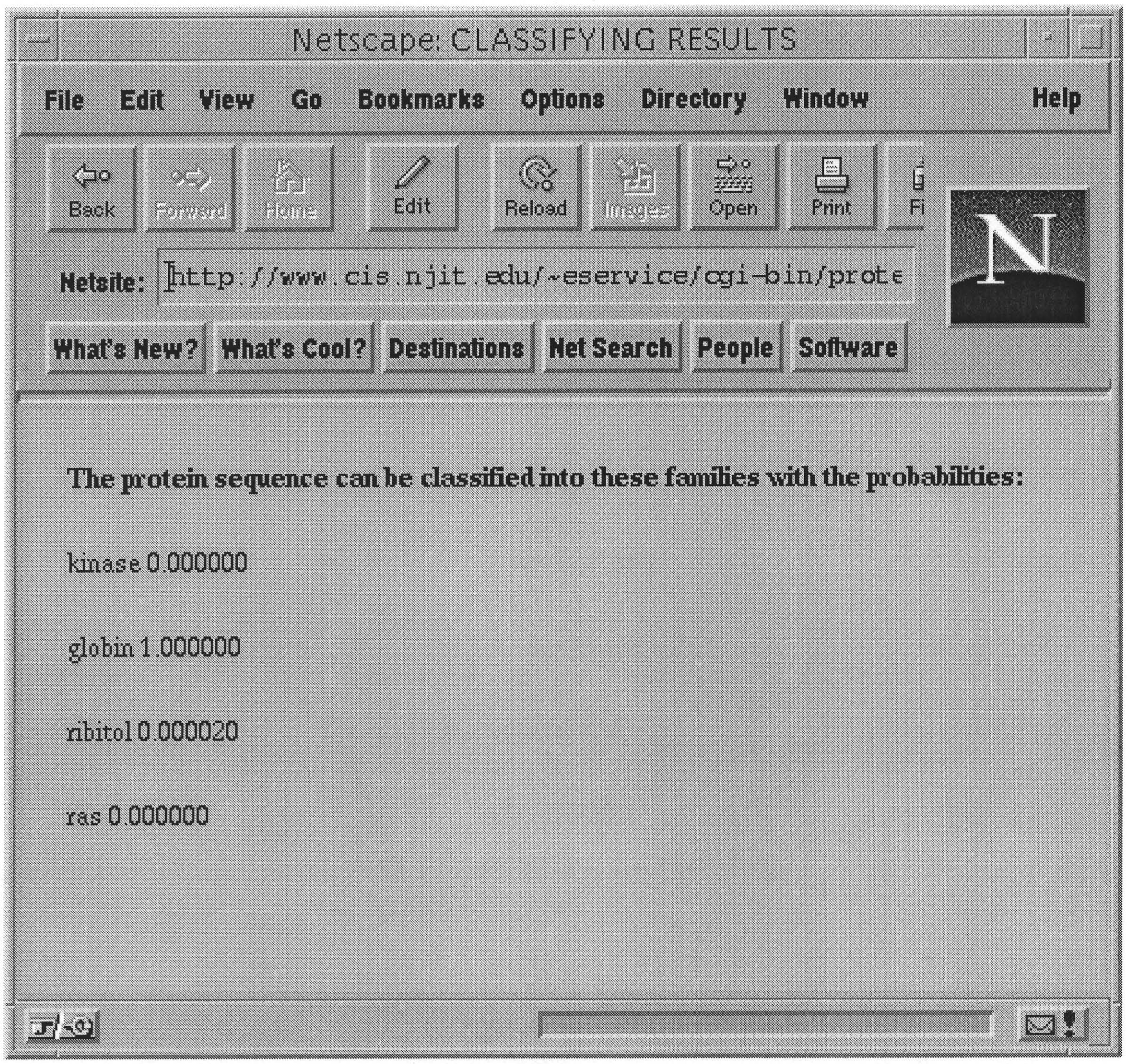
Choose the form of sequence submission:

[✓ Type in a 65 nucleotide long DNA sequence in the FASTA format in the box below.] Click here to cut and paste a sample sequence.

Use a sample promoter sequence (this sequence will be displayed after clicking the **Submit Request** button).

```
>carAB-P1, promoter
AAAAAAATCCCGCCATTAAGTTGACTTTTAGCGCCCATATCTCCAGAATGCCGCCGTTTG
```

Figure 5.4 The interface for submitting a promoter query sequence.



http://www.cis.njit.edu/~eservice/cgi-bin/prote

The protein sequence can be classified into these families with the probabilities:

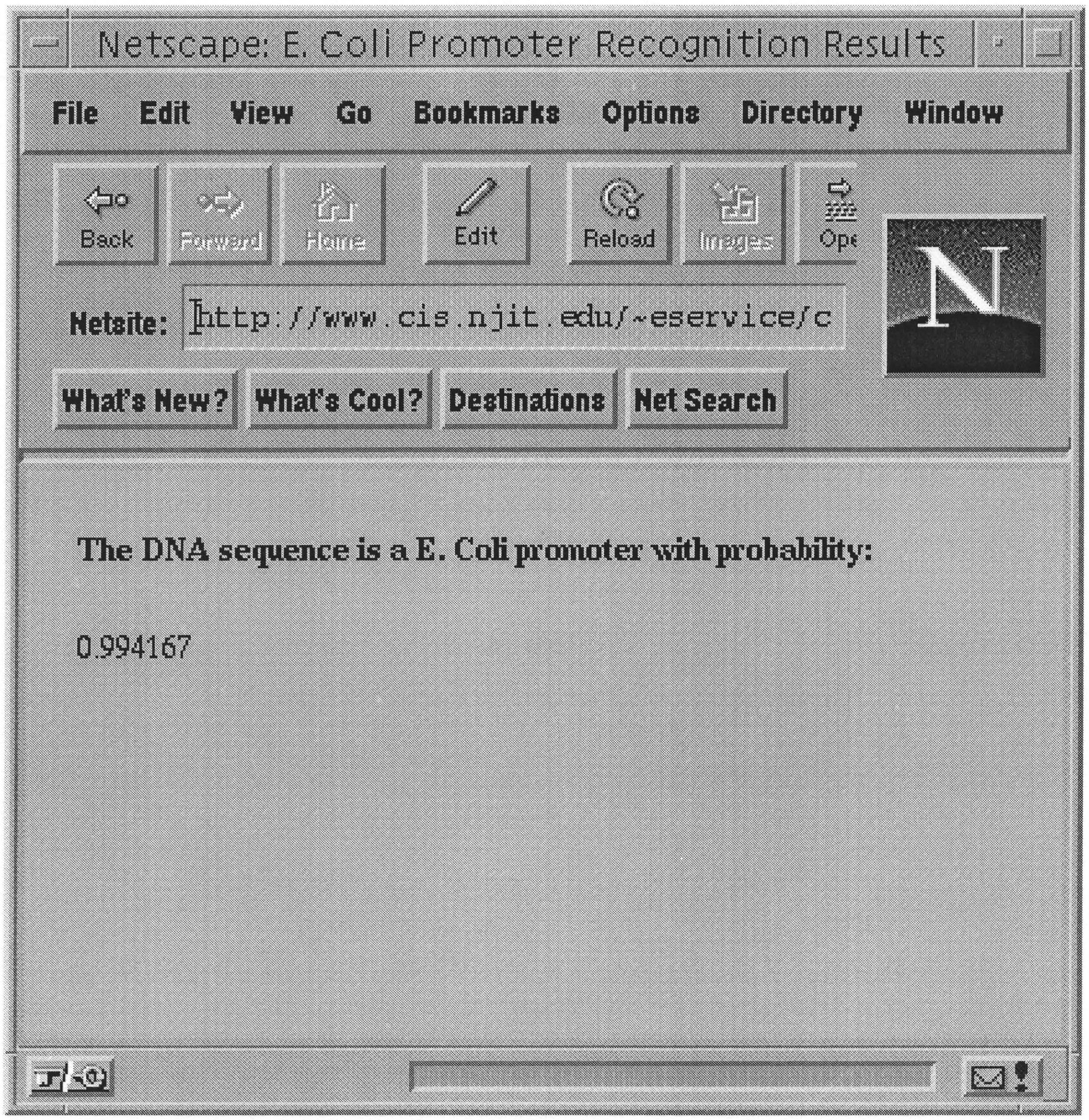
kinase 0.000000

globin 1.000000

ribitol 0.000020

ras 0.000000

Figure 5.5 The classification results of a protein query sequence.



```
http://www.cis.njit.edu/~eservice/c
```

The DNA sequence is a E. Coli promoter with probability:

0.994167

Figure 5.6 The recognition result of a promoter query sequence.

CHAPTER 6
CONCLUSIONS

In this dissertation, we have presented new techniques for two biological sequence classification problems. In the protein sequence classification problem, we have presented a Bayesian neural network approach to classifying protein sequences. The main contributions of our protein sequence classification method include (1) the development of new algorithms for extracting the global similarity and the local similarity from the sequences that are used as input features of the Bayesian neural network; (2) the development of new measures for evaluating the significance of 2-gram patterns and frequently occurring motifs used in classifying the sequences; (3) experimental studies in which we compare the performance of the proposed BNN classifier with two other classifiers, namely BLAST and SAM, on four different superfamilies of sequences in the PIR protein database.

In the promoter recognition problem, we have proposed a new technique to recognize E. Coli promoter sequences. We first use a Bayesian MAP EM algorithm to locate the binding sites of the promoter sequences. We then align promoters with respect to the two binding sites as well as the transcriptional start site. Significant features within promoters are extracted according to their information contents. These features are then represented by the orthogonal encoding method and fed to neural networks. Empirical study shows that the proposed approach achieves better results when comparing with existing methods on the same dataset. This happens because our EM algorithm is able to precisely locate the binding sites of the promoter sequences.

In the future, we will combine neural networks and our recently developed graph matching software to compare 3D protein structures. As the size of the Protein Data Bank becomes larger and larger (over 7000), it is important to develop new protein structure classification algorithms [63].

As protein structure comparison is a computationally intensive task, we propose a two phase classification process. The first phase is the protein primary structure (sequence) classification process. This phase is a classification process at a coarse granularity, as described in Chapter 3. The first phase can significantly reduce the computational time by eliminating unnecessary comparisons. The result of the first phase is a set of possible candidates. The second phase of the classification process is the protein 3D structure comparison process, where we compare the query protein structure with a few possible candidates. Thus, the second phase focuses on a refined granularity. We plan to study this research problem, conducting structural classification and prediction in the future.

APPENDIX A

Program Listing

This appendix includes the source code for the genome mining tool.

```
/***********************************************************************/
/*                                                                     */
/*    (c) Copyright 2000                                               */
/*    All rights reserved                                              */
/*    Programs written by Qicheng Ma (NJIT)                            */
/*    RA in the group of Jason T. L. Wang (New Jersey Institute        */
/*    of Technology) and Dennis Shasha (New York University)           */
/*                                                                     */
/*    Permission to use, copy, modify, and distribute this             */
/*    software and its documentation for any purpose and without       */
/*    fee is hereby granted, provided that this copyright              */
/*    notice appears in all copies.  Programmer(s) makes no            */
/*    representations about the suitability of this                    */
/*    software for any purpose.  It is provided "as is" without        */
/*    express or implied warranty.                                     */
/*                                                                     */
/***********************************************************************/

//the format of the input file is assumed to be FASTA

#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <string.h>
#include <stdio.h>              // for sprintf()
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#include <sys/time.h>
#define imin(a,b)       (((a)<(b))?(a):(b))
#define imax(a,b)       (((a)<(b))?(b):(a))

#define Max_len 3000
#define MAXLGH Max_len
#define MAX_MOTIF_LEN 200
#define MIN_SIMILARITY -1
#define INVALID  -1
#define TOTAL 751
#define EQUAL_SEQUENCES_FASTA_SCORE 918
#define ROUND 10
```

```
struct sample {
 char  *string;
 int actual_length;
 double two_gram[22][20]; /* include 20*20 2-gram of aa and 6*6 2-gram ee */
 int flag; //flag=1 means that it is in the sample set
 int training_set_flag;//flag=1 means in training set, =0 means in the test set
 double *selected_two_gram_features_values;
 double nearest_distance_from_cluster_centers;
 double correlation_coefficient;// correlation_coefficient with average frequency , pos_frequencies.
 double motif_score;
};

struct two_gram_md {
 double md;
 int aa_i;
 int aa_j;
};

struct component { //for each component of a cluster
 int index; // index to sequence array
 component *next;
};

struct score_element {
 int value;
 int cluster_ID;
 int row_sequence_number;
 int column_sequence_number;//the row sequence and column sequence that
                        // contribue to this score
 score_element *next;
};

struct cluster { //for each cluster
 int cluster_ID;
 component  *elements;
 score_element  *resemblance_vector;
 int max_similarity_column;
 int max_similarity;
 int max_row_sequence;
 int max_column_sequence;
 int center_sequence_number; // a center sequence is the sequence that
                      // has the highest counts value in a cluster
 cluster *next;
};
```

```
struct motif{
  float compression;
  int distance;
  int occurrence;
  char *motif;
};

sample *read_file(char *, int ,int &,double);
sample *read_file2(char *, int ,int &,double);
sample *read_file3(char *, int ,int &,int);
int filter(sample *,int,char *);
int filter2(sample *,int,char *,float);
int filter3(sample *,int,char *,float);
int filter4(sample *,int,char *);
int count_2_gram(sample *,int, double [][20],double);
int mapping(char);
int mapping2(char);
void calculate_dispersion(sample *,int,double [][20],double [][20]);
void mahalonobis_distance(double [][20],double [][20],double [][20],double [][20],double [][20]);
void calculate_correlation_coefficient(sample *,int ,double [][20]);
void sort(double [][20],two_gram_md *);
void get_discriminating_frequencies(sample *,int ,two_gram_md *,int);
void relief(sample *,int , sample *, int ,double [][20]);
int find_near_hit(sample *,int , int );
int find_near_miss(sample *,int , sample *, int );
double distance(sample *,int ,sample *,int );
void generate_pca_data(sample *,int ,sample *,int );
int read_pca_data(sample *,int,sample *,int,double [][20],double [][20]);
void get_motif_score(sample *,int,motif *,int);
short match(char *,char *);
void  classify_by_motif(sample *,int,sample *,int);
motif *read_in_motif(char *,int &);
void scale_feature_values(sample *,int,sample *,int ,int );
void  write_feature_values(sample *,int ,sample *,int, int );
double randomNumber(void);
long randomSeed();
char *OUTPUT;
long unsigned int nextrandom;  /* seed for random number generator */

main(int argc,char *argv[])
{
  sample *pos_samples,*neg_samples;
  int pos_num,neg_num,actual_pos,actual_neg,i,j;
  int number_of_two_grams, number_of_clusters,number_of_motifs;
  double ratio,pos_frequencies[22][20],neg_frequencies[22][20]; /* mean */
  double md[22][20],pos_dispersion[22][20],neg_dispersion[22][20];/*deviation*/
  double training_test_ratio;
  two_gram_md  one_dimension[440];
  cluster *all_clusters;
  motif *motifs;
  int *counts;/* record the connection times of each sequence */
  int md_or_relief_or_FK;
  int min_len,max_len;
  int show_feature;
  ofstream output;
  if(argc!=15){
    cout<<"Usage: "<<argv[0]<<" positive_data_file positive_data_num  negative_data_file
negative_data_num  sample_ratio training_test_ratio number_of_two_grams number_of_clusters
motifs_file number_of_motifs OUTPUT md_or_relief_or_FK seed show_feature "<<endl;
    return -1;
  }
  pos_num=atoi(argv[2]);
  neg_num=atoi(argv[4]);
  ratio=atof(argv[5]);
  training_test_ratio=atof(argv[6]);
  number_of_two_grams=atoi(argv[7]);
  number_of_clusters=atoi(argv[8]);
  number_of_motifs=atoi(argv[10]);
  OUTPUT=argv[11];
  md_or_relief_or_FK=atoi(argv[12]);
  nextrandom=atoi(argv[13]);
  show_feature=atoi(argv[14]);
  pos_samples=read_file3(argv[1],pos_num,actual_pos,1);
  neg_samples=read_file3(argv[3],neg_num,actual_neg,0);

  if(!pos_samples||!neg_samples)
  {
    cout<<"Error in reading the input file"<<endl;
    return -1;
  }
  filter2(pos_samples,actual_pos,argv[1],training_test_ratio);
  filter2(neg_samples,actual_neg,argv[3],training_test_ratio);
  filter3(pos_samples,actual_pos,argv[1],training_test_ratio);
  filter3(neg_samples,actual_neg,argv[3],training_test_ratio);


  count_2_gram(pos_samples,actual_pos,pos_frequencies,ratio);
```

```
count_2_gram(neg_samples,actual_neg,neg_frequencies,ratio);
if (md_or_relief_or_FK==0) {
  calculate_dispersion(pos_samples,actual_pos,pos_frequencies,pos_dispersion);
  calculate_dispersion(neg_samples,actual_neg,neg_frequencies,neg_dispersion);
  mahalonobis_distance(pos_frequencies,neg_frequencies,pos_dispersion,neg_dispersion,md);
  sort(md,one_dimension);

  if(show_feature==1) {
    output.open("feature_file",ios::out);
    if(!output) {
      cerr<<"Error! Can not open feature file"<<endl;
      return 0;
    }
    for(i=0;i<number_of_two_grams;i++)
      output<<one_dimension[i].aa_i<<" "<<one_dimension[i].aa_j<<endl;
    for(i=0;i<22;i++)
      if(i!=21)
        for(j=0;j<20;j++) {
          output<<pos_frequencies[i][j]<<" ";
        }
      else
        for(j=0;j<16;j++) {
          output<<pos_frequencies[i][j]<<" ";
        }
     output<<endl;
     output.close();
     return 0;
  }

get_discriminating_frequencies(pos_samples,actual_pos,one_dimension,number_of_two_grams);

get_discriminating_frequencies(neg_samples,actual_neg,one_dimension,number_of_two_grams);

}
else if (md_or_relief_or_FK==1) {
  relief(pos_samples,actual_pos,neg_samples,actual_pos,md);
  sort(md,one_dimension);

get_discriminating_frequencies(pos_samples,actual_pos,one_dimension,number_of_two_grams);

get_discriminating_frequencies(neg_samples,actual_neg,one_dimension,number_of_two_grams);

}
```

```
  else {
   generate_pca_data(pos_samples,actual_pos,neg_samples,actual_neg);
    if
(read_pca_data(pos_samples,actual_pos,neg_samples,actual_neg,pos_frequencies,neg_frequencies)
<0)
      return 0;
    calculate_dispersion(pos_samples,actual_pos,pos_frequencies,pos_dispersion);
    calculate_dispersion(neg_samples,actual_neg,neg_frequencies,neg_dispersion);
    mahalonobis_distance(pos_frequencies,neg_frequencies,pos_dispersion,neg_dispersion,md);
    sort(md,one_dimension);

get_discriminating_frequencies(pos_samples,actual_pos,one_dimension,number_of_two_grams);

get_discriminating_frequencies(neg_samples,actual_neg,one_dimension,number_of_two_grams);
  }

  calculate_correlation_coefficient(pos_samples,actual_pos,pos_frequencies);
  calculate_correlation_coefficient(neg_samples,actual_neg,pos_frequencies);
  motifs=read_in_motif(argv[9],number_of_motifs);
  get_motif_score(pos_samples,actual_pos,motifs,number_of_motifs);
  get_motif_score(neg_samples,actual_neg,motifs,number_of_motifs);
  write_feature_values(pos_samples,actual_pos,neg_samples,actual_neg,number_of_two_grams);
  return 0;
}

 sample *read_file(char *file_name,int total,int &actual_total,double ratio)

// the fuction is to read total samples
//the format of the file is assumed to be FASTA
// return NULL if there is an error
//if the current sequence contain illegal amino acid, the sequence is ignored

{
  ifstream infile(file_name);
  char buffer[Max_len],ch;
  int i,count,error_flag; // count the length of the current sample
  sample *samples;
  samples= new  sample [total];
  if(!infile) {
    cout<<"Can not open sample file"<<endl;
    return NULL;
  }
```

```
if (!samples)
{
 cout<<"Can not allocate enough memory"<<endl;
 return NULL;
}
ch=infile.get();
i=0;
while (ch!=EOF) {
 if(ch!='>') // assume that the first character is '>'
 {
  cout<<"The format of the sample file is not FASTA format"<<endl;
  return NULL;
 }
 while((ch=infile.get())!='\n') // skip  the first line of a sample
  ;
 count=-1;
 error_flag=0;
 while((ch=infile.get())!='>'&&ch!=EOF) {
  if (ch==' '|ch=='\n'|ch=='\t')  //skip new line character
   continue;
  if(mapping(ch)==-1)
     error_flag=1;
  else  {
   count++;
   if(count>Max_len-1) {
    cout<<"Sample is longer than Max_len"<<endl;
    return NULL;
   }
   buffer[count]=ch;
  }
 }
 if(error_flag)
  continue;
 samples[i].string=new char [count+2];
 if (!samples[i].string)
 {
  cout<<"Can not allocate enough memory"<<endl;
  return NULL;
 }
 strncpy(samples[i].string,buffer,count+1);
 samples[i].string[count+1]='\0';
 samples[i].actual_length=count+1;
```

```
/*
   if(randomNumber()<=ratio)
     samples[i].training_set_flag=1;
   else
     samples[i].training_set_flag=0;
*/
   samples[i].training_set_flag=((int) randomNumber()*10*ROUND ) /10;
   i++;
  }
  actual_total=i;
  infile.close();
  return samples;
}
 sample *read_file2(char *file_name,int total,int &actual_total,double ratio)

// the fuction is to read total samples, BUT illegal AA are ignored.
//the format of the file is assumed to be FASTA
// return NULL if there is an error
//if the current sequence contain illegal amino acid, the sequence is ignored

{
  ifstream infile(file_name);
  char buffer[Max_len],ch;
  int i,count,error_flag; // count the length of the current sample
  int training_count=0;
  sample *samples;
  samples= new  sample [total];
  if(!infile) {
   cout<<"Can not open sample file"<<endl;
   return NULL;
  }

  if (!samples)
  {
   cout<<"Can not allocate enough memory"<<endl;
   return NULL;
  }
  ch=infile.get();
  i=0;
  while (ch!=EOF) {
   if(ch!='>') // assume that the first character is '>'
   {
     cout<<"The format of the sample file is not FASTA format"<<endl;
```

```
      return NULL;
    }
    while((ch=infile.get())!='\n') // skip  the first line of a sample
      ;
    count=-1;
    error_flag=0;
    while((ch=infile.get())!='>'&&ch!=EOF) {
      if (ch==' '|ch=='\n'|ch=='\t')  //skip new line character
        continue;
      if(mapping(ch)==-1)
           error_flag=1;
      else  {
        count++;
        if(count>Max_len-1) {
          cout<<"Sample is longer than Max_len"<<endl;
          return NULL;
        }
        buffer[count]=ch;
      }
    }
/*
    if(error_flag)
      continue;
*/
    samples[i].string=new char [count+2];
    if (!samples[i].string)
    {
      cout<<"Can not allocate enough memory"<<endl;
      return NULL;
    }
    strncpy(samples[i].string,buffer,count+1);
    samples[i].string[count+1]='\0';
    samples[i].actual_length=count+1;
    if(randomNumber()<=ratio){
      samples[i].training_set_flag=1;
      training_count++;
    }
    else
      samples[i].training_set_flag=0;
    i++;
  }
  actual_total=i;
  infile.close();
```

```
 return samples;
}
 sample *read_file3(char *file_name,int total,int &actual_total, int pos_neg_flag)

// Read Cathy's data
// the fuction is to read total samples, BUT illegal AA are ignored.
//the format of the file is assumed to be FASTA
// return NULL if there is an error
//if the current sequence contain illegal amino acid, the sequence is ignored

{
 ifstream infile(file_name);
 char buffer[Max_len],ch;
 int i,count,error_flag; // count the length of the current sample
 int training_count=0;
 sample *samples;
 samples= new  sample [total];
 if(!infile) {
  cout<<"Can not open sample file"<<endl;
  return NULL;
 }
 if (!samples)
 {
  cout<<"Can not allocate enough memory"<<endl;
  return NULL;
 }
 ch=infile.get();
 i=0;
 while (ch!=EOF) {
  if(ch!='>') // assume that the first character is '>'
  {
   cout<<"The format of the sample file is not FASTA format"<<endl;
   return NULL;
  }
  while((ch=infile.get())!='\n') // skip  the first line of a sample
   ;
  count=-1;
  error_flag=0;
  while((ch=infile.get())!='>'&&ch!=EOF) {
   if (ch==' '|ch=='\n'|ch=='\t')  //skip new line character
    continue;
   if(mapping(ch)==-1)
      error_flag=1;
      else  {
        count++;
        if(count>Max_len-1) {
          cout<<"Sample is longer than Max_len"<<endl;
          return NULL;
        }
        buffer[count]=ch;
      }
    }
/*
    if(error_flag)
      continue;
*/
    samples[i].string=new char [count+2];
    if (!samples[i].string)
    {
      cout<<"Can not allocate enough memory"<<endl;
      return NULL;
    }
    strncpy(samples[i].string,buffer,count+1);
    samples[i].string[count+1]='\0';
    samples[i].actual_length=count+1;
    if(pos_neg_flag==1) /* positive */
      if(i<554)
        samples[i].training_set_flag=1;
      else
        samples[i].training_set_flag=0;
    else /* negative */
      if(i<1108)
        samples[i].training_set_flag=1;
      else
        samples[i].training_set_flag=0;
/*
    if(randomNumber()<=ratio){
      samples[i].training_set_flag=1;
      training_count++;
    }
    else
      samples[i].training_set_flag=0;
*/

    i++;
  }
  actual_total=i;
  infile.close();
  return samples;
}
//write to a file the filtered training data
int filter2(sample *samples, int total, char *original_file_name,float training_test_ratio)
{
  int i,j;
  char real_file_name[100];
  ofstream  output;
  sprintf(real_file_name,"%s.training_%.2f",original_file_name,training_test_ratio);
  output.open(real_file_name,ios::out);
  if(!output) {
    cerr<<"Error! Can not open file "<<real_file_name<<" to write."<<endl;
    return -1;
  }
  for(i=0;i<total;i++) {
    if(samples[i].training_set_flag==1) {
      output<<">pir|S|"<<original_file_name<<" training "<<i<<endl;
      for(j=0;j<samples[i].actual_length;j++) {
        if((j+1)%30==0&&j!=0)
            output<<endl;
        output<<samples[i].string[j];
      }
      output<<endl;
    }
  }
  output.close();
  return 0;
}
//write to a file the filtered data
int filter(sample *samples, int total, char *original_file_name)
{
  int i,j;
  char real_file_name[100];
  ofstream  output;
  strcpy(real_file_name,original_file_name);
  strcat(real_file_name,".filtered");//append suffix .filtered
  output.open(real_file_name,ios::out);
  if(!output) {
    cerr<<"Error! Can not open file "<<real_file_name<<" to write."<<endl;
    return -1;
  }
```

```
 for(i=0;i<total;i++) {
    output<<">"<<endl;
    for(j=0;j<samples[i].actual_length;j++) {
     if((j+1)%30==0&&j!=0)
        output<<endl;
     output<<samples[i].string[j];
    }
    output<<endl;
 }
 output.close();
 return 0;
}
//write to a file the filtered test data
int filter3(sample *samples, int total, char *original_file_name,float training_test_ratio)
{
 int i,j;
 char real_file_name[100];
 ofstream  output;
 sprintf(real_file_name,"%s.test_%.2f",original_file_name,training_test_ratio);
 output.open(real_file_name,ios::out);
 if(!output) {
   cerr<<"Error! Can not open file "<<real_file_name<<" to write."<<endl;
   return -1;
 }
 for(i=0;i<total;i++) {
   if(samples[i].training_set_flag==0) {
    output<<">pir|S|"<<original_file_name<<" test "<<i<<endl;
    for(j=0;j<samples[i].actual_length;j++) {
     if((j+1)%30==0&&j!=0)
        output<<endl;
     output<<samples[i].string[j];
    }
    output<<endl;
   }
 }
 output.close();
 return 0;
}
//write to a file the filtered ROUND cross_validation data
int filter4(sample *samples, int total, char *original_file_name)
{
 int i,j,k;
 char real_file_name_t[30],real_file_name_s[30],buffer[5];
 ofstream  output_t,output_s;
```

```
  for(k=0;k<ROUND;k++) {
    strcpy(real_file_name_t,original_file_name);
    sprintf(buffer,"%d",k);
    strcat(real_file_name_t, buffer);
    strcat(real_file_name_t, "training");
    strcpy(real_file_name_s,original_file_name);
    strcat(real_file_name_s, buffer);
    strcat(real_file_name_s, "test");
    output_t.open(real_file_name_t,ios::out);
    output_s.open(real_file_name_s,ios::out);
    if(!output_t||!output_s) {
      cerr<<"Error! Can not open file to write."<<endl;
      return -1;
    }
    for(i=0;i<total;i++) {
      if(samples[i].training_set_flag!=k) {
        output_t<<">"<<endl;
        for(j=0;j<samples[i].actual_length;j++) {
          if((j+1)%30==0&&j!=0)
            output_t<<endl;
          output_t<<samples[i].string[j];
        }
        output_t<<endl;
      }
      else {
        output_s<<">"<<endl;
        for(j=0;j<samples[i].actual_length;j++) {
          if((j+1)%30==0&&j!=0)
            output_s<<endl;
          output_s<<samples[i].string[j];
        }
        output_s<<endl;
      }

    }
    output_t.close();
    output_s.close();
  }
  return 0;
}
// calculate 2-gram for each sequence in the samples and
// calculate frenquencies of each 2_gram in the set
// return -1 if an error is encountered
```

```
int count_2_gram(sample *samples,int total,double frequencies[][20],double ratio)
{
 int i,j,k,n,flag,training_flag,total_length=0;
 int ei,ej;
 char ch1;
 for(i=0;i<22;i++)
  for(j=0;j<20;j++)
   frequencies[i][j]=0;
 flag=0;
 for(k=0;k<total;k++) {
  for(i=0;i<22;i++) {
   for(j=0;j<20;j++)
      samples[k].two_gram[i][j]=0;
  }
/*
  if(samples[k].training_set_flag==1)
   if(randomNumber()<ratio)
*/
    flag=1;
  samples[k].flag=flag;
  ch1=samples[k].string[0];
  i=mapping(ch1);
  ei=mapping2(ch1);
  for(n=0;n<samples[k].actual_length-1;n++) {
   ch1=samples[k].string[n+1];
   j=mapping(ch1);
   ej=mapping2(ch1);
   if(j==-1) {
    cerr<<"Can not count 2 gram successfully in the "<<k+1<<"th sequence. "<<endl;
    return -1;
   }
   samples[k].two_gram[i][j]++;
   samples[k].two_gram[(ei*6+ej)/20+20][(ei*6+ej)%20]++;
   if(flag) {
      frequencies[i][j]++;
    frequencies[(ei*6+ej)/20+20][(ei*6+ej)%20]++;
   }
   i=j;
   ei=ej;
  }
  for(i=0;i<22;i++){
   for(j=0;j<20;j++)
      samples[k].two_gram[i][j]/=samples[k].actual_length-1;
  }
```

```
    if(flag)
      total_length+=samples[k].actual_length-1;
    flag=0;
  }
  for(i=0;i<22;i++)
    for(j=0;j<20;j++)
      frequencies[i][j]/=total_length;
  return 1;
}
//calculate the the square of the dispersion for each 2_grams
void calculate_dispersion(sample *samples, int total, double frequencies[][20],double
dispersion[][20])
{
  int i,j,k,count;
  for(i=0;i<22;i++)
    for(j=0;j<20;j++)
      dispersion[i][j]=0;
  for(i=0;i<22;i++)
    for(j=0;j<20;j++) {
      count=0;
      for(k=0;k<total;k++) {
          if(samples[k].flag) {
            count++;
            dispersion[i][j]+=(samples[k].two_gram[i][j]-
frequencies[i][j])*(samples[k].two_gram[i][j]-frequencies[i][j]);
        }
      }
      dispersion[i][j]/=count;
  }
  return;
}
//calculate mahalonobis distance MD
void mahalonobis_distance(double pos_frequencies[][20],double neg_frequencies[][20],double
pos_dispersion[][20],double neg_dispersion[][20],double md[][20])
{
  int i,j;
  for(i=0;i<22;i++)
    if(i!=21)
      for(j=0;j<20;j++)
        md[i][j]=(pos_frequencies[i][j]-neg_frequencies[i][j])*(pos_frequencies[i][j]-
neg_frequencies[i][j])/(pos_dispersion[i][j]+neg_dispersion[i][j]);
    else
      for(j=0;j<16;j++)
        md[i][j]=(pos_frequencies[i][j]-neg_frequencies[i][j])*(pos_frequencies[i][j]-
neg_frequencies[i][j])/(pos_dispersion[i][j]+neg_dispersion[i][j]);

 return;
}
void calculate_correlation_coefficient(sample *samples,int total,double pos_frequencies[][20])
{
 int i,j,k;
 double x, y;
 double sigma_xy,sigma_x,sigma_y,sigma_x_square,sigma_y_square;
 for(k=0;k<total;k++){
   sigma_xy=sigma_x=sigma_y=sigma_x_square=sigma_y_square=0;
   samples[k].correlation_coefficient=0;
   for(i=0;i<22;i++)
     if(i!=21)
       for(j=0;j<20;j++) {
         x=samples[k].two_gram[i][j];
         y=pos_frequencies[i][j];
         sigma_xy+=x*y;
         sigma_x+=x;
         sigma_y+=y;
         sigma_x_square+=x*x;
             sigma_y_square+=y*y;
       }
     else
       for(j=0;j<16;j++) {
         x=samples[k].two_gram[i][j];
         y=pos_frequencies[i][j];
         sigma_xy+=x*y;
         sigma_x+=x;
         sigma_y+=y;
         sigma_x_square+=x*x;
             sigma_y_square+=y*y;
       }

   samples[k].correlation_coefficient=(436*sigma_xy-
sigma_x*sigma_y)/(sqrt(436*sigma_x_square-sigma_x*sigma_x)*sqrt(436*sigma_y_square-
sigma_y*sigma_y));
//   cout<<samples[k].correlation_coefficient<<endl;
  }
 return;
}
// map a amino acid to a integer in [0,19]
// return -1 in case of illegal amino acid
```

```
int mapping(char aa)
{
 if(aa=='A')
   return 0;
 else if (aa=='R')
   return 1;
 else if (aa=='N')
   return 2;
 else if (aa=='D')
   return 3;
 else if (aa=='C')
   return 4;
 else if (aa=='E')
   return 5;
 else if (aa=='Q')
   return 6;
 else if (aa=='G')
   return 7;
 else if (aa=='H')
   return 8;
 else if (aa=='I')
   return 9;
 else if (aa=='L')
   return 10;
 else if (aa=='K')
   return 11;
 else if (aa=='M')
   return 12;
 else if (aa=='F')
   return 13;
 else if (aa=='P')
   return 14;
 else if (aa=='S')
   return 15;
 else if (aa=='T')
   return 16;
 else if (aa=='W')
   return 17;
 else if (aa=='Y')
   return 18;
 else if (aa=='V')
   return 19;
```

```
  else { //error
    return -1;
  }

}
// map a amino acid to a integer in [0,5] for 6 exchange groups
// return -1 in case of illegal amino acid
int mapping2(char aa)
{
  if(aa=='H'||aa=='R'||aa=='K')
    return 0;
  else if (aa=='D'||aa=='E'||aa=='N'||aa=='Q')
    return 1;
  else if (aa=='C')
    return 2;
  else if (aa=='S'||aa=='T'||aa=='P'||aa=='A'||aa=='G')
    return 3;
  else if (aa=='M'||aa=='I'||aa=='L'||aa=='V')
    return 4;
  else if (aa=='F'||aa=='Y'||aa=='W')
    return 5;
  else { //error
    return -1;
  }

}
void sort(double md[][20],two_gram_md one_dimension[])
{
  int i,j,max_index;
  two_gram_md temp;
  for(i=0;i<22;i++)
    for(j=0;j<20;j++){
      one_dimension[20*i+j].md=md[i][j];
      one_dimension[20*i+j].aa_i=i;
      one_dimension[20*i+j].aa_j=j;
    }
  for(i=0;i<439;i++) {
    max_index=i;
    for(j=i+1;j<440;j++)
      if(one_dimension[max_index].md<one_dimension[j].md)
         max_index=j;
    if(i!=max_index) {
      temp=one_dimension[i];
      one_dimension[i]=one_dimension[max_index];
      one_dimension[max_index]=temp;
    }
  }
/*
  for(i=0;i<440;i++)
    cout<<one_dimension[i].md<<" "<<one_dimension[i].aa_i<<"
"<<one_dimension[i].aa_j<<endl;
*/
  return ;
}
// print the first n most discriminating frequencies
void get_discriminating_frequencies(sample *samples,int total,two_gram_md one_dimension[],int
n)
{
   int i,k;
   for(k=0;k<total;k++) {
     samples[k].selected_two_gram_features_values=new double [n];
     for(i=0;i<n;i++)

samples[k].selected_two_gram_features_values[i]=samples[k].two_gram[one_dimension[i].aa_i][o
ne_dimension[i].aa_j];
//     free(samples[k].two_gram);
   }
   return;
}
int read_pca_data(sample *pos_samples,int pos_num,sample *neg_samples,int neg_num,double
pos_frequencies[][20],double neg_frequencies[][20])
{
   int i,k,j;
   char lookahead[80];
   double previous_value;
   ifstream ptd,psd,ntd,nsd;
   ptd.open("ptd1.dat17",ios::in);
   psd.open("psd1.dat17",ios::in);
   ntd.open("ntd1.dat17",ios::in);
   nsd.open("nsd1.dat17",ios::in);
   if(!ptd||!psd||!ntd||!nsd) {
     cout<<"Can not open file"<<endl;
     return -1;
   }
   for(i=0;i<22;i++)
     for(j=0;j<20;j++) {
```

```
    pos_frequencies[i][j]=0;
    neg_frequencies[i][j]=0;
  }
  for(k=0;k<pos_num;k++) {
   if(pos_samples[k].training_set_flag==1) {
    ptd.getline(lookahead,80); // get rid of "{"
    if(strstr(lookahead,"{")==NULL) {
     cout<<"begining of Input data format error"<<" pos "<<k<<endl;
     return -1;
    }
    for(j=0;j<436;j++) {
     ptd.getline(lookahead,80);
     if(strstr(lookahead," 10")!=NULL&&j>0) {
      j--;
      pos_samples[k].two_gram[j/20][j%20]=0;
      pos_frequencies[j/20][j%20]-=previous_value;
     }
     else {
      previous_value=atof(lookahead);
      pos_samples[k].two_gram[j/20][j%20]=previous_value;
      pos_frequencies[j/20][j%20]+=previous_value;
     }
    }
    ptd.getline(lookahead,80); // get rid of "}"
    if(strstr(lookahead,"}")==NULL) {
     cout<<"End of Input data format error"<<" pos "<<k<<endl;
     return -1;
    }
   }
   else {
    psd.getline(lookahead,80); // get rid of "{"
    if(strstr(lookahead,"{")==NULL) {
     cout<<"begining of Input data format error"<<" pos "<<k<<endl;
     return -1;
    }
    for(j=0;j<436;j++) {
     psd.getline(lookahead,80);
     if(strstr(lookahead," 10")!=NULL&&j>0) {
      j--;
      pos_samples[k].two_gram[j/20][j%20]=0;
      pos_frequencies[j/20][j%20]-=previous_value;
     }
     else{
        previous_value=atof(lookahead);
        pos_samples[k].two_gram[j/20][j%20]=previous_value;
        pos_frequencies[j/20][j%20]+=previous_value;
          }
    }
    psd.getline(lookahead,80); // get rid of "{"
    if(strstr(lookahead,"}")==NULL) {
      cout<<"End of Input data format error"<<" pos "<<k<<endl;
      return -1;
    }
  }
 // free(pos_samples[k].two_gram);
}
for(k=0;k<neg_num;k++) {
  if(neg_samples[k].training_set_flag==1)  {
    ntd.getline(lookahead,80); // get rid of "{"
    if(strstr(lookahead,"{")==NULL) {
      cout<<"begining of Input data format error"<<" neg "<<k<<endl;
      return -1;
    }
    for(j=0;j<436;j++) {
      ntd.getline(lookahead,80);
      if(strstr(lookahead," 10")!=NULL&&j>0) {
        j--;
        neg_samples[k].two_gram[j/20][j%20]=0;
        neg_frequencies[j/20][j%20]-=previous_value;
      }
      else {
        previous_value=atof(lookahead);
        neg_samples[k].two_gram[j/20][j%20]=previous_value;
        neg_frequencies[j/20][j%20]+=previous_value;

      }
    }
    ntd.getline(lookahead,80); // get rid of "{"
    if(strstr(lookahead,"}")==NULL) {
      cout<<"End of Input data format error"<<" neg "<<k<<endl;
      return -1;
    }
  }
  else {
    nsd.getline(lookahead,80); // get rid of "{"
    if(strstr(lookahead,"{")==NULL) {
        cout<<"begining of Input data format error"<<" neg "<<k<<endl;
        return -1;
      }
      for(j=0;j<436;j++) {
        nsd.getline(lookahead,80);
        if(strstr(lookahead," 10")!=NULL&&j>0) {
          j--;
          neg_samples[k].two_gram[j/20][j%20]=0;
          neg_frequencies[j/20][j%20]-=previous_value;
        }
        else {
          previous_value=atof(lookahead);
          neg_samples[k].two_gram[j/20][j%20]=previous_value;
          neg_frequencies[j/20][j%20]+=previous_value;
        }
      }
      nsd.getline(lookahead,80);
      if(strstr(lookahead,"}")==NULL) {
        cout<<"End of Input data format error"<<" neg "<<k<<endl;
        return -1;
      }
    }
    //free(neg_samples[k].two_gram);
  }
  for(i=0;i<22;i++)
    for(j=0;j<20;j++) {
      pos_frequencies[i][j]/=pos_num;
      neg_frequencies[i][j]/=neg_num;
  }
  cout<<ptd.eof();
  cout<<psd.eof();
  cout<<ntd.eof();
  cout<<nsd.eof();

  return 0;
}
void relief(sample *pos_samples,int pos_num, sample *neg_samples, int neg_num,double
weight[][20])
{
  int i,j,k,near_hit,near_miss;
  double temp;//weight for each feature[i][j]
  for(i=0;i<22;i++)
    for(j=0;j<20;j++)
      weight[i][j]=0;
  for(k=0;k<pos_num;k++)
    if(pos_samples[k].flag) {
      near_hit=find_near_hit(pos_samples,pos_num,k);
      near_miss=find_near_miss(pos_samples,k,neg_samples,neg_num);
      for(i=0;i<22;i++)
          for(j=0;j<20;j++) {
            temp=pos_samples[k].two_gram[i][j]-neg_samples[near_miss].two_gram[i][j];
            weight[i][j]+=temp*temp;
            temp=pos_samples[k].two_gram[i][j]-pos_samples[near_hit].two_gram[i][j];
            weight[i][j]-=temp*temp;
        }
    }
  for(k=0;k<neg_num;k++)
    if(neg_samples[k].flag) {
      near_hit=find_near_hit(neg_samples,neg_num,k);
      near_miss=find_near_miss(neg_samples,k,pos_samples,pos_num);
      for(i=0;i<22;i++)
          for(j=0;j<20;j++) {
            temp=neg_samples[k].two_gram[i][j]-pos_samples[near_miss].two_gram[i][j];
            weight[i][j]+=temp*temp;
            temp=neg_samples[k].two_gram[i][j]-neg_samples[near_hit].two_gram[i][j];
            weight[i][j]-=temp*temp;
        }
    }
  return;
}
// return the index value of the near_hit
int find_near_hit(sample *samples,int total, int n)
{
  int  k,min_index;
  double min_distance=100000,dist;
  for(k=0;k<total;k++) {
    if(k==n)
      continue;
    if((dist=distance(samples,k,samples,n))<min_distance) {
      min_index=k;
      min_distance=dist;
    }
  }
  return min_index;
}
// return the index of the near_miss  in s2
int find_near_miss(sample *s1,int n, sample *s2, int total)
{
  int  k,min_index;
  double min_distance=100000,dist;
  for(k=0;k<total;k++)
    if((dist=distance(s1,n,s2,k))<min_distance) {
      min_index=k;
      min_distance=dist;
    }
  return min_index;
}
//return the distance between s1[n1] and s2[n2]
double distance(sample *s1,int n1,sample *s2,int n2)
{
  double result=0;
  int i,j;
  for(i=0;i<22;i++)
    for(j=0;j<20;j++)
      result+=(s1[n1].two_gram[i][j]-s2[n2].two_gram[i][j])*(s1[n1].two_gram[i][j]-
s2[n2].two_gram[i][j]);
  result=sqrt(result);
  return result;
}
void generate_pca_data(sample *pos_samples,int pos_num,sample *neg_samples,int neg_num)
{
  ofstream output;
  int n,i,j;
  output.open("pos_training",ios::out);
  if(!output) {
    cout<<"Error in open a file"<<endl;
    return;
  }
  for(n=0;n<pos_num;n++)
    if(pos_samples[n].training_set_flag==1) {
      for(i=0;i<22;i++)
        for(j=0;j<20;j++)
          output<<pos_samples[n].two_gram[i][j]<<" ";
      output<<endl;
    }
  output.close();

  output.open("neg_training",ios::out);
  if(!output) {
    cout<<"Error in open a file"<<endl;
    return;
  }
  for(n=0;n<neg_num;n++)
    if(neg_samples[n].training_set_flag==1) {
      for(i=0;i<22;i++)
        for(j=0;j<20;j++)
          output<<neg_samples[n].two_gram[i][j]<<" ";
      output<<endl;
    }
  output.close();

  output.open("pos_test",ios::out);
  if(!output) {
    cout<<"Error in open a file"<<endl;
    return;
  }
  for(n=0;n<pos_num;n++)
    if(pos_samples[n].training_set_flag!=1) {
      for(i=0;i<22;i++)
        for(j=0;j<20;j++)
          output<<pos_samples[n].two_gram[i][j]<<" ";
      output<<endl;
    }
  output.close();

  output.open("neg_test",ios::out);
  if(!output) {
    cout<<"Error in open a file"<<endl;
    return;
  }
  for(n=0;n<neg_num;n++)
    if(neg_samples[n].training_set_flag!=1) {
      for(i=0;i<22;i++)
        for(j=0;j<20;j++)
          output<<neg_samples[n].two_gram[i][j]<<" ";
      output<<endl;
    }
  output.close();
}
void get_motif_score(sample *samples,int sample_total,motif *motifs,int motif_total)
{
  int i,j;
  double score;
  for(i=0;i<sample_total;i++) {
    samples[i].motif_score=INVALID;
    for(j=0;j<motif_total;j++)
```

```
      if(match(motifs[j].motif,samples[i].string)<=motifs[j].distance){
          score=motifs[j].compression;
        if(score>samples[i].motif_score)
          samples[i].motif_score=score;
      }
  }
  return;
}
short match(char *P,char *D)
{
        register short m,n,i,j,k,r,t,m1,m2,m3,m12,di_1,TE;
        short E[MAXLGH];
        m=strlen(P);
        n=strlen(D);
        E[0]=0;
        for(i=1;i<=m;i++)if(P[i-1]=='*')E[i]=E[i-1];
        else E[i]=E[i-1]+1;
        for(i=1;i<=n;i++){
                E[0]=i;
                TE=i-1;
                di_1=D[TE];
                for(j=0;j<m;j++){
                        if(P[j]=='*')r=t=0;
                        else if(P[j]==di_1){
                                r=0;
                                t=1;
                        } else r=t=1;
                        m1=TE+r;
                        TE=E[j+1];
                        m2=TE+t;
                        m3=E[j]+t;
                        m12=imin(m1,m2);
                        E[j+1]=imin(m12,m3);
                      /*  Core++;*/
                }
        }
        return(E[m]);
}
void classify_by_motif(sample *pos_samples,int actual_pos,sample *neg_samples,int actual_neg)
{
  int i;
  int false_pos,false_neg,pos_test,neg_test;
  pos_test=neg_test=false_pos=false_neg=0;
  for(i=0;i<actual_pos;i++)
```

```
    if(pos_samples[i].training_set_flag==0) {
      pos_test++;
      if(pos_samples[i].motif_score==INVALID)
        false_pos++;
    }
  for(i=0;i<actual_neg;i++)
    if(neg_samples[i].training_set_flag==0) {
      neg_test++;
      if(neg_samples[i].motif_score!=INVALID)
        false_neg++;
    }
  cout<<"pos_test is"<<pos_test<<" neg_test is"<<neg_test<<endl;
  cout<<"Precision is "<<1-float(false_pos+false_neg)/(pos_test+neg_test);
  return;
}
// return the fasta score between two sequences
// retrun -1 if there is an error.
int get_fasta_score(sample *samples1,int n1, sample *samples2,int n2)
{
  int i;
  ifstream in1;
  ofstream out1,out2;
  char filename1[10]="seq1";
  char filename2[10]="seq2";
  out1.open(filename1,ios::out);
  out2.open(filename2,ios::out);
  if(!out1||!out2) {
    cerr<<"Can not open file "<<filename1<<" or "<<filename2<<endl;
    return -1;
  }
  out1<<">"<<endl;
  for(i=0;i<samples1[n1].actual_length;i++)
    out1<<samples1[n1].string[i];
  out1.close();
  out2<<">"<<endl;
  for(i=0;i<samples2[n2].actual_length;i++)
    out2<<samples2[n2].string[i];
  out2.close();
  system("fasta seq1 seq2 >temp1");
  system("perl get_score.pl temp1>result");
  in1.open("result",ios::in);
  if(!in1) {
    cerr<<"Can not open file: result"<<endl;
    return -1;
  }
  in1>>i;
  in1.close();
  system("/bin/rm -rf seq1 seq2 temp1 result");
// cout<<"("<<n1<<","<<n2<<")="<<i<<endl;
  return i;
}
int get_fasta_score2(sample *samples1,int t1, sample *samples2,int t2)
{
  static int *scores;
  int n1,n2;
  long i,j,index,total;
  ifstream in;
  static calling_times=0;
  n1=t1;
  n2=t2;
  if(calling_times==0) {
    calling_times++;
    total=TOTAL*(TOTAL-1)/2;
    scores=new int [total];
    if(!scores) {
      cerr<<"Can not allocate memory"<<endl;
      return -1;
    }
    in.open("scores",ios::in);
    if(!in) {
      cerr<<"Can not open file scores"<<endl;
      return -1;
    }
    for(i=0;i<total;i++){
      in>>scores[i];//skip a number
      in>>scores[i];//skip a number
      in>>scores[i];
    }
  }
  //calculate index value
  //assume n2>n1;
  if(n2<n1) {
    i=n2;
    n2=n1;
    n1=i;
  }
  else if (n1==n2)
    return MIN_SIMILARITY;
  j=0;
  for(i=0;i<n1;i++)
    j+=(750-i);
  j+=n2-n1-1;
//  cout<<"("<<t1<<","<<t2<<")="<<scores[j]<<endl;
  return scores[j];
}
// read motifs and their weights from a file into variable motifs
motif *read_in_motif(char *motif_file,int &total)
{
  int i,len;
  motif *motifs;
  ifstream infile;
  char buffer[MAX_MOTIF_LEN];
  motifs=new motif [total];
  if(!motifs) {
    cerr<<"Can not allocate memory!"<<endl;
    return NULL;
  }
  infile.open(motif_file,ios::in);
  if(!infile) {
    cerr<<"Can not open "<<motif_file<<endl;
    return NULL;
  }
  i=0;
  while(i<total&&!infile.eof()) {
    infile>>motifs[i].compression;
    infile>>motifs[i].distance;
    infile>>motifs[i].occurrence;
    infile>>buffer;
    len=strlen(buffer);
    motifs[i].motif=new char [len+1];
    strncpy(motifs[i].motif,buffer,len);
    motifs[i].motif[len]='\0';
//    cout<<motifs[i].compression<<" "<<motifs[i].distance<<" "<<motifs[i].occurrence<<"
"<<motifs[i].motif<<endl;;
    i++;
  }
  total=i;
  infile.close();
  return motifs;
}
```

```
 /* scale all values to [-1,1] with mean =0 */
void scale_feature_values(sample *pos_samples, int pos_total,sample *neg_samples,int
neg_total,int number_of_two_grams)
{
  int i,j;
  double max,min,mid_range,range;
/*
  for(i=0;i<number_of_two_grams;i++) {//scale selected_two_gram_feature_values
    min=imin(pos_samples[0].selected_two_gram_features_values[i],\
pos_samples[1].selected_two_gram_features_values[i]);
    max=imax(pos_samples[0].selected_two_gram_features_values[i],\
pos_samples[1].selected_two_gram_features_values[i]);
    for(j=2;j<pos_total;j++)
      if(pos_samples[j].selected_two_gram_features_values[i]<min)
        min=pos_samples[j].selected_two_gram_features_values[i];
      else if(pos_samples[j].selected_two_gram_features_values[i]>max)
        max=pos_samples[j].selected_two_gram_features_values[i];

    for(j=0;j<neg_total;j++)
      if(neg_samples[j].selected_two_gram_features_values[i]<min)
        min=neg_samples[j].selected_two_gram_features_values[i];
      else if(neg_samples[j].selected_two_gram_features_values[i]>max)
        max=neg_samples[j].selected_two_gram_features_values[i];
    mid_range=(max+min)/2;
    range=max-min;
    range/=2;
    for(j=0;j<pos_total;j++)
      pos_samples[j].selected_two_gram_features_values[i]= \
(pos_samples[j].selected_two_gram_features_values[i]-mid_range)/range;
    for(j=0;j<neg_total;j++)
      neg_samples[j].selected_two_gram_features_values[i]= \
(neg_samples[j].selected_two_gram_features_values[i]-mid_range)/range;
  } */
  /* scale nearest_distance_from_cluster_centers
  min=imin(pos_samples[0].nearest_distance_from_cluster_centers,\
pos_samples[1].nearest_distance_from_cluster_centers);
  max=imax(pos_samples[0].nearest_distance_from_cluster_centers,\
pos_samples[1].nearest_distance_from_cluster_centers);
  for(j=2;j<pos_total;j++)
    if(pos_samples[j].nearest_distance_from_cluster_centers<min)
      min=pos_samples[j].nearest_distance_from_cluster_centers;
    else if(pos_samples[j].nearest_distance_from_cluster_centers>max)
      max=pos_samples[j].nearest_distance_from_cluster_centers;
  for(j=0;j<neg_total;j++)
    if(neg_samples[j].nearest_distance_from_cluster_centers<min)
      min=neg_samples[j].nearest_distance_from_cluster_centers;
    else if(neg_samples[j].nearest_distance_from_cluster_centers>max)
      max=neg_samples[j].nearest_distance_from_cluster_centers;
  mid_range=(max+min)/2;
  range=max-min;
  range/=2;
  for(j=0;j<pos_total;j++)
    pos_samples[j].nearest_distance_from_cluster_centers=\
(pos_samples[j].nearest_distance_from_cluster_centers-mid_range)/range;
  for(j=0;j<neg_total;j++)
    neg_samples[j].nearest_distance_from_cluster_centers=\
(neg_samples[j].nearest_distance_from_cluster_centers-mid_range)/range;
*/
/*
  // scale correlation_coefficient
  min=imin(pos_samples[0].correlation_coefficient,\
pos_samples[1].correlation_coefficient);
  max=imax(pos_samples[0].correlation_coefficient,\
pos_samples[1].correlation_coefficient);

  for(j=2;j<pos_total;j++)
    if(pos_samples[j].correlation_coefficient<min)
      min=pos_samples[j].correlation_coefficient;
    else if(pos_samples[j].correlation_coefficient>max)
      max=pos_samples[j].correlation_coefficient;
  for(j=0;j<neg_total;j++)
    if(neg_samples[j].correlation_coefficient<min)
      min=neg_samples[j].correlation_coefficient;
    else if(neg_samples[j].correlation_coefficient>max)
      max=neg_samples[j].correlation_coefficient;
  mid_range=(max+min)/2;
  range=max-min;
  range/=2;
  for(j=0;j<pos_total;j++)
    pos_samples[j].correlation_coefficient=\
(pos_samples[j].correlation_coefficient-mid_range)/range;
  for(j=0;j<neg_total;j++)
    neg_samples[j].correlation_coefficient=\
(neg_samples[j].correlation_coefficient-mid_range)/range;
*/
  // scale motif_score
```

```
min=imin(pos_samples[0].motif_score,\
pos_samples[1].motif_score);
  max=imax(pos_samples[0].motif_score,\
pos_samples[1].motif_score);
  for(j=2;j<pos_total;j++)
    if(pos_samples[j].motif_score<min)
      min=pos_samples[j].motif_score;
    else if(pos_samples[j].motif_score>max)
      max=pos_samples[j].motif_score;
  for(j=0;j<neg_total;j++)
    if(neg_samples[j].motif_score<min)
      min=neg_samples[j].motif_score;
    else if(neg_samples[j].motif_score>max)
      max=neg_samples[j].motif_score;
  mid_range=(max+min)/2;
  range=max-min;
  range/=2;
  for(j=0;j<pos_total;j++)
    pos_samples[j].motif_score=\
(pos_samples[j].motif_score-mid_range)/range;
  for(j=0;j<neg_total;j++)
    neg_samples[j].motif_score=\
(neg_samples[j].motif_score-mid_range)/range;
  return;
}
/*
void  write_feature_values(sample *pos_samples,int actual_pos,sample *neg_samples,int
actual_neg, int number_of_two_grams)
{
  int i,j,n;
  char features_file[30],target_file[30];
  ofstream out1,out2;
  strcpy(features_file,OUTPUT);
  strcat(features_file,".i");
  strcpy(target_file,OUTPUT);
  strcat(target_file,".t");
  out1.open(features_file,ios::out);
  out2.open(target_file,ios::out);
  if(!out1||!out2) {
    cerr<<"Error in openning files"<<endl;
    return;
  }
  out1<<setprecision(6)<<setiosflags(ios::fixed);
```

```
 i=j=0;
 while(i<actual_pos&&j<actual_neg) {
   if(randomNumber()<0.5){//switch to positive set
     if(pos_samples[i].training_set_flag==1) {//write training data
       for(n=0;n<number_of_two_grams;n++)
             out1<<pos_samples[i].selected_two_gram_features_values[n]<<" ";
//        out1<<pos_samples[i].nearest_distance_from_cluster_centers<<" ";
       out1<<neg_samples[j].correlation_coefficient<<" ";
          out1<<pos_samples[i].motif_score;
       out1<<endl;
       out2<<1<<endl;
     }
     i++;
   }
   else {//switch to negative set
     if(neg_samples[j].training_set_flag==1) {//write training data
       for(n=0;n<number_of_two_grams;n++)
             out1<<neg_samples[j].selected_two_gram_features_values[n]<<" ";
//        out1<<neg_samples[j].nearest_distance_from_cluster_centers<<" ";
      out1<<neg_samples[j].correlation_coefficient<<" ";
          out1<<neg_samples[j].motif_score;
       out1<<endl;
       out2<<0<<endl;
     }
     j++;
   }
 }
 while(i<actual_pos) {
   if(pos_samples[i].training_set_flag==1) {//write training data
     for(n=0;n<number_of_two_grams;n++)
             out1<<pos_samples[i].selected_two_gram_features_values[n]<<" ";
//      out1<<pos_samples[i].nearest_distance_from_cluster_centers<<" ";
    out1<<pos_samples[i].correlation_coefficient<<" ";
    out1<<pos_samples[i].motif_score;
     out1<<endl;
     out2<<1<<endl;
   }
   i++;
 }
 while(j<actual_neg) {
     if(neg_samples[j].training_set_flag==1) {//write training data
       for(n=0;n<number_of_two_grams;n++)
             out1<<neg_samples[j].selected_two_gram_features_values[n]<<" ";
```

```
//      out1<<neg_samples[j].nearest_distance_from_cluster_centers<<" ";
      out1<<neg_samples[j].correlation_coefficient<<" ";
     out1<<neg_samples[j].motif_score;
      out1<<endl;
      out2<<0<<endl;
     }
     j++;
  }
  out1<<"Test data begin"<<endl;
  out2<<"Test data begin"<<endl;
  i=j=0;
  while(i<actual_pos&&j<actual_neg) {
   if(randomNumber()<0.5){//switch to positive set
    if(pos_samples[i].training_set_flag==0) {//write test data
     for(n=0;n<number_of_two_grams;n++)
          out1<<pos_samples[i].selected_two_gram_features_values[n]<<" ";
//      out1<<pos_samples[i].nearest_distance_from_cluster_centers<<" ";
      out1<<pos_samples[i].correlation_coefficient<<" ";
      out1<<pos_samples[i].motif_score;
      out1<<endl;
      out2<<1<<endl;
    }
    i++;
   }
   else {//switch to negative set
    if(neg_samples[j].training_set_flag==0) {//write test data
     for(n=0;n<number_of_two_grams;n++)
          out1<<neg_samples[j].selected_two_gram_features_values[n]<<" ";
//      out1<<neg_samples[j].nearest_distance_from_cluster_centers<<" ";
      out1<<neg_samples[j].correlation_coefficient<<" ";
      out1<<neg_samples[j].motif_score;
      out1<<endl;
      out2<<0<<endl;
    }
    j++;
   }
  }
  while(i<actual_pos) {
   if(pos_samples[i].training_set_flag==0) {//write test data
    for(n=0;n<number_of_two_grams;n++)
          out1<<pos_samples[i].selected_two_gram_features_values[n]<<" ";
//    out1<<pos_samples[i].nearest_distance_from_cluster_centers<<" ";
    out1<<pos_samples[i].correlation_coefficient<<" ";
  out1<<pos_samples[i].motif_score;
```

```
      out1<<endl;
      out2<<1<<endl;
    }
    i++;
  }
  while(j<actual_neg) {
      if(neg_samples[j].training_set_flag==0) {//write test data
        for(n=0;n<number_of_two_grams;n++)
              out1<<neg_samples[j].selected_two_gram_features_values[n]<<" ";
//        out1<<neg_samples[j].nearest_distance_from_cluster_centers<<" ";
      out1<<neg_samples[j].correlation_coefficient<<" ";
            out1<<neg_samples[j].motif_score;
        out1<<endl;
        out2<<0<<endl;
      }
      j++;
  }
  out1.close();
  out2.close();
  return;
}
*/
void  write_feature_values(sample *pos_samples,int actual_pos,sample *neg_samples,int
actual_neg, int number_of_two_grams)
{
  int i,j,n;
  char features_file[30],target_file[30];
  ofstream out1,out2;
  strcpy(features_file,OUTPUT);
  strcat(features_file,".i");
  strcpy(target_file,OUTPUT);
  strcat(target_file,".t");
  out1.open(features_file,ios::out);
  out2.open(target_file,ios::out);
  if(!out1||!out2) {
    cerr<<"Error in openning files"<<endl;
    return;
  }
  out1<<setprecision(6)<<setiosflags(ios::fixed);
  i=j=0;
  while(i<actual_pos) {
      if(pos_samples[i].training_set_flag==1) {//write training data
        for(n=0;n<number_of_two_grams;n++)
```

```
          out1<<pos_samples[i].selected_two_gram_features_values[n]<<" ";
 //     out1<<pos_samples[i].nearest_distance_from_cluster_centers<<" ";
       out1<<pos_samples[i].correlation_coefficient<<" ";
          out1<<pos_samples[i].motif_score;
       out1<<endl;
       out2<<1<<endl;
      }
      i++;
    }
  while(j<actual_neg) {
     if(neg_samples[j].training_set_flag==1) {//write training data
       for(n=0;n<number_of_two_grams;n++)
           out1<<neg_samples[j].selected_two_gram_features_values[n]<<" ";
//       out1<<neg_samples[j].nearest_distance_from_cluster_centers<<" ";
      out1<<neg_samples[j].correlation_coefficient<<" ";
          out1<<neg_samples[j].motif_score;
       out1<<endl;
       out2<<0<<endl;
      }
      j++;
  }
// out1<<"Test data begin"<<endl;
// out2<<"Test data begin"<<endl;
  i=j=0;
  while(i<actual_pos) {
     if(pos_samples[i].training_set_flag==0) {//write test data
       for(n=0;n<number_of_two_grams;n++)
           out1<<pos_samples[i].selected_two_gram_features_values[n]<<" ";
//       out1<<pos_samples[i].nearest_distance_from_cluster_centers<<" ";
       out1<<pos_samples[i].correlation_coefficient<<" ";
       out1<<pos_samples[i].motif_score;
       out1<<endl;
       out2<<1<<endl;
      }
      i++;
  }
  while(j<actual_neg) {
     if(neg_samples[j].training_set_flag==0) {//write test data
       for(n=0;n<number_of_two_grams;n++)
           out1<<neg_samples[j].selected_two_gram_features_values[n]<<" ";
//       out1<<neg_samples[j].nearest_distance_from_cluster_centers<<" ";
      out1<<neg_samples[j].correlation_coefficient<<" ";
      out1<<neg_samples[j].motif_score;
```

```
        out1<<endl;
        out2<<0<<endl;
      }
      j++;
   }
  out1.close();
  out2.close();
  return;
}

/*
   The random number generator, it generates a pseudo-random number between
   0 and 1, starting with the seed given above.
*/
double randomNumber()
{
  unsigned int x;
  double dx,r;

  r=0;
  while (r<1E-24)
      {
      nextrandom=nextrandom*1103515245 + 12345;
      x=(nextrandom/65536) % 32768;
      dx=x;
      r=dx/32768;
      }
  return r;
}
/* return a seed for random number generation */
long randomSeed()
{
  struct timeval tp;

 gettimeofday(&tp,NULL);
 return tp.tv_sec;
}

 int find_number_of_clusters(cluster *all_clusters)
 {
  int count=0;
  cluster *p;
  p=all_clusters;
  while(p!=NULL) {
    count++;
    p=p->next;
   }
   return count;
  }
  // samples can be pos_samples or neg_samples
  void get_nearest_distance_from_cluster_centers(sample *pos_samples,int pos_total,sample
*samples,int total,cluster *all_clusters,int total_cluster)
  {
   int i,j,center,score;
   cluster *p;
   if(pos_samples==samples)/* for positive set, call get_fasta_score2 */
    for(i=0;i<total;i++) {
     samples[i].nearest_distance_from_cluster_centers=INVALID;
     p=all_clusters;
     while(p!=NULL) {
        center=p->center_sequence_number;
        if(center!=i)
          score=get_fasta_score2(pos_samples,center,samples,i);
      else
        score=EQUAL_SEQUENCES_FASTA_SCORE;
        if(score>samples[i].nearest_distance_from_cluster_centers)
        samples[i].nearest_distance_from_cluster_centers=score;
        p=p->next;
     }
    }
   else // for negative set,call get_fasta_score instead
    for(i=0;i<total;i++) {
     samples[i].nearest_distance_from_cluster_centers=INVALID;
     p=all_clusters;
     while(p!=NULL) {
        center=p->center_sequence_number;
        score=get_fasta_score(pos_samples,center,samples,i);
        if(score>samples[i].nearest_distance_from_cluster_centers)
        samples[i].nearest_distance_from_cluster_centers=score;
        p=p->next;
     }
   }
   return;
  }
```

REFERENCES

1. C. F. Allex, J. W. Shavlik, and F. R. Blattner. Neural network input representations that produce accurate consensus sequences from DNA fragment assemblies. *Bioinformatics* **15**(9), 723–728, 1999.

2. S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped Blast and PSI-Blast: A new generation of protein database search programs. *Nucleic Acids Research* **25**(17), 3389–3402, 1997.

3. A. Apostolico and R. Giancarlo. Sequence alignment in molecular biology. *Journal of Computational Biology* **5**(2), 173–196, 1998.

4. R. Ash. *Information Theory.* Interscience Publishers, New York, 1965.

5. T. L. Bailey and W. N. Grundy. Classifying proteins by family using the product of correlated p-values. In *Proceedings of the Third Annual International Conference on Computational Molecular Biology,* 1999, pp. 10–14.

6. T. L. Bailey and C. P. Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning* **21**, 51–83, 1995.

7. A. Bairoch. The PROSITE dictionary of sites and patterns in proteins, its current status. *Nucleic Acids Research* **21**, 3097–3103, 1993.

8. W. C. Barker, J. S. Garavelli, H. Huang, P. B. McGarvey, B. Orcutt, G. Y. Srinivasarao, C. Xiao, L. S. Yeh, R. S. Ledley, J. F. Janda, F. Pfeiffer, H. W. Mewes, A. Tsugita, and C. H. Wu. The protein information resource (PIR). *Nucleic Acids Research* **28**(1), 41–44, 2000.

9. M. Ben-Bassat. Use of distance measures, information measures and error bounds in feature evaluation. In: P. R. Krishnaiah, L. N. Kanal (eds.), *Classification, Pattern Recognition and Reduction of Dimensionality: Handbook of Statistics,* volume 2, North-Holland publishing company, North-Holland, 1982, pp 773–791.

10. J. O. Berger. Statistical Decision Theory and Bayesian Analysis. Springer-Verlag, New York, New York, 1985.

11. C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, New York, New York, 1995.

12. A. Brazma, I. Jonassen, I. Eidhammer I., and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology* **5**(2), 279–305, 1998.

13. A. Brazma, I. Jonassen, E. Ukkonen, and J. Vilo. Discovering patterns and subfamilies in biosequences. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, 1996, pp. 34–43.

14. W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering* **8**(2), 195–210, 1996.

15. A. Califano. SPLASH: Structural pattern localization and analysis by sequential histograms. *Bioinformatics*, 2000.

16. L. R. Cardon and G. D. Stormo. Expectation maximization algorithm for identifying protein-binding sites with variable lengths from unaligned DNA fragments. *Journal of Molecular Biology* **223**(1), 159–170, 1992.

17. N. A. Chuzhanova, A. J. Jones, and S. Margetts. Feature selection for genetic sequence classification. *Bioinformatics* **14**(2), 139–143, 1998.

18. M. W. Craven and J. W. Shavlik. Machine learning approaches to gene recognition. *IEEE Expert* **9**(2), 2–10, 1994.

19. E. M. Crowley, K. Roeder, and M. Bina. A statistical model for locating regulatory regions in genomic DNA. *Journal of Molecular Biology* **268**(1), 8–14, 1997.

20. M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis* **1**(3),1997. Electronic Journal: http://www-east.elsevier.com/ida.

21. M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure* **15** suppl. 3, 345–358, 1978.

22. A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, Series B, **39**, 1–38, 1977.

23. T. G. Dietterich. Machine learning research: Four current directions. *AI Magazine* **18**(4), 97–136, 1997.

24. S. Eddy. Profile hidden Markov models. *Bioinformatics*, **14**(9), 755–763, 1999.

25. S. Eddy, G. Mitchison, and R. Durbin. Maximum discrimination hidden Markov models of sequence consensus. *Journal of Computational Biology* **2**, 9–23, 1995.

26. K. A. Frenkel. The human Genome project and informatics. *Communications of the ACM* **34**(11), 41–51, 1991.

27. D. J. Galas, M. Eggert, and M. S. Waterman. Rigorous pattern-recognition methods for DNA sequences: Analysis of promoter sequences from E. Coli. *Journal of Molecular Biology* **186**(1), 117–128, 1985.

28. W. N. Grundy and T. L. Bailey. Family pairwise search with embedded motif models. *Bioinformatics* **15**(6), 463–470, 1999.

29. W. N. Grundy, T. L. Bailey, C. P. Elkan, and M. E. Baker. Meta-MEME: Motif-based hidden Markov models of protein families. *Computer Applications in the Biosciences* **13**(4), 397–406, 1997.

30. D. C. Hanselman. Mastering MATLAB 5: A comprehensive tutorial and reference. Prentice Hall, Upper Saddle River, NJ, 1998.

31. C. B. Harley and R. P. Reynolds. Analysis of E. Coli promoter sequences. *Nucleic Acids Research* **15**(5), 2343–2361, 1987.

32. R. Hart, A. Royyuru, G. Stolovitzky, and A. Califano. Systematic and automated discovery of patterns in PROSITE families. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology*, 2000.

33. D. Haussler. A brief look at some machine learning problems in genomics. In *Proceedings of the Tenth Annual Conference on Computational Learning Theory*, 1997, pp. 109–113.

34. S. Henikoff and J. G. Henikoff. Automated assembly of protein blocks for database searching. *Nucleic Acids Research* **19**, 6565–6572, 1991.

35. S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the USA* **89**, 10915–10919, 1992.

36. H. Hirsh and M. Noordewier. Using background knowledge to improve inductive learning of DNA sequences. In *Proceedings of the Tenth Conference on Artificial Intelligence for Applications*, 1994, pp. 351–357.

37. R. Karchin and R. Hughey. Weighting hidden Markov models for maximum discrimination. *Bioinformatics* **14**(9), 772–782, 1998.

38. S. Knudsen. Promoter2.0: For the recognition of PolII promoter sequences. *Bioinformatics* **15**(5), 356–361, 1999.

39. A. Krogh, M. Brown, I.S. Mian, K. Sjolander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology* **235**(5), 1501–1531, 1994.

40. D. Kulp, D. Haussler, M. G. Reese, and F. H. Eeckman. A generalized hidden Markov model for the recognition of human genes in DNA. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, 1996, pp. 134–142.

41. C. E. Lawrence and A. A. Reilly. An expectation-maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins: Structure, Function, and Genetics* **7**, 41–51, 1990.

42. S. Lisser and H. Margalit. Compilation of E. Coli mRNA promoter sequences. *Nucleic Acids Research* **21**(7), 1507–1516, 1993.

43. D. J. C. Mackay. The evidence framework applied to classification networks. *Neural Computation* **4**(5), 698–714, 1992.

44. I. Mahadevan and I. Ghosh. Analysis of E. Coli promoter structures using neural networks. *Nucleic Acids Research* **22**(11), 2158-2165, 1994.

45. G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions.* John and Wiley, New York, New York, 1997.

46. G. Mengeritsky and T. F. Smith. Recognition of characteristic patterns in sets of functionally equivalent DNA sequences. *Computer Applications in the Biosciences* **3**(3), 223–227, 1 987.

47. D. W. Opitz and J. W. Shavlik. Connectionist theory refinement: Genetically searching the space of network topologies. *Journal of Artificial Intelligence Research* **6**, 177–209, 1997.

48. O. N. Ozoline, A. A. Deev, and E. N. Trifonov. DNA bendability- A novel feature in E. Coli promoter recognition. *Journal of Biomolecular Structure and Dynamics* **16**(4), 825- -831, 1999.

49. O. N. Ozoline, A. A. Deev, and M. V. Arkhipova. Non-canonical sequence elements in the promoter structure. Cluster analysis of promoters recognized by E. Coli RNA polymerase. *Nucleic Acids Research* **25**(23), 4703–4709, 1997.

50. R. H. Lathrop, N. R. Steffen, M. Raphael, S. Deeds-Rubin, M. J. Pazzani, P. J. Cimoch, D. M. See, and J. G. Tilles. Knowledge-based avoidance of drug-resistant HIV mutants. *AI Magazine* **20**(1), 13–25, 1999.

51. W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the USA* **85**(8), 2444–2448, 1988.

52. A. G. Pedersen, P. Baldi, S. Brunak, and Y. Chauvin. Characterization of prokaryotic and eukaryotic promoters using hidden Markov models. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, 1996, pp. 182–191.

53. A. G. Pedersen and J. Engelbrecht. Investigations of E. Coli promoter sequences with artificial neural networks: New signals discovered upstream of the transcriptional start point. In *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, 1995, pp. 292–299.

54. L. Pickert, I. Reuter, F. Klawonn, and E. Wingender. Transcription regulatory region analysis using signal detection and fuzzy clustering. *Bioinformatics* **14**(3), 244–251, 1998.

55. L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE **77**(2), 257–286, 1989.

56. J. Rissanen. Modeling by shortest data description. *Automatica* **14**, 465–471, 1978.

57. S. Salzberg. A decision tree system for finding genes in DNA. Technical Report CS-97-03, Johns Hopkins University, 1997.

58. T. J. Santner. *The Statistical Analysis of Discrete Data.* Springer-Verlag, New York, New York, 1989.

59. T. D. Schneider and R. M. Stephens. Sequence logos: A new way to display consensus sequences. *Nucleic Acids Research* **18**(20), 6097–6100, 1990.

60. C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal* **27**, 379–423, 623–656, 1948.

61. V. V. Solovyev and K. S. Makarova. A novel method of protein sequence classification based on oligopeptide frequency analysis and its application to search for functional sites and to domain localization. *Computer Applications in the Biosciences* **9**(1), 17–24, 1993.

62. R. Staden. Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Research* **12**(1), 505–519, 1984.

63. M. B. Swindells, C. A. Orengo, D. T. Jones, E. G. Hutchinson, and J. M. Thornton. Contemporary approaches to protein structure classification. *BioEssays* **20**, 884-891, 1998.

64. G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence* **70**, 119–165, 1994.

65. J. T. L. Wang, G. W. Chirn, T. G. Marr, B. A. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: Some preliminary results. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Minneapolis, Minnesota, May 1994, pp. 115–125.

66. J. T. L. Wang, T. G. Marr, D. Shasha, B. A. Shapiro, G. W. Chirn, and T. Y. Lee. Complementary classification approaches for protein sequences. *Protein Engineering* **9**(5), 381–386, 1996.

67. J. T. L. Wang, S. Rozen, B. A. Shapiro, D. Shasha, Z. Wang, and M. Yin. New techniques for DNA sequence classification. *Journal of Computational Biology* **6**(2), 209–218, 1999.

68. J. T. L. Wang, B. A. Shapiro, and D. Shasha (eds.). *Pattern Discovery in Biomolecular Data: Tools, Techniques and Applications.* Oxford University Press, New York, 1999.

69. X. Wang, J. T. L. Wang, D. Shasha, B. A. Shapiro, S. Dikshitulu, I. Rigoutsos, and K. Zhang. Automated discovery of active motifs in three dimensional molecules. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, Newport Beach, California, August 1997, pp. 89–95.

70. C. H. Wu. Artificial neural networks for molecular sequence analysis. *Computers and Chemistry* **21**(4), 237–256, 1997.

71. C. H. Wu, M. Berry, Y. S. Fung, and J. McLarty. Neural networks for full-scale protein sequence classification: Sequence encoding with singular value decomposition. *Machine Learning* **21**, 177–193, 1995.

72. C. H. Wu and J. McLarty. *Neural Networks and Genome Informatics.* Elsevier Science, 2000.

73. C. H. Wu, G. Whitson, J. McLarty, A. Ermongkonchai, and T. C. Chang. Protein classification artificial neural system. *Protein Science* **1**(5), 667–677, 1992.

74. S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM* **35**(10), 83–91, 1992.