INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9212149

Study of secondary structure of protein sequences by linear algebra

Hsieh, Wei-hua, Ph.D.

Iowa State University, 1991

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Study of secondary structure of protein sequences by linear algebra

by

Wei-hua Hsieh

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Department: Mathematics
Major: Applied Mathematics

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa
1991

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I am deeply grateful to Dr. James L. Cornette for his guidance and encouragement during my graduate study. Without his help, this dissertation would never have been possible.

I am also indebted to Dr. Vincent A. J. Sposito for his kindly help on both MPSX and OSL software.

Finally, I am grateful to the Computation Center of Iowa State University for the Block Grant support for computing.

1. INTRODUCTION

A protein is a long-chain, linear polymer of amino acids, often cross-linked but never branched. Each monomer has a side chain, which is usually one of the 20 common types: alanine, arginine, asparagine, aspartic acid, cysteine, glutamine, glutamic acid, glycine, histidine, isoleucine, leucine, lysine, methionine, phenylalanine, proline, serine, threonine, tryptophan, tyrosine, and valine. The so-called primary structure of a protein is its sequence of these amino acids. A common feature of a protein's three-dimensional structure is the alpha-helix, which consists of a segment of some 7 to 20 amino acids folded into the form of a helix. Two-state prediction of the secondary structure of a protein usually entails the classification of each of its amino acids as either "helical" or "nonhelical." Three-state prediction, on the other hand, entails classification of the structure of each amino acid as alpha-helix, beta-sheet, or coil. Beta-sheet is a kind of secondary structure, and thus neither alpha-helix nor beta-sheet structures are referred to as coil.

In 1974 [1], it was asserted that because protein folding occurs with no outside assistance, the resulting three-dimensional native conformation depends solely upon the primary sequence. Furthermore, the influence of the secondary structure of an amino acid by other amino acids in the protein decreases with "residue distance," i.e., much information about the conformation of an amino acid is carried by

a "long enough" block with the observed amino acid in the middle. In 1984 Kabsch and Sander [2] found that identical pentapeptides had different conformations. This observation tells us that long-range interaction should be considered upon determination of the secondary structure. In short, we can say that two identical strings of amino acids tend to have identical three-dimensional shapes and that two similar but nonidentical strings tend to have similar conformations. The role of blocks of amino acids in the determination of protein secondary structures remains in question, however.

This research is directed to study "locally" the relation between primary structures and secondary structures of proteins and to provide a methodology for identifying secondary structures. Specific objectives are as follows:

1. **Similarity scale.** To construct a similarity scale assigning a value to each amino acid. For example, a seven amino acid sequence can be represented by a seven-dimensional real vector or by a single point in R^7. The Euclidean distance between two points determines the similarity of the corresponding two amino acid strings.

2. **Similarity matrix.** To construct a 20×20 similarity matrix that is symmetric and in which each row (and column) corresponds to a particular amino acid. The "similarity score" of two amino acid sequences of the same length is determined by addition of entries chosen from the similarity matrix. (The number of chosen entries is the length of the amino acid strings.)

3. **Single separation plane.** To locate amino acid strings of length k, say, in the 20k-dimensional real space, and to then find an acceptable plane to "separate"

middle amino acid alpha-helical strings from middle amino acid non-alpha-helical strings. The problem is thus transformed into a partition problem.

4. **Pairs of separation planes.** To construct several pairs of parallel planes, thereby making partitioning possible.

All computing jobs were performed on the IBM 360 or the Digital Work Station 3100. Programs, which were written in C, are listed in Appendix B. Software— IBM Mathematical Programming System (MPSX) and Optimization Subroutine Library (OSL)—was used to perform Simplex computations. Kabsch & Sander's protein secondary structure assignments, listed in Appendix A, were used to train and to test prediction schemes.

2. LITERATURE REVIEW

2.1 Introduction

We will begin by quoting from the works of prominent researchers on the significance of predicting secondary protein structure from the primary protein sequence:

- Chou and Fasman [3]:

  > As such, multi-state prediction models serve as important starting conformations for calculations of protein folding based on energy minimization. (Page 87)

- Qian and Sejnowski [10]:

  > This approach is not meant to be an alternative to other methods that have been developed to study protein folding that take biophysical properties explicitly into account, such as the methods of free energy minimization... Rather, our method provides additional constraints to reduce the search space for these methods. For example, a good prediction for the secondary structure could be used as the initial conditions for energy minimization... (Page 866)

2.2 Chou and Fasman's Conformational Parameters

Chou and Fasman's predictive method [3], which is one of the earliest prediction schemes, outlines three basic steps:

Step 1: Use the database, which contains some proteins with known structures, to compute the conformational parameters P_α (for helix) and P_β (for sheet) for each of the 20 amino acids.

The helix conformational parameter is

$$P_\alpha = \frac{f_\alpha}{< f_\alpha >},$$

where f_α is the frequency of residues in the helix and where $< f_\alpha >$ is the average frequency of all residues in helical regions. Similar expressions hold for P_β.

Example. Suppose that there are 4741 residues in 29 proteins for which 1798 residues are alpha-helical and 930 are beta-sheet residues. Then

$$< f_\alpha >= \frac{1798}{4741} \simeq 0.379, \text{ and}$$

$$< f_\beta >= \frac{930}{4741} \simeq 0.196.$$

Suppose that, among the 4741 residues, 434 are Alanines, for which 234 are alpha-helical and 71 are beta-sheet. Then, for Ala,

$$f_\alpha = \frac{234}{434} \simeq 0.539, \text{ and}$$

$$f_\beta = \frac{71}{434} \simeq 0.164.$$

So the conformational parameters P_α and P_β for Ala are

$$P_\alpha = \frac{f_\alpha}{< f_\alpha >} = \frac{0.539}{0.379} \simeq 1.422 \text{ and}$$

Table 2.1: Assignment of region 179–183 of Carboxypeptitase A (Based on 15 proteins): $(Ihihh)_\alpha$ and $(HHihH)_\beta$.

	179	180	181	182	183
Carboxypeptitase A	Ile	Val	Asp	Phe	Val
Helical assignment	I	h	i	h	h
P_α value	1.00	1.14	0.98	1.12	1.14
beta-sheet assignment	H	H	i	h	H
P_β value	1.60	1.65	0.80	1.28	1.65

$$P_\beta = \frac{f_\beta}{< f_\beta >} = \frac{0.164}{0.196} \simeq 0.837.$$

Step 2: Use the conformational parameters to assign each residue as a former, an indifferent, and a breaker.

Helical assignments: H_α, strong alpha-former; h_α, alpha-former; I_α, weak alpha-former; i_α, alpha-indifferent; b_α, alphabreaker; B_α, strong alphabreaker.

Beta-sheet assignments: H_β, strong beta-former; h_β, beta-former; I_β, weak beta-former; i_β, beta-indifferent; b_β, beta-breaker; B_β, strong beta-breaker.

Example. As in the example for Step 1, Alanine was assigned as a strong helix former (H_α, $P_\alpha = 1.422$) and as an indifferent for beta-sheet (i_β, $P_\beta = 0.837$).

Example. See Table 2.1.

Step 3: Use a set of empirical rules to locate the secondary structures of proteins.

There are three basic rules for predicting secondary structures. Let $< P_\alpha >$ and $< P_\beta >$ denote the averages of P_α and P_β values for the residues in the segment under consideration, respectively. Because rules and conditions are somewhat ambiguous, we will describe those aspects of the rules and the conditions that have been used in a program written by Minoru Kanehisa and list the titles of the unused conditions in the basic rules.

- *Rule 1. (Search for helical regions)*

  – *Helix Nucleation*

  Step A. Assign 2 to H_α and h_α residues; 1 to I_α residues; and 0 to i_α, b_α, and B_α residues. Let us call these values (2, 1, and 0) the "α-tendency quantity" of residues. The α-tendency quantity of a k-residue sequence in length is the sum of the α-tendency quantity of the k residues in this sequence.

  Example. The α-tendency quantity of the five-residue sequence in the previous example, i.e. $(Ihihh)_\alpha$, is $1+2+0+2+2=7$.

  Consider all amino acid sequences of length six with an α-tendency quantity of at least 8. Modify each sequence by keeping its longest contiguous subsequence such that the first and the last amino acids have an α-tendency quantity of 1 or 2. Then assign the structure alpha-helix to all the amino acids remaining after modification.

  – *Helix Propagation and Termination*

  Step B. Assign 1 to H_α and h_α residues; 0 to I_α and i_α residues; and -2 to b_α and B_α residues. Let us call these values the "$\bar{\alpha}$-tendency quantity."

  Example. The $\bar{\alpha}$-tendency quantity of $(Ihihh)_\alpha$ is $0+1+0+1+1=3$.

  Extend the helical segments found in Step 1 in both direction by considering their adjacent tetrapeptides. For the C-terminal end extension, shift along the protein one amino acid every time until find a tetrapeptide having $< P_\alpha > < 1$ and ($\bar{\alpha}$-tendency quantity) ≤ 0. Then assign the structure alpha-helix to the residues between the helical segment and

the last tetrapeptide (not included). Do the same for the N-terminal end extension.

– *Proline as Helix Breaker*

Step C. The helical segment in a new (or observed) protein has been identified in Steps 1 and 2. The object of Step 3 is to modify the located helical segments by considering the Prolines in these helical segments according to rules (a)–(c):

(a) If position 4, 5, or 6, counted from the N-terminal end to the C-terminal end, is occupied by a Proline, then the helical segment is shortened by deleting the residues "before" this Proline in the segment. The new, shortened helical segment should also obey rules (a), (b), and (c).

(b) The minimum length of a helical segment is 6. Otherwise, cancel the helix assigned to that segment.

(c) After (a) and (b), the first six residues in every helical segment have been definitely assigned a secondary structure, namely, a alpha-helix. So we construct the subsequences of these segments by cutting the residues before the first proline found after the seventh (included) position. Then go to (a).

– *Helix Boundaries*

• *Rule 2. (Search for beta-sheet regions)*

– *Beta-Sheet Nucleation*

Step A. Assign 1 to H_β and h_β residues; and 0 to i_β, b_β, and B_β

residues. Let us call these values (1 and 0) the "β-tendency quantity" of residues. The β-tendency quantity of a sequence of length k is the sum of the β-tendency quantity of the k residues in this sequence.

Consider all amino acid sequences of length 5 with a β-tendency quantity of at least 3. Modify each sequence by keeping its longest contiguous subsequence such that the first and the last amino acids have a β-tendency quantity of 1. Then assign the structure beta-sheet to all the amino acids remaining after modification.

- *Beta-Sheet Propagation and Termination*

  Step B. Assign 1 to H_β and h_β residues; 0 to i_β residues; and -2 to b_β and B_β residues. Let us call these values the "$\bar{\beta}$-tendency quantity."

  Extend the beta-sheet segments found in Step A in both directions by considering their adjacent tetrapeptides. For the C-terminal end extension, shift along the protein one amino acid every time until there is a tetrapeptide having $< P_\beta > < 1$ and ($\bar{\beta}$-tendency quantity) ≤ 0. Then assign the structure beta-sheet to the residues between the beta-sheet segment and the last tetrapeptide (not included). Do the same for the N-terminal end extension.

- *Strong Beta-sheet Breakers*

- *Beta-sheet Boundaries*

- *Rule 3. (Overlapping alpha- and beta- regions)*

2.3 Robson *et al.*'s Information Theory Method

In this section, we will explain the basic idea of Robson *et al.* 's information theory method [5, 6, 7, 8]. Theoretically, the purpose of this method is to estimate the real value of the information [6]

$$I\left(S_j; R_1, R_2, \ldots, R_{last}\right),$$

which reads, " the information that the residues at the first (R_1), the second (R_2), and so on, up to the last position $\left(R_{last}\right)$ carry regarding the conformation of the jth residue $\left(S_j\right)$."

If there are two possible conformational states, say A and $\bar{A}$, for each residue, then to predict the structure of the jth residue, we simply compare the two values:

$$I_1 \equiv I\left(S_j = A; R_1, R_2, \ldots, R_{last}\right)$$

and

$$I_2 \equiv I\left(S_j = \bar{A}; R_1, R_2, \ldots, R_{last}\right).$$

The larger value defines the conformational state of the jth residue.

Because of the observation that the effect of residues

$$R_{j-m}, R_{j-m+1}, \ldots, R_j, \ldots, R_{j+m-1}, \text{ and } R_{j+m}$$

plays a dominant role for some integer m (it is claimed that the best choice is m = 8), an approximation for I_1 is thus

$$I_1 = I\left(S_j = A; R_1, R_2, \ldots, R_{last}\right) \simeq \sum_{i=-m}^{m} I\left(S_j = A; R_{j+i}\right).$$

A similar expression holds for I_2. So

$$I_1 - I_2 \simeq \sum_{i=-m}^{m} \left[I\left(S_j = A; R_{j+i}\right) - I\left(S_j = \bar{A}; R_{j+i}\right) \right]. \tag{2.1}$$

Definition. $I\left(S_j = A; R_{j+i}\right) = \log \frac{P\left(S_j = A | R_{j+i}\right)}{P\left(S_j = A\right)}$, where $P\left(S_j = A \mid R_{j+i}\right)$ is the conditional probability that the conformation at the jth position(S_j) is A, given the type of residue at position j + i $\left(R_{j+i}\right)$; and where $P\left(S_j = A\right)$ is the probability that the conformation at the jth position is A.

Definition. $I\left(S_j = A : \bar{A}; R_{j+i}\right) = I\left(S_j = A; R_{j+i}\right) - I\left(S_j = \bar{A}; R_{j+i}\right)$.

From Eq. (2.1), we have

$$I_1 - I_2 \simeq \sum_{i=-m}^{m} I\left(S_j = A : \bar{A}; R_{j+i}\right). \tag{2.2}$$

For notational convenience, let us remove the suffixes j and j + i; let us also replace A and $\bar{A}$ by 1 and 2, respectively, in the right hand side of Eq. (2.2) and explain how to evaluate $I\left(S_j = A : \bar{A}; R_{j+i}\right)$, that is, how to find $I(S = 1 : 2; R)$.

Let θ_{SR} denote the probability of the combination (S,R), where S stands for one of the conformations 1 and 2 and where R stands for one of the 20 amino acids. Then

$$\theta_{SR} \geq 0,$$

$$\sum_{S=1}^{2} \sum_{R=1}^{20} \theta_{SR} = 1$$

and

$$\begin{aligned} I(S = 1 : 2; R) &= \log \frac{P(S = 1 \mid R)}{P(S = 1)} - \log \frac{P(S = 2 \mid R)}{P(S = 2)} \\ &= \log \frac{P(S = 1 \mid R)}{P(S = 2 \mid R)} - \log \frac{P(S = 1)}{P(S = 2)} \\ &= \log \frac{\theta_{1R}}{\theta_{2R}} - \log \frac{\theta_{1\bullet}}{\theta_{2\bullet}}, \end{aligned} \tag{2.3}$$

where $\theta_{S\bullet} = \sum_{R=1}^{20} \theta_{SR}$.

Suppose that f_{SR} is the number of appearances of the combination (S,R) in the database D, say, and that $\sum_{S=1}^{2} \sum_{R=1}^{20} f_{SR} = f_{\bullet\bullet}$ is fixed. When the values of f_{1R}, f_{2R}, $f_{1\bullet}$, and $f_{2\bullet}$ are great, then, from Eq. (2.3),

$$I\,(S = 1:2; R) \simeq \log \frac{f_{1R}}{f_{2R}} - \log \frac{f_{1\bullet}}{f_{2\bullet}}. \tag{2.4}$$

On the other hand, if the size of the database is too small, Robson *et al.* apply **Bayes's decision theory**, based on a database, to determine the posterior expected value of

$$I\,(S = 1:2; R)$$

(see Note 2). The likelihood function and the *prior* probability density of the unknown parameters θ_{SR}'s are then proportional to

$$\prod_{S=1}^{2} \prod_{R=1}^{20} \theta_{SR}^{f_{SR}}$$

and

$$\prod_{S=1}^{2} \prod_{R=1}^{20} \theta_{SR}^{-1},$$

respectively. The posterior density, say

$$\pi\left[\theta_{11}, \theta_{12}, \cdots \theta_{1k}, \theta_{21}, \theta_{22}, \cdots \theta_{2k} \mid D\right], \text{ where } k = 20,$$

of θ_{SR}'s is then proportional to the product of the *prior* probability and the likelihood function, i.e.,

$$\pi\left[\theta_{11}, \theta_{12}, \cdots \theta_{1k}, \theta_{21}, \theta_{22}, \cdots \theta_{2k} \mid D\right] \propto \prod_{S=1}^{2} \prod_{R=1}^{20} \theta_{SR}^{f_{SR}-1},$$

where $k = 20$.

Property 1 [14]. The marginal posterior density function of $\theta_{1\bullet}$ is beta with parameters $f_{1\bullet}$ and $f_{2\bullet}$.

Property 2 [14]. The marginal posterior density function of $\frac{\theta_{1R}}{\theta_{\bullet R}}$ is beta with parameters f_{1R} and f_{2R} .

Hence, the expected value of $I(S=1,2;R)$, given data D, is

$$\begin{aligned}
& E[I(S=1:2;R) \mid D] \\
= \; & E\left[\log\frac{\theta_{1R}}{\theta_{2R}} - \log\frac{\theta_{1\bullet}}{\theta_{2\bullet}} \mid D\right] \\
= \; & E\left[\log\frac{\theta_{1R}}{\theta_{\bullet R}-\theta_{1R}} - \log\frac{\theta_{1\bullet}}{1-\theta_{1\bullet}} \mid D\right] \\
= \; & E\left[\log\frac{\theta_{1R}/\theta_{\bullet R}}{1-\left(\theta_{1R}/\theta_{\bullet R}\right)} - \log\frac{\theta_{1\bullet}}{1-\theta_{1\bullet}} \mid D\right] \\
= \; & E\left[\log\frac{\theta_{1R}/\theta_{\bullet R}}{1-\left(\theta_{1R}/\theta_{\bullet R}\right)} \mid D\right] - E\left[\log\frac{\theta_{1\bullet}}{1-\theta_{1\bullet}} \mid D\right] \\
= \; & \int_0^1 \left(\log\frac{x}{1-x}\right) g\left(x; f_{1R}, f_{2R}\right) dx - \int_0^1 \left(\log\frac{y}{1-y}\right) g\left(y; f_{1\bullet}, f_{2\bullet}\right) dy \\
= \; & T\left(f_{1R}, f_{2R}\right) - T\left(f_{1\bullet}, f_{2\bullet}\right), \qquad (2.5)
\end{aligned}$$

where

$$g(u;m,n) = \frac{(m+n-1)\,!}{(m-1)!\,(n-1)!} u^{m-1}(1-u)^{n-1}$$

is the beta density with parameters $m>0$ and $n>0$ and where

$$T(m,n) = \int_0^1 \left(\log\frac{x}{1-x}\right) g(x;m,n)\, dx.$$

Property 3. $T(m,n) = \#(\mathrm{m}) - \#(\mathrm{n})$, where both m and n ≥ 2 and where

$$\#(k) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k-1}. \qquad (2.6)$$

Property 3 is a good result that will make application of the information theory method relatively easy. For this reason, we will give the proof of Property 3 here.

Proof of property 3:

$$\begin{aligned}
T(m,n) &= \int_0^1 \log\frac{u}{1-u} g(u;m,n)\,du \\
&= \frac{(m+n-1)!}{(m-1)!(n-1)!}\left\{\int_0^1 (\log u)\,u^{m-1}(1-u)^{n-1}\,du\right. \\
&\quad \left. -\int_0^1 [\log(1-u)]\,u^{m-1}(1-u)^{n-1}\,du\right\} \\
&= \frac{(m+n-1)!}{(m-1)!(n-1)!}\left[\int_0^1 (\log u)\,u^{m-1}(1-u)^{n-1}\,du\right. \\
&\quad \left. -\int_0^1 (\log v)(1-v)^{m-1}v^{n-1}\,dv\right] \\
&= \frac{(m+n-1)!}{(m-1)!(n-1)!}[U(m,n)-U(n,m)],
\end{aligned}$$

where

$$U(m,n)=\int_0^1 (\log u)\,u^{m-1}(1-u)^{n-1}\,du.$$

Integrating by parts, we have

$$\begin{aligned}
U(m,n) &= \int_0^1 (\log u)\,u^{m-1}(1-u)^{n-1}\,du \\
&= \left[\frac{u^m}{m}(\log u)(1-u)^{n-1}\right]_{u=0}^{u=1} \\
&\quad -\int_0^1 \frac{u^{m-1}}{m}(1-u)^{n-1}\,du \\
&\quad +\int_0^1 \frac{u^m}{m}(\log u)(n-1)(1-u)^{n-2}\,du \\
&= 0-\frac{1}{m}\int_0^1 u^{m-1}(1-u)^{n-1}\,du \\
&\quad +\frac{n-1}{m}\int_0^1 u^m(\log u)(1-u)^{n-2}\,du \\
&= \frac{-(m-1)!(n-1)!}{m(m+n-1)!}+\frac{n-1}{m}U(m+1,n-1).
\end{aligned}$$

Multiplying both sides by $\frac{(m+n-1)!}{(m-1)!(n-1)!}$, we have the recursive formula

$$
\begin{aligned}
& \frac{(m+n-1)!}{(m-1)!\,(n-1)!}U(m,n) \\
= \; & -\frac{1}{m}+\frac{(m+n-1)!}{m!\,(n-2)!}U(m+1,n-1) \\
= \; & -\frac{1}{m}-\frac{1}{m+1}+\frac{(m+n-1)!}{(m+1)!\,(n-3)!}U(m+2,n-2) \\
= \; & \cdots \\
= \; & -\left(\frac{1}{m}+\frac{1}{m+1}+\cdots+\frac{1}{m+n-2}\right) \\
& +\frac{(m+n-1)!}{(m+n-2)!0!}U(m+n-1,1) \\
= \; & -\left(\frac{1}{m}+\frac{1}{m+1}+\cdots+\frac{1}{m+n-2}\right) \\
& +\frac{(m+n-1)!}{(m+n-2)!0!}\int_0^1 (\log u)\,u^{m+n-2}\,du \\
= \; & -\left(\frac{1}{m}+\frac{1}{m+1}+\cdots+\frac{1}{m+n-2}\right) \\
& +(m+n-1)\left[\left(\frac{u^{m+n-1}}{m+n-1}\log u\right)_0^1-\int_0^1\frac{u^{m+n-2}}{m+n-1}\,du\right] \\
= \; & -\left(\frac{1}{m}+\frac{1}{m+1}+\cdots+\frac{1}{m+n-2}\right)-(m+n-1)\left[\frac{u^{m+n-1}}{(m+n-1)^2}\right]_0^1 \\
= \; & -\left(\frac{1}{m}+\frac{1}{m+1}+\cdots+\frac{1}{m+n-2}+\frac{1}{m+n-1}\right).
\end{aligned}
$$

Hence,

$$
\begin{aligned}
T(m,n) &= -\left(\frac{1}{m}+\frac{1}{m+1}+\cdots+\frac{1}{m+n-2}+\frac{1}{m+n-1}\right) \\
&\quad +\left(\frac{1}{n}+\frac{1}{n+1}+\cdots+\frac{1}{n+m-2}+\frac{1}{n+m-1}\right) \\
&= \#(m)-\#(n).
\end{aligned}
$$

Note 1. Because Eq.2.6 is valid only for m and n $\geq$ 2, Robson *et al.* set $f_{1R}=2$ whenever $f_{1R}=1$ or 0. Let a similar condition hold for f_{2R}, $f_{1\bullet}$, and $f_{2\bullet}$.

After this modification, we can use Eq. (2.6) to find the posterior expected value of $I(S=1:2;R)$ from Eq. (2.5).

Note 2. For large m and n,

$$\log \frac{m}{n} = \int_n^m \frac{1}{x}\,dx \simeq T(m,n),$$

and Eq. (2.4) and (2.5) are approximately equal.

Now, go back to Eq. (2.2). Instead of using the prediction scheme

$$S_j = \begin{cases} A & \text{for } I_1 - I_2 > 0, \\ \bar{A} & \text{for } I_1 - I_2 < 0, \end{cases}$$

Robson *et al.* use that of

$$S_j = \begin{cases} A & \text{for } I_1 - I_2 > \text{DC}, \\ \bar{A} & \text{for } I_1 - I_2 < \text{DC}, \end{cases}$$

where DC is an **adjustable parameter** chosen to improve prediction.

Note 3. Chou & Fasman's conformation parameters can be expressed as

$$P_\alpha = \frac{P(S=\alpha \mid R)}{P(S=\alpha)} = e^{I(S=\alpha;R)}$$

where $P(S=\alpha \mid R)$ is the probability that the structure of the residue R is alpha-helix and $P(S=\alpha)$ is the probability that the structure of a residue is alpha-helix, and as

$$P_\beta = \frac{P(S=\beta \mid R)}{P(S=\beta)} = e^{I(S=\beta;R)}.$$

The information theory method considers the influence of residues at different positions on the conformation of the middle amino acid in a block of amino acids and so predicts one amino acid each time. Although information regarding the different

amino acids at different positions is not included in the conformational parameters, this method predicts the structures of several sequential amino acids every time. (Four sets of conformational parameters are constructed that are different from the two sets of conformational parameters we have introduced, in terms of the influence of amino acids on the N-terminal end or on the C-terminal end of the conformation of the entire block. The program for Chou & Fasman's method, mentioned before, used only the two sets of parameters that we have introduced.)

2.4 Levin *et al.*'s Similarity Matrix

Applying the assumption that short homologous sequences of amino acid have identical secondary structure tendencies and using a symmetric matrix called a secondary structure similarity matrix, Levin *et al.* [9] assigned sequence similarity scores between all two sequences seven residues in length. The empirically determined similarity matrix(Table 2.2) was developed and optimized using the Kabsch & Sander database (See note 3 below).

Example. The similarity score between the sequences STNGIYW and ATSLVFW is 1(S and A) + 2(T and T) + 0(N and S) +(-1)(G and L)+ 1(I and V) + 1(Y and F) + 2(W and W) = 6.

Example. The similarity score between the sequences STNGIYW and ATSGVFL is 1 + 2 + 0 + 2 + 1 + 1 + 0 = 7.

Definition. A **training set** is a set of proteins with known structures that is used to establish a prediction scheme. The set that is used to test the performance of a prediction scheme is called a **testing set.**

Prediction requires four steps:

Table 2.2: Levin *et al.*'s secondary structure similarity matrix.

	G	P	D	E	A	N	Q	S	T	K	R	H	V	I	M	C	L	F	Y	W
G	2	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	0	-1	-1	-1	-1
P	0	3	0	-1	-1	0	0	0	0	0	0	0	-1	-1	-1	0	-1	-1	-1	-1
D	0	0	2	1	0	1	0	0	0	0	0	0	-1	-1	-1	0	-1	-1	-1	-1
E	0	-1	1	2	1	0	1	0	0	0	0	0	-1	-1	-1	0	-1	-1	-1	-1
A	0	-1	0	1	2	0	0	1	0	0	0	0	0	0	0	0	0	-1	-1	-1
N	0	0	1	0	0	3	1	0	0	1	0	0	-1	-1	-1	0	-1	-1	-1	-1
Q	0	0	0	1	0	1	2	0	0	0	0	0	-1	-1	-1	0	-1	-1	-1	-1
S	0	0	0	0	1	0	0	2	0	0	0	0	-1	-1	-1	0	-1	-1	-1	-1
T	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	-1	-1	-1
K	0	0	0	0	0	1	0	0	0	2	1	0	-1	-1	-1	0	-1	-1	-1	-1
R	0	0	0	0	0	0	0	0	0	1	2	0	-1	-1	-1	0	-1	-1	-1	0
H	0	0	0	0	0	0	0	0	0	0	0	2	-1	-1	-1	0	-1	-1	0	-1
V	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	2	1	0	0	1	0	0	0
I	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	1	2	0	0	0	1	0	0
M	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	0	0	2	0	2	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	-1	-1	-1
L	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	1	0	2	0	2	0	0	0
F	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	1	0	-1	0	2	1	0
Y	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	-1	0	1	2	0
W	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	0	0	0	-1	0	0	0	2

Step 1. Choose a seven amino acid sequence from the testing set. (The first block chosen from a protein is composed of the first seven residues on the N-terminal end. To choose the second sequence, and so on, shift along the protein sequence one residue.)

Step 2. Use the similarity matrix to find the similarity score between the amino acid block chosen in step 1 and every amino acid block in the training set. A block in the training set is accepted if the score is greater than or equal to seven; otherwise, it is rejected. For an accepted block, the score is the credit that the corresponding residues in the two blocks have the same conformation. Accumulate the credits for all types of conformation for each residue in the testing block.

Step 3. If every block in the testing set has been chosen for a comparison (in step 2), then go to step 4. Otherwise, go to step 1.

Step 4. For each residue in the testing set, compare the "sums of credits" for all types of conformation. The conformation assigned to the observed residue is the conformation with the greatest value.

Note 1. In step 4, each "sum of credits" is multiplied by a different constant before being compared. The constants, called **decision constants**, are different for different conformations. The purpose is " to avoid overprediction of helix and underprediction of aperiodic structure" ([9]); page 305).

Note 2. It was observed by Levin and Garnier that if the number of amino acids in a block is increased from seven to eight, the percentage accuracy will fall. If no cut-off (i.e., number 7 in step 2) is used, the entire protein sequence will be predicted as aperiodic (the most commonly observed conformation in the training set).

Note 3. Initially, an arbitrary assignment for the matrix was made in order to

construct the similarity matrix. The principal diagonal entries were 2's, 1 was given for pairs of amino acids considered to have properties in common, −1 was given for dissimilar amino acid pairs, and 0 was given elsewhere. The initial matrix was then optimized by making "rational changes" [9] and by observing their effects on prediction accuracy.

2.5 Qian & Sejnowski's Feedforward Neural Network Models

A feedforward neural network model is composed of two or more layers of processing units with feedforward connections from all the units in one layer to those units one layer above. The bottom layer is the input layer, the top layer is the output layer, and the other layers are hidden layers (See Figure 2.1) .

A processing unit sums the signals presented to it, with weights assigned to each input signal, and computes an output to be sent to the next layer.

In Sejnowski and Qian's models[10], inputs are the strings of contiguous amino acids, and outputs are the conformational tendencies of the middle amino acids in the input strings. For a block of length 13, for example, the input layer consists of 13 groups of processing units, with 20^1 units in each group. Each input group encodes one amino acid of the input block, and each unit in a group represents one of the 20 amino acids. Thus, only one unit in each input group is active, which will give an output 1, and the output of the other units is 0. This is called a local encoding

[1] Sejnowski and Qian, in fact, used 21 units in each group to predict the structures of the first and the last six (for block length 13) amino acids in each protein, the 21st unit being associated with a spacer. Note that the sequence of amino acids was concatenated to form a long string for each of the training and testing sets, with spacers between the proteins to separate them during training.

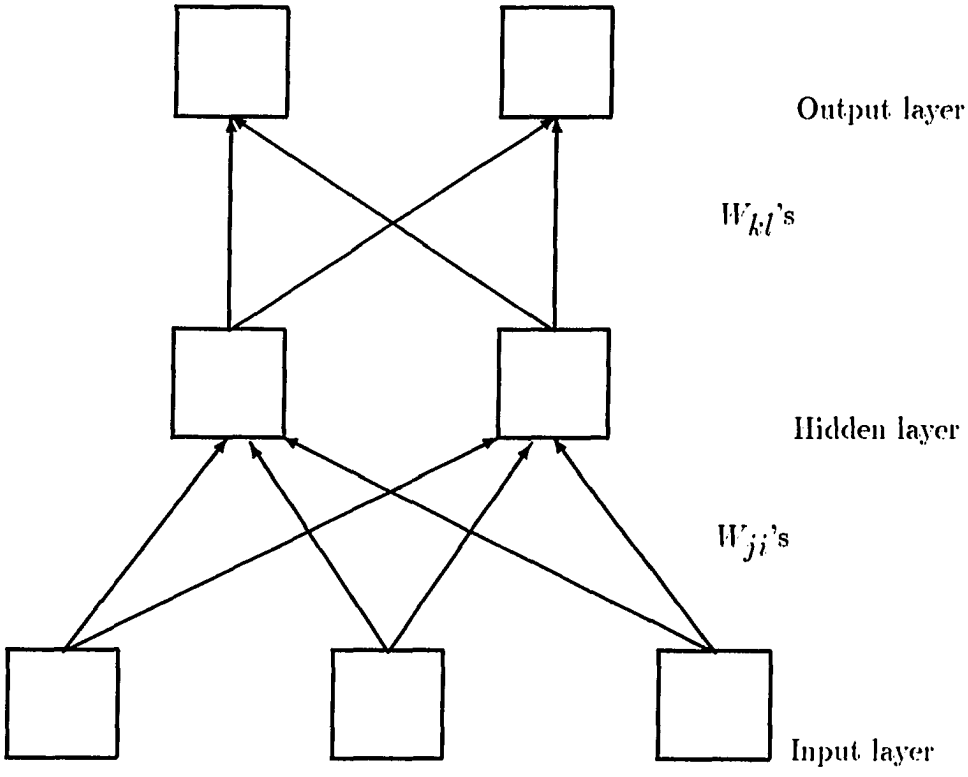

Figure 2.1: A feedforward neural network with one hidden layer. The bottom layer is the input layer, the top layer is the output layer, and the layer between is the hidden layer. W_{ji} is the weight assigned to the connection from the ith unit in the input layer to the jth unit in the hidden layer.

scheme.

For a unit i, say, which is not in the input layer, the total input E_i to unit i and the output (or state) S_i from unit i are determined by means of equations

$$E_i = \left(\sum_j W_{ij} S_j \right) + b_i$$

and

$$S_i = F\left(E_i\right) = \frac{1}{1 + e^{-E_i}},$$

where W_{ij} is the strength of the connection from unit j to unit i, j's are those units one layer below unit i, S_j is the output (or state) of unit j ,and b_i is the bias of unit i.[2]

The output layer consists of three units (that is, for three-state prediction; it consists of two units for two-state prediction) representing the secondary structures alpha-helix, beta-sheet, and coil. The structure corresponding to the greatest value among the outputs of the output units was assigned to the middle amino acid in the input protein block.

To train a neural network is to modify the weights between layers of units (and biases in units) to obtain a desired input-output mapping (i.e., type of middle amino acid of blocks and their secondary structures). The input set (or training set) contains blocks of amino acids chosen from proteins whose secondary structures are determined, and the output set contains three types of secondary structures (alpha-helix, beta-sheet, and coil) for three-state prediction (or two types—alpha-helix and coil—for two-state prediction). In Qian and Sejnowski[10], "Initially, the weights in the neural network were assigned randomly with values uniformly distributed in

[2]If unit i is in the input layer, i.e., an input unit, then $S_i = E_i$.

the range [-0.3,0.3]." Then a block chosen from the training set was input, and the output observed. If a desired output resulted, the original weights were kept and another block input. Otherwise, weights were modified[3] by means of the **learning algorithm** described below, and another block was input. This process was repeated until a "good result" was achieved.

The learning algorithm used incorporated the **generalized delta rule** and **back-propagation** [11]. For a pattern p (or for a block of amino acids here) with inappropriate mapping results, the learning algorithm increases the current weights W_{ji}'s by the amounts $\Delta_p W_{ji}$'s, where[4]

$$
\begin{aligned}
\Delta_p W_{ji} &= \eta \delta_{pj} o_{pi}, \\
o_{pi} &\quad \text{is the output of unit } i, \\
\delta_{pj} &= \begin{cases} \left(t_{pj} - o_{pj}\right) f'\left(net_{pj}\right) & \text{if unit } j \text{ is in the output layer,} \\ \left(\sum_k \delta_{pk} W_{kj}\right) f'\left(net_{pj}\right) & \text{if unit } j \text{ is in the hidden layer,} \end{cases} \\
f(x) &= \frac{1}{1+e^{-x}} \text{ is the activative function,} \\
net_{pj} &= \sum_i W_{ji} o_{pi} \text{ is the total input to unit } j, \\
t_{pj} &\quad \text{is the target output of the output unit } j \qquad (2.7) \\
&\quad \text{(i.e., j is in the output layer;} \\
&\quad t_{pj} = 1 \text{ if the structure of the middle amino acid in p is simply} \\
&\quad \text{the structure represented by unit } j, \text{ and 0 otherwise.), and}
\end{aligned}
$$

[3] Same as below (next footnote).

[4] According to Sejnowski and Rosenberg[4], "To reduce the average error for all the input patterns, these gradients must be averaged over all the training patterns before updating the weights. In practice, it is sufficient to average over several inputs before updating the weights."

η is the constant of proportionality representing the learning rate.

The bias b_i for a unit i can be considered the weight between i and a unit, with fixed output 1 connected to i only.

3. PARTITION IN LINEAR SPACE

3.1 Introduction

We will describe two different protein-sequence encoding schemes that will be used in this and the next chapters. Then we will apply the encoding schemes to discuss the works mentioned in Chapter 2.

3.2 Transformation from Alphabetical Segment to Linear Space — Encoding Schemes

The least complicated general encoding scheme is that of assigning a quantity such as hydrophobicity to each amino acid. In this manner, an amino acid segment of length k, say, corresponds to a k-dimensional real vector or to a single point in R^k. We will use this encoding scheme to conduct certain experiments in the next chapter.

The most general encoding scheme, that is, the local encoding scheme, is to assign each amino acid to a 20-dimensional unit vector. We arrange the 20 amino acids in the order A,R,N,D,C,Q,E,G,H,I,L,K,M,F,P,S,T,W,Y,V and then use the ith column of the 20×20 identity matrix to represent the ith capital letter or amino acid. So an amino acid segment of length k corresponds to a single point in the (20k)-dimensional real space (R^{20k}) with the ith, say, 20 coordinates representing the ith amino acid

counted from the N-terminal end of the segment. In some instances, we will also use a $20 \times k$ matrix to represent a k amino acid sequence in length with its ith column corresponding to the ith amino acid counted from the N-terminal end of the segment.

3.3 Conformation Parameters

Chou and Fasman[3] established, in total, 19 sets of conformational parameters (P_α, P_β, etc.) "$\cdots$ showing the conformational potentials for 20 amino acids in the helical, beta-sheet, coil, and beta-turn regions, as well as the frequency of residues at helix and beta-sheet boundary and central regions." Each set of parameters contains 20 real numbers, and each real number corresponds to one of the 20 amino acids. As mentioned above regarding the least complicated encoding scheme assigning hydrophobicity to each amino acid, each set of conformational parameters may be used to assign a "number" to each of the 20 amino acids. Then each of the sets of 20 numbers may be tested for the ability to represent quantitatively the secondary structure tendency of amino acids. In the next chapter, we will describe an experiment attempting to establish an optimum set of 20 numbers, called a similarity scale, from which to assign a value to each amino acid. This experiment arose from the question of whether we can construct only one set of 20 numbers that will give us sufficient information about protein secondary structures and that can be used to make objective predictions.

3.4 Information Theory Method

From Eq. (2.2), i.e.,

$$I_1 - I_2 \simeq \sum_{i=-m}^{m} I\left(S_j = A : \bar{A}; R_{j+i}\right),$$

the information theory method is used to construct a table T, say, of 20(2m + 1) discrete numbers for two-state predictions. Twenty is the number of different amino acids and the number of rows in T. 2m + 1 is the length of protein blocks considered and the number of columns in T. We can rearrange the 20(2m + 1) numbers into a 20(2m + 1)-dimensional column vector **N** such that the first 20 entries of **N** constitute the first column of T, the second 20 entries of **N** the second column of T, and so on. If we use the most general encoding scheme to encode blocks of (2m + 1) amino acids in length such that each segment is represented by a single point in 20(2m + 1)-dimensional real space, then the structure of the middle amino acid of an observed segment **P** is predicted to be

a) alpha-helical if $\mathbf{N} \bullet \mathbf{P} + DC > 0$ and

b) coil otherwise,

where DC, or the **decision constant**, is a fixed real number and where $\bullet$ is the usual dot product in $R^{20(2m+1)}$. Geometrically, the information theory method ultimately constructs a plane in 20(2m + 1)-dimensional real space to "separate" alpha-helical amino acids from coil amino acids. But the information theory is used to find a normal vector only, not to find a decision constant. According to Garnier *et al.*, "DC is an adjustable parameter which is chosen with the aim of producing optimal predictions. It is a function only of $S_j \cdots$" (page 105). Moreover, "the choice of a correct set of decision constants is a definitive step towards improvement

of a predictive program. Whatever its physical significance, the improvement is such that this choice is critical" (page 112). Thus, to choose the best decision constant for two-state prediction is to "shift" a plane, whose direction is determined by the information theory method, thereby obtaining a best or sufficiently good partition result, i.e., minimal misplaced points.

In the next chapter, we will describe a linear programming model attempting to find an acceptable normal vector **N** and a constant DC simultaneously to constitute a plane.

3.5 Neural Network Models

Qian and Sejnowski[10] used several neural network models including one layer, two layers, and three layers of weights. Seemingly, "networks with no hidden units performed as well as networks with hidden units on the non-homologous training set..." (page 881). So we concentrate on the neural network model without hidden units. Note that Qian and Sejnowski performed a three-state prediction, which will be discussed in Chapter 4. For convenience, we will discuss here a model for two-state prediction which is similar to one for three-state prediction.

A two-state prediction neural network model for segments 2m + 1 amino acids in length contains 20(2m + 1) input units, two output units (one for alpha helix and the other for coil) and two groups of weights connected from input units to output units. We can consider the two groups of weights as two 20(2m + 1)-dimensional real vectors N_1 (for alpha-helix) and N_2 (for coil), say, and let the biases associated with N_1 and N_2 be b_1 and b_2, respectively. As in the previous section, we use the most general encoding scheme. Then the secondary structure of the middle amino acid of

observed block **P** is predicted to be alpha-helical if

$$\mathbf{N_1 \bullet P} + b_1 > \mathbf{N_2 \bullet P} + b_2,$$

i.e.,

$$\mathbf{N \bullet P} + b > 0,$$

where

$$\mathbf{N} = \mathbf{N_1} - \mathbf{N_2} \text{ and } b = b_1 - b_2;$$

and coil otherwise. So the network is used to construct a plane, i.e., to find **N** and b, as was done in the information theory method.

Qian and Sejnowski gave an example of two sets of input-output mapping having the same joint probability[1] but that can be learned by two different networks[10](page 869). They thus claimed that "this observation will be used to explain why the neural network method yields better results than the information theory method of Robson and Suzuki (1976)" [10] (page 869).

One difference between the information theory method and the neural network model is the means of finding the constant, which is called the **decision constant** in the information theory method and the **bias** in the neural network model. As mentioned in Chapter 2, biases can be learned as well as weights, but the decision constant is not determined by the information theory method. Instead, Robson *et al.* used

> a rapid optimization program, capable of making tens of thousands of predictions for various combination of decision constants...

[1] This means that the information theory method will result in the same planes for these two sets.

> This program was designed to trace the highest percentage of correctly predicted residues for the four conformations by varying the decision constants independently for alpha-helix, beta-sheet, extended chain, and reverse turns. [8](page 108)

Note: Use one decision constant for two-state prediction and three decision constants for four-state prediction. This methodology will be discussed in Chapter 4.

3.6 Discussion

About Chou and Fasman's conformational parameters method, Garnier and Robson [8] stated, "Unfortunately, some of their rules are qualitative rather than quantitative, and being open to interpretation they have not always yielded such promising results in the hands of other workers..."(page 98).

Chou and Fasman[3] themselves remarked that "the predicted conformation is more reliable if all the prediction rules [2] of Chou & Fasman are followed and quantitative calculations performed for the $< P_\alpha >$, $< P_\beta >$, $< P_t >$, and P_t values of the predicted secondary structures" (page 139).

Additionally, Qian and Sejnowski pointed out that the "existing methods for predicting secondary structures" were not entirely reliable although they cited Robson *et al.*'s as the most reliable.

Sejnowski and Qian[10] used 106 proteins, which was called a training set, to train their neural networks and then used 15 proteins, which was called a testing set and was **nonhomologous** with the corresponding training set, to test the performance of networks. Protein secondary structure assignments were from Kabsch &

[2] Only 2 out of 19 [3] sets of conformational parameters were used in the program describing Chou & Fasman's prediction rules.

Sander and based upon the Brookhaven databank of protein structures. The average success rates, or

$$\frac{\text{Total number of correctly predicted amino acids}}{\text{Total number of predicted amino acids}},$$

for Qian & Sejnowski 's testing set, are 53% for Robson's information theory method, 50% for Chou-Fasman method, and 62.7% for the Qian-Sejnowski neural network method[3] [10]. Evidently, the performances of Robson's and of Chou & Fasman's methods were not as successful as they had announced. According to Qian and Sejnowski [10],

> The original measures of accuracy reported by these authors were based in part on the same proteins from which they derived their method, and these proteins are equivalent to our training set.
>
> However, these methods should be compared on proteins with structures that were not used in or homologous with those in the training set. (page 878)

A good prediction scheme should be quantitative and objective. Good performance, although the goal, is not the only factor requiring consideration. Different researchers should obtain the same prediction results when using the same method on the same database. With this fact in mind, we apply a powerful mathematical tool—**linear programming**— to develop models with which to study the secondary structure of proteins. We consider all amino acids in the training set at the same time but do not consider the performance on the testing set during "training."

[3]Kabsch and Sander [2] used Chou & Fasman's methods to test 62 proteins with more than 1000 residues. For three-state prediction, the overall prediction accuracy is 55%–56% for Robson's method and 50% for Chou & Fasman's.

Even Qian and Sejnowski introduced subjectivity into their method. In [10], "All the amino acids in the training set were sampled once before starting again. This random sampling process was adopted to prevent erratic oscillations in the performance that occured when the amino acids were sequentially sampled. The performance of the network on the testing set was monitored frequently during training and the set of weights was kept that achieved the best average success rate on the testing set."(page 871) This is a difference between Linear Programming Models and Neural Network Models.

To conclude this chapter, let us quote Qian & Sejnowski's[10] comment about Levin *et al.* 's [9] similarity matrix:

> ... our method should be faster because a set of weights obtained through training can be used for predicting secondary structures for all new proteins. The method of Levin *et al.*, on the other hand, requires an exhaustive search of the whole database for every seven-amino acid sequence in the new protein. (page 879)

4. LOCAL STUDY OF PROTEIN SEQUENCES IN SEGMENTS

4.1 Introduction

In the first two sections, we represent a k-amino acid sequence by means of a $20 \times k$ matrix (see Chapter 3) and to construct both a similarity scale and a similarity matrix.

In the last two sections, we relate a segment of length k to a single point in the 20k-dimensional real space and search for partition planes.

Slightly modified training and testing sets of Qian & Sejnowski are used to develop and to test prediction schemes, respectively. Proteins in the testing set are nonhomologous with proteins in the training set (see Appendix A).

Segments of amino acids in a protein are chosen by our shifting along the protein from the N-terminal end to the C-terminal end one amino acid at a time. The prediction schemes point to the secondary structure of the middle amino acid in a segment; thus, the first $\frac{k-1}{2}$ and the last $\frac{k-1}{2}$ amino acids of a protein in the testing set, in which k is the number of amino acids in a segment, will not be predicted. Moreover, information regarding the secondary structures of amino acids in the "head" part or the "tail" part of a protein in the training test is not used.

The accuracy of a prediction result is described by means of the fraction

$$\frac{\text{Total number of correctly predicted amino acids}}{\text{Total number of predicted amino acids}}.$$

In the last two sections, we obtain a "plane" partitioning and the accuracy of a partition result is described by means of the fraction

$$\frac{\text{Total number of correctly placed amino acids}}{\text{Total number of located amino acids}}.$$

4.2 Similarity Scale for Two-state Prediction

4.2.1 Introduction

The similarity scale V, a 20-dimensional column vector derived, by the method described in next section, from the Dayhoff substitution matrix, is used to assign a real number to each amino acid. Thus, a k-amino acid-long string, represented by a $20 \times k$ matrix M (as mentioned in Chapter 3), is transformed into either a k-dimensional column vector or a single point $M^T V$ in R^k.

The similarity scale is designed to cause a relatively small Eucledian distance, transformed from two similar strings, between the two points. Because the 20×20 Dayhoff substitution matrix (Table 4.1) is often used to determine sequence homology, it was used first to measure the similarity of two amino acid strings.

4.2.2 Method

Let $D(M, N)$ denote the Dayhoff similarity score [1] of two amino acid segments, say A and B, represented by matrices M and N, respectively. We say that the two

[1] The method used to evaluate the Dayhoff similarity score of two strings is similar to Levin *et al.*'s similarity matrix.

Table 4.1: Dayhoff's substitution matrix.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-4	1	1	1	-6	-3	0
R	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2
N	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-4	-1	1	0	-4	-2	-2
D	0	-1	2	4	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2
C	-2	-4	-4	-5	12	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2
Q	0	1	1	2	-5	4	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2
E	0	-1	1	3	-5	2	4	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2
G	1	-3	0	1	-3	-1	0	5	-2	-3	-4	-2	-3	-5	-1	1	0	-7	-5	-1
H	-1	2	2	1	-3	3	1	-2	6	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5	2	-2	2	1	-2	-1	0	-5	-1	4
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-3	-4	-2
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6	0	-2	-2	-1	-4	-2	2
F	-4	-4	-4	-6	-4	-5	-5	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1
P	1	0	-1	-1	-3	0	-1	-1	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-2	-3	-1
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17	0	-6
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	-2
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

strings A and B are similar when the Dayhoff similarity score $D(M,N)$ is relatively great and dissimilar when $D(M,N)$ is small. Suppose that P and Q are the two points in R^k that were transformed from A and B, respectively. Then the square of the distance between P and Q is equal to

$$V^T(M-N)(M-N)^T V,$$

a relatively small distance if and only if $D(M,N)$ is relatively great.

The best similarity scale, or V, is supposed to

1. $$\text{MINIMIZE}_{\|V\|_2=1} \sum_{i<j} V^T\left(M_i - M_j\right)\left(M_i - M_j\right)^T V, \tag{4.1}$$

where M_i's and M_j's are $20 \times k$ matrices representing k-amino acid sequences chosen from the training set satisfying the two conditions

(a) $D\left(M_i, M_j\right) \geq S_1$, where S_1 is a fixed number, and

(b) M_i and M_j represent middle amino acid alpha-helical strings;

2. $$\text{MINIMIZE}_{\|V\|_2=1} \sum_{i<j} V^T\left(N_i - N_j\right)\left(N_i - N_j\right)^T V, \tag{4.2}$$

where N_i's and N_j's are $20 \times k$ matrices representing k-amino acid sequences chosen from the training set satisfying the two conditions

(a) $D\left(N_i, N_j\right) \geq S_2$, where S_2 is a fixed number, and

(b) N_i and N_j represent middle amino acid non-alpha-helical strings;

and

3. $$\text{MAXIMIZE}_{\|V\|_2=1} \sum_{i<j} V^T\left(M_i - N_j\right)\left(M_i - N_j\right)^T V, \tag{4.3}$$

where M_i's and N_j's are $20 \times k$ matrices representing k-amino acid sequences chosen from the training set satisfying the two conditions

(a) $D\left(M_i, N_j\right) \leq S_3$, where S_3 is a fixed number, and

(b) M_i and N_j represent strings for which the middle amino acid has different secondary structures.

Note: Condition 1 (a) is used to collect similar strings in the sense that their Dayhoff similarity score is greater than or equal to a fixed number. Similar expression hold for conditions 2 (a) and 3 (a).

Under the same conditions[2], combining 1, 2, and 3 and attempting to find an acceptable V, we

$$\begin{aligned}
&\text{MINIMIZE}_{\|V\|_2=1} \left[\sum_{i<j} V^T \left(M_i - M_j\right)\left(M_i - M_j\right)^T V \right. \\
&+ \sum_{i<j} V^T \left(N_i - N_j\right)\left(N_i - N_j\right)^T V \\
&\left. - \sum_{i<j} V^T \left(M_i - N_j\right)\left(M_i - N_j\right)^T V \right] \\
= \ &\text{MINIMIZE}_{\|V\|_2=1} \ V^T \left[\sum_{i<j} \left(M_i - M_j\right)\left(M_i - M_j\right)^T \right. \\
&+ \sum_{i<j} \left(N_i - N_j\right)\left(N_i - N_j\right)^T \\
&\left. - \sum_{i<j} \left(M_i - N_j\right)\left(M_i - N_j\right)^T \right] V \\
= \ &\text{MINIMIZE}_{\|V\|_2=1} \ V^T (M + N - W) V, \qquad (4.4)
\end{aligned}$$

[2]Condition 1 (a) only holds for the matrix M, condition 2 (a) only holds for the matrix N, and condition 3 (a) only holds for the matrix W in Eq. 4.4.

where

$$M = \sum_{i<j} \left(M_i - M_j\right)\left(M_i - M_j\right)^T \text{ and}$$

$$N = \sum_{i<j} \left(N_i - N_j\right)\left(N_i - N_j\right)^T,$$

and where

$$W = \sum_{i<j} \left(M_i - N_j\right)\left(M_i - N_j\right)^T.$$

Because M + N − W is a 20×20 real, symmetrical matrix, according to the spectrum decomposition theorem [15],

$$M + N - W = \sum_{i=1}^{20} \xi_i V_i V_i^T,$$

where the real numbers ξ_i are the eigenvalues and the vectors V_i are associated orthonormal eigenvectors of the matrix M+N−W. Therefore, the optimal V satisfying Eq. 4.4 is the unit eigenvector corresponding to the smallest eigenvalue of the matrix M+N−W.

An acceptable V can be obtained by means of another approach. In Eq. 4.3, the matrix

$$\left(M_i - N_j\right)\left(M_i - N_j\right)^T$$

is positive definite, as is the matrix

$$W = \sum_{i<j} \left(M_i - N_j\right)\left(M_i - N_j\right)^T = \sum_{i=1}^{20} \eta_i O_i O_i^T,$$

where the positive numbers, η_i, are the eigenvalues associated with the eigenvectors O_i's of W. Thus, the inverse of W, denoted by W^{-1}, exists and can be expressed as

$$W^{-1} = \sum_{i=1}^{20} \frac{1}{\eta_i} O_i O_i^T.$$

The unit vector maximizing V^TWV and minimizing $V^T W^{-1}$V is the unit eigenvector corresponding to the largest eigenvalue of W, so an acceptable similarity scale V is the unit vector minimizing $V^T \left(M + N + W^{-1}\right) V$. Let V be the unit eigenvector corresponding to the smallest eigenvalue of the matrix $M + N + W^{-1}$.

Note that if we delete Eq. 4.2 and replace condition b in Eq. 4.1 by "M_i and M_j represent strings for which the middle amino acid has same secondary structures (alpha-helix, beta-sheet, or coil)", then the resulting matrices $C - W$ and $C + W^{-1}$ can be used to do 3-state predictions.

4.2.3 Prediction Procedure — Five Nearest Neighbors

According to similarity scale design, we use the Euclidean distance between two points in R^k, which have been transformed from two k-amino acid sequences, to measure the similarity of the secondary structure of the middle amino acids in the two segments.

To predict the structure of an amino acid in a new protein, we first locate in R^k the k-amino acid segment, which is chosen from the new protein with the observed residue in the middle, and then find its five nearest neighbors from the training set. The secondary structures of the middle residues in the five neighbors determine the structure of the target residue. If the middle residue of 4 (3 or 5) of the neighbors have the same structure, then this structure is assigned to the target residue (Fig. 4.1). Otherwise, no prediction is made.

Because the distances between a target residue and some of its neighbors may be too great, we set a fixed real number, called a **threshold**, as the maximum accepted distance between a target residue and its neighbors. The distance between a target

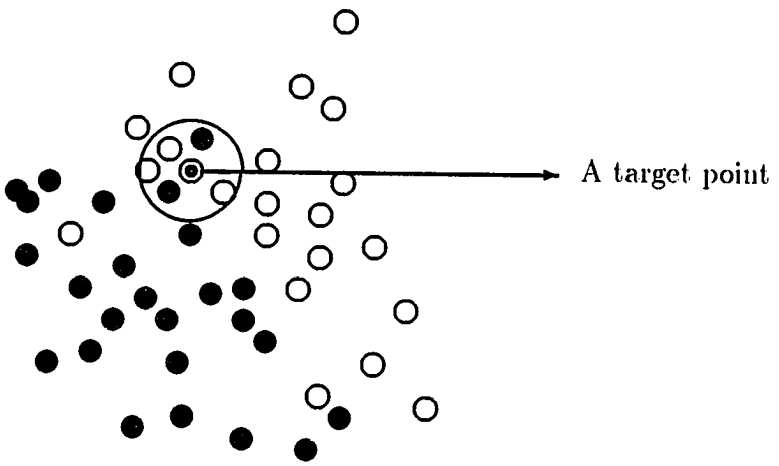

Figure 4.1: Five nearest neighbors.

residue and each of its five neighbors should be less than or equal to the threshold; otherwise no prediction can be made.

The information contained in the training set may or may not be sufficient to allow prediction of a residue in a new protein; it therefore seems reasonable to set a threshold and to avoid making predictions in certain instances.

4.2.4 Results and Discussion

The results using matrix $M + N - W$ for 7-amino acid segments in the testing set (see Appendix A) are listed in Table 4.2. Prediction results shown in column P indicate the overpredictions of coil. When U = D = 7, performances for thresholds of different values are slightly different. The derived similarity scale V is listed in Table 4.3. There are only five nonzero entries in V, and the five corresponding residues, A, R, N, D, and C, will dominate structure assignments. The similarity scale derived from the example (U, D) = (7, -7) is more acceptable and perform better than do the previously mentioned scale and is listed in Table 4.3.

The 3-state prediction results using matrix $C - W$ for 7-amino acid segments in the testing set are listed in Table 4.4. Prediction results shown in column P indicate the overpredictions of coil.

Table 4.5 presents results obtained using the matrix $M + N + W^{-1}$. Similarity scales are listed in Table 4.6. The 3-state prediction results using matrix $C + W^{-1}$ for 7-amino acid segments in the testing set are listed in Table 4.7. Coil was overpredicted as it was when the matrix $C - W$ was used.

Table 4.2: Two-state predictions using the similarity scale derived from the matrix $M + N - W$ and the 5-nearest neighbors method (Predict when 4 or 5 neighbors have same structure).

NP[a]	Thr[b]	U[c]	D[d]	P[e]		PA[f]	$\#_p$[g]
				73	410		
				(0.004359)[h]	(0.004359)[i]		
				(0.000000)[j]	(0.000000)[k]		
				144	1482		
				(0.004366)	(0.004359)		
7	0.5	7	7	(0.000000)	(0.000000)	74%	2109

[a] NP is the number of residues in a segment.

[b] Threshold.

[c] U is the minimal Dayhoff similarity score that two segments can be treated similar. That is $S_1 = S_2 = D$, where S_1 is the fixed number in condition 1 (a) and S_2 is the fixed number in condition 2 (a),

[d] D is maximal Dayhoff similarity score that two segments can be treated dissimilar. That is $S_3 = D$, where S_3 is the fixed number in condition 3 (a).

[e] Prediction results. See Table 4.16. There are 3312 7-amino acid segments in the testing set, which is listed in Appendix A and in which 842 are alpha-helical and 2470 are non-alpha-helical. Note that there are 4884 alpha-helical segments, which are segments with middle residue alpha-helix, and 11,956 coil segments in the training set, which is listed in Appendix A. There are 3162 17-amino acid segments in the testing set.

[f] PA = Prediction accuracy.

[g] $\#_p$ is the number of segments in the testing set that were predicted.

[h] The maximal distance between the correctly predicted alpha-helical segments and their neighbors.

[i] The maximal distance between the observed alpha-helical segments, which are predicted to be coil, and their neighbors.

[j] The minimal distance between the correctly predicted alpha-helical segments and their neighbors.

[k] The minimal distance between the observed alpha-helical segments, which are predicted to be coil, and their neighbors.

Table 4.2 (Continued.)

NP	Thr	U	D	P		PA	$\#_p$
				78	410		
				(0.999994)	(0.004359)		
				(0.000000)	(0.000000)		
				145	1482		
				(0.995634)	(0.004359)		
7	1	7	7	(0.000000)	(0.000000)	74%	2115
				78	410		
				(0.999994)	(0.004359)		
				(0.000000)	(0.000000)		
				145	1482		
				(0.995634)	(0.004359)		
7	5	7	7	(0.000000)	(0.000000)	74%	2115
				56	424		
				(0.063808)	(0.063829)		
				(0.000000)	(0.000000)		
				84	1531		
				(0.063833)	(0.063829)		
7	0.5	7	-7	(0.000000)	(0.000000)	76%	2095
				38	424		
				(0.400973)	(0.312849)		
				(0.000001)	(0.000000)		
				61	1469		
				(0.400972)	(0.282662)		
17	0.5	7	-7	(0.000000)	(0.000000)	76%	1992
				55	424		
				(0.047778)	(0.047788)		
				(0.000000)	(0.000000)		
				87	1505		
				(0.047790)	(0.047788)		
7	0.5	12	-12	(0.000000)	(0.000000)	75%	2071

Table 4.3: The similarity scales derived from the matrix $M+N-W$ in the examples in previous table.

residue	(U, D, NP)=(7, 7, 7)	(7, -7, 7)	(7, -7, 17)	(12, -12, 7)
A	0.997815	0.000000	0.000000	0.000000
R	-0.066023	0.000000	0.000000	0.000000
N	0.002535	0.000000	0.000000	0.000000
D	-0.000208	0.000000	0.000000	0.000000
C	0.000004	0.000000	0.000000	0.000000
Q	0.000000	0.000000	0.000003	0.000000
E	0.000000	-0.000008	-0.000090	0.000003
G	0.000000	0.000092	0.000725	-0.000040
H	0.000000	-0.000591	-0.001508	0.000290
I	0.000000	0.030991	0.071284	-0.016680
L	0.000000	-0.967061	-0.958101	0.975670
K	0.000000	0.252636	0.277383	-0.218602
M	0.000000	-0.002827	-0.004038	0.001829
F	0.000000	0.000034	0.000058	-0.000020
P	0.000000	-0.000002	-0.000003	0.000001
S	0.000000	0.000000	0.000000	0.000000
T	0.000000	0.000000	0.000000	0.000000
W	0.000000	0.000000	0.000000	0.000000
Y	0.000000	0.000000	0.000000	0.000000
V	0.000000	0.000000	0.000000	0.000000

Table 4.4: Three-state predictions using the similarity scale derived from the matrix $C-W$ and the 5-nearest neighbors method (Predict when 4 or 5 neighbors have same structures).

NP	Thr	U	D	P[a]			PA	$\#_p$
				55	22	158		
				(0.054492)	(0.001735)	(0.054492)		
				(0.000000)	(0.000000)	(0.000000)		
				30	34	146		
				(0.054519)	(0.001797)	(0.036515)		
				(0.000000)	(0.000000)	(0.000000)		
				50	57	419		
				(0.036515)	(0.054164)	(0.054507)		
7	0.5	7	-7	(0.036515)	(0.000000)	(0.000000)	52%	971
				56	25	151		
				(0.042106)	(0.000443)	(0.042106)		
				(0.000000)	(0.000000)	(0.000000)		
				31	38	152		
				(0.042117)	(0.000451)	(0.033850)		
				(0.000000)	(0.000000)	(0.000000)		
				53	64	408		
				(0.033850)	(0.042042)	(0.042114)		
7	0.5	12	-12	(0.033850)	(0.000000)	(0.000000)	51%	978

[a]See Table 4.33.

Table 4.5: Two-state predictions using the similarity scale derived from the matrix $M + N + W^{-1}$ and the 5-nearest neighbors method (Predict when 4 or 5 neighbors have same structures).

NP	Thr	U	D	P		PA	$\#_p$
				79	404		
				(0.000014)	(0.000637)		
				(0.000000)	(0.000000)		
				141	1407		
				(0.000014)	(0.000650)		
7	0.5	7	7	(0.000000)	(0.000000)	73%	2031
				79	404		
				(0.000014)	(0.000637)		
				(0.000000)	(0.000000)		
				141	1413		
				(0.000014)	(0.942219)		
7	1	7	7	(0.000000)	(0.000000)	73%	2037
				79	404		
				(0.000014)	(0.000637)		
				(0.000000)	(0.000000)		
				141	1413		
				(0.000014)	(0.942219)		
7	5	7	7	(0.000000)	(0.000000)	73%	2037
				66	372		
				(0.001052)	(0.002690)		
				(0.000000)	(0.000000)		
				114	1244		
				(0.000667)	(0.002092)		
17	0.5	7	-7	(0.000000)	(0.000000)	73%	1796
				79	404		
				(0.000014)	(0.000637)		
				(0.000000)	(0.000000)		
				141	1407		
				(0.000014)	(0.000650)		
7	0.5	7	-7	(0.000000)	(0.000000)	73%	2031
				79	402		
				(0.000011)	(0.000426)		
				(0.000000)	(0.000000)		
				143	1405		
				(0.000011)	(0.000437)		
7	0.5	12	-12	(0.000000)	(0.000000)	73%	2029

Table 4.6: The similarity scales derived from the matrix $M + N + W^{-1}$ in the examples in previous tables.

residue	(U, D, NP) = (7, 7, 7)	(7, -7, 17)	(7, -7, 7)	(7, 12, -12)
A	0.000000	0.000000	0.000000	0.000000
R	0.000000	0.000000	0.000000	0.000000
N	0.000000	0.000000	0.000000	0.000000
D	0.000000	0.000000	0.000000	0.000000
C	0.000000	0.000000	0.000000	0.000000
Q	0.000000	0.000000	0.000000	0.000000
E	0.000000	0.000000	0.000000	0.000000
G	0.000000	0.000000	0.000000	0.000000
H	0.000000	0.000000	0.000000	0.000000
I	0.000000	0.000000	0.000000	0.000000
L	0.000000	0.000000	0.000000	0.000000
K	0.000000	0.000000	0.000000	0.000000
M	0.000000	0.000000	0.000000	0.000000
F	-0.000001	0.000004	-0.000001	0.000001
P	-0.000077	0.000115	-0.000077	0.000078
S	-0.000444	0.000752	-0.000444	0.000422
T	-0.003688	0.006673	-0.003688	0.003336
W	-0.999575	0.999473	-0.999575	0.999707
Y	-0.028906	0.031739	-0.028906	0.023961
V	-0.000729	0.000993	-0.000729	0.000559

Table 4.7: Three-state predictions using the similarity scale derived from the matrix $C + W^{-1}$ and the 5-nearest neighbors method (Predict when 4 or 5 neighbors have same structures).

NP	Thr	U	D	P			PA	$\#p$
				79	33	120		
				(0.000013)	(0.000038)	(0.000011)		
				(0.000000)	(0.000000)	(0.000000)		
				50	44	123		
				(0.000008)	(0.000026)	(0.000013)		
				(0.000000)	(0.000000)	(0.000000)		
				90	45	373		
				(0.000013)	(0.000026)	(0.000653)		
7	0.5	7	-7	(0.000013)	(0.000000)	(0.000000)	52%	957
				79	34	125		
				(0.000010)	(0.000031)	(0.000009)		
				(0.000000)	(0.000000)	(0.000000)		
				51	45	121		
				(0.000007)	(0.000021)	(0.000010)		
				(0.000000)	(0.000000)	(0.000000)		
				92	48	367		
				(0.000010)	(0.000021)	(0.000448)		
7	0.5	12	-12	(0.000010)	(0.000000)	(0.000000)	51%	962

4.3 Similarity Matrix for Two-state Prediction

4.3.1 Introduction

The similarity matrix is a 20 × 20 matrix, the i,jth entry of which marks the similarity between amino acids i and j. The matrix may be based on the chemical or physical similarities between amino acids or an implicit similarity based on the frequency with which amino acid i has mutated (non-lethally) to amino acid j. The similarity scale and the five nearest neighbors method consider the Euclidean distance

between two points as the structural similarity between two corresponding residues. The similarity matrix considers the inner product of two vectors, which corresponds to two protein segments, as the similarity of the structures of the two middle residues in the two segments. Thus, the points with known structures, i.e., points in the training set, lying on one side of the plane determined by the observed segment will determine the secondary structure of the middle residue in the observed protein segment.

4.3.2 Method

We construct a similarity matrix S using certain small eigenvalues and corresponding eigenvectors of the matrix $M + N + W^{-1}$ (or $M + N - W$), which was discussed in the previous section. That is,

$$S = \sum_{i=1}^{m} \frac{1}{\eta_{(i)}} O_{(i)} O_{(i)}^T,$$

where $m < 20$ and where $\eta_{(i)}$'s are some significant, small eigenvalues associated with the eigenvectors $O_{(i)}$'s of the matrix $M + N + W^{-1}$. By significant, we mean that the smallest unused eigenvalue is greater than the greatest used one and that the difference is relatively great. For example, suppose the eigenvalues listed, from smallest to largest, are 1.0, 1.2, 1.7, 2.0, 7.4, 7.6,$\cdots$, 12, and 14; then the first four eigenvalues will be chosen. We use the matrix S, as Levin *et al.* use their similarity matrix, to define the similarity score of two amino acid segments.

Suppose that S_1 and S_2 are two k-amino acid segments represented by the two $20 \times k$ matrices U_1 and U_2, respectively, as in the previous section. Then the similarity

score, $S(S_1, S_2)$, of S_1 and S_2, as defined by matrix S, is

$$S(S_1, S_2) = \sum_{i=1}^{m} \left[\frac{1}{\eta_{(i)}} \left(O_{(i)}^T U_1 \right) \bullet \left(O_{(i)}^T U_2 \right) \right], \tag{4.5}$$

where $\bullet$ is the usual inner product in k-dimensional real space.

Temporarily, let $m = 1, \eta_{(1)} = \eta, \text{and} O_{(1)} = O$ in Eq. 4.5, i.e.,

$$S(S_1, S_2) = \frac{1}{\eta} \left(O^T U_1 \right) \bullet \left(O^T U_2 \right).$$

If P_1 and P_2 are the two points in R^k transformed from S_1 and S_2 , respectively, by O, as in the previous section, then

$$P_1 = O^T U_1,$$

$$P_2 = O^T U_2,$$

and

$$S(S_1, S_2) = \frac{1}{\eta} P_1 \bullet P_2.$$

The matrix S would work well if the following two hypotheses were true:

1. The points transformed from the k-amino acid segments by eigenvector O are well distributed. That is, there exists a plane L in R^k such that most middle residue alpha-helical points lie on one side of L (the points lying on the other side are called "misplaced" points) and most non-alpha-helical points lie on the other side (see Fig. 4.2).

2. The plane L passes through the origin (see Fig. 4.2).

We can classify k-amino acid segments in the training set into two groups, A and B, according to the helical or nonhelical structure of the middle residue,

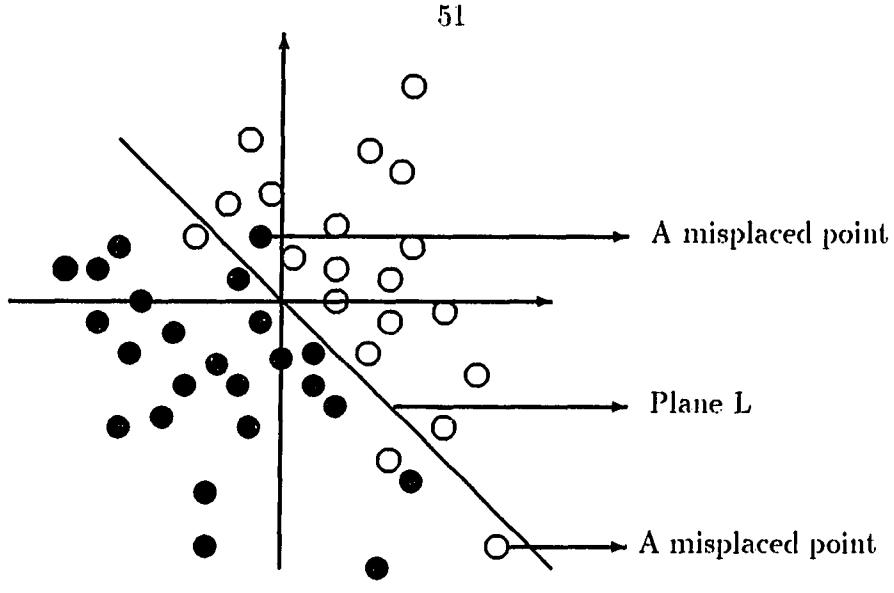

Figure 4.2: The transformed points are well distributed.

For an element S_1 in A that is not misplaced in R^k, most segments T in the training set such that

$$S(S_1, T) > C,$$

for some constant C, would belong to group A if the above two hypotheses were true. A similar phenomenon would hold true for group B (Fig. 4.3).

Hence, to predict the structure of the middle residue of a new amino acid segment, S_2, we compare the sums of the $S(S_2, T)$ values for segments T in groups A and B for which $S(S_2, T) > C$.

4.3.3 Prediction Procedure

If for $S(S_2, T) > C$, where S_2 is a new amino acid segment whose structure is unknown, the sums of the $S(S_2, T)$ values for segments T in group A is greater than

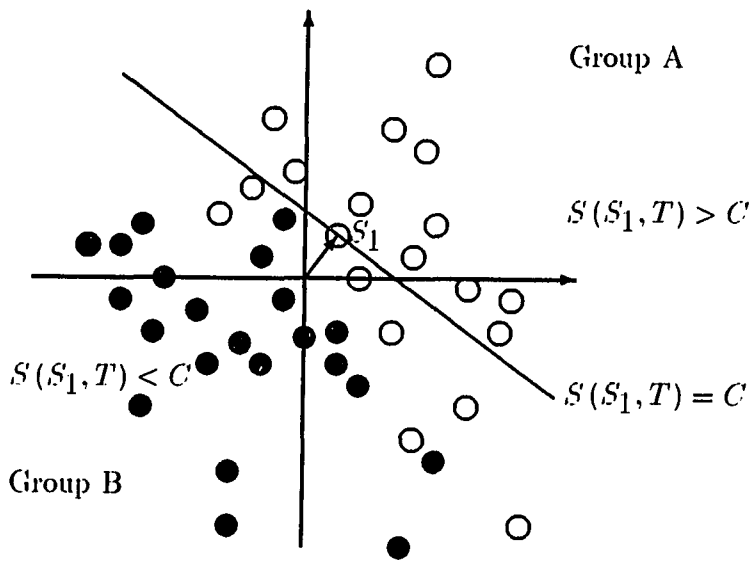

Figure 4.3: Most points T belong to the group A for which $S(S_1, T) > C$.

that in group B, then the alpha-helix structure is assigned to the middle residue of S_2. Otherwise, the structure assigned is coil. Let us call this prediction procedure the Levin-like scheme.

Another prediction procedure is similar to the 5-nearest neighbors method. We find the 5 segments T from the training set with greatest $S(S_2, T)$ values. Then the secondary structures of the middle residues in the 5 segments determine the structure of the target residue. If the middle residue of 4 (3 or 5) of the 5 segments have the same structure, then this structure is assigned to the target residue. Otherwise, no prediction is made.

4.3.4 Results and Discussion

Tables 4.8, 4.9, 4.10, and 4.11 present results of two-state predictions obtained using different matrices ($M + N - W$ or $M + N + W^{-1}$) and prediction schemes (Levin-like or 5-nearest neighbors). Similarity matrices are band matrices.

Tables 4.12 and 4.13 present results of three-state predictions obtained using matrices $C - W$ and $C + W^{-1}$, respectively, and the 5-nearest neighbors prediction scheme. Similarity matrices are band matrices. Coil remains overpredicted. Note that there are 4884 alpha-helical segments, 3884 beta-sheet segments, and 8072 coil segments in the training set when the length of the segment is seven, and there are 4524 alpha-helical segments, 3634 beta-sheet segments, and 7652 coil segments when the length of the segment is seventeen.

Table 4.8: Two-state predictions using the similarity matrix derived from the matrix $M + N - W$ and the Levin-like method.

NP	U	D	λ	SCORE[a]	P		PA	$\#_p$
					0	840		
7	7	-5	5	7	0	2472	75%	3312

[a]SCORE is the minimal similarity value between the observed segment in testing set and any segment in training set that are considered similar.

Table 4.9: Two-state predictions using the similarity matrix derived from the matrix $M + N + W^{-1}$ and the Levin-like method.

NP	U	D	λ	SCORE	P		PA	$\#_p$
					0	840		
7	7	-5	3	7	0	2472	75%	3312

Table 4.10: Two-state predictions using the similarity matrix derived from the matrix $M + N - W$ and the 5-nearest neighbors method (Predict when 4 or 5 neighbors have same structures).

NP	Thr	U	D	λ	P		PA	$\#_p$
					100	355		
					(188,795,908)	(176,997,325)		
					(3,735,217)	(1,258,731)		
					178	1453		
					(176,038,425)	(182,352,343)		
7	0.5	7	-5	5	(1,346,108)	(1,038,586)	74%	2086

Table 4.11: Two-state predictions using the similarity matrix derived from the matrix $M + N + W^{-1}$ and the 5-nearest neighbors method (Predict when 4 or 5 neighbors have same structures).

NP	Thr	U	D	λ	P		PA	$\#_p$
					64	386		
					(787,608)	(1,149,940)		
					(903)	(124)		
					169	1102		
					(1,165,708)	(1,149,448)		
7	0.5	7	-5	3	(306)	(61)	68%	1721

Table 4.12: Three-state predictions using the similarity matrix derived from the matrix $C - W$ and the 5-nearest neighbors method (Predict when 4 or 5 neighbors have same structures).

NP	Thr	U	D	λ	P			PA	$\#_p$
					94	29	127		
					(280,867,391)	(288,710,681)	(288,712,170)		
					(5,095,399)	(52,866,430)	(2,031,192)		
					71	60	148		
					(256,763,271)	(228,986,542)	(276,302,636)		
					(8,401,568)	(8,149,886)	(1,599,557)		
					109	63	374		
					(275,017,426)	(249,545,024)	(280,385,919)		
7	10^4	7	-5	5	(275,017,426)	(4,501,798)	(3,326,436)	49%	1075
					92	29	115		
					(280,867,391)	(288,710,681)	(288,712,170)		
					(54,781,622)	(52,866,430)	(52,880,575)		
					69	59	140		
					(256,763,271)	(228,986,542)	(276,302,636)		
					(52,950,522)	(59,405,761)	(51,862,433)		
					105	59	349		
					(275,017,426)	(249,545,024)	(280,385,919)		
7	5×10^7	7	-5	5	(275,017,426)	(55,070,816)	(50,877,254)	49%	1017
					80	33	97		
					(1,135,257,107)	(1,140,401,086)	(1,028,756,924)		
					(318,006,616)	(351,858,070)	(204,966,534)		
					51	40	107		
					(1,017,372,169)	(1,376,094,007)	(1,109,190,057)		
					(422,400,762)	(357,545,242)	(323,577,768)		
					103	41	311		
					(1,116,703,446)	(983,976,949)	(1,271,005,388)		
17	5×10^7	7	-5	5	(1,116,703,446)	(324,759,996)	(184,949,291)	50%	863
					77	33	98		
					(951,979,541)	(943,400,934)	(861,639,626)		
					(266,433,217)	(293,302,073)	(172,023,641)		
					53	42	108		
					(843,388,304)	(1,147,947,682)	(927,787,508)		
					(355,775,251)	(297,746,735)	(267,309,152)		
					101	39	312		
					(937,624,840)	(822,945,509)	(1,064,905,273)		
17	5×10^7	10	-10	5	(937,624,840)	(268,429,753)	(152,627,097)	50%	863

Table 4.13: Three-state predictions using the similarity matrix derived from the matrix $C + W^{-1}$ and the 5-nearest neighbors method (Predict when 4 or 5 neighbors have same structures).

NP	Thr	U	D	λ	P			PA	$\#_p$
					46	23	81		
					(527,404)	(525,997)	(781,386)		
					(11,091)	(12,237)	(11,067)		
					43	17	57		
					(792,111)	(626,213)	(781,017)		
					(11,094)	(12,331)	(11,098)		
					81	30	146		
					(626,202)	(372,651)	(780,878)		
7	10^4	7	-5	3	(626,202)	(12,237)	(11,009)	40%	521
					6	1	30		
					(527,404)	(525,997)	(781,386)		
					(391,021)	(394,350)	(390,559)		
					15	4	17		
					(792,111)	(626,213)	(781,017)		
					(390,531)	(371,328)	(390,576)		
					11	0	29		
					(626,202)	()	(780,878)		
7	3×10^5	7	-5	3	(626,202)	()	(371,325)	35%	113
					21	9	91		
					(970,613)	(1,149,355)	(1,310,932)		
					(315,701)	(316,217)	(312,155)		
					24	15	92		
					(1,328,353)	(813,704)	(1,803,345)		
					(320,111)	(315,688)	(320,068)		
					37	20	186		
					(1,329,871)	(1,310,689)	(1,803,347)		
17	3×10^5	7	-5	3	(1,329,871)	(315,687)	(315,017)	45%	498

Table 4.13 (Continued.)

NP	Thr	U	D	λ	P			PA	$\#_p$
					41	11	153		
					(970,613)	(1,149,355)	(1,310,932)		
					(36,760)	(37,136)	(34,143)		
					35	20	125		
					(1,328,353)	(813,704)	(1,803,345)		
					(34,101)	(33,888)	(36,675)		
					71	24	280		
					(1,329,871)	(1,310,689)	(1,803,347)		
17	3×10^4	7	-5	3	(1,329,871)	(33,888)	(32,528)	45%	760
					24	9	100		
					(514,177)	(606,262)	(706,196)		
					(34,150)	(161,664)	(30,245)		
					25	15	93		
					(715,686)	(416,884)	(958,559)		
					(34,148)	(161,373)	(34,204)		
					38	21	202		
					(716,436)	(706,064)	(958,560)		
17	3×10^4	10	-10	3	(716,436)	(34,150)	(30,232)	46%	527

4.4 Single Separation Plane

4.4.1 Introduction

The most general encoding scheme introduced in Chapter 3 will be used in the last two sections of Chapter 4. That is, a k-amino acid segment is represented by k 20-dimensional unit vectors and transformed into a single point in 20k-dimensional real space.

The ultimate purpose of the information theory method and the neural network model is to construct a plane in the $(20 \times k)$-dimensional real space, where k is the number of amino acids in a segment. The structure assigned to the middle amino acid in a new segment is determined by the location of the segment in space. It is alpha-helical if the point transformed from a segment lies on one side of the plane; it is coil (or nonalpha-helix) if the point lies on the other side. We will construct a linear programming model in this section and attempt to find an acceptable separating plane.

4.4.2 Method

Two-state prediction. A segment of amino acids is defined to be alpha-helical if the structure of its middle amino acid is in an alpha-helix. Similar procedure is followed for a non-alpha-helical structure. Suppose that there are $m + r$ k-amino acid segments in the training set, within which m segments are alpha-helical, and r segments non-alpha-helical. Let A be an $m \times 20k$ matrix with each row composed of k 20-dimensional unit vectors and corresponding to an alpha-helical segment in the training set. Similarly, let B be an $r \times 20k$ matrix representing the r non-alpha-

helical segments in the training set. To find a normal vector **N** and a constant C to constitute a plane, we solve the minimization problem (For convenience, let k = 7.):

$$\text{Minimize} \quad \sum_{i=1}^{m+r} Z_i$$

$$\begin{aligned}
\text{subject to} \qquad A \cdot \mathbf{N} + C_m + Z_\alpha &\geq 0, \\
B \cdot \mathbf{N} + C_r + Z_c &\leq 0, \\
N_1 + N_2 + \cdots + N_{20} &= 0, \\
N_{21} + N_{22} + \cdots + N_{40} &= 0, \\
N_{41} + N_{42} + \cdots + N_{60} &= 0, \\
N_{61} + N_{62} + \cdots + N_{80} &= 0, \\
N_{81} + N_{82} + \cdots + N_{100} &= 0, \\
N_{101} + N_{102} + \cdots + N_{120} &= 0, \\
N_{121} + N_{122} + \cdots + N_{140} &= 0, \\
\mathbf{N} \bullet \mathbf{N} &= 1, \text{ and} \\
Z_i &\geq 0, \text{ where i = 1, 2,}\cdots\text{, m + r,}
\end{aligned}$$

$$\begin{aligned}
\text{where} \qquad \mathbf{N} &= (N_1, N_2, \cdots, N_{140})^T, \\
Z_\alpha &= (Z_1, Z_2, \cdots, Z_m)^T, \\
Z_c &= (Z_{m+1}, Z_{m+2}, \cdots, Z_{m+r})^T, \\
C_m &= (C, C, \cdots, C)^T, \text{ a constant vector in } R^m, \text{ and} \\
C_r &= (C, C, \cdots, C)^T, \text{ a constant vector in } R^r,
\end{aligned}$$

are unknowns.

Solving such a minimization problem will yield a unit vector **N** and a constant C such that the sum of the distances (Z_i) from the "misplaced points" to the plane $\mathbf{X} \bullet \mathbf{N} + C = 0$ is minimized (Fig. 4.4). The Z_i value associated with a correctly placed point is 0. The seven constraints $N_i + N_{i+1} + \cdots + N_{i+19}$, for i = 1, 21, 41, 61, 81, 101, and 121, eliminate some trivial solutions for which the resulting plane contains all the points in the training set and thus will not classify alpha-helical and coil points. For example,

$$\begin{cases} N = \frac{1}{\sqrt{40}}(\underbrace{1,1,\cdots,1}_{20},\underbrace{-1,-1,\cdots,-1}_{20},0,0,\cdots,0,0)^T, \\ C = 0, \text{ and} \\ Z_i = 0, \text{ where } i = 1,2,\cdots,m+r, \end{cases}$$

and

$$\begin{cases} N = \frac{1}{\sqrt{140}}(\underbrace{1,1.1,\cdots\cdots\cdots,1,1}_{140})^T, \\ C = \frac{-7}{\sqrt{140}}, \text{ and} \\ Z_i = 0, \text{ where } i = 1,2,\cdots,m+r \end{cases}$$

are two trivial solutions, but do not satisfy the constraint $N_1 + N_2 + \cdots + N_{20} = 0$.

On the other hand, because we are concerned with the relative influence of the 20 different amino acids at each position on the structure of the middle amino acid of k-amino acid-long sequences, the seven constraints are quite acceptable.

The nonlinear constraint $\mathbf{N} \bullet \mathbf{N} = 1$ is first replaced with $\mathbf{N} \bullet N_0 = 1$, where $\|N_0\| = 1$ is given, thus turning it into a linear constraint for which the **Simplex method** can solve for a solution of $\bar{N}_1$. Normalizing $\bar{N}_1$ to obtain N_1 and replacing

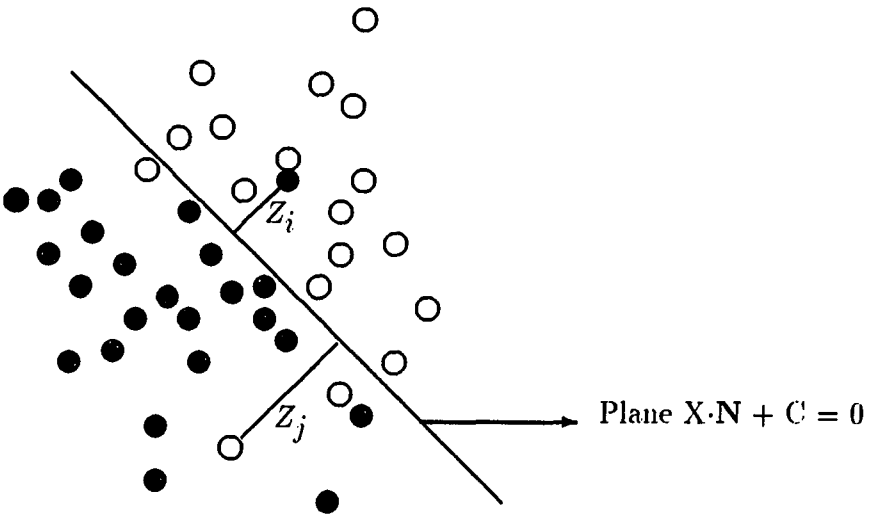

Figure 4.4: The nonzero Z_i is the distance from the plane to the "misplaced" point associated with Z_i.

$\mathbf{N} \bullet \mathbf{N}$ by $\mathbf{N} \bullet N_1 = 1$, we solve the minimal problem by the Simplex method again and normalize the solution to obtain N_2, etc.

The initial guess, N_0, is given as

$$N_0 = \frac{\bar{A} - \bar{B}}{\|\bar{A} - \bar{B}\|_2},$$

where

$$\bar{A} = \frac{\sum_{i=1}^{m} A_i}{m} \text{ and } \bar{B} = \frac{\sum_{i=1}^{r} A_i}{r}, \text{ where}$$

A_i's are the rows of the matrix A and

B_i's are the rows of the matrix B.

That is, $\bar{A}$ and $\bar{B}$ are the centroids (or averages) of the alpha-helical points and the coil points, respectively, and N_0 is the normalization of the difference between centroids.

Suppose there is a plane, namely $X \bullet \mathbf{N} + C = 0$, able to separate alpha-helical from non-alpha-helical points. That is, $A_i \cdot \mathbf{N} + C \geq 0$ for any row A_i of the matrix A, and $B_i \cdot \mathbf{N} + C \leq 0$ for any row B_i of the matrix B. Then

$$\bar{A} \cdot \mathbf{N} + C \geq 0$$

and

$$\bar{B} \cdot \mathbf{N} + C \leq 0,$$

where

$$\bar{A} = \frac{\sum_{i=1}^{m} A_i}{m} \text{ and } \bar{B} = \frac{\sum_{i=1}^{r} A_i}{r}.$$

So

$$\bar{A} \cdot \mathbf{N} - \bar{B} \cdot \mathbf{N} \geq 0,$$

i.e.,

$$(\bar{A} - \bar{B}) \cdot \mathbf{N} \geq 0,$$

i.e.,

$$\frac{\bar{A} - \bar{B}}{\|\bar{A} - \bar{B}\|_2} \cdot \mathbf{N} \geq 0 \text{ provided } \|\bar{A} - \bar{B}\|_2 \neq 0.$$

Thus, let

$$N_0 = \frac{\bar{A} - \bar{B}}{\|\bar{A} - \bar{B}\|_2}.$$

Three-state prediction. First, we describe Qian & Sejnowski's and Robson *et al.*'s three-state prediction schemes and explain how to make a three-state prediction.

Qian and Sejnowski did not undertake a two-state prediction in [10]; instead they constructed three tables for a three-state prediction using a two layer neural network. These three tables include the weights from the input units to the three output units, as well as the bias in each output unit. If we rearrange the numbers in each table as three 20k-dimensional vectors N_α, N_β, and N_c and let the three biases be B_α, B_β, and B_c, respectively, then for a new amino acid segment P, Qian and Sejnowski use the greatest of the three numbers

$$N_\alpha \bullet P + B_\alpha,$$

$$N_\beta \bullet P + B_\beta,$$

and

$$N_c \bullet P + B_c$$

to determine the secondary structure of the middle residue of P.

Assume that

$$N_{\alpha\beta} = N_\alpha - N_\beta,$$

$$N_{\alpha c} = N_\alpha - N_c,$$

$$N_{\beta c} = N_\beta - N_c,$$

$$B_{\alpha\beta} = B_\alpha - B_\beta,$$

$$B_{\alpha c} = B_\alpha - B_c,$$

and that

$$B_{\beta c} = B_\beta - B_c;$$

then the three planes

$$I_{\alpha\beta}\colon\ N_{\alpha\beta} \bullet X + B_{\alpha\beta} = 0,$$

$$I_{\alpha c}\colon\ N_{\alpha c} \bullet X + B_{\alpha c} = 0,$$

and

$$I_{\beta c}\colon\ N_{\beta c} \bullet X + B_{\beta c} = 0$$

partition the 20k-dimensional real space into several regions, and each region is a "alpha region," a " beta region," or a "coil region" (Fig. 4.5). The secondary structure assigned to the middle residue of a new segment P depends upon in which region or in what kind of region P is located. For example, an alpha-helix will be assigned if P is in an alpha region.

The three planes

$$P_\alpha\colon\ N_\alpha \bullet P + B_\alpha = 0,$$

$$P_\beta\colon\ N_\beta \bullet P + B_\beta = 0,$$

and

$$P_c\colon\ N_c \bullet P + B_c = 0,$$

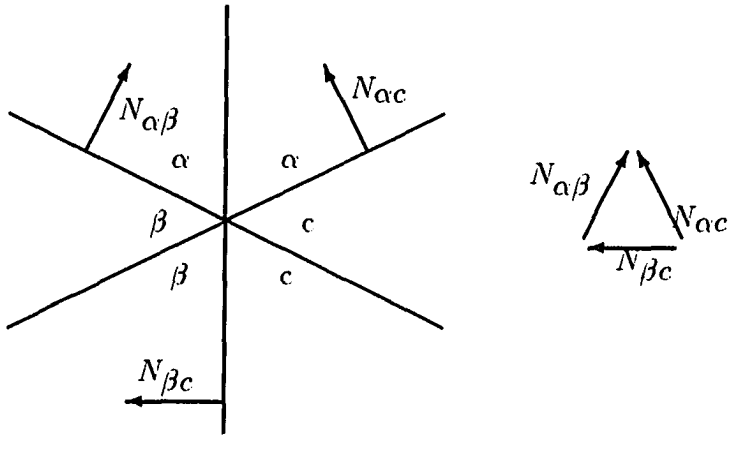

Figure 4.5: Three-state prediction in three-dimensional space. The intersection of the three planes is a line. The three vectors $N_{\alpha\beta}$, $N_{\alpha c}$, and $N_{\beta c}$ are linearly dependent.

which are derived from the neural network model; and the magnitude of the three vectors N_α, N_β, and N_c, i.e.,

$$\|N_\alpha\|_2,$$

$$\|N_\beta\|_2,$$

and

$$\|N_c\|_2,$$

determine how 20k-dimensional real space is partitioned. Suppose that Q is a point on the plane $I_{\alpha\beta}$, and suppose that the Euclidean distances from Q to P_α and from Q to P_β are d_α and d_β, respectively. Then

$$\|N_\alpha\| \cdot d_\alpha = \|N_\beta\| \cdot d_\beta.$$

Let us call the three planes $I_{\alpha\beta}, I_{\alpha c}$, and $I_{\beta c}$ the "indifference planes"; the Euclidean distance from a point T to P_x times $\|N_x\|$, where x = α, β, and c, the "statistical distance" between T and P_x; and the three vectors N_α, N_β, and N_c the "distance vectors" (See Fig. 4.6).

Applying Eq. 2.5, the information theory method for three-state prediction constructs three tables for alpha-nonalpha, beta-nonbeta, and coil-noncoil separations, respectively. In exactly the same procedure as used in the neural network model, the three tables are used to construct three indifference planes if the three decision constants for each table are determined. For a fixed database, the three unique distance vectors for the information theory method are determined, and the three decision constants chosen to yield a good performance; that is, the three indifference planes are "shifted" to cause a good partition for the points in the training set. The statistical distances from a point to the three indifference planes will reflect the structural

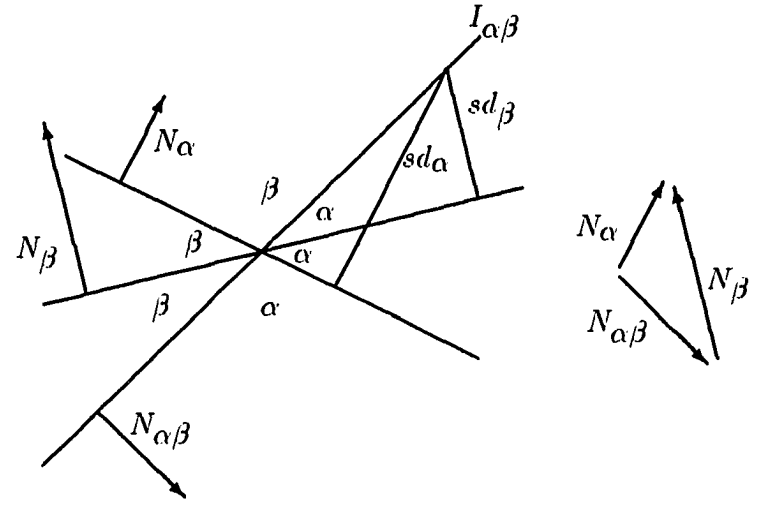

Figure 4.6: The indifference plane $I_{\alpha\beta}$ in three-dimensional space. $N_{\alpha\beta} = N_\alpha - N_\beta$. The two statistical distances sd_α and sd_β are equal.

tendency of the middle residue of the segment corresponding to the point (Fig. 4.5). On the other hand, in the neural network model, both the three distance vectors and the three biases are changable. The three indifference planes should not only be "shifted" but also be "rotated" to achieve a good performance upon partition. Thus, the neural network model can, although not consistently[3], yield a better result than can the information theory method.

The goal of the linear programming model for three-state prediction is to construct three planes directly to partition the 20k-dimensional space. For two-state prediction, we construct a plane to "separate" the alpha-helical points from the non-alpha-helical points. The three planes for three-state prediction "separate" the alpha-helical points from the beta-sheet points, the alpha-helical points from the coil points, and the beta-sheet points from the coil points. We use the same model for two-state prediction but use different A and B matrices to find the three planes. If A and B represent the alpha-helical and the beta-sheet points, respectively, and if the coil points are discarded, then the resulting plane can be used for alpha-beta separation (Fig. 4.6). The other two planes are constructed similarly (Fig. 4.5). The resulting three planes will partition the space into several regions, each of which will be categorized as an alpha-helix region, a beta-sheet region, a coil region, or a fuzzy region. We assign no structure to points lying on a fuzzy region.

Note that the intersection of the convex hulls of the alpha-helical points, the beta-sheet points, and the coil points may not be empty, i.e., "mixed region(s) (or

[3]To achieve a desired input-output mapping, the purpose of the neural network model is to minimize the quadratic function, which is the sum of the differences of the target outputs and the corresponding outputs of the network over all inputs. Because of the existence of local minima, the performance of a network depends upon starting weights and training procedures.

fuzzy set(s))" exist(s). Therefore, a point is not predicted if it is lying in a mixed region.

4.4.3 Prediction Procedure

Two-state prediction. Suppose that the plane $X \bullet \mathbf{N} + C = 0$ results from the linear programming model. Then the secondary structure of the middle amino acid of a segment, corresponding to point P, is predicted to be (Fig. 4.7) alpha-helical if

$$P \bullet \mathbf{N} + C > 0$$

and non-alpha-helical if

$$P \bullet \mathbf{N} + C < 0.$$

Three-state prediction. Suppose that the three planes

$$X \bullet \mathbf{N}_{\alpha\beta} + C_{\alpha\beta} = 0,$$

$$X \bullet \mathbf{N}_{\alpha\mathbf{c}} + C_{\alpha c} = 0, \text{ and}$$

$$X \bullet \mathbf{N}_{\beta\mathbf{c}} + C_{\beta\mathbf{c}} = 0$$

are the resulting partition planes. To assign a secondary structure to the middle residue of a block of k amino acids, each of the three partition planes will be used to determine one of two possible structures: alpha-helix or beta-sheet, alpha-helix or coil, and beta-sheet or coil, respectively. We predict only when two out of three structures are the same and assign the structure to the target residue (Fig. 4.8).

4.4.4 Smoothing Algorithm

A helix or a sheet in a protein is composed of several contiguous amino acids. A prediction scheme may mistakenly assign non-helix to a residue in the middle of

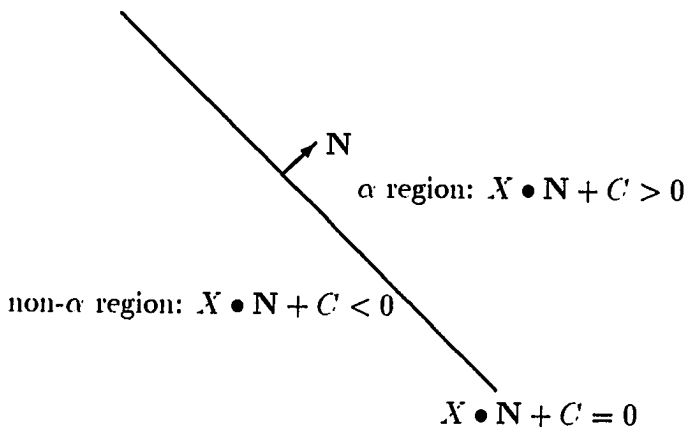

Figure 4.7: Two-state prediction.

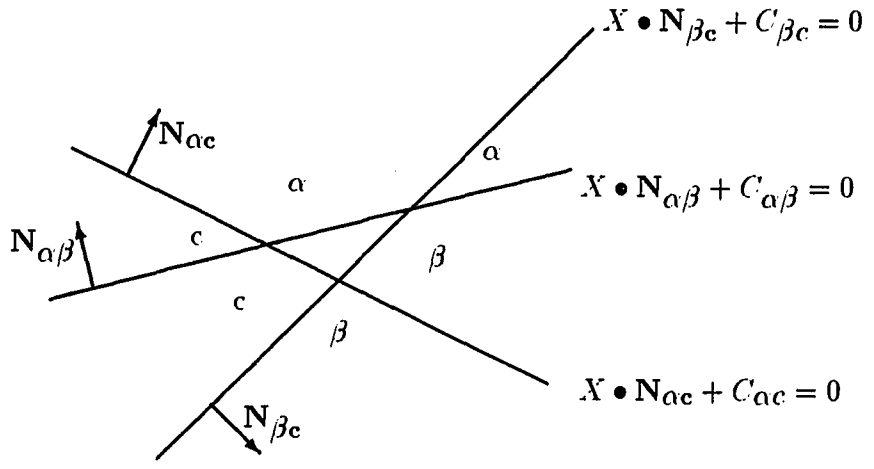

Figure 4.8: Three-state prediction. The structure assigned to a point depends upon where this point is located in space. We do not predict if a point lies in the middle triangle region.

a block of helix residues, which would appear to be a discontinuity in the prediction sequence. The purpose of the smoothing algorithm is to offset or to eliminate the discontinuity trend. If an amino acid is assigned a structure different from that assigned to the left two and the right two amino acids, which have the same structure, then the structure of the middle acid must be changed so that all five acids have the same structure.

We applied the smoothing algorithm in two-state prediction, and accuracy percentages improved slightly.

4.4.5 Results and Discussion

A. Artificial Database.

We use Eqs. 2.5 and 2.4, which have been derived from the information theory method with and without Baysian smoothing, respectively, to construct artificial databases using the first 30[4] or the first 50[5] proteins in the training set. We assign "structure" to the proteins according to the information theory prediction. To do so, we first compute the appropriate information theory vector (a 140-dimensional vector because we use segments of length 7) using one of the two equations and either 30 or 50 proteins. We let the adjustable parameter be zero and construct a 140-dimensional plane, L, passing through the origin. The artificial database is established by changing the structural assignments to the residues in the 30 or the 50 proteins whenever necessary so that all alpha-helical points lie on one side of plane L and so that all non-alpha-helical points lie on the other side. Then the linear

[4]The total number of amino acids in the 30 proteins is 3707.

[5]The total number of amino acids in the 50 proteins is 7251.

programming model in the Method section uses the artificial database to find the plane M by means of MPSX software and an IBM machine. The partition result is subsequently tested on the artificial database.

The resulting planes M can, except for some points on M, separate alpha-helical points from non-alpha-helical points in the artificial database (See Table 4.14). From the experiments, we found that a plane M can be constructed in one iteration by means of the linear programming model, which separates the alpha-helical from the non-alpha-helical points; on the other hand, the number of iterations has little influence on the performance of the partition for the 30 or the 50 proteins.

We tested the partition accuracy performed by plane L on the 30 or 50 proteins before changing structures to establish an artificial database (see Table 4.15). We found that the use of Bayes' decision theory has little influence on the performance of partitions of the 30 or 50 proteins.

Table 4.14: The partition result on artificial database performed by the planes, which were derived from the linear programming model and calculated by the software MPSX, for seven amino acid segments in the set of 30 or of 50 proteins.

	#[a]	S[b]		I[c]	CPU[d]	%[e]	Del[f]	S		I	CPU	%	Del	S		I	CPU	%	Del
		1587	42																
	30	7	1841	1	421	99	36												
		3125	17					3107	40										
B[g]	50	56	3725	1	1030	99	14	37	3749	2	1261	99	4						
		1636	20																
	30	38	1790	1	405	99	29												
		3156	24					3122	48					3150	27				
$\bar{B}$[h]	50	46	3704	1	1274	99	7	16	3722	2	3049	99	29	35	3708	3	1071	99	17

[a] # = The number of proteins used.

[b] The separation result for the 30 or the 50 proteins (see Table 4.16 for details.). Note that we ran the MPSX in double precision, but only five numbers after digital points are used to form the planes. Because all the output solutions from MPSX were optimal with objective values 0, any resulting plane will separate all alpha-helical points from nonalpha points, except for some points lying on the plane. The "misplaced" points in this column, in fact, lie on the corresponding plane derived from the MPSX.

[c] I is the number of iterations performed in the linear programming model.

[d] In seconds.

[e] % = SA = separation accuracy.

[f] Del is the number of deleted points, which may or may not be on the real plane constructed by the MPSX, but which lie on the plane used here and were not partitioned.

[g] Baysian information theory method.

[h] Non-Baysian information theory method.

Table 4.15: The partition results of the planes, which were derived from the information theory method, for seven-amino acid segments in the sets of 30 or of 50 proteins.

	#	S[a]		SA[b]	Del[c]
Baysian	30	1013	410	70%	0
		638	1452		
	50	1768	750	69%	0
		1380	3039		
Non-Baysian	30	1020	403	70%	0
		650	1440		
	50	1783	735	69%	0
		1401	3018		

[a]The separation results for the 30 or the 50 proteins.

[b]SA = Separation accuracy.

[c]Del = The number of points on the plane.

Table 4.16: The four numbers in the entries of the S or P column in some tables have the same roles as do A, B, C, and D.

Real structure	Predicted structure	
	alpha-helix	nonalpha-helix
alpha-helix	A	B
nonalpha-helix	C	D

B. Real Database.

An acceptable normal vector will reflect the influences of different amino acids at different positions on the structures of middle residues in segments. Algebraically, the entries in an acceptable normal vector will be different to quantitatively indicate the relative influences; geometrically, the resulting plane will separate most points of different types and itself contain only a few points.

Two-state Prediction. We use MPSX software and the first 10 proteins (1027 residues) in the training set to test the performance of the linear programming model for seven-amino acid segments. The solution, which includes a normal vector N and a constant C, is of the the form[6]

$$\begin{cases} N_{95} = 19 \times (-a), & \text{where a is a real number,} \\ N_i = a & \text{for } i = 81 - 100 \text{ except } 95, \\ N_j = 0 & \text{otherwise, and} \\ C = -a. \end{cases}$$

The plane will contain all points except the points whose corresponding segments include prolines at the fifth positions (counted from the N-terminal end to the C-terminal end), and for which the structures of the middle residues are predicted to be non-alpha-helical. Thus, from the viewpoint of separation, this plane does not achieve our goal; on the other hand, the normal vector indicates that proline is a helix-breaker when appearing in at the fifth position but gives no information about the other residues.

When we add the constraint $N_{95} = 0$, similar output results. This time, proline in the sixth position is a helix-breaker. As we continue this process, the resulting

[6]The iterative linear programming problem converged in one iteration.

normal vectors indicate that proline is a helix-breaker at positions 4, 5, and 6; and that tryptophan is a helix-breaker at positions 3, 4, 5, 6, and 7, and that tryptophan is a helix-former at the first and second positions.

Another experiment was conducted to add instead the constraint

$$-0.3 \leq N_i \leq 0.3, \text{ for } i = 75, 95, 115, 18, 38, 58, 78, 98, 118, 138,$$

to restrict in the normal vector the ranges of some specific entries found in the previous experiment. We solved the problem until it converged. Using the resulting planes of each iteration, we list the separation results for the first 10 proteins in Table 4.17. Note that the normal vector derived from the fifth iteration is not in an acceptable form as almost all points are predicted to be non-alpha-helical (see Table 4.17). For example, the normal vector derived from the ninth iteration is

$$\begin{cases} N_i = -0.3 & \text{if } i = 78, 95, 98, 118, 138, \\ N_{122} = 0.71482, & \\ N_j = 0.01579 & \text{if } j = 61 - 80 \text{ and } 101 - 120, \text{ except 78 and 118;} \\ N_k = 0.03333 & \text{if } k = 81 - 100, \text{ except 95 and 98;} \\ N_l = -0.02305 & \text{if } l = 121 - 140, \text{ except 122 and 138;} \\ N_m = 0 & \text{otherwise; and} \\ C = -0.04187. & \end{cases}$$

Note that $N_{122} = 0.71482$ indicates that arginine is a helix-former when in the seventh position. We modified the added constraint as

$$\begin{cases} -0.3 \leq N_i \leq -0.01 & \text{if } i = 75, 95, 115, 58, 78, 98, 118, 138, \text{ and} \\ 0.01 \leq N_j \leq 0.3 & \text{if } j = 18, 38, 122, \end{cases}$$

and used the normal vector of the fourth iteration above as the starting vector; the iteration scheme converged, and the resulting normal vectors were accepted. Separation accuracy for the 10 proteins is about 73%. Results are listed in Table 4.18.

We also used the second 10 proteins in the training set to test the model. The resulting normal vectors indicate that proline at positions 5 and 6, methionine at the positions 2, 5, 4, 6; and tyrosine at the positions 3, 5, 4, and 2 are helix-breakers and that tryptophan at the positions 6 and 7 is a helix-former, which contradicts the indications of the first 10 proteins. Because the normal vector and the resulting plane are dominated by the given ten proteins, this phenomenon is not surprising.

Table 4.17: The 10 iterations of the single plane for two-state separation with segment length seven on 10 proteins.

Ite.[a]	OBJ[b]	CPU	DOT[c]	S[d]		SA[e]	$\#_s$[f]
				270 (283)	115 (107)		
1st	28.36468	208.63	1.000003	129 (132)	439 (445)	74% (75%)	(14) 953
				244 (252)	138 (138)		
2nd	18.82253	195.43	0.619989	109 (108)	455 (469)	74% (75%)	(21) 946
				255 (256)	132 (134)		
3rd	16.21761	195.73	0.491218	123 (106)	450 (471)	73% (75%)	(7) 960
				264 (278)	118 (112)		
4th	15.12556	199.56	0.426878	141 (144)	419 (433)	73% (74%)	(25) 942

[a] Ite. = Iteration.

[b] OBJ is the objective value of the linear programming problem. That is, OBJ is the sum of the distances from the "misplaced" points to the separation plane.

[c] DOT is the inner product of the normal vector and the given initial vector.

[d] The partition performed by the plane on the 10 proteins. The results of using the smoothing algorithm are in parentheses. See Table 4.16.

[e] SA = Separation accuracy.

[f] $\#_s$ is the number of segments separated by the plane. The number of points lying on the plane is in parentheses.

Table 4.17 (Continued.)

Itc.	OBJ	CPU	DOT	S		SA	$\#_s$
				17	373		
				(1)	(389)		
				10	567	60%	(0)
5th	11.98900	96.16	0.167121	(7)	(570)	(59%)	967
				12	378		
				(1)	(389)		
				7	570	60%	(0)
6th	8.30203	37.90	0.142542	(5)	(572)	(59%)	967
				12	378		
				(1)	(389)		
				7	570	60%	(0)
7th	7.68239	58.85	0.137115	(5)	(572)	(59%)	967
				12	378		
				(1)	(389)		
				7	570	60%	(0)
8th	7.66635	39.54	0.136975	(5)	(572)	(59%)	967
				12	378		
				(1)	(389)		
				7	570	60%	(0)
9th	7.66632	51.85	0.136975	(5)	(572)	(59%)	967
				12	378		
				(1)	(389)		
				7	570	60%	(0)
10th	7.66632	41.15	0.136975	(5)	(572)	(59%)	967

Table 4.18: The last 5 iterations after modification of a constraint.

Ite.	OBJ	CPU	DOT	S		SA	#s
				245 (242)	141 (148)		
				119	451	73%	(11)
5th	14.71520	243.05	0.394234	(102)	(475)	(74%)	956
				252 (255)	134 (135)		
				127	436	73%	(18)
6th	13.85828	254.18	0.359923	(107)	(470)	(75%)	949
				263 (271)	120 (119)		
				139	425	73%	(20)
7th	13.79683	210.37	0.356741	(126)	(451)	(75%)	947
				263 (271)	120 (119)		
				139	425	73%	(20)
8th	13.78685	228.46	0.356317	(126)	(451)	(75%)	947
				238 (239)	141 (151)		
				106	465	74%	(17)
9th	13.78409	188.31	0.356048	(91)	(486)	(75%)	950
				244 (245)	129 (145)		
				118	435	73%	(41)
10th	13.78297	158.31	0.355848	(102)	(475)	(74%)	926

For our problems, the MPSX software required much more time than did the OSL software. But when we used the OSL software to find a single separation plane for the 101 proteins in the training set and for length seven, the calculation remained unfinished after 9619.72 CPU seconds.

For two-state prediction, the beta-sheet structure, as important a structure as alpha-helix, is considered coil. This assumption will make the distribution of points more ambiguous and will limit the accuracy of two-state prediction.

Early results regarding secondary structures of proteins included few beta-sheets; for example, myoglobin and haemoglobin have no beta-sheet structures; thus, only two-state prediction was studied before.

Three-state prediction. First, we used the first 10 proteins in the training set to find three single planes for alpha-beta, alpha-coil, and beta-coil separations, respectively. Second, we used the first 20 proteins to do the same thing. Third, we use the second 20 proteins to do it. The outputs of all linear programming problems were optimal.

The results for the first 10 proteins are listed in Tables 4.19 and 4.20. Each plane is obtained without iteration. Furthermore, the alpha-beta plane is obtained by allowing N_{130} to be 0 (we allowed N_{130} to be a relatively great positive number when predicting, so that whenever isoleucine is at the seventh position, the segment will tend to be alpha-helical rather than beta-sheet), and the alpha-coil plane is obtained by allowing N_{95} to be 0 (we allowed N_{95} to be a relatively small negative number when predicting, so that whenever proline is at the fifth position, the segment will tend to be coil rather than alpha-helical.).

Because the number of beta-sheet points is far smaller than the number of either of the other two structures in the 10 proteins, much more time was required to construct the alpha-coil plane, and a great objective value resulted. Note that a great objective value may result from the distribution of points in space.

Table 4.19: The three single separation planes for three-state prediction for seven-amino acid segments on the first 10 proteins in the training set.

	alpha-beta[a]	alpha-coil	beta-coil
CPU[b]	28.76	177.64	44
OBJ	0.00000	27.56092	0.00000
1st[c]	0.10604	0.07288	0.11615
2nd	0.10106	0.06939	0.09881
3rd	0.12525	0.09334	0.08914
4th	0.19647	0.10866	0.12971
5th	0.14873	0.10247	0.11118
6th	0.13549	0.10978	0.09641
7th	0.11230	0.10407	0.10583
C[d]	0.17629	-0.02952	-0.13055

[a]The single plane separating the alpha-helical points from the beta-sheet points.

[b]In seconds; MPSX software.

[c]The average of magnitudes of the first 20 entries in the normal vector.

[d]The constant term in the equation of the plane.

Table 4.20: Separation results regarding the three planes mentioned in the previous table.

S[a]			SA	$\#_s$	ab[b]	ac	bc	Del[c]
238	7	104						
9	33	6						
87	17	341	85%	842	35	1	23	66

[a]See Table 4.33.

[b]ab is the number of points contained in the alpha-beta plane.

[c]Del is the number of points lying in the fuzzy region.

The average of magnitudes of the middle 20 entries (fourth position) in each normal vector in Table 4.19 is relatively greater than that of the other entries. This reflects the fact that structure of a residue is determined mainly by the residue itself. Furthermore, because the averages of the entries on the C-terminal end are greater than the averages on the N-terminal end of the alpha-coil plane, the alpha or the coil tendency of the structure of a residue is dominated by C-terminal residues.

The constant of the alpha-beta plane is a relatively great positive number compared with the averages of entries. Thus, alpha-helix may be overpredicted in alpha-beta prediction. Similarly, coil may be overpredicted in beta-coil prediction. On the other hand, the signs of the three constants are the same as the corresponding differences of the constants in Fig. 9 of [10]; and the sum of the three constants is 0.01622, a number near zero.

Results regarding the first 20 proteins in the training set are listed in Tables 4.21 and 4.22. For iteration 1, the alpha-coil plane was obtained by allowing N_{95} (tend to coil) to be 0; the beta-coil plane was obtained by allowing N_{75} (tends to coil) to be 0. For iteration 2, we used the normal vectors from iteration 1 as the initial vectors and

allow N_{95} (tends to beta-sheet) to be 0 to get the alpha-beta plane, allowed $N_{95} = N_{78} = N_{115} = 0$, where N_{78} tends to alpha-helix rather than to coil and the other two tend to coil, to get the alpha-coil plane, and allowed $N_{75} = N_{63} = N_{133} = 0$ (all tend to coil) to get the beta-coil plane. Because we set more entries equal to 0 in iteration 2 than in iteration 1, it is not the real second iteration.

The information contained in the tables for the first 20 proteins is similar to that for the first 10 proteins.

Table 4.21: The three single separation planes for three-state prediction for seven-amino acid segments in the first 20 proteins in the training set.

	Iteration 1			Iteration 2		
	alpha-beta	alpha-coil	beta-coil	alpha-beta	alpha-coil	beta-coil
CPU	232.73	620.38	323.10	270.72	762.69	420.95
OBJ	22.77856	71.92611	31.17186	20.83535	60.84273	60.58094
1st	0.06475	0.03941	0.04922	0.06205	0.03572	0.12916
2nd	0.07197	0.05889	0.05211	0.06961	0.05706	0.14681
3rd	0.06808	0.08343	0.06536	0.06534	0.08380	0.15399
4th	0.08047	0.11318	0.19206	0.07551	0.08373	0.21087
5th	0.09405	0.07422	0.06498	0.07537	0.06718	0.18197
6th	0.06633	0.09746	0.04554	0.06428	0.07708	0.12202
7th	0.06902	0.09969	0.05002	0.07091	0.09401	0.10881
C	0.07453	-0.03785	-0.15509	0.08594	0.00504	-0.14950

Table 4.22: Separation results performed by the planes in the previous table.

Iteration 1								
S			SA	$\#_s$	ab	ac	bc	Del
457	65	222						
74	177	93						
217	82	710	67%	2097	10	1	27	53
Iteration 2								
S			SA	$\#_s$	ab	ac	bc	Del
401	75	260						
65	170	102						
180	86	735	67%	2074	16	0	9	89

Results regarding the second 20 proteins in the training set are presented in Tables 4.23 and 4.24. Separation accuracy is 52%, percentage inferior to those for the first 20 proteins. Note that the average of the middle 20 entries of the normal vector of the alpha-coil plane is a small number compared with others and thus will not dominate the prediction. As a result, accuracy diminishes.

Table 4.23: The three single separation planes for three-state prediction for seven-amino acid segments on the second 20 proteins in the training set.

	alpha-beta	alpha-coil	beta-coil
CPU	644.31	2061.35	511.09
OBJ	72.54383	144.73585	75.30725
1st	0.06642	0.08712	0.03722
2nd	0.08047	0.09116	0.10199
3rd	0.08598	0.09383	0.08301
4th	0.09264	0.06259	0.09753
5th	0.07908	0.08035	0.06710
6th	0.07693	0.06103	0.08220
7th	0.07001	0.06010	0.07886
C	0.09854	0.00459	-0.12038

Table 4.24: Separation results performed by the three planes in the previous table.

S			SA	$\#_s$	ab	ac	bc	Del
452	145	339						
128	195	165						
363	172	611	52%	2570[a]	14	2	7	130

[a] $2570+23+130+6\times 20+2\times 6+2 = 2857$, where 2857 is the number of residues in the 20 proteins; 23 is the number of points on the planes; 130 is the number of points in the fuzzy region; the last number 2 on the lefthand side is the two unknown residues; and the 2 in the term 2×6 comes from the splitting of the two proteins containing one unknown residue (see Appendix A for details).

The performance of three-state prediction seems better than that of two-state prediction. But not only is finding single separation planes quite timeconsuming but also it is somewhat artificial to set certain entries to equal 0.

4.5 Pairs of Separation Planes

4.5.1 Introduction

Because of the existence of a "fuzzy set[7]" or plural sets[8], it is impossible to separate completely, for example, alpha-helical points from non-alpha-helical points by means of a single plane. Several pairs of parallel planes, however, can perform this job [13]. The first pair of parallel planes partitions the whole space into three

[7] If the convex hulls of, for example, the alpha-helical points and the coil points intersect, then the intersection of the two sets is called a fuzzy set.

[8] The percentage of correctly predicted structures of Qian & Sejnowski's neural network without hidden units for the training set is about 63. The 63% performance of separation by the three indifference planes seemingly reflects the ill-distribution of the points in space, which corroborates the observation made in the previous section. Nevertheless, the performance of the same neural network for the testing set is about 63%, a good result.

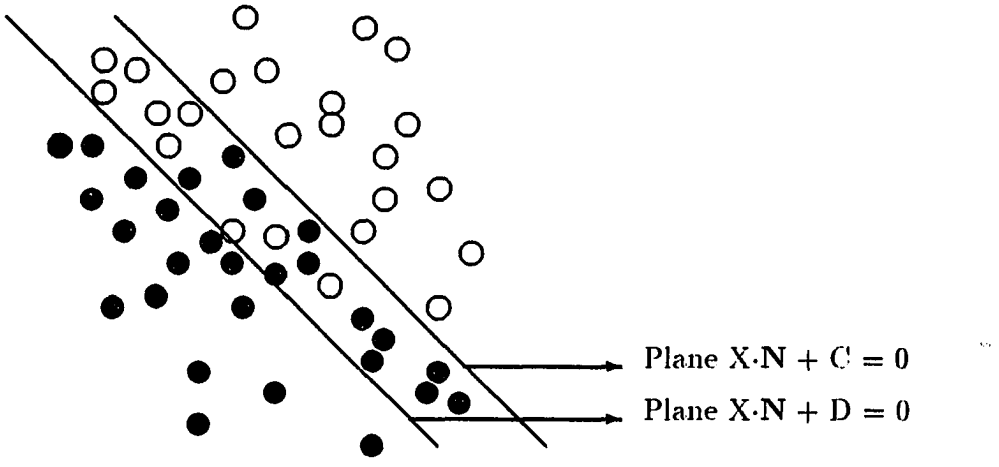

Figure 4.9: One pair of parallel separation planes.

parts so that, for alpha and nonalpha separation, the points belonging to the fuzzy set, which is the intersection of the two convex hulls of the two sets containing the alpha-helical points and the non-alpha-helical points, lie between the two parallel planes; the points lying on one side, which is not the region between the two planes, are all alpha-helical points; the non-alpha-helical points, which are not between the two planes, lie on the other side. The two parallel planes are constructed as closely as possible. One of the two parts not between the pair is considered an alpha-helical region, and the other a non-alpha-helical region (Fig. 4.9).

After discarding the points located on the two sides not between the first pair

of planes, we construct the second space partitioning pair, which is between the first pair, again into three parts so that the points belonging to the fuzzy set lie in between. Thus, one side contains only alpha-helical points; the other side only non-alpha-helical points. The distance between the second pair of planes is also as small as possible (Fig. 4.10). We then construct the third pair, the fourth pair, and so on, until all points are completely partitioned.

The pairs of parallel planes partition the entire space into several regions, each of which is categorized as either alpha-helical or non-alpha-helical (Fig. 4.11). The structure of the middle residue of a new segment is determined according to the type of region in which the segment is located.

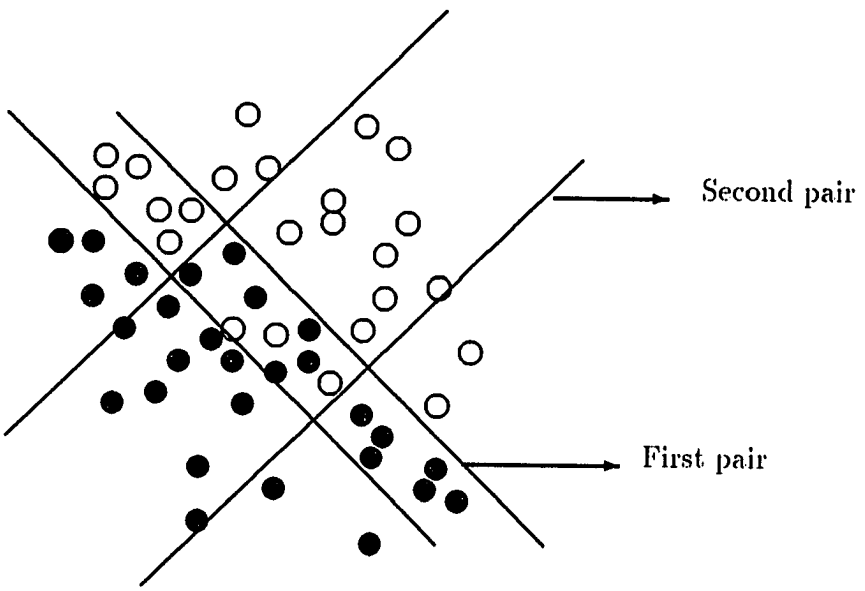

Figure 4.10: Two pairs of parallel separation planes.

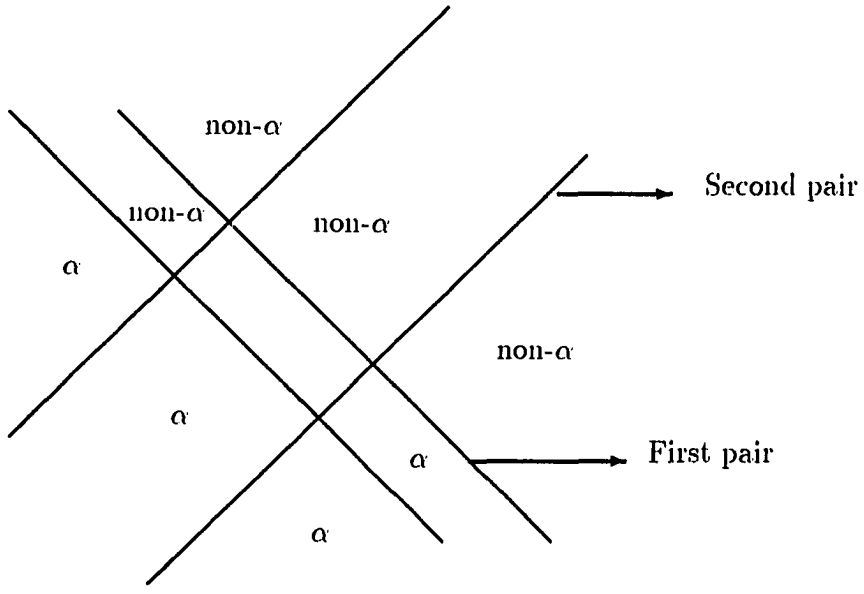

Figure 4.11: The categorization by the two pair of parallel planes in Fig. 4.10, in which the solid disks represent the alpha-helical points, and the circles represent the non-alpha-helical points.

Because of the limitation of local prediction, we did not attempt to construct sufficiently many pairs of parallel planes to partition the whole space; neither did we make predictions regarding the points lying on the regions neither partitioned or categorized.

We tested several different lengths of amino acid sequences, and found that the result for length 25 is not better than the result for length 17, which is the better choice.

4.5.2 Method

We solve the minimization problem below to find a normal vector **N** and two constants C and D to constitute the two parallel planes

$$X \bullet \mathbf{N} + C = 0 \text{ and } X \bullet \mathbf{N} + D = 0.$$

For the sake of convenience, we will use length seven to describe the minimization problem for the separation of alpha-helical points from non-alpha-helical points. We do likewise for other examples.

Minimize $C - D$

Subject to

$$\begin{aligned}
A \cdot \mathbf{N} + C_m &\geq 0, \\
B \cdot \mathbf{N} + D_r &\leq 0, \\
N_1 + N_2 + \cdots + N_{20} &= 0, \\
N_{21} + N_{22} + \cdots + N_{40} &= 0, \\
N_{41} + N_{42} + \cdots + N_{60} &= 0, \\
N_{61} + N_{62} + \cdots + N_{80} &= 0, \\
N_{81} + N_{82} + \cdots + N_{100} &= 0, \\
N_{101} + N_{102} + \cdots + N_{120} &= 0, \\
N_{121} + N_{122} + \cdots + N_{140} &= 0, \text{ and} \\
\mathbf{N} \bullet \mathbf{N} &= 1,
\end{aligned}$$

where

m is the number of alpha-helical points,

r is the number of non-alpha-helical points,

A is an $m \times 140$ matrix representing alpha-helical points, and

B is an $r \times 140$ matrix representing non-alpha-helical points,

and where

$\mathbf{N} = (N_1, N_2, \cdots, N_{140})^T$,

$C_m = (C, C, \cdots, C)^T$, a constant vector in R^m, and

$D_r = (D, D, \cdots, D)^T$, a constant vector in R^r,

are unknowns.

Note that C − D is greater than or equal to zero if there is a fuzzy set with a point P such that

$$P \bullet N + C \geq 0 \text{ and } P \bullet N + D \leq 0.$$

The difference between these two inequalities implies that C − D ≥ 0, where C is equal to D if and only if the resulting plane contains the fuzzy set; this special condition was not fulfilled in our experiments. On the other hand, C < D if no fuzzy set exists; again, this condition did not arise.

A. Type I Partition.

Type I partition is accomplished by means of three groups of parallel planes separating the alpha-helical points from the beta-sheet points, the alpha-helical points from the coil points, and the beta-sheet points from the coil points. To construct the first group of planes, i.e., to separate alpha-helical points from beta-sheet points, the matrix A in the linear programming model represents the alpha-helical points, the

matrix B represents the beta-sheet points, and the coil points are discarded. For the construction of the second group of planes, the matrix A represents the alpha-helical points, and the matrix B represents the coil points. Similarly, to construct the third group, let A represent the beta-sheet points, and let B represent the coil points.

B. Type II Partition.

Type II partition is similar to type I partition except, that type II partition is the separation of alpha-helical points from non-alpha-helical points, which includes the beta-sheet points and the coil points; of beta-sheet points from nonbeta-sheet points, which includes the alpha-helical points and coil points; and of the coil points from the noncoilpoints, which include the alpha-helical points and the beta-sheet points. Thus, the type II partition is also performed by three groups of parallel planes.

4.5.3 Prediction Procedure

A. For Type I Partition.

Each group of parallel planes partitions the 20k-dimensional space, i.e., R^{20k}, where k is the number of amino acids in each segment. For a new segment, the structure assigned to the middle residue depends upon the location of the segment in R^{20k}. We consider the partition performed by each group of planes independently, and one of two possible conformations will be given from each group if the point is not located in a mixed region such as the middle rectangular region in Fig. 4.11. The two possible conformations are alpha-helical and beta-sheet from the first group; alpha-helix and coil from the second group; and beta-sheet and coil from the third group. If two out of the three structures derived from the three groups of planes are

the same, then the same structure is assigned to the observed residue. Otherwise, we make no prediction.

B. For Type II Partition.

We consider the partitions performed by each of the three groups of parallel planes indepedently. For a new protein segment P, each group of planes will determine a number d as follows. For the sake of convenience, we will use the $\alpha- \sim \alpha$ group to describe how the number d is determined. (Similar expressions hold for the other two cases.) If P lies in the mixed region, d is 0. Otherwise, P lies either on an alpha-helical or a non-alpha-helical region, which is determined by a pair of parallel planes, L_1 and L_2, in the group. If P lies on an alpha-helical region, d is the distance from P to the closer one of the two planes L_1 and L_2; otherwise, d is the negative of the distance from P to the closer one of them.

Let d_i, where $i = 1, 2, 3$, be the number determined by the ith group of planes. If at least one of the d_i's is positive, the conformation (alpha-helix, beta-sheet, or coil) associated with the largest number will be assigned to the observed residue; if only one of the d_i's is 0 and the other two are negative, then the structure assigned to the residue will be alpha-helical if $d_1 = 0$, beta-sheet if $d_2 = 0$, and coil if $d_3 = 0$.; otherwise, no prediction is made.

4.5.4 Results and Discussion

First, we used the first 50 proteins, which include 7251 amino acids, [9] in the training set to test the convergence of the iterative linear programming scheme both for two-state separation (alpha-helical and non-alpha-helical) and for length seven, which is the number of residues in a segment. To construct the first pair of parallel planes for nine iterations, OSL[10] software was used. Prediction performance were also tested on the testing set, which included 15 proteins and 3402 amino acids,[11] by pairs of planes from each iteration. (See Tables 4.25 and 4.26.).

In Table 4.25, the numbers in the "norm" column decrease to 1, which signifies that the normal vectors approach a unit vector and that the distances between pairs of parallel planes decrease to about 0.54.

Note that because numbers were truncated after the 8th position of digital points, the "misplaced" points in the "S" column are, in fact, contained in either one of the corresponding planes. Results in Table 4.25 were obtained by means of the truncated numbers.

The performances, which include both number of partitioned points and accuracy of separation of the second and third iterations in the training set including 50 proteins, are better than that of the other iterations, and slightly better results are implied in the testing set. Iteration 1, which achieved a 85% accuracy, has a rela-

[9] There are $7251 - 50 \times 6 - 2 \times 6 - 2 = 6937$ seven-amino acid segments, for which the last number 2 on the lefthand side is the two unknown residues, and the 2 in the term 2×6 comes from the splitting of the two proteins containing one unknown residue (see Appendix A for details).

[10] OSL software was used to find the parallel planes in the last section, in which all outputs were optimal for their corresponding programming problems.

[11] There are $3402 - 15 \times 6 = 3312$ seven-amino acid segments in the testing set.

tively large non-alpha-helical tendency; on the other hand, from results in the "P" column, we found that the iterations could reduce overprediction of non-alpha-helical structure.

In Table 4.26, the numbers in each column are decreasing and the numbers in each row are increasing, which means that no oscillation is occurring; on the other hand, the top numbers in each column, from left top corner to right bottom corner, approach 1, which means that the sequence of unit normal vectors converge.

Table 4.25: The nine iterations for the first pair of parallel planes for two-state separation with segment length seven on 50 proteins.

Ite.[a]	Norm[b]	CPU	Dis.[c]	S[d]		SA[e]	$\#_s$[f]	P[g]		PA[h]	$\#_p$[i]
				168	45			37	31		
1st	1.375548	165.03[j]	0.686854	13	769	94%	995	40	358	85%	166
				209	10			48	29		
2nd	1.042482	135.72[k]	0.617340	14	746	98%	979	70	339	80%	486
				191	5			45	25		
3rd	1.014400	151.64[l]	0.600303	14	718	98%	928	69	330	80%	469
				182	23			48	22		
4th	1.013297	143.65	0.588287	31	660	94%	896	68	318	80%	456
				169	26			44	23		
5th	1.017352	148.69	0.571633	16	611	95%	822	73	291	78%	431
				173	9			54	25		
6th	1.016371	154.81	0.554744	25	587	96%	794	81	275	76%	435
				177	30			54	27		
7th	1.005795	169.75	0.549540	22	546	93%	775	76	262	75%	419
				190	37			59	28		
8th	1.004754	158.00	0.544411	21	542	93%	790	84	255	74%	426
				190	40			58	28		
9th	1.003725	166.55	0.541059	23	539	92%	792	83	246	73%	415

[a]Ite. = Iteration.

[b]Norm = The 2-norm of the normal vector of the parallel planes before normalization.

[c]Dis. is the distance between the two planes of the pair.

[d]The partition performed by the pair on the training set. See Table 4.16

[e]SA = Separation accuracy.

[f]$\#_s$ is the number of segments in the training set that were separated by the pair.

[g]The prediction performed by the pair on the testing set.

[h]PA = Prediction accuracy.

[i]$\#_p$ is the number of segments in the testing set that were predicted by the pair.

[j]The software MPSX was also used to solve the same problem, and the exact output resulted, but CPU time was 2863.11 seconds, or about 17 times 165.03.

[k]The MPSX required 1733.09 seconds to perform the same problem, or about 13 times 135.72.

[l]The MPSX required 1051.37 seconds, or about seven times 151.64.

Table 4.26: The inner product of the normalized normal vectors.

	I[a]	1st[b]	2nd	3rd	4th	5th	6th	7th	8th
I									
1st	0.726984								
2nd	0.614294	0.959249							
3rd	0.563561	0.923962	0.985805						
4th	0.522771	0.886338	0.957054	0.986877					
5th	0.474034	0.831045	0.908925	0.949743	0.982944				
6th	0.431682	0.768766	0.849476	0.896202	0.943668	0.983893			
7th	0.414767	0.737033	0.823874	0.872135	0.924434	0.970592	0.994238		
8th	0.402402	0.710829	0.797834	0.846155	0.901236	0.952657	0.981834	0.995269	
9th	0.390250	0.695449	0.782921	0.828789	0.884130	0.938559	0.970783	0.987395	0.996289

[a]The initial guess.

[b]The unit normal vector of the pair of parallel planes of the first iteration.

Two-state prediction. Table 4.28 lists the results obtained from ten pairs of parallel planes for two-state prediction for the seven amino acid segments in the training set (101 proteins and 17,460 residues [12]). Note that each pair of planes is the output of a linear programming problem and that each problem has been iterated only once.

About one-third of the points in the training set and about one-third of the points in the testing set were partitioned by these ten pairs of planes. The ratio of the number of points predicted in the testing set to the number separated in the training set by each pair of planes is about $\frac{1}{5}$, which is about the ratio of the number of points in the testing set (3312) to that in the training set (16840). It seems that the distribution of the points in the training set, is similar to that in the testing

[12]When the length of segments is seven, there are 4884 alpha-helical points, 3884 beta-sheet points, and 8072 coil points in the training set.

set. In other words, it seems possible to diminish accuracy on separation and, at the same time, to improve accuracy on prediction. The 63% accuracy on both separation and prediction for three-state prediction performed by Qian & Sejnowski's two-layer neural network evidently reflects this phenomenon.

Because the number of points partitioned by pairs after the fifth pair is smaller than 100 in the testing set and smaller than 500 in the training set, figures much smaller than the number of points in the testing and training sets, and from the results presented in Table 4.25, the iteration of planes will not increase the number of partitioned points, for the alpha-helix and nonalpha-helix points are terribly mixed. Thus, two-state prediction is limited, especially when one single separation plane is being used.

As can be seen from the results in Table 4.25, it might be possible to overcome the overprediction of non-alpha-helical structure by iterating each pair of planes more frequently. Note that the four chains of 2SOD, that is, 2SODo, 2SODy, 2SODb, and 2SODg, which have identical primary structures (see Table 4.27), are included in the training set so that the structures of the middle residues of certain identical segments, which are from different chains of 2SOD, will be different, and so that these segments will be contained in the fuzzy set. This situation will limit the performance of the linear programming model. Note that Sejnowski uses only one of the four chains. But two of the 4SBV chains are contained in Sejnowski's training set, which also will cause the problem mentioned here.

Table 4.29 lists the normal vectors of the first, the second, the sixth, and the tenth pair of parallel planes. Relatively great and positive numbers in the vector support alpha-helical conformation; on the other hand, relatively small and negative numbers

Table 4.27: The secondary structures of the four proteins 2SOD's. (c: coil; α : alpha-helix.)

	51	52	53	54	55	56	57	58	59	60
	N	T	Q	G	C	T	S	A	G	P
2SODo	c	c	c	α	α	α	c	c	c	c
2SODy	c	c	c	c	α	α	α	α	c	c
2SODb	c	c	c	c	c	c	c	c	c	c
2SODg	c	c	c	c	c	c	c	c	c	c
	131	132	133	134	135	136	137	138	139	140
	E	S	T	K	T	G	N	A	G	S
2SODo	c	c	c	c	c	c	c	c	c	c
2SODy	c	c	c	c	c	c	c	c	c	c
2SODb	c	c	c	c	c	c	c	c	c	c
2SODg	c	α	α	α	α	c	c	c	c	c

support non-alpha-helical conformation. Normal vectors reflect some information about the contribution to each type of secondary structure by amino acids at each position. For example, that proline is a helix breaker is reflected by relatively small negative numbers, especially when on the C-terminal end; glutamic acid, which is a helix former on the N-terminal end and a strong β-sheet breaker, is represented by relatively positive, large numbers; alanine, leucine, and methionine are helix formers, especially when on the C-terminal end; and lysine and arginine are helix formers when on the C-terminal end.

Table 4.30 lists the inner products of the unit normal vectors of the ten pairs of planes in Table 4.28. The smallest number in the table is 0.770179, which indicates that the "rotation angle" between any two normal vectors of pairs of planes is small. Thus, the partitioned points in the training set, roughly speaking, are distributed well in the sense that alpha-helical points lie on one side of the "smooth" multiplane determined by the ten pair of planes and that the non-alpha-helical lie on the other side.

On the other hand, because the "rotation angles" between later normal vectors are smaller than between former vectors, that is, the values of inner products are greater and, from Table 4.28, the numbers of partitioned points are smaller, unpartitioned points are terribly mixed.

Table 4.28: The two-state prediction for seven-amino acid segments in the training set.

Pair	CPU	Dis.	S		SA	$\#_s$	P		PA	$\#_p$
			123[a]	3			18	10		
1st	2112.07	0.830707	17	1239	99%	1382	9	222	93%	259
			89	35			13	9		
2nd	673.66	0.825133	34	451	89%	609	6	71	85%	99
			98	46			11	14		
3rd	641.10	0.801053	19	575	91%	738	9	108	84%	142
			69	54			13	14		
4th	1486.59	0.754658	19	481	88%	623	5	98	85%	130
			37	28			9	9		
5th	1413.89	0.721548	31	457	89%	553	5	77	86%	100
			54	34			15	17		
6th	574.83	0.723060	16	289	87%	393	5	53	76%	90
			32	37			4	5		
7th	1176.24	0.697520	2	231	87%	302	5	42	82%	56
			40	46			7	8		
8th	1188.54	0.665975	19	204	79%	309	2	43	83%	60
			46	33			5	12		
9th	583.55	0.660534	11	273	88%	363	3	52	79%	72
			33	44			6	4		
10th	1079.24	0.602637	5	260	86%	342	4	52	88%	66
			621	360			101	102		
All[b]			173	4460	91%	5614[c]	53	818	86%	1071[d]

[a]The distance from any of the 123 points to the plane, which is one of the two planes such that all points lying on one side of it are alpha-helical, is between 2×10^{-1} and 10^{-3}, and the average of the distances is about 6.6×10^{-2}; the distance from any of the 17 points below, which are, in fact, on the planes, to the same plane is $10^{-16}, 10^{-8}$, or 9×10^{-9}, and the average of the distances is about 8.2×10^{-9}.

[b]The prediction performed by the 10 pair of parallel planes on the testing set.

[c]$17,460 = 5614 + 11,226 + 101 \times 6 + 2 \times 6 + 2$, where 11,226 is the number of seven-residue segments that were not predicted, 101 is the number of proteins in the training set, the last number 2 on the righthand side is the two unknown residues, and 2 in the term 2×6 is the splitting of the two proteins containing one unknown residue (see Appendix A for details).

[d]The total number of residues in the testing set is $3402 = 1074 + 6 \times 15 + 2238$, where 15 is the number of proteins in the testing set, and 2238 is the number of seven-residue segments not predicted.

Table 4.29: The normal vectors of the second, the sixth, and the tenth pair of parallel planes. The numbers in the first two rows (S_1 and S_2) are the constant terms in the equations of the planes.

S_1	0.562056	0.577381	0.430183	0.347987
S_2	-0.540403	-0.649251	-0.567913	-0.548244
(Position[a] 1)				
Residue	First pair	Second pair	Sixth pair	Tenth pair
A	0.005468	0.023201	0.152979	0.133138
R	0.075784	-0.002298	-0.090865	-0.027708
N	-0.027001	0.013161	-0.024840	0.011972
D	0.012582	0.073869	0.030170	0.001611
C	-0.075841	-0.188956	-0.072478	-0.030083
Q	0.046783	0.072394	0.063079	-0.019587
E	0.052384	0.123321	0.069411	0.090187
G	-0.039885	-0.106068	-0.148528	-0.114755
H	-0.118235	-0.074113	-0.045119	-0.049452
I	0.057331	0.022446	-0.041844	0.017746
L	0.007934	-0.001955	0.072088	0.091612
K	0.020523	0.021503	0.096915	-0.035816
M	0.054548	0.036884	0.088450	-0.048180
F	0.104333	0.075377	0.022726	0.053950
P	-0.014487	-0.062668	-0.105774	-0.027409
S	-0.023121	-0.080881	-0.065163	-0.137276
T	-0.014793	-0.077147	-0.044238	-0.062517
W	-0.102116	0.024986	0.069910	0.187907
Y	-0.028044	-0.052483	0.008151	-0.017310
V	0.005854	0.159425	-0.035032	-0.020970

[a]The seven positions in a seven-residue segment counted from the N-terminal end to the C-terminal end.

Table 4.29 (Continued.)

(Position 2)				
Residue	First pair	Second pair	Sixth pair	Tenth pair
A	0.045093	0.029227	0.142521	0.123351
R	0.055546	0.081957	-0.047947	-0.012378
N	0.058329	-0.042048	-0.046644	-0.032181
D	0.000786	0.026099	0.009698	0.069807
C	-0.047457	-0.061859	-0.045108	-0.033583
Q	0.036759	0.024259	0.009035	0.019301
E	0.126698	0.191933	0.145885	0.114969
G	-0.169641	-0.121068	-0.142464	-0.145763
H	0.006137	-0.069331	0.120596	0.031903
I	0.016483	-0.064552	-0.046767	-0.000462
L	0.018670	0.050717	0.020999	0.074480
K	0.058949	0.081957	0.014386	-0.001659
M	-0.045756	0.173114	0.070370	-0.031784
F	0.068024	0.022642	0.008217	0.075258
P	-0.109245	-0.102885	-0.041128	-0.137754
S	-0.131533	-0.171643	-0.112820	-0.108265
T	0.047411	0.002168	-0.040282	-0.092920
W	-0.009872	-0.024579	0.112022	0.152410
Y	-0.064212	-0.058216	-0.119822	-0.062986
V	0.038831	0.032108	-0.010747	-0.001746

Table 4.29 (Continued.)

(Position 3)				
Residue	First pair	Second pair	Sixth pair	Tenth pair
A	0.126256	0.146744	0.131173	0.135130
R	-0.138408	-0.190755	-0.037535	0.003153
N	-0.029649	-0.018291	0.059980	-0.081202
D	0.068559	0.073613	-0.027327	-0.067306
C	-0.132751	-0.129127	-0.089149	0.042620
Q	0.080537	0.131961	0.048502	0.087698
E	0.128712	0.192274	0.161219	0.140859
G	-0.172999	-0.146743	-0.245617	-0.285380
H	0.106713	0.079022	-0.016901	0.002581
I	-0.005223	0.029718	-0.033945	-0.017736
L	0.080268	0.039253	0.076742	0.120847
K	0.088117	0.155358	0.029748	0.062711
M	0.112866	0.181004	0.126291	0.060487
F	0.040457	0.040304	0.051391	0.040711
P	-0.222775	-0.179198	-0.017648	-0.132435
S	-0.108231	-0.106331	-0.108601	-0.130372
T	-0.086350	-0.091742	-0.106110	-0.058208
W	0.100315	-0.021127	0.108701	0.192544
Y	0.030288	-0.074060	-0.131281	-0.090717
V	-0.066702	-0.111876	0.020367	-0.026287

Table 4.29 (Continued.)

(Position 4)				
Residue	First pair	Second pair	Sixth pair	Tenth pair
A	0.142896	0.199983	0.194278	0.175250
R	0.004754	0.052975	-0.053563	-0.129452
N	-0.042031	-0.174259	-0.075199	-0.129751
D	-0.015089	-0.023765	-0.016046	-0.011489
C	-0.121938	-0.097754	-0.064410	-0.034377
Q	0.090440	0.049671	0.032683	0.052776
E	0.097150	0.148013	0.239403	0.205570
G	-0.193050	-0.188745	-0.309389	-0.311595
H	0.068712	0.123824	-0.012159	0.084714
I	0.070439	0.051175	0.062845	0.075084
L	0.096907	0.092492	0.163936	0.163283
K	0.103493	0.060392	0.058118	0.079596
M	0.082474	0.253915	0.191165	0.075010
F	0.106430	0.034728	0.126296	0.100280
P	-0.288980	-0.280663	-0.258778	-0.216434
S	-0.114968	-0.116040	-0.098338	-0.066000
T	-0.140555	-0.136529	-0.134740	-0.190537
W	0.110963	0.101281	0.054159	0.049397
Y	0.004331	-0.081653	-0.042490	0.013743
V	-0.062376	-0.069041	-0.057772	0.014933

Table 4.29 (Continued.)

(Position 5)				
Residue	First pair	Second pair	Sixth pair	Tenth pair
A	0.118934	0.156359	0.205538	0.191152
R	0.017251	-0.040424	-0.012232	0.054929
N	0.012465	-0.039508	-0.079924	-0.143338
D	0.018445	-0.132412	-0.015457	-0.053463
C	0.118664	0.120237	0.057823	0.047078
Q	0.089266	0.142451	0.149401	0.134881
E	0.134009	0.162850	0.125426	0.142774
G	-0.225810	-0.147345	-0.184491	-0.178410
H	0.078037	0.155141	0.047186	0.048708
I	0.024924	-0.146599	0.051970	0.053970
L	0.112561	0.170007	0.154249	0.217666
K	0.026765	0.033289	0.121996	0.112276
M	0.122904	0.168971	0.081381	0.141501
F	0.040046	0.049393	0.065518	0.094098
P	-0.504376	-0.393860	-0.475040	-0.635442
S	-0.117844	-0.143294	-0.181919	-0.152118
T	-0.097216	-0.157069	-0.190969	-0.147975
W	0.124782	0.232236	0.155041	0.129333
Y	-0.038784	-0.045908	-0.042967	-0.038010
V	-0.055021	-0.144517	-0.032531	-0.019611

Table 4.29 (Continued.)

(Position 6)				
Residue	First pair	Second pair	Sixth pair	Tenth pair
A	0.109120	0.096613	0.110384	0.144374
R	0.038913	-0.052571	0.008892	-0.042541
N	-0.046140	-0.143566	-0.033289	-0.164165
D	-0.087881	-0.129604	-0.003509	-0.036396
C	-0.073372	0.018285	-0.024963	0.078636
Q	0.073900	0.060365	0.072767	0.002120
E	0.088422	0.203885	0.104430	0.000751
G	-0.113065	-0.042271	-0.194709	-0.070879
H	0.070081	0.097030	0.009805	0.078960
I	0.059484	0.015630	0.013997	-0.020926
L	0.164443	0.150914	0.143743	0.149503
K	0.089018	0.062699	0.090791	0.151096
M	0.122436	0.151649	0.146560	0.127470
F	0.106376	0.055721	0.068483	0.085905
P	-0.435547	-0.348022	-0.286135	-0.369852
S	-0.230493	-0.227542	-0.163053	-0.155935
T	-0.175159	-0.093028	-0.135169	-0.092005
W	0.122516	0.122271	0.069373	0.180369
Y	-0.000458	-0.042634	-0.021564	-0.050577
V	0.117405	0.044174	0.023167	0.004093

Table 4.29 (Continued.)

(Position 7)				
Residue	First pair	Second pair	Sixth pair	Tenth pair
A	0.038625	0.104614	0.095047	0.112871
R	0.038471	-0.117105	0.037433	-0.035137
N	-0.029258	-0.131734	-0.112439	0.010356
D	-0.069508	-0.138639	-0.056579	-0.163198
C	-0.004213	-0.044918	-0.115212	0.036656
Q	0.079837	0.027054	0.100444	0.134919
E	0.012102	0.085059	-0.005741	0.030828
G	-0.115141	-0.061117	-0.118914	-0.119195
H	0.093065	0.223832	0.158140	0.058783
I	0.048033	-0.083979	-0.009986	0.048465
L	0.099986	0.133732	0.063374	0.070257
K	0.082442	0.086673	0.064925	0.103084
M	0.146835	0.231794	0.182329	0.188279
F	0.008356	-0.014745	0.118561	0.088459
P	-0.279265	-0.315760	-0.266188	-0.305580
S	-0.103607	-0.112449	-0.137529	-0.159087
T	-0.176345	-0.109046	-0.166916	-0.154156
W	0.136209	0.211843	0.189341	0.139417
Y	-0.031782	0.016978	0.027404	-0.082980
V	0.025154	0.007911	-0.047493	-0.003042

Table 4.30: The inner product of the normalized normal vectors of the ten pair of parallel planes.

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th
1st									
2nd	0.851750								
3rd	0.815257	0.812966							
4th	0.792132	0.770657	0.861484						
5th	0.808601	0.788934	0.871070	0.901453					
6th	0.827218	0.811528	0.866814	0.885454	0.896754				
7th	0.820555	0.797946	0.851350	0.900840	0.908771	0.916297			
8th	0.841345	0.783296	0.841941	0.850165	0.890116	0.916648	0.945680		
9th	0.800363	0.784424	0.856642	0.856868	0.880685	0.878995	0.925763	0.916722	
10th	0.823918	0.770179	0.844505	0.863859	0.889108	0.880333	0.917776	0.919611	0.925669

Three-state prediction. Three-state prediction improves the overprediction problem. Tables 4.31 and 4.32 list the results of type I partition and of type II partition, respectively, for 17-amino acid segments in the training set (101 proteins[13]) and in the testing set (15 proteins[14]). Separation and prediction performances of the two partition schemes, type I and type II, are slightly different, but much more computer time was required to construct the planes for type II partition.

The beta-sheet structure is more difficult to predict than is alpha-helix (or coil). For example, as can be seen from Table 4.31, 74% accuracy was achieved by using three pairs of $\alpha - c$ planes, 1 pair of $\alpha - \beta$ planes, and 1 pair of $\beta - c$ planes; when the numbers of each type of pair, $\alpha - \beta$, $\alpha - c$, and $\beta - c$, are 2, 2, and 2, and 2, 3, and 2, the percentages are similar, but the numbers of separated points and of predicted points for the latter are significantly greater.

[13]The 101 proteins include 4524 alpha-helical points, 3634 beta-sheet points, and 7652 coil points when the length of the segment is seventeen.

[14]The 15 proteins includes 3162 length 17 segments

The number of separated points is about five times that of the number of predicted points, which is about the ratio of the number of points in the training set to that in the testing set. We have made the same observation in previous sections. (See Table 4.28).

Table 4.34 lists results when the segment length is 7 or 25, results not as good as when length is 17.

Table 4.31: The three-state prediction of Type I for 17-amino acid segments. (Note: because the misplaced points in the "S" column lie on the corresponding plane derived from the OSL, we set 0.00000001 as threshold for the case 4, 4, and 4 to avoid predicting these points. The separation result is improved, but the prediction results are slightly differet. Similar expressions hold for the cases listed in subsequent tables.)

# of $\alpha-\beta$[a]	# of $\alpha-$c[b]	# of $\beta-$c	S[c]			SA	$\#_s$	P			PA	$\#_p$
			436	35	23			58	16	7		
			18	400	16			18	79	11		
2	1	1	65	19	750	90%	1762	23	27	114	71%	353
			635	20	65			78	14	15		
			22	324	24			20	56	17		
1	3	1	75	15	1089	90%	2269	33	16	192	74%	411
			767	35	41			98	16	10		
			29	400	21			32	79	14		
2	2	1	83	19	946	90%	2341	41	27	159	71%	176
			635	26	104			78	17	21		
			22	435	63			20	67	41		
1	3	2	75	21	1570	89%	2951	33	26	270	72%	573
			767	48	65			98	24	14		
			29	560	55			32	98	32		
2	2	2	83	28	1319	90%	2954	41	37	222	70%	598
			896	48	104			128	24	21		
			32	560	63			36	98	41		
2	3	2	99	28	1570	89%	3400	51	37	270	70%	706
			635	34	124			78	21	28		
			22	510	83			20	76	47		
1	3	3	75	57	1864	88%	3404	33	36	330	72%	669
			843	53	65			111	32	14		
			46	631	55			39	111	32		
3	2	2	97	31	1319	89%	3140	51	44	222	68%	656

[a] The number of pair of parallel planes separating the alpha-helical points from the beta-sheet points.

[b] c = coil.

[c] See Table 4.33.

Table 4.31 (Continued.)

# of $\alpha-\beta$	# of $\alpha-c$	# of $\beta-c$	S			SA	$\#_s$	P			PA	$\#_p$
			997	53	104			150	32	21		
			58	631	63			44	111	41		
3	3	2	115	31	1570	88%	3622	62	44	270	69%	775
			997	77	124			150	44	28		
			58	773	83			44	137	47		
3	3	3	115	89	1864	87%	4180	62	63	330	68%	905
			1281	70	132			195	52	49		
			65	956	98			63	156	68		
4	4	4	120	100	2436	89%	5258	83	84	432	66%	1182

Table 4.32: The three-state prediction of Type II for 17-amino acid segments.

# of $\alpha - \sim \alpha$	# of $c - \sim c$	# of $\beta - \sim \beta$	S			SA	$\#_s$	P			PA	$\#_p$
			510	1	1			52	19	15		
			2	438	0			14	64	18		
1	1	1	8	11	1055	99%	2026	36	24	175	70%	417
			796	1	3			104	19	15		
			7	440	0			30	66	20		
2	1	1	18	11	1106	98%	2382	50	24	188	69%	516
			804	19	6			106	33	16		
			7	661	6			29	109	22		
2	1	2	19	39	1234	97%	2795	50	43	207	69%	615
			823	19	12			110	33	23		
			7	668	8			29	112	25		
2	2	2	22	41	1508	97%	3108	50	43	256	70%	681
			1032	26	33			160	43	36		
			14	832	33			40	139	39		
3	3	3	31	61	1853	95%	3915	71	67	326	68%	921
			1217	33	53			188	54	47		
			26	962	58			55	161	56		
4	4	4	65	72	2146	93%	4632	89	84	397	66%	1131

Table 4.33: The nine numbers in the entries of the S or P column in some tables have the same role as A, B, C, D, E, F, G, H, and I, respectively.

Real structure	Predicted structure		
	alpha-helix	beta-sheet	coil
alpha-helix	A	B	C
beta-sheet	D	E	F
coil	G	H	I

Table 4.34: The three-state prediction of type I and type II for amino acid segments with lengths seven and 25.

Type I; Length = 7														
# of $\alpha-\beta$	# of $\alpha-c$	# of $\beta-c$	S			SA	$\#_s$	P			PA	$\#_p$
			120	3	19			17	5	7		
			2	171	7			4	26	4		
2	2	2	1	17	660	95%	1000	6	13	105	79%	187
			189	12	28			29	8	10		
			7	223	16			4	31	7		
3	3	3	9	26	895	93%	1405	7	16	154	80%	266
Type II; Length = 7														
# of $\alpha-\sim\alpha$	# of $c-\sim c$	# of $\beta-\sim\beta$	S			SA	$\#_s$	P			PA	$\#_p$
			211	0	2			31	11	3		
			5	139	6			6	16	8		
2	2	1	12	0	567	97%	942	9	6	94	77%	184
Type I; Length = 25														
# of $\alpha-\beta$	# of $\alpha-c$	# of $\beta-c$	S			SA	$\#_s$	P			PA	$\#_p$
			368	10	9			40	15	10		
			8	431	4			16	50	12		
1	1	1	32	35	840	94%	1737	29	22	122	67%	316
			468	23	9			67	18	10		
			13	537	4			24	65	12		
2	1	1	42	45	840	93%	1981	40	34	122	65%	392
			768	23	30			111	18	20		
			25	537	8			40	65	20		
2	2	1	91	45	1103	92%	2630	66	34	180	64%	551

4.6 Conclusion

A reliable prediction scheme should be both quantitative and objective. Obviously, the information theory method and the neural network models are largely quantitative; nonetheless, both are somewhat subjective. For example, Robson *et al.* use decision constants to improve prediction results, and Qian & Sejnowski observe performances of networks on a testing set. A great accuracy prediction result on a certain testing set does not necessarily imply such accuracy for subsequent. Yet, using a set of numbers to predict is more economical than exhaustively searching as in Levin's similarity matrix and is more stable than using Chou & Fasman's conformational parameters method.

We believe that a basic requirement for a reliable prediction scheme is that of achieving a good performance on the training set. The linear programming model was trained on a set of known structure proteins, which form the constraints of the optimization problem, and so the partition on the training set is optimized in some sense.

Because of the terribly distributed points in space, coil is overpredicted by both our similarity scale and our similarity matrix. On the other hand, in the prediction of secondary structures of new proteins, both methods require exhaustive search and are timecomsuming.

BIBLIOGRAPHY

[1] C. B. Anfinsen, E. Haber, M. Sela, and F. H. White. Jr., Proc. Natl. Acad. Sci. U.S. 47, 1309 (1961).

[2] W. Kabsch and C. Sander. Proc. Natl. Acad. Sci. USA 81, 1075-1078 (1984).

[3] P. Y. Chou and G. D. Fasman. Advances in Enzymology 47, 45-148 (1978).

[4] T. J. Sejnowski and R. R. Rosenberg. "Parallel networks that learn to pronounce English text." Compl. Syst. 1, 145–168 (1987).

[5] B. Robson and R. H. Pain. "Analysis of the code relating sequence to conformation in proteins: Possible implications for the mechanism of formation of helical regions." J. Mol. Biol. 58, 237–259 (1971).

[6] B. Robson. "Analysis of the code relating sequence to conformation in globular proteins." Biochem. J. 141, 853–867 (1974).

[7] B. Robson and E. Suzuki. "Conformational properties of amino acid residues in globular proteins." J. Mol. Biol. 107, 327–356 (1976).

[8] J. Garnier, D. J. Osguthorpe and B. Robson. "Analysis of accuracy and implications of simple methods for predicting the secondary structure of globular proteins." J. Mol. Biol. 120, 97–120 (1978).

[9] J. M. Levin, B. Robson and J. Garnier. "An algorithm for secondary structure determination in proteins based on sequence similarity." FEBS Letters 205, 303–308 (1986).

[10] N. Qian and T. J. Sejnowski. "Predicting the secondary structure of globular proteins using neural network models." J. Mol. Biol. 202, 865–884 (1988).

[11] D. E. Rumelhart, G. E. Hinton and R. J. Williams. "Learning internal representations by error propagation." Parallel Distributed Processing. 1. Ed. D. E. Rumelhart, J. L. McClelland and the PDP Research Group.

[12] W. Kabsch and C. Sander. "How good are predictions of protein secondary structure?" FEBS Letters 155, number 2,179–182 (1983).

[13] O. L. Mangasarian. "Multisurface method of pattern separation." IEEE Transactions on Information Theory 14, 801–807 (1968).

[14] S. S. Wilks. Mathematical Statistics. New York: Wiley, 1962.

[15] P. Lancaster and M. Tismenetsky. The Theory of Matrices. 2nd ed. New York: Academic press, 1985.

APPENDIX A. DATABASE

A.1 Training Set

Code	Protein name	L[a]
1AZU	Azurin	128
1BP2	Phospholipase A2	123
1CC5	Cytochrome c5 (oxidized)	83
1CCR	Cytochrome c (rice)	111
1CPV	Calcium-bindind parvalbumin b	108
1CRN	Crambin	46
1CTX	α-Cobratoxin	71
1CY3	Cytochrome c3	118
1CYC	Ferrocytochrome c	103
1ECD	Haemoglobin (deoxy)	136
1EST	Tosyl-elastase	240
1FC2c	Immunoglobulin FC-Frag B complex	55[b]
1FC2d	Immunoglobulin FC-Frag B complex	224
1FDHa	Haemoglobin (deoxy, human fetal)	111
1FDHg	Haemoglobin (deoxy, human fetal)	146
1FDX	Ferredoxin	54
1FX1	Flavodoxin	148
1GCN	Glucagon (pH 6 - pH 7 form)	29
1GCR	γ-Crystallin	174
1GF1	Insulin-like growth factor	70
1GF2	Insulin-like growth factor	67
1GP1a	Glutathione peroxidase	198[c]
1GP1b	Glutathione peroxidase	198
1HDSa	Haemoglobin	141
1HDSb	Haemoglobin	145

[a] L = Length.

[b] The last three residues, in which the type of the first two residues are unknown, are not included.

[c] The type of the 45th amino acid in both of the two proteins 1GP1a and 1GP1b is not shown in our database. We split each protein as two sequences. The first sequence is composed of the first 44 amino acids of the original sequence, and the second sequence is composed of the amino acids from the 46th to the last. The split two sequences are treated as two proteins.

Training set (Continued.)

Code	Protein name	L
1HIP	High potential iron protein	85
1LZ1	Lysozyme	130
1LZT	Lysozyme, triclinic crystal form	129
1MBD	Myoglobin (deoxy, pH 8.4)	153
1MBS	Myoglobin (met)	153
1MLTa	Melittin	26
1P2P	Phospholipase A2	124
1PFC	Fragment of IgG	113
1PPT	Avian prancreatic polypeptide	36
1REIa	Immunoglobulin B-J fragment V	107
1RHD	Rhodanese	293
1RN3	Ribonuclease	124
1SN3	Scorpion neurotoxin (variant 3)	65
1TIMa	Triose phosphate isomerase	247
2APP	Acid proteinase, penicillopepsin	323
3LDH	Lactate dehydrogenase complex	329
2APR	Acid protease	325
2AZAa	Azurin	129
2B5C	Cytochrome b5 (oxidized)	93
2CAB	Carbonic anhydrase form b	260
2CCYa	Cytochrome c (prime)	128
2CYP	Cytochrome c peroxidase	294
2DHBa	Haemoglobin (horse, deoxy)	141
2DHBb	Haemoglobin (horse, deoxy)	146
2GCH	γ-Chymotrypsin a	241[a]
2GN5	Gene 5/DNA binding protein	87
2IG2l	Immunoglobunlin G1	216
2IG2h	Immunoglobunlin G1	455
2INSa	Insulin	21
2INSb	Insulin	29
2KAIa	Kallikrein a	80
2KAIb	Kallikrein a	152
2KAIi	Kallikrein a	58
2LDX	Lactate dehydrogenase	331
2LH1	Leghaemoglobin (acetate, met)	153
2MCPh	Ig Fab mcpc603/phosphocholine	222
2MCPl	Ig Fab mcpc603/phosphocholine	220
2PABa	Prealbumin (human plasma)	127
2RHE	Immunoglobulin B-J fragment V-MN	114

[a]The type of the 14th, 15th, 47th, and 48th amino acids in the protein 2GCH are not shown in our database and are simply deleted.

Training set (Continued.)

Code	Protein name	L
2SGA	Proteinase A	181
2SNS	Staphylococcal nuclease complex 5	149
2SODo	Cu,Zn superoxide dismutase	151
2SODy	Cu,Zn superoxide dismutase	151
2SODb	Cu,Zn superoxide dismutase	151
2SODg	Cu,Zn superoxide dismutase	151
2SSI	Streptomyces subtilisin inhibito	113
2STV	Satellite tobacco necrosis virus	195
2TAAa	Taka-amylase	478
2TBVa	Tomato bushy stunt virus	387
3ADK	Adenylate kinase	194
3C2C	Cytochrome c2 (reduced)	112
3CNA	Concanavalin A	237
3FXC	Ferredoxin	98
3HHBa	Haemoglobin (deoxy)	141
3HHBb	Haemoglobin (deoxy)	146
3ICB	Calcium-binding protein	75
3PCY	Plastocyanin (Hg^{2+} substituted)	99
3PGK	Phosphoglycerate kinase complex	415
3PGM	Phosphoglycerate mutase	241
3RP2a	Rat mast cell protease	224
3SGBe	Proteinase B	185
3SGBi	Proteinase B	56
3TLN	Thermolysin	316
451C	Cytochrome c551 (reduced)	82
4APE	Acid proteinase, endothiapepsin	330
4CTSa	Citrate synthase complex	437
4DFRa	Dihydrofolate reductase	159
4FXN	Flavodoxin (semiquinone form)	138
4MDHa	Cytoplasmic malate dehydrogenase	333
4SBVa	Southern bean mosaic virus coat protein	260
4SBVc	Southern bean mosaic virus coat protein	260
5CPA	Carboxypeptidase	307
5PTI	Trypsin inhibitor	58
5RXN	Rubredoxin (oxidized)	54
6ADHa	Alcohol dehydrogenase complex	374
8CATa	Catalase	506

A.2 Testing Set

Code	Protein name	L
1ABP	1-Arabinose-binding protein	306
1ACX	Actinoxanthin	108
1HMQa	Haemerythrin (met)	113
1IGEa	Fe fragment (model)	322
1NXB	Neutrotoxin b	62
1PPD	2-hydroxyethylthiopapain d	212
1PYP	Inorganic pyrophosphatase	285
2ACT	Actinidin (sulphhydryl proteinase)	220
2ALP	α-Lytic protease	198
2CDV	Cytochrome c3	107
3GRS	Glutathione reductase	478
2SBT	Subtilisin novo	275
3GPDr	Glyceraldehyde-3-P-dehydrogenase	334
6APIa	Modified α-1-antitrypsin	347
6APIb	Modified α-1-antitrypsin	35

APPENDIX B. PROGRAMS

B.1 Main Program

```
#include <stdio.h>
#include <malloc.h>
#include <math.h>

#define RES  20
#define NP    7      /* NP = number of residues in a segment.     */
#define RESNP  RES*NP
#define NUM   20     /* # of proteins in training set used.       */

#define USCORE  7    /* Similarity value for different structures.*/
#define SCORE   7    /* Similarity value for the same structure.  */
#define K_NEAR  5    /* For k-nearest neighbors.                  */
#define THR     5    /* Threshold for k-nearest neighbors.        */

#define RATE  1.0    /* Learning rate. n1'=n0+RATE*(n1-n0).       */
                     /* where n0 is the initial vector.           */
                     /* 0.0 <= RATE <= 1.0.                       */
#define INFO  0      /* Construct artificial database if INFO=1.  */
                     /* Warning: Let INFO=0 to use real database. */
#define BAY   0      /*BAY=1 if Baysian is used and when INFO=1.  */
#define ITE   0   /* ITE = 1 only if iteration is executed.        */
#define TEST  0   /* If TEST=1, then use the output of mpsx to     */
                  /*  separate artificial data.                    */
#define PARL  0   /* If PARL=1, then find parallel planes.         */
#define PAIR  10  /* PAIR is the number of pairs of parallel lines. */
                  /* Warning: To construct the initial guess       */
                  /* to find, e.g., the 2nd pair of lines, let     */
                  /* PAIR=1 and ITE=0; then let PAIR=2 to do       */
                  /* iterations for the 2nd pair (let ITE=1)       */
                  /* or to find the 3rd pair (let ITE=0).          */
                  /* Warning: PAIR != 0 in any situation.          */
                  /* For 3-state prediction of type 1, let         */
                  /* PAIR=PAIRAB (PAIRAC, PAIRBC) for A-B (A-C, B-C)*/
                  /* separation. For 3-state prediction of type 2, */
                  /* let PAIR=PAIRA (PAIRB, PAIRC) for             */
                  /* A-~A (B-~B, C-~C) separation.                 */
```

```
#define BIAS 0.0
#define CA 0.00000000
#define CB -0.00000000

#define PAIRAB 3      /* For 3-state prediction of type I.          */
#define PAIRAC 3      /* The number of pairs of parallel            */
#define PAIRBC 3      /* planes in each group.                      */

#define PAIRA  2      /* For 3-state prediction of type II.         */
#define PAIRB  1      /* (Parallel planes)                          */
#define PAIRC  2      /* Let STRUC2=2.                              */

#define FIRST 0       /* To construct the first pair at iteration 1, */
                      /* let FIRST==1; otherwise, let FIRST==0      */
                      /* (Let ITE==0 when FIRST == 1).              */
#define STRUC1  (1)   /* For 3-state prediction (single or pairs).  */
#define STRUC2  (2)   /* 1 --- helix; 0 --- beta; (-1) --- coil     */
                      /* For type II separation, let STRUC2==2 and  */
                      /* use 3-state function,i.e., let STATE==3.   */
                      /* If STRUC2==2 & STRUC1==1(0, -1), then      */
                      /* A vs. ~A (B vs. ~B, C vs. ~C).             */

#include "happy"      /* Use the first 20 intergers to represent the
                         20 residues. */

/* The following blocks of included functions contain planes derived
   from either MPSX or OSL.*/

/*#include "mpsxout.c"       Single plane separation.*/

/*
#include "ab_mpsxout.c"      3 single planes separation.
#include "ac_mpsxout.c"      Type 1 separation.
#include "bc_mpsxout.c"
*/

/*
#include "pab0.c"         3-state prediction of type 1.
#include "pac0.c"         Pairs of parallel planes.
#include "pbc0.c"         Example:  pab0.c contains the first pair
```

```
                                        for alpha-beta separation.
#include "pab1.c"
#include "pac1.c"
#include "pbc1.c"

#include "pab2.c"
#include "pac2.c"
#include "pbc2.c"

#include "pab3.c"
#include "pac3.c"
#include "pbc3.c"
*/

/*
#include "pa0.c"            3-state prediction of type 2.
#include "pb0.c"            Pairs of parallel planes.
#include "pc0.c"            Example: pa1.c contains the second pair
                                     for alpha and non-alpha separation.

#include "pa1.c"
#include "pb1.c"
#include "pc1.c"

#include "pa2.c"
#include "pb2.c"
#include "pc2.c"

#include "pa3.c"
#include "pb3.c"
#include "pc3.c"
*/

#undef DATA
#define DATA    2
#if     DATA == 1
#define TR_HYD      "tr.hyd"
#define TR_STR      "tr.str"
#define TE_HYD      "te.hyd"
#define TE_STR      "te.str"
```

```
#elif   DATA == 2
#define TR_HYD       "trytr.hyd"   /* Kabsch and Sander's assignments.*/
#define TR_STR       "trytr.str"
#define TE_HYD       "tryte.hyd"
#define TE_STR       "tryte.str"
#endif


#include "ptr_read.c"              /* To read the training set.       */
#include "pte_read.c"              /* To read the testing  set.       */

#undef SYSTEM
#define SYSTEM  5
#if SYSTEM   == 1
#define SIMI         "S_M+N-W.c"   /* Similarity scale.               */
#elif SYSTEM == 2
#define SIMI         "S_M+N+invW.c"
#elif SYSTEM == 3
#define SIMI         "M+N-W.c"     /* Similarity matrix.              */
#elif SYSTEM == 4
#define SIMI         "M+N+invW.c"
#elif SYSTEM == 5
#define SIMI         "partition.c" /* To construct separation planes. */
#endif
#include  SIMI


#undef STATE
#define  STATE 3
#if   STATE == 2
#define FUN          "_2state_fun.c"
#elif STATE == 3
#define FUN          "_3state_fun.c"
#endif
#include FUN


#undef SCHEME
#define SCHEME  7
#if   SCHEME == 1
```

```
#define METHOD       "_2_dayhoff_levin.c"    /* Levin-like scheme.   */
#elif SCHEME == 2
#define METHOD       "_2ave_dayhoff_levin.c"
#elif SCHEME == 3
#define METHOD       "_3k_dayhoff_levin.c"   /* k-nearest neighbors. */
#elif SCHEME == 4
#define METHOD       "_2k_dayhoff_levin.c"
#elif SCHEME == 5
#define METHOD       "c_super.c"
#elif SCHEME == 6
#define METHOD       "m_super.c"
#elif SCHEME == 7
#define METHOD       "3c_super.c"
#elif SCHEME == 8
#define METHOD       "pr_super.c"
#elif SCHEME == 9
#define METHOD       "31pr_super.c"          /* Type I partition.   */
#elif SCHEME ==10
#define METHOD       "32pr_super.c"          /* Type II partition.  */
#endif
#include METHOD

/* external variables */

int   n_tr;            /* Total # of amino acids in the training set. */
int   n_te;            /* Total # of amino acids in the testing set.  */
int   num;             /* Total # of proteins in the training set.    */
int   num_te;          /* Total # of proteins in the testing set.     */
double  day[21][21];   /* Similarity matrix.                          */

double hsieh[20000],wei[5000];
int fan[20000],ching[5000],hua,hub;
                    /* hsieh contains residues in the training set.  */
                    /* wei contains residues in the testing set.     */
                    /* fan contains structures in the training set.  */
                    /* ching contains structures in the testing set. */
                    /* hua = length of hsieh.                        */
                    /* hub = length of wei.                          */
/* Single separation planes. */
double  normal[RESNP],shift;
```

```
double ab_plane[RESNP],ac_plane[RESNP],bc_plane[RESNP],ab,ac,bc;

/* Pairs of parallel lines.  */
double pr[PAIR][RESNP+2];
double pab[PAIRAB][RESNP+2],pac[PAIRAC][RESNP+2],pbc[PAIRBC][RESNP+2];
double pa[PAIRA][RESNP+2],pb[PAIRB][RESNP+2],pc[PAIRC][RESNP+2];

main()
{
tr_read();        /* Read the training set. */
te_read();        /* Read the testing set.  */

/*
mpsxout();        Single separation plane.
                  Use the function mpsxout.c*/

ab_mpsxout();   /*3 single separation planes for 3-state prediction.*/
ac_mpsxout();   /*Use the function ac_mpsxout.c.*/
bc_mpsxout();

/*
pab0();         Type I planes.
pac0();         Use the function pac0.c.
pbc0();

pab1();
pac1();
pbc1();

pab2();
pac2();
pbc2();

pab3();
pac3();
pbc3();
*/

/*
pa0();            Type II planes.
```

```
pb0();
pc0();

pa1();
pb1();
pc1();

pa2();
pb2();
pc2();

pa3();
pb3();
pc3();
*/

system();
method();
}
```

B.2 Functions

```
/*partition.c
1. Always let STATE==3.
2. Let STRUC2==2 (and STATE==3); then it's a 2-state separation:
   if STRUC1==1, then alpha vs. non-alpha; if STRUC1==0,then beta vs.
   non-beta; if STRUC1==(-1), then coil vs. non-coil.
3. This function can construct an artificial data base (let INFO = 1)
   using the information theory method with (BAY=1) or without (BAY!=1)
   Baysian. In the latter case (BAY!=1), we suppose that the database
   is large enough.
   Note: An artificial database will not be constructed when doing
   3-state prediction.
4.When PARL=1, this function will construct the input of either OSL or
  MPSX. To construct the input for 3-state seperation of both types,
  assign different values to STRUC1 & STRUC2 in the main function.
*/

#include "nrerror.c"
 /*Standard error handler.                                          */
#include "dvector.c"
 /*Allocates a double vector with range [nl..nh].                   */
#include "imatrix.c"
 /* Allocates an integer matrix with range [nrl..nrh][ncl..nch]. */
#include "d_dmatrix.c"
 /* Allocates a double matrix with range [nrl..nrh][ncl..nch].    */
#include "ivector.c"
 /* Allocates an integer vector with range [nl..nh].              */
 /*See ''Numerical recipes in C.'' Press et al.                    */

#define MOD NP-1

void system()
{
extern int fan[20000],hua,n_tr;
extern double hsieh[20000],normal[RESNP],shift;
extern double ab_plane[RESNP],ac_plane[RESNP],bc_plane[RESNP];
extern double pr[PAIR][RESNP+2];
```

```
extern double pab[PAIRAB][RESNP+2],pac[PAIRAC][RESNP+2],
              pbc[PAIRBC][RESNP+2];
extern double pa[PAIRA][RESNP+2],pb[PAIRB][RESNP+2],pc[PAIRC][RESNP+2];

int i,j=0,jj,kk=0,qq=0,k,l,s,p,q,w,bug=0; /*j = # of points */
double ss=0.0,zz,dot_pdt;
double  *v,*u,**f,**b,mm,x,y;
int **h,*pat1,*pat0;
int yy=0;/*yy = # of middle residue STRUC1 sequences in training set*/
int xx=0;/*xx = # of middle residue STRUC2 sequences in training set*/
int a_a=0,a_na=0,na_a=0,na_na=0,del=0,str;
double group;

v = dvector(1,RESNP);
u = dvector(1,RESNP);
h = imatrix(1,n_tr,1,NP);
f = d_dmatrix(1,RES*2,1,NP); /* Table of conditional probabilities. */
b = d_dmatrix(1,RES*2,1,NP); /* Table of I(x=1:0;y) (for Baysian).  */
pat1 = ivector(1,PAIR);      /* e.g., pat1[1] = # of alpha-helices
                                partitioned by 1st pair of planes.  */
pat0 = ivector(1,PAIR);      /* pat0 is for non-alpha helices.      */

for(i=1;i<=RESNP;++i) {v[i]=0.0;u[i]=0.0;}
for(i=1;i<=RES*2;++i)
  for(k=1;k<=NP;++k)
    f[i][k] = 0.0;

/****** Construct the matrices h & f ********************/
/* The j x NP matrix h records all sequences of length NP in the
   training set. The sign of the first column of h records
   structures ('+' for STRUC1 and '-' for STRUC2) of the middle
   residues. The RES*2 by NP matrix f is the contingency table in
   which odd and even rows are for STRUC1 and STRUC2 amino acids,
   respectively.
*/
for(i=0;i<=hua-NP;++i)
  {
  if(hsieh[i+NP-1] == 0.0) i = i+NP;
  if(fan[i+(NP-1)/2] == STRUC1) {yy += 1;mm=(1.0);}
  else if(fan[i+(NP-1)/2]==STRUC2 || STRUC2==2) {xx += 1;mm=(-1.0);}
```

```
   else goto delete;
   j = yy+xx;
     for(k=0;k<=NP-1;++k)
       {
       l = RES*k;
       s = hsieh[i+k];
       h[j][k+1] = s;
       if(mm==(-1.0)) {v[l+s] += mm;f[s*2][k+1] += 1.0;}
       else {u[l+s] += mm;f[s*2-1][k+1] += 1.0;}
       }
   if(mm == (-1.0)) h[j][1] = (-h[j][1]);
   delete: if(fan[i+(NP-1)/2]!=STRUC1&&fan[i+(NP-1)/2]!=STRUC2&&STRUC2!=2)
              ++del;
   }
if(STRUC1==1 && STRUC2==0)
    printf("\n#of coil in the training set = %d\n\n",del);
else if(STRUC1==1&&STRUC2==(-1))
    printf("\n# of beta sheet in the training set = %d\n\n",del);
else if(STRUC1==0&&STRUC2==(-1))
    printf("\n# of alpha helex in the training set = %d\n\n",del);

del = 0;

/* Construct artificial database.
   Let STRUC1==1 and STRUC2==2 to construct an artificial database
   for alpha and non-alpha separation. For the case STRUC2 != 2, we
   should modify the program to construct a 3-state artificial database.
*/

if(INFO == 1 && STRUC2==2)
{
/* Copy f[Xi][Yj] to b[Xi][Yj](for Baysian) */
for(i=1;i<=NP;++i)
  for(k=1;k<=RES*2;++k)
    b[k][i] = f[k][i];

/* Find mm=#(f1)-#(f2) where f1=yy and f2=xx */
mm = 0.0;
for(i=1;i<=yy-1;++i)
  mm += (1.0/i);
```

```
for(i=1;i<=xx-1;++i)
  mm -= (1.0/i);

/*Find the conditional probabilities p(Xi|Yj) and then replace f[Xi][Yj].
 Find #(f1y)-#(f2y)-#(f1)+#(f2) and then replace b[odd][]. */
for(i=1;i<=NP;++i)
  for(k=2;k<=RES*2;k=k+2)
    {
    if(f[k][i]==0.0) {f[k][i]+=1.0;b[k][i]+=2.0;} /* modification */
    if(f[k-1][i]==0.0) {f[k-1][i]+=1.0;b[k-1][i]+=2.0;}

    w=b[k-1][i]-1;
    b[k-1][i]=0.0;
    for(l=1;l<=w;++l)
      b[k-1][i] += (1.0/l);
    w=b[k][i]-1;
    for(l=1;l<=w;++l)
      b[k-1][i] -= (1.0/l);
    b[k-1][i] -= mm;

    zz = f[k][i]+f[k-1][i];
    f[k][i] /= zz;
    f[k-1][i] /= zz;
    }

/* Information theory method with (let BAY=1) or
   without (let BAY!=1) Baysian */
x = 1.0*xx/j;
y = 1.0*yy/j;  /* y=p(alpha) and x=p(non-alpha) where p=probability */
               /*  y & mm: for alpha; x & zz: for non-alpha.       */
  for(i=1;i<=j;++i)
    {
    mm = 0.0;
    zz = 0.0;
    if(h[i][1] > 0)
      {
      if(BAY==1)  mm += b[h[i][1]*2-1][1];
      else {mm+=(log(f[h[i][1]*2-1][1]/y));zz+=(log(f[h[i][1]*2][1]/x));}
      }
    else
```

```
      {
      if(BAY==1)  mm += b[-h[i][1]*2-1][1];
      else
        {mm+=(log(f[-h[i][1]*2-1][1]/y));zz+=(log(f[-h[i][1]*2][1]/x));}
      }
    for(k=2;k<=NP;++k)
      {
      if(BAY==1) mm+=b[h[i][k]*2-1][k];
      else
        {
        mm += (log(f[h[i][k]*2-1][k]/y));
        zz += (log(f[h[i][k]*2][k]/x));
        }
      }
    if(BAY==1) zz=0.0;
    if(mm>zz)
      {
      if(h[i][1] > 0) a_a += 1;
      else {na_a += 1; h[i][1]=(-h[i][1]);}
      }
    else if(mm<zz)
      {
      if(h[i][1]>0) {a_na += 1; h[i][1]=(-h[i][1]);}
      else {na_na += 1;}
      }
    else del+=1;
    }
printf("\n\n*************************************");
printf("\n* The result on the training set    *");
if(BAY==1)
printf("\n* (information with Baysian)        *");
else
printf("\n* (information without Baysian)     *");
printf("\n*************************************");
printf("\n          predicted  structures");
printf("\n          helix      non-helix    sum\n");
printf("\n");
printf("    helix   %d         %d         %d\n",a_a,a_na,a_a+a_na);
printf("\n");
printf("non-helix   %d         %d         %d\n",na_a,na_na,na_a+na_na);
```

```
printf("\n");
printf("# of deleted residues = %d\n",del);
printf("(# of correctly predicted residues)/(total # of residues-del)");
printf(" = %f\n",(a_a+na_na)*1.0/(a_a+na_a+a_na+na_na-del));

a_a=0;
a_na=0;
na_a=0;
na_na=0;
del=0;

for(i=0;i<=hub-NP;++i)
  {
  mm=0.0;
  zz=0.0;
  if(wei[i+NP-1] == 0.0) i = i+NP;
  str = ching[i+(NP-1)/2];
    for(k=0;k<=NP-1;++k)
      {
      s = wei[i+k];
      if(BAY==1) mm+=b[s*2-1][k+1];
      else
        {
        mm += (log(f[s*2-1][k+1]/y));
        zz += (log(f[s*2][k+1]/x));
        }
      }
    if(BAY==1) zz=0.0;
    if(mm>zz)
      {
      if(str==STRUC1) a_a += 1;
      else na_a += 1;
      }
    else if(mm<zz)
      {
      if(str==STRUC1) a_na += 1;
      else {na_na += 1;}
      }
    else del+=1;
  }
```

```
printf("\n\n*************************************");
printf("\n* The result on the testing set     *");
if(BAY==1)
printf("\n* (information with Baysian)        *");
else
printf("\n* (information without Baysian)     *");
printf("\n*************************************");
printf("\n          predicted  structures");
printf("\n          helix      non-helix    sum\n");
printf("\n");
printf("    helix   %d         %d         %d\n",a_a,a_na,a_a+a_na);
printf("\n");
printf("non-helix   %d         %d         %d\n",na_a,na_na,na_a+na_na);
printf("\n");
printf("# of deleted residues = %d\n",del);
printf("(# of correctly predicted residues)/(total # of residues-del)");
printf(" = %f\n",(a_a+na_na)*1.0/(a_a+na_a+a_na+na_na-del));

a_a=0;
a_na=0;
na_a=0;
na_na=0;
del=0;

/* Use the single separation plane (two-state) to test performance
   on the artificial database. normal[] is the normal vector. */
if(TEST==1)
  {
  for(i=1;i<=j;++i)
    {
    zz=0.0;
    if(h[i][1]>0) zz+=normal[h[i][1]-1];
    else zz+=normal[-h[i][1]-1];
    for(k=2;k<=NP;++k)
      zz += normal[h[i][k]+(k-1)*RES-1];
    zz+=(shift);
    if(h[i][1]>0)
      {
      if(zz>0) a_a+=1;
```

```
      else if(zz<0) a_na+=1;
      else del += 1;
      }
    else
      {
      if(zz>0) na_a += 1;
      else if(zz<0) na_na += 1;
      else del += 1;
      }
    }

printf("\n\n***********************************************");
printf("\n* The result on the artificial training set    *");
printf("\n***********************************************");
printf("\n          predicted  structures");
printf("\n          helix      non-helix    sum\n");
printf("\n");
printf("    helix   %d         %d         %d\n",a_a,a_na,a_a+a_na);
printf("\n");
printf("non-helix   %d         %d         %d\n",na_a,na_na,na_a+na_na);
printf("\n");
printf("# of deleted residue = %d\n",del);
printf("(# of correctly predicted residue)/(total # of residue-del)");
printf(" = %f\n",(a_a+na_na)*1.0/(a_a+na_a+a_na+na_na-del));
  }
  }
/* End of if(INFO==1 && STRUC2==2) */

if(PARL==1 && FIRST!=1)
if(STRUC2 != 2)
{
if(STRUC1==1&&STRUC2==0)
  for(i=0;i<=PAIR-1;++i)
    for(w=1;w<=RESNP+2;++w)
      pr[i][w-1] = pab[i][w-1];
else if(STRUC1==1&&STRUC2==(-1))
  for(i=0;i<=PAIR-1;++i)
    for(w=1;w<=RESNP+2;++w)
      pr[i][w-1] = pac[i][w-1];
else if(STRUC1==0&&STRUC2==(-1))
```

```
  for(i=0;i<=PAIR-1;++i)
    for(w=1;w<=RESNP+2;++w)
      pr[i][w-1] = pbc[i][w-1];
}
else
{
if(STRUC1==1)
  for(i=0;i<=PAIR-1;++i)
    for(w=1;w<=RESNP+2;++w)
      pr[i][w-1] = pa[i][w-1];
else if(STRUC1==0)
  for(i=0;i<=PAIR-1;++i)
    for(w=1;w<=RESNP+2;++w)
      pr[i][w-1] = pb[i][w-1];
else if(STRUC1==(-1))
  for(i=0;i<=PAIR-1;++i)
    for(w=1;w<=RESNP+2;++w)
      pr[i][w-1] = pc[i][w-1];
}

/* Delete the points seperated by pairs of parallel planes. */
if(PARL==1)
{
for(i=1;i<=PAIR;++i)  {pat0[i]=0; pat1[i]=0;}
if(PAIR >= 2)
{
for(i=1;i<=j;++i)
  for(w=0;w<=PAIR-2;++w)
    {
    mm = 0.0;
    if(h[i][1] > 0) mm += pr[w][h[i][1]-1];
    else  mm += pr[w][-h[i][1]-1];
    for(k=2;k<=NP;++k)
      mm += pr[w][h[i][k]+(k-1)*RES-1];
    if(mm+pr[w][RESNP] < CB)
      {
      pat0[w+1] += 1;

      /* To construct the new centroids,
         delete the partitioned points. */
```

```
        if(h[i][1] > 0)
          {
          for(k=1;k<=NP;++k) u[h[i][k]+(k-1)*RES] -= 1.0;
          yy -= 1;
          }
        else if(h[i][1] < 0)
          {
          v[-h[i][1]] += 1.0;
          for(k=2;k<=NP;++k) v[h[i][k]+(k-1)*RES] += 1.0;
          xx -= 1;
          }
        h[i][1] = 0;
        /* To indicate the deleted row (or NP-amino acid segment). */
        break;
        }
      else if(mm+pr[w][RESNP+1] > CA)
        {
        pat1[w+1] += 1;

        if(h[i][1] > 0)
          {
          for(k=1;k<=NP;++k) u[h[i][k]+(k-1)*RES] -= 1.0;
          yy -= 1;
          }
        else if(h[i][1] < 0)
          {
          v[-h[i][1]] += 1.0;
          for(k=2;k<=NP;++k) v[h[i][k]+(k-1)*RES] += 1.0;
          xx -= 1;
          }
        h[i][1] = 0;
        break;
        }
      }
s = 0;
for(i=1;i<=PAIR-1;++i)
  {
  printf("\n# of '%d'('%d') residues partitioned by the %dth pair
          = %d(%d)\n",STRUC1,STRUC2,i,pat1[i],pat0[i]);
  s += (pat0[i]+pat1[i]);
```

```
  }
  printf("Total # of residues partitioned by the first %d pairs
          = %d\n",PAIR-1 ,s);
}   /* End of "PAIR>=2" */
if(FIRST!=1)
{
for(i=1;i<=j;++i)
    if(h[i][1] != 0)
      {
      mm = 0.0;
      if(h[i][1] > 0) mm += pr[PAIR-1][h[i][1]-1];
      else  mm += pr[PAIR-1][-h[i][1]-1];
      for(k=2;k<=NP;++k)
        mm += pr[PAIR-1][h[i][k]+(k-1)*RES-1];
      if(mm+pr[PAIR-1][RESNP] < CB)
        {
        pat0[PAIR] += 1;
        if(ITE!=1) /*To construct the centroids for next pair. */
          {
          if(h[i][1] > 0)
            {
            for(k=1;k<=NP;++k) u[h[i][k]+(k-1)*RES] -= 1.0;
            yy -= 1;
            }
          else
            {
            v[-h[i][1]] += 1.0;
            for(k=2;k<=NP;++k) v[h[i][k]+(k-1)*RES] += 1.0;
            xx -= 1;
            }
          h[i][1] = 0;
          }
        }
      else if(mm+pr[PAIR-1][RESNP+1] > CA)
        {
        pat1[PAIR] += 1;
        if(ITE!=1)
          {
          if(h[i][1] > 0)
            {
```

```
            for(k=1;k<=NP;++k) u[h[i][k]+(k-1)*RES] -= 1.0;
            yy -= 1;
            }
          else
            {
            v[-h[i][1]] += 1.0;
            for(k=2;k<=NP;++k) v[h[i][k]+(k-1)*RES] += 1.0;
            xx -= 1;
            }
          h[i][1] = 0;
          }
        }
      }

printf("# of '%d' residues partitioned by the new pair =
        %d\n",STRUC1,pat1[PAIR]);
printf("# of '%d' residues partitioned by the new pair =
        %d\n",STRUC2,pat0[PAIR]);
printf("Total # of residues partitioned by the new pair =
        %d\n",pat1[PAIR]+pat0[PAIR]);
s = 0;
for(k=1;k<=PAIR;++k) s+=(pat0[k]+pat1[k]);
printf("\nTotal # of residues that have been partitioned = %d\n",s);
printf("# of residues left = %d\n\n",j-s);
}    /* End of "FIRST!=1"*/
}    /* End of "PARL==1" */

/* Construct the initial guess */
for(i=1;i<=RESNP;++i)
  v[i]=(v[i]/xx)+(u[i]/yy);       /* Difference of centroids.*/
for(i=1;i<=RESNP;++i) ss += (v[i]*v[i]); /* Normalization. */
zz = sqrt(ss);
for(i=1;i<=RESNP;++i) v[i] /= (zz);

/*Find the inner product of the initial vector and the normalized
  output vector when iteration is executed.
  Use the output vector of either MPSX or OSL to replace v[i].*/
if(ITE == 1)
{
if(PARL==1)
```

```
  for(i=0;i<=RESNP-1;++i) normal[i] = pr[PAIR-1][i];
else if(STRUC2 != 2)
  {
  if(STRUC1==1 && STRUC2==0)
    for(i=0;i<=RESNP-1;++i) normal[i]=ab_plane[i];
  else if(STRUC1==1 && STRUC2==(-1))
    for(i=0;i<=RESNP-1;++i) normal[i]=ac_plane[i];
  else if(STRUC1==0 && STRUC2==(-1))
    for(i=0;i<=RESNP-1;++i) normal[i]=bc_plane[i];
  }

ss=0.0;
for(i=1;i<=RESNP;++i)
  ss += (normal[i-1]*normal[i-1]);
zz = sqrt(ss);
printf("The 2-norm of the normal vector is %f\n",zz);
if(PARL==1)
{
printf("The distance between the two planes = ");
printf("ZSUM/(2-norm of the normal) = %f\n",
       (pr[PAIR-1][RESNP]-pr[PAIR-1][RESNP+1])/zz);
}
for(i=1;i<=RESNP;++i)
  u[i] = normal[i-1]/zz;

dot_pdt = 0.0;
for(i=1;i<=RESNP;++i)
  dot_pdt += (u[i]*v[i]);
printf("\ndot_pdt = %f\n",dot_pdt);
if(dot_pdt < 0.0 && PARL!=1) goto terrible;

for(i=1;i<=RESNP;++i) v[i] += (RATE*(u[i]-v[i]));
/*for(i=1;i<=RESNP;++i) printf("  %f\n",v[i]);*/
}

/*goto terrible;*/ /*Use this statement when input is not constructed.*/

/*print JCL for MPSX */

printf("\n//PRIMAL JOB    ,MSGLEVEL=1\n");
```

```
printf("//* CHECKPOINT=[NO]\n");
printf("/*JOBPARM  BIN=246,CLASS=G\n");
printf("/*JOBPARM  L=500\n");
printf("//S1 EXEC DPLMPROC,PBDISP=NEW,\n");
printf("//  TIME.MPSGO=(2,10),REGION.MPSGOX=15M,\n");
printf("//  TIME.MPSGOX=(68,55)\n");
printf("//MPSGO.SYSIN DD *\n");
printf("         PROGRAM\n");
printf("         INITIALZ\n");
printf("         MOVE(XDATA,'INPUT%d')\n",NUM);
printf("         MOVE(XPBNAME,'PRIMAL')\n");
printf("         MOVE(XOBJ,'ZSUM')\n");
printf("         MOVE(XRHS,'RHS')\n");
printf("         XEPS=0.001\n");
printf("         CONVERT\n");
printf("         SETUP('MIN','BOUNDS','B1','SCALE')\n");
printf("         OPTIMIZE\n");
printf("         SAVE\n");
printf("         XEPS=0.0\n");
printf("         RESTORE\n");
printf("         OPTIMIZE\n");
printf("         SOLUTION\n");
printf("         EXIT\n");
printf("         PEND\n");
printf("/*\n");
printf("//MPSGOX.MATRIX1 DD UNIT=SCRTCH,SPACE=(CYL,(10),,CONTIG)\n");
printf("//MPSGOX.SCRATCH1 DD UNIT=SCRTCH,SPACE=(CYL,(10),,CONTIG)\n");
printf("//MPSGOX.SCRATCH2 DD UNIT=SCRTCH,SPACE=(CYL,(10),,CONTIG)\n");
printf("//MPSGOX.ETA1 DD UNIT=SCRTCH,SPACE=(CYL,(40),,CONTIG)\n");
printf("//MPSGOX.ETA2 DD UNIT=SCRTCH,SPACE=(CYL,(40),,CONTIG),SEP=ETA1\n");
printf("//MPSGOX.SYSIN DD *\n");


/* Print JCL for OSL */
/*
printf("\n//PRIMAL  JOB\n");
printf("//*  CHECKPOINT=[NO]\n");
printf("/*JOBPARM   BIN=246,CLASS=G\n");
printf("/*JOBPARM   L=500\n");
printf("//S1  EXEC  PGM=SAM12,REGION=50M,TIME=(60,20)\n");
```

```
printf("//STEPLIB  DD  DSN=V.U9229.LP.LOAD12,DISP=SHR\n");
printf("//FT06F001  DD  SYSOUT=A\n");
printf("//FT98F001 DD *\n");
printf("    0    2    1\n");
*/

/*Construct the initial tableau which is the input of
  OSL or MPSX software*/
printf("NAME          INPUT%d\n",NUM);
printf("ROWS\n");
printf(" N  ZSUM\n");
jj = 0;       /* To count the # of points not partitioned
                 by previously constructed pairs of planes. */
for(i=1;i<=j;++i)
  if(h[i][1]!=0) {jj += 1; printf(" G  R%d\n",jj);}
  for(i=jj+1;i<=NP+1+jj;++i)
    printf(" E  R%d\n",i);

printf("COLUMNS\n");
for(i=1;i<=RESNP+1+j;++i)
  {
  if(i>RESNP+1)
   {
   if(PARL!=1) /* For single separation planes. */
   {
   if(i<10)
     {
     printf("    C%d        ZSUM        1.0\n",i);
     printf("    C%d        R%d         1.0\n",i,i-RESNP-1);
     }
   else if(i<100)
     {
     printf("    C%d        ZSUM       1.0\n",i);
     printf("    C%d       R%d         1.0\n",i,i-RESNP-1);
     }
   else if(i<1000)
     {
     printf("    C%d       ZSUM        1.0\n",i);
     printf("    C%d      R%d         1.0\n",i,i-RESNP-1);
     }
```

```
 else if(i<10000)
   {
   printf("    C%d       ZSUM        1.0\n",i);
   printf("    C%d     R%d         1.0\n",i,i-RESNP-1);
   }
 else if(i<100000)
   {
   printf("    C%d       ZSUM        1.0\n",i);
   printf("    C%d    R%d         1.0\n",i,i-RESNP-1);
   }
 else if(i<1000000)
   {
   printf("    C%d       ZSUM        1.0\n",i);
   printf("    C%d   R%d         1.0\n",i,i-RESNP-1);
   }
 else printf("Modify the format of input data.");
 } /* End of if(PARL != 1) */
 } /* End of if(i>RESNP+1) */
else
  {
  kk=0;
  for(k=1;k<=NP;++k)
    if(i<=RES*k) {p=k;break;}
  for(k=1;k<=j;++k)
    if(h[k][1]==0) kk+=1;
    else if((h[k][p]==i-RES*(p-1))||(h[k][p]==RES*(p-1)-i))
      {
      if(i<10)
        {
        if(h[k][1]>0)  printf("    C%d        R%d         1.0\n",i,k-kk);
        else           printf("    C%d        R%d         -1.0\n",i,k-kk);
        }
      else if(i<100)
        {
        if(h[k][1]>0)  printf("    C%d       R%d         1.0\n",i,k-kk);
        else           printf("    C%d       R%d         -1.0\n",i,k-kk);
        }
      else if(i<1000)
        {
        if(h[k][1]>0)  printf("    C%d      R%d         1.0\n",i,k-kk);
```

```
        else            printf("    C%d      R%d          -1.0\n",i,k-kk);
        }
      else if(i<10000)
        {
        if(h[k][1]>0)  printf("    C%d      R%d          1.0\n",i,k-kk);
        else           printf("    C%d      R%d          -1.0\n",i,k-kk);
        }
      else if(i<100000)
        {
        if(h[k][1]>0)  printf("    C%d     R%d          1.0\n",i,k-kk);
        else           printf("    C%d     R%d          -1.0\n",i,k-kk);
        }
      else if(i<1000000)
        {
        if(h[k][1]>0)  printf("    C%d    R%d          1.0\n",i,k-kk);
        else           printf("    C%d    R%d          -1.0\n",i,k-kk);
        }
      else printf("*******Modify the format of input data 3*******");
      }

/* Put the initial vector; add (NP-1) constraints; add the constraint
   (sum of the first 20 entries of the normal vector)=0. */
    if(i<=RESNP)
      {
      if(i<10)
       {
       printf("    C%d        R%d       %f\n",i,jj+1,v[i]);
       if(p==1)
         {
         for(w=1;w<=MOD;++w) printf("    C%d        R%d        1.0\n",
                                    i,jj+1+w);
         printf("    C%d        R%d        1.0\n",i,jj+NP+1);
         }
       }
      else if(i<100)
        {
        printf("    C%d        R%d       %f\n",i,jj+1,v[i]);
        if(p==1)
          {
          for(w=1;w<=MOD;++w) printf("    C%d        R%d          1.0\n",
```

```
                                        i,jj+1+w);
            printf("    C%d        R%d          1.0\n",i,jj+NP+1);
            }
          for(w=2;w<=NP;++w)
            if(p==w) {printf("    C%d        R%d          -1.0\n",i,jj+p);
                      break;}
          }
        else if(i<1000)
          {
          printf("    C%d       R%d       %f\n",i,jj+1,v[i]);
          for(w=2;w<=NP;++w)
            if(p==w) {printf("    C%d       R%d          -1.0\n",i,jj+p);
                      break;}
          }
        else if(i<10000)
        printf("    C%d      R%d      %f\n",i,jj+1,v[i]);
        else if(i<100000)
        printf("    C%d    R%d    %f\n",i,jj+1,v[i]);
        else printf("*****Modify the format of input data 4*****");
        }

      /* Set constant term for single separation plane. */
      if(i==RESNP+1 && PARL!=1)
        {
        for(q=1;q<=j;++q)
          {
          if(h[q][1]>0)  printf("    C%d       R%d          1.0\n",i,q);
          else           printf("    C%d       R%d          -1.0\n",i,q);
          }
        }
      } /* End of "else" of the statement if(i>RESNP+1). */
if(PARL==1)
  {
  if(i==RESNP+1)
    {
    qq=0;
    printf("    C%d       ZSUM         1.0\n",i);
    for(q=1;q<=j;++q)
      {
      if(h[q][1]!=0) qq+=1;
```

```
      if(h[q][1]>0)  printf("    C%d       R%d          1.0\n",i,qq);
      }
    }
  else if(i==RESNP+2)
    {
    qq=0;
    printf("    C%d       ZSUM         -1.0\n",i);
    for(q=1;q<=j;++q)
      {
      if(h[q][1]!=0) qq+=1;
      if(h[q][1]<0)  printf("    C%d       R%d          -1.0\n",i,qq);
      }
    }
  }
 } /* End of the COLUMN part */
printf("RHS\n");
  printf("    RHS       R%d         1.0\n",jj+1);
printf("BOUNDS\n");
for(i=1;i<=RESNP+1;++i)
  printf(" FR B1        C%d\n",i);
if(PARL==1) printf(" FR B1        C%d\n",RESNP+2);

printf("ENDATA\n");
printf("/*\n");
terrible:
if(dot_pdt < 0.0)
printf("\n The inner product of the initial vector and
          the output vector is less than 0.\n");
}
```

```
/* happy */
/* Use the first 20 integers to represent the 20 amino acids. */

#define  AA  1
#define  RR  2
#define  NN  3
#define  DD  4
#define  CC  5
#define  QQ  6
#define  EE  7
#define  GG  8
#define  HH  9
#define  II  10
#define  LL  11
#define  KK  12
#define  MM  13
#define  FF  14
#define  PP  15
#define  SS  16
#define  TT  17
#define  WW  18
#define  YY  19
#define  VV  20
```

```
/* _3state_fun.c */
/*
 Use 1 to represent alpha-helix, 0 to represent beta-sheet,
 and -1 to represent coil.
 ' ' = coil; <----> = helix; else = sheet
*/
int func(c)
char c;
{
int i;
switch(c)
  {
  case '<' : i=1;break;
  case '-' : i=1;break;
  case '=' : i=1;break;
  case '>' : i=1;break;
  case ' ' : i=(-1);break;
  default  : i=0;
  }
return(i);
}
```

```
/*ptr_read.c*/
void tr_read()
{
extern int n_tr;
extern double hsieh[20000];
extern int fan[20000],hua;
extern int  num;        /* num = # of proteins. */

double tr_hyd;
int    tr_str;
int  i,p,b,q;
int  k = 0;
int  l = -1;
int  anum = 0;        /* anum = # of amino acids + # of proteins */

FILE  *fptr;
FILE  *sptr;
char str,ftr;

fptr = fopen(TR_HYD, "r");
sptr = fopen(TR_STR, "r");

for(i=0;i<=20000;++i)
    {
    str = fgetc(sptr);
    ftr = fgetc(fptr);
    anum += 1;
    switch(ftr)
    {
    case 'A' : tr_hyd = AA;tr_str = func(str); break;
    case 'R' : tr_hyd = RR;tr_str = func(str); break;
    case 'N' : tr_hyd = NN;tr_str = func(str); break;
    case 'D' : tr_hyd = DD;tr_str = func(str); break;
    case 'C' : tr_hyd = CC;tr_str = func(str); break;
    case 'Q' : tr_hyd = QQ;tr_str = func(str); break;
    case 'E' : tr_hyd = EE;tr_str = func(str); break;
    case 'G' : tr_hyd = GG;tr_str = func(str); break;
    case 'H' : tr_hyd = HH;tr_str = func(str); break;
    case 'I' : tr_hyd = II;tr_str = func(str); break;
    case 'L' : tr_hyd = LL;tr_str = func(str); break;
```

```
    case 'K' : tr_hyd = KK;tr_str = func(str); break;
    case 'M' : tr_hyd = MM;tr_str = func(str); break;
    case 'F' : tr_hyd = FF;tr_str = func(str); break;
    case 'P' : tr_hyd = PP;tr_str = func(str); break;
    case 'S' : tr_hyd = SS;tr_str = func(str); break;
    case 'T' : tr_hyd = TT;tr_str = func(str); break;
    case 'W' : tr_hyd = WW;tr_str = func(str); break;
    case 'Y' : tr_hyd = YY;tr_str = func(str); break;
    case 'V' : tr_hyd = VV;tr_str = func(str); break;
    case '*' : k+=1; tr_hyd=0.0; break;
    case 'X' : tr_hyd=0.0; break;
    case '^' : tr_hyd=100.0; k+=1; break;
    default  : anum-=1;l-=1;break;
    }

    l += 1;
    hsieh[l] = tr_hyd;      /* Amino acid sequences.*/
    fan[l] = tr_str;        /* Structure sequences. */

    if(ftr == '*' || ftr ==  '^')
      {
      num += 1;
      if(ftr == '^') goto end;
      if(num == NUM) goto end;
      }
    }

end: printf("\nThe total number of protein in tr.hyd is %d\n",num);
hua = l;
n_tr = anum - k;
printf("\nThe total number of amino acid in tr.hyd is %d\n", n_tr);
fclose(fptr);
fclose(sptr);
}
```

```
/*31pr_super.c
  This subroutine uses 3 groups of several pairs of
  parallel planes determined by OSL to perform 3-state
  predictions.
  For 3-state prediction of type 1 only.
  Type 1: alpha vs. beta, alpha vs. coil, and beta vs. coil.
*/

void method()
{
extern int fan[20000],ching[5000],hua,hub;
extern double hsieh[20000],wei[5000];
extern double pab[PAIRAB][RESNP+2],pac[PAIRAC][RESNP+2];
extern double pbc[PAIRBC][RESNP+2];

int i,k,l,s,w,str,del=0,s1,s2,s3;
int a_a,a_b,a_c,b_a,b_b,b_c,c_a,c_b,c_c;
double mm;

a_a = 0;
a_b = 0;
a_c = 0;
b_a = 0;
b_b = 0;
b_c = 0;
c_a = 0;
c_b = 0;
c_c = 0;

   /* ************training set***************** */
for(i=0;i<=hua-NP;++i)
  {
  if(hsieh[i+NP-1] == 0.0) i = i+NP;
  str = fan[i+(NP-1)/2];

  /* alpha-beta seperation */
  for(w=0;w<=PAIRAB-1;++w)
    {
    mm = 0.0;
    for(k=0;k<=NP-1;++k)
```

```
    {
    l=RES*k;
    s = hsieh[i+k];
    mm += pab[w][l+s-1];
    }
  if(mm+pab[w][RESNP] < CB)          {s1=0;break;}
  else if(mm+pab[w][RESNP+1] > CA)   {s1=1;break;}
  else if(w==PAIRAB-1)                s1=2;
  }

/* alpha-coil seperation */
for(w=0;w<=PAIRAC-1;++w)
  {
  mm = 0.0;
  for(k=0;k<=NP-1;++k)
    {
    l=RES*k;
    s = hsieh[i+k];
    mm += pac[w][l+s-1];
    }
  if(mm+pac[w][RESNP] < CB)          {s2=(-1);break;}
  else if(mm+pac[w][RESNP+1] > CA)   {s2=1;break;}
  else if(w==PAIRAC-1)                s2=2;
  }

/* beta-coil seperation */
for(w=0;w<=PAIRBC-1;++w)
  {
  mm = 0.0;
  for(k=0;k<=NP-1;++k)
    {
    l=RES*k;
    s = hsieh[i+k];
    mm += pbc[w][l+s-1];
    }
  if(mm+pbc[w][RESNP] < CB)          {s3=(-1);break;}
  else if(mm+pbc[w][RESNP+1] > CA)   {s3=0;break;}
  else if(w==PAIRBC-1)                s3=2;
  }
```

```
/* Prediction of the ith point. */
if(s1==1)
  {
  if(s2==1)
    {
    if(str==1)          a_a += 1;
    else if(str==0)     b_a += 1;
    else                c_a += 1;
    }
  else if(s2==(-1))
    {
    if(s3==(-1))
      {
      if(str==(-1))       c_c += 1;
      else if(str==1)     a_c += 1;
      else                b_c += 1;
      }
    else del += 1;
    }
  else del += 1;
  }
else if(s1==0)
  {
  if(s3==0)
    {
    if(str==0)          b_b += 1;
    else if(str==1)     a_b += 1;
    else                c_b += 1;
    }
  else if(s3==(-1))
    {
    if(s2==(-1))
      {
      if(str==(-1))     c_c += 1;
      else if(str==0)   b_c += 1;
      else              a_c += 1;
      }
    else del += 1;
    }
  else del += 1;
```

```
    }
  else
    {
    if(s2==(-1))
      {
      if(s3==(-1))
        {
        if(str==(-1))      c_c += 1;
        else if(str==0)    b_c += 1;
        else               a_c += 1;
        }
      else del += 1;
      }
    else del += 1;
    }
  }

printf("\n# of pairs of parallel planes for A-B = %d\n",PAIRAB);
printf("\n# of pairs of parallel planes for A-C = %d\n",PAIRAC);
printf("\n# of pairs of parallel planes for B-C = %d\n",PAIRBC);

printf("\nCA = %.10f, CB = %.10f\n",CA,CB);

printf("\n*************************************");
printf("\n* The result on the training set    *");
printf("\n*************************************");
printf("\n          predicted  structures");
printf("\n          helix      beta      coil    sum\n");
printf("\n");
printf("    helix   %d         %d        %d     %d\n",
       a_a,a_b,a_c,a_a+a_b+a_c);
printf("\n");
printf("    beta    %d         %d        %d     %d\n",
       b_a,b_b,b_c,b_a+b_b+b_c);
printf("\n");
printf("    coil    %d         %d        %d     %d\n",
       c_a,c_b,c_c,c_a+c_b+c_c);
printf("\n");
printf("Total # of points not predicted = %d\n",del);
printf("Total# of points predicted = %d\n",
```

```
       a_a+a_b+a_c+b_a+b_b+b_c+c_a+c_b+c_c);
printf("Total # of points in the training set = %d\n",
       del+a_a+a_b+a_c+b_a+b_b+b_c+c_a+c_b+c_c);
printf("(Total # of correctly predicted residues)/
       (total # of predicted residues)=");
printf("%f\n",(a_a+b_b+c_c)*1.0/(a_a+a_b+a_c+b_a+b_b+b_c+c_a+c_b+c_c));

   /* ************testing set***************** */
a_a = 0;
a_b = 0;
a_c = 0;
b_a = 0;
b_b = 0;
b_c = 0;
c_a = 0;
c_b = 0;
c_c = 0;
del = 0;

for(i=0;i<=hub-NP;++i)
  {
  if(wei[i+NP-1] == 0.0) i = i+NP;
  str = ching[i+(NP-1)/2];

  /* alpha-beta seperation */
  for(w=0;w<=PAIRAB-1;++w)
    {
    mm = 0.0;
    for(k=0;k<=NP-1;++k)
      {
      l=RES*k;
      s = wei[i+k];
      mm += pab[w][l+s-1];
      }
    if(mm+pab[w][RESNP] < CB)          {s1=0;break;}
    else if(mm+pab[w][RESNP+1] > CA)   {s1=1;break;}
    else if(w==PAIRAB-1)                s1=2;
    }

  /* alpha-coil seperation */
```

```
for(w=0;w<=PAIRAC-1;++w)
  {
  mm = 0.0;
  for(k=0;k<=NP-1;++k)
    {
    l=RES*k;
    s = wei[i+k];
    mm += pac[w][l+s-1];
    }
  if(mm+pac[w][RESNP] < CB)          {s2=(-1);break;}
  else if(mm+pac[w][RESNP+1] > CA)   {s2=1;break;}
  else if(w==PAIRAC-1)                s2=2;
  }

/* beta-coil seperation */
for(w=0;w<=PAIRBC-1;++w)
  {
  mm = 0.0;
  for(k=0;k<=NP-1;++k)
    {
    l=RES*k;
    s = wei[i+k];
    mm += pbc[w][l+s-1];
    }
  if(mm+pbc[w][RESNP] < CB)          {s3=(-1);break;}
  else if(mm+pbc[w][RESNP+1] > CA)   {s3=0;break;}
  else if(w==PAIRBC-1)                s3=2;
  }

/* Prediction of the ith point. */
if(s1==1)
  {
  if(s2==1)
    {
    if(str==1)          a_a += 1;
    else if(str==0)     b_a += 1;
    else                c_a += 1;
    }
  else if(s2==(-1))
    {
```

```
    if(s3==(-1))
      {
      if(str==(-1))      c_c += 1;
      else if(str==1)    a_c += 1;
      else               b_c += 1;
      }
    else del += 1;
    }
  else del += 1;
  }
else if(s1==0)
  {
  if(s3==0)
    {
    if(str==0)          b_b += 1;
    else if(str==1)     a_b += 1;
    else                c_b += 1;
    }
  else if(s3==(-1))
    {
    if(s2==(-1))
      {
      if(str==(-1))     c_c += 1;
      else if(str==0)   b_c += 1;
      else              a_c += 1;
      }
    else del += 1;
    }
  else del += 1;
  }
else
  {
  if(s2==(-1))
    {
    if(s3==(-1))
      {
      if(str==(-1))      c_c += 1;
      else if(str==0)    b_c += 1;
      else               a_c += 1;
      }
```

```
      else del += 1;
      }
    else del += 1;
    }
  }

printf("\n*************************************");
printf("\n* The result on the testing set     *");
printf("\n*************************************");
printf("\n          predicted  structures");
printf("\n          helix      beta      coil    sum\n");
printf("\n");
printf("    helix   %d         %d        %d       %d\n",
        a_a,a_b,a_c,a_a+a_b+a_c);
printf("\n");
printf("    beta    %d         %d        %d      %d\n",
        b_a,b_b,b_c,b_a+b_b+b_c);
printf("\n");
printf("    coil    %d         %d        %d      %d\n",
        c_a,c_b,c_c,c_a+c_b+c_c);
printf("\n");
printf("Total # of points not predicted = %d\n",del);
printf("Total # of points predicted = %d\n",
       a_a+a_b+a_c+b_a+b_b+b_c+c_a+c_b+c_c);
printf("Total # of points in the testing set = %d\n",
       del+a_a+a_b+a_c+b_a+b_b+b_c+c_a+c_b+c_c);
printf("(Total # of correctly predicted residues)/
       (total # of predicted residues)=");
printf("%f\n",(a_a+b_b+c_c)*1.0/(a_a+a_b+a_c+b_a+b_b+b_c+c_a+c_b+c_c));
}
```

```
/* Examples of training set and testing set */
/* Residues (A subset of tryte.hyd.) */

ENLKLGFLVKQPEEPWFQTEWKFADKAGKDLGFEVIKIAV
PDGEKTLNAIDSLAASGAKGFVICTPDPKLGSAIVAKARG
YDMKVIAVDDQFVNAKGKPMDTVPLVMMAATKIGERQGQE
LYKEMQKRGWDVKESAVMAITANELDTARRRTTGSMDALK
AAGFPEKQIYQVPTKSNDIPGAFDAANSMLVQHPEVKHWL
IVGMNDSTVLGGVRATEGQGFKAADIIGIGINGVDAVSEL
SKAQATGFYGSLLPSPDVHGYKSSEMLYNWVAKDVEPPKF
TEVTDVVLITRDNFKEELEKKGLGGK*
APAFSVSPASGASDGQSVSVSVAAAGETYYIAQCAPVGGQ
DACNPATATSFTTDASGAASFSFTVRKSYAGQTPSGTPVG
SVDCATDACNLGAGNSGLNLGHVALTFG*
SIPPEVKFNKPFVFLMIEQNTKSPLFMGKVVNPTQ^

/* Structures (A subset of tryte.str.) */

      AAA        <------------>       AAA
  <------------>        B          <-------
>     B                      C  <---------
------->         DDD       <=><-----------
->        DD         <------------>        D
DD        <------->              EE      <-->
           EE        <------------>
  C    E          <=>        *
  AAAAA           AAAAAAA     BBBBBBB CC  C
C         BBB         AAAAA   DDDDD      DDD
DDD       BBBBB                *
    DDD     AAAAAAA     AAAAAAAA      ^
```