PROTEIN SECONDARY STRUCTURE PREDICTION USING NEURAL NETWORKS AND SUPPORT VECTOR MACHINES

by

Lipontseng Cecilia Tsilo

A thesis submitted to Rhodes University in partial fulfillment of the requirements for the degree of

Master of Science

in

Mathematical Statistics

Department of Statistics

February 2008

Abstract

Predicting the secondary structure of proteins is important in biochemistry because the 3D structure can be determined from the local folds that are found in secondary structures. Moreover, knowing the tertiary structure of proteins can assist in determining their functions. The objective of this thesis is to compare the performance of Neural Networks (NN) and Support Vector Machines (SVM) in predicting the secondary structure of 62 globular proteins from their primary sequence. For each NN and SVM, we created six binary classifiers to distinguish between the classes' helices (H) strand (E), and coil (C). For NN we use Resilient Backpropagation training with and without early stopping. We use NN with either no hidden layer or with one hidden layer with 1,2,...,40 hidden neurons. For SVM we use a Gaussian kernel with parameter fixed at $\gamma = 0.1$ and varying cost parameters C in the range [0.1,5]. 10-fold cross-validation is used to obtain overall estimates for the probability of making a correct prediction. Our experiments indicate for NN and SVM that the different binary classifiers have varying accuracies: from 69% correct predictions for coils vs. non-coil up to 80% correct predictions for stand vs. non-strand. It is further demonstrated that NN with no hidden layer or not more than 2 hidden neurons in the hidden layer are sufficient for better predictions. For SVM we show that the estimated accuracies do not depend on the value of the cost parameter. As a major result, we will demonstrate that the accuracy estimates of NN and SVM binary classifiers cannot distinguish. This contradicts a modern belief in bioinformatics that SVM outperforms other predictors.

keywords: Neural Networks, Support Vector Machines, Protein Secondary Structure Prediction

Acknowledgements

Many people have contributed to making my research work both successful and fruitful. First I must thank Professor Gunther Jäger for support throughout my research. It would not have been possible without his patience, suggestions, and encouragement. His guidance has made a difference to both my work and my life.

I could not have completed this work without the motivation, love, inspiration, and massive support of my husband Kopano, and daughter Nthabiseng Matsaseng. I am grateful to have you as part of my life.

Last but not least, I give my appreciation to my parents, the rest of my family and friends. Also thanks to Louise Asmal of the Canon Collins Education Trust, Dr. Sirion Robertson from Faculty of Pharmacy (Rhodes University) and Mr. Muhammad Jamal. You came into my life when most needed.

Contents

List of Figures

List of Tables

Basic Notation and Abbreviations

PSSP: Protein Secondary Structure Prediction

EM: Electron Microscopy

NMR: Nuclear Magnetic Resonance

MSA : Multiple Sequence Alignment

XOR: Exclusive OR

NN: Neural Networks

MLPs: Multi-Layer Perceptrons

SVM: Support Vector Machines

MSA: Multiple Sequence alignment

DSSP: Dictionary of Secondary Structure Prediction

STRIDE: STRuctural IDEntification

DEFINE

PDB: Protein Data Bank

PSIBLAST: Position Specific Iterated Blast

ES: Early Stopping

WES: Without Early Stopping

SOV: Segment Overlap Measure

QP: Quadratic Programming

$3D$: three-dimensional structures

$P(\alpha)$: conformational parameters for α

P_α: are the structural parameters

$\langle w, x \rangle$: inner product of w and x

M: margin

$K(x_i, x_j)$: kernel function

$\mathcal{F}$: feature space

$x \epsilon X$: input space

$y \epsilon Y = \{+1, -1\}$: output space

Chapter 1

Research Introduction

1.1 Introduction

This chapter presents an introduction to the study. The background of the research is discussed in Section 1.2. The aims that the study hopes to achieve are in given in Section 1.3 and Section 1.4 gives an outline of how the thesis is organised.

1.2 Background of Research

Predicting the structure of proteins from their primary sequence is becoming increasingly important in biochemistry. The key to the wide variety of functions shown by individual proteins is in their three dimensional structure adopted by this sequence. In order to understand protein function at the molecular level, it is important to study the structure adopted by a particular sequence. This is one of the greatest challenges in Bioinformatics.

There are 4 types of structures; Primary structure, Secondary structure, Tertiary structure and Quartenary structure [53]. Secondary structure prediction is an important intermediate step in this process because 3D structure can be determined from the local folds that are found in secondary structures. To set the background for the research study, the research context is explored. The objectives of the research and a summary of research methodology used are then presented. Furthermore, the thesis organisation is outlined.

There are different databases that record available protein sequences and their tertiary structures. However, sequence-structure gap is rapidly increasing. There are about 5 million protein sequences available from `http://www.ebi.ac.uk/trembl/` and about fifty thousand protein sequences that are available in `http://www.rcsb.org/pdb/`. Different techniques have been developed that can predict secondary structure of proteins from their amino acids sequences. They are based on different algorithms, such as Statistical Analysis (Chou and Fasman, 1974), Information theory, Bayesian Statistics and Evolutionary Information (Sen, Jernigan, Garnier, and Kloczkowski, 2005), Neural Networks (Holley and Kurplus, 1989; Qian and Sejnowski, 1988), Nearest Neighbour Methods (Salamov and Salovyev, 1995), a combination of multiple alignment and Neural Networks (Rost and Sander, 1993). For these approaches, the accuracy levels are in the range 65–80%.

This thesis examines the prediction of secondary structure of proteins from their sequences. The two methods of prediction that are used are Neural Networks and Support Vector Machines. Neural Networks have wide applications in pattern classification. They are useful for classification and function approximation. There are different types of networks used for different applications. The most commonly used is the Multilayer Feed Forward Networks. For these networks, there are 3 types of layers - input layer, hidden layers and output layer. The input layer consists of neurons that receive information (inputs) from the external environment. The hidden layer accepts information from the input layer and communicates it to other hidden layers. The information from the last hidden layer is sent to the output layer. Also for this type of networks, the layers are fully connected respectively and preceding layers are not allowed to communicate the information back to the other layers of a network. For many networks, one hidden layer is sufficient to approximate any given function with the required precision [23].

Practical applications of Neural Networks most often employ Supervised Learning. For supervised learning, training data that includes both the input and the desired result (the target value) is provided. After successful training, input data is presented to the Neural Network and the Neural Network will compute an output value that approximates the desired result. However, if the outputs do not approximate the desired outputs well, more training is performed. This means that the parameters of the network are adjusted until the output is close to the target value [3]. For training of Neural Networks, Resilient Backpropagation [40] is used. Once training has been performed, the network is evaluated to see whether the training process was successful. This evaluation process can be done using the test set. The purpose

of test set is to see if the network is able to identify unfamiliar data, and classify them into different classes. Neural Networks have been applied in secondary structure prediction. Some of the applications can be found on the work of [22], and [38].

Support Vector Machines are machine learning algorithms implemented for classification and regression [18]. For classification, Support Vector Machines operate by finding a separating hyperplane in the space of possible inputs. This hyperplane attempts to split the positive examples from the negative examples. The split will be chosen to have the largest distance from the hyperplane to the nearest of the positive and negative examples. Data points that are at the margin are called Support Vectors. These data points are very important in the theory of Support Vector Machines because they can be used to summarise information contained in the dataset. [18]. The hyperplane with a maximum margin allows more accurate classification of new points [52]. However, not all problems can be linearly separated by a hyperplane. For such problems, the resulting algorithm is formally similar, except that a non linear transformation of the data into a feature space is performed. This allows the algorithm to fit the maximum-margin hyperplane in the transformed feature space. Kernels are used to perform the mapping [14]. Support Vector Machines have previously been shown to predict the secondary structure of proteins [24], [32].

1.3 Goals of the Research

The main objective of this study is to predict the secondary structure of the proteins using two machine learning algorithms, Neural Networks and Support Vector Machines. This will be done by monitoring and measuring the performance of the algorithms using their prediction accuracies as a standard measure of performance. To achieve the objectives, the software Matlab Version 7.4.0.287 (R2007a) will be used to perform the experiments.

1.4 Research Method

Different approaches have been proposed to achieve prediction of secondary structures from the amino acid sequences [22], [38] . The following methodology is employed:

1. The main aim is to train the Neural Network and a Support Vector Machine to respond to the sequence of proteins when the predictions of the secondary structures are known. We also want to achieve recognition of amino acid patterns associated with the secondary structures. To perform the required classification of primary sequences into their secondary sequences, Matlab programs or codes will be set up. For Neural Networks, the toolbox Version 5.0.2 (R2007a) will be used while for SVM, Matlab codes by Schwaighofer, A. (2002) will be used.

2. The data to be used consists of 62 proteins from Rost and Sander (1983) database available from `http://www.anteprot-pbil.ibcp.fr/`. It contains a protein name, its primary and secondary sequences.

3. Preparation for the data for processing is done in steps. The first step performed is pre-processing. The data is presented in letters and the purpose of pre-processing is to convert those letters into real numbers. To achieve this, orthogonal coding, a similar coding scheme adopted by Holley and Karplus (1989) is used. The second step is secondary structure assignment. Secondary structures are classified into 8 categories H,G,E,B,I,T,S and the last category is for unclassified structures. This are reduced to 3 categories of H,E and C by using a secondary structure assignment called DSSP.

4. The Matlab codes implemented are for two class problems. And the following binary classifiers are created: the One-Against-All classifiers (helix vs. no helix, sheet vs. no sheet and coil vs. no coil) and the One-against-One classifiers (helix vs. sheet, sheet vs. coil and coil vs. helix).

5. 10-fold cross validation will be used to estimate the performance of the selected model. It also assists in comparing the two machine learning algorithms since it ensures equal and consistent partitioning of the data.

6. For the comparison of both Neural Network and Support Vector Machines, the overall

accuracy given by

$$\hat{P}(correct) = \frac{1}{N}\sum_{j=1}^{c} N_{jj}$$

is used. Here N is the number of all samples and N_{jj} is the number of correctly classified samples in class j. This accuracy measure gives the proportion of correctly classified samples.

1.5 Thesis Organisation

The thesis is organised in the following chapters:

Chapter 1: Research Introduction

This chapter introduces the research of study. It identifies the goals of the research and the research methodology. Furthermore, it outlines the structure of the remaining chapters.

Chapter 2: Protein Structure Prediction

This chapter explores the theoretical information of Protein Structure Prediction.

Chapter 3: Neural Networks

This chapter discusses the theoretical and background information of Neural Networks. The aim of this chapter is to develop an understanding of how Neural Networks can be used to achieve classification on data sets.

Chapter 4: Support Vector Machines

Support Vector Machine theoretical and background information is discussed in this chapter. The aim of this chapter also is to develop an understanding of how Support Vector Machines are implemented to perform classification tasks.

Chapter 5: Estimating the accuracy of a classifier

In this chapter, the estimates to assess the accuracy of a classifier are derived. This important part of Statistics is essential as it provides the research with a standard measure for comparison for both Neural Networks and Support Vector Machines.

Chapter 6: Data and Methods

This chapter develops the methodology to be used to aid comparison of the performance of Neural Networks and Support Vector Machines. The software used for Neural Networks and Support Vector Machines implementation is discussed. A summary of the type of data used and the accuracy measures to enable an objective comparison for the performance of Neural Networks and Support Vector Machines is given in this chapter.

Chapter 7: Results

This chapter contains the results of the simulations performed for Neural Networks and Support Vector Machines. Different network architectures in Neural Networks and variations over the cost parameter for Support Vector Machines are considered. A comparison of the performance of Neural Networks and Support Vector Machines is made. The results are achieved using codes implemented specifically for this study with Matlab and those implemented by Anton Schwaighofer (2002) for classification purposes. The codes are given in Appendix 3.

Chapter 8: Conclusion

This chapter concludes the research. It discusses the achievements of this research. The main observations of the thesis are highlighted in this chapter. Comparison of the results achieved in this study is made relative to literature. Further developments to improve the results achieved in this study are also discussed.

Appendix 1: Results of Neural Networks

Appendix 1 shows the results of Neural Networks achieved using Matlab.

Appendix 2: Results of Support Vector Machines

This appendix shows the results of Support Vector Machines also achieved with Matlab.

Appendix 3: Matlab Codes

Matlab codes used to achieve the results of the study are shown in this appendix. They have been developed for both Neural Networks and Support Vector Machines.

Appendix 4: Dataset

This appendix gives the data used in the study to achieve the research objectives. The data is arranged in rows by protein name, primary and secondary structures.

Chapter 2

Protein Structure Prediction

2.1 Introduction

This chapter gives an introduction to proteins and the prediction of their secondary structure. It provides a basis of the study. Section 2.2 discusses the functions of proteins. Section 2.3 relates structure to function. The importance of prediction and different approaches to prediction are discussed in Section 2.4.

2.2 Proteins

Proteins are made of simple building blocks called amino acids. According to [51], an amino acid is " a compound consisting of a carbon atom to which are attached a primary amino group, a carboxylic acid group, a side chain (R group), and an H atom. Also called an α amino acid." There are 20 different amino acids that can occur in proteins. Their names are abbreviated in a three letter code or a one letter code. The amino acids and their letter codes are given in Table 2.1 [5]:

Table 2.1: Amino acids

Glycine	Gly	G	Tyrosine	Try	Y
Alanine	Ala	A	Methionine	Mer	M
Serine	ser	S	Tryptophan	Trp	W
Threonine	Thr	T	Asparagine	Asn	N
Cysteine	Cys	C	Glutamine	Gln	Q
Valine	Val	V	Histidine	His	D
Isoleucine	Ile	I	Aspartic Acid	Asp	D
Leucine	Leu	L	Glutamic Acid	Glu	E
Proline	Pro	P	Lysine	Lys	K
Phenylalanine	Phe	F	Arginine	Arg	R

2.2.1 Functions of proteins

Proteins are important for living organisms. For example, our body fabric is made of protein molecules. Some of the proteins are found in the food we eat. Proteins also help to build their own protein molecules. They serve as hormones, receptors, storage, defense, enzymes and as transporters of particles in our bodies [53].

2.3 Protein Structure

2.3.1 Structure

There are four different structure types of proteins, namely Primary, Secondary, Tertiary and Quartenary structures. Primary structure refers to the amino acid sequence of a protein. It provides the foundation of all the other types of structures. Secondary structure refers to the arrangement of connections within the amino acid groups to form local structures. α *helix*, β *strand* are some examples of structures that form the local structure. Tertiary structure is the three dimensional folding of secondary structures of a polypeptide chain. Quartenary structure is formed from interactions of several independent polypeptide chains [53]. The four structures of proteins are shown in Figure 2.1.

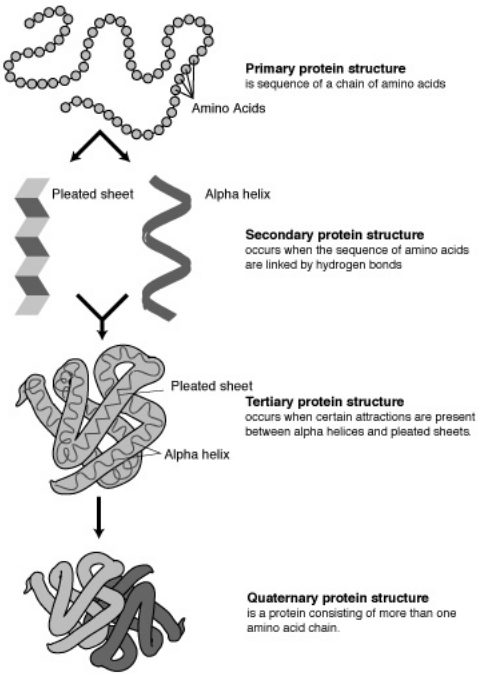

Figure 2.1: Protein structure
(from http://www.genome.gov//Pages/Hyperion/DIR/VIP/Glossary/Illustration/protein.cfm *accessed 15-01-2008 18:48 pm*)

2.3.2 Relationship between structure and function

There exists a relationship between protein structure and function. Proteins with similar sequences and structures have similar functions. Moreover, similar sequences in proteins imply that they also have similar structures. However, similar structures in proteins may have different sequences and different functions, [32]. The following section discusses the importance of prediction and different approaches to prediction.

2.4 Prediction

2.4.1 Importance of prediction

The primary structure of proteins can be used to predict its tertiary structure. It is through the tertiary structure of the protein that we can derive its properties as well as how they function in an organism. Currently, there exist databases with sequence structures of proteins whose tertiary structures are known e.g. Research Collaboratory for Structural Bioinformatics (RCSB) available from `http://www.rcsb.org/pdb/`. As of 29 January 2008, RCSB reported 48,638 available structures in the protein data bank. The results are achieved based on the following experimental approaches: X-ray, Nuclear Magnetic Resonance (NMR), Electron Microscopy (EM), and (other) which are the methods not mentioned. Table 2.2 shows a breakdown of structures available in RCSB from different experimental methods.

Table 2.2: Protein Data Bank current holdings breakdown as at Tuesday 29 January 2008

Experimental methods	Protein structures	Nuclei Acids	Protein/NA Complexes	Other	Total
X-ray	38541	1016	1770	24	41351
NMR	6080	802	137	7	7026
Electron Microscopy	112	11	41	0	164
Other	87	4	4	2	97
Total	44820	1833	1952	33	48638

From Table 2.2, X-ray is the most successful of all the methods since it has helped to determine about 85% of the structures available in the data bank. The number 48,638 represent all the structures determined until the end of January 2008. Knowledge of the tertiary structure of proteins can lead to rapid progress in the fields of protein engineering and drug design. However, there are still more known sequences than known structures. As of 15 January 2008, Release 37.7 of UniProtKB/TrEMBL contains 5,139,891 protein sequences. The protein sequences are available from `http://www.ebi.ac.uk/trembl/`. Because of a wide gap between tertiary structures (~50,000) and primary structures (~5,000,000), prediction is important in order to close this gap. However, there are limitations in some experimental approaches that are used for prediction. For example, X-ray crystallography and (NMR) spectroscopy. Both

methods are limited in time and in the form and size of the proteins that can be studied [50].

2.4.2 Secondary structure prediction

Secondary structure prediction means predicting the secondary structure of a protein from its primary sequence. It is important because knowledge of secondary structure helps in the prediction of tertiary structure. This is very interesting for proteins whose sequences do not show any similarities with the sequences of proteins in the database. Secondary structure has two properties, hydrogen bond patterns, and backbone geometry. Hydrogen bonded features include turns, bridges, $\alpha-helices$, $\beta-ladders$ and $\beta-sheets$ while bends, chirality, SS bonds and Solvent exposure are features which are determined geometrically [26].

Some of the computationally-based methods that can be used to achieve the secondary predictions include Neural Networks, Support Vector Machines, the Chou and Fasman Method, GOR V, Rost/Sander with frequency profiles and Nearest Neighbor Methods. These approaches will be reviewed next.

2.4.2.1 Neural Networks

The Neural Networks considered are usually of the feed forward type. The approach [22] uses a network that receives an input vector representing a segment of primary amino acid sequence.

- The input layer encodes a moving window in the amino acid sequence and secondary structure prediction is made for the central residue in the window. The length of the window can be varied.
- There are 21 input positions for the amino acids. 20 positions for the 20 amino acids (refer to table 2.1) and one to provide a null input used when no amino acid appears in that window. This occurs when the moving window overlaps the end of a sequence. For example for a window size of 5, suppose that we have "KLMNA" as our sequence. In that case the central residue that will be predicted for is "M". However, to predict for "N", the window overlaps the sequence end and we have the following sequence "LMNA*". In this case, (*) will correspond to the 21^{st} input unit.
- For a group of 21 inputs, the input corresponding to the amino acid type at that window position is set to 1 and all other inputs are set to 0. This binary vector can be of the

form, $(1,0,0,0...,0)$ or $(0,0,1,0,...,0)$ etc. Hence the network has an input layer of 5 $\times$21 neurons e.g. for a window of length 5.

- The output layer consists of 2 units and the coding for the secondary structure of an amino acid is $\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \alpha$, $\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \beta$, $\begin{pmatrix} 0 \\ 0 \end{pmatrix} = coil$.
- Training is performed using Resilient Backpropagation [40].

This method has been applied on 62 proteins from the Kabsch and Sander database [26]. The window size used was 17. The three quality indices were used, the probability of correct prediction of a residue to be in state S, $PC(S)$ [22] for the three states α, β, and c, the total percentage correct of predicted Q_3 and the correlation coefficients C_S [38]. $PC(S)$ gives the percent of residues that are correctly predicted to be in state S i.e. α, β or *coil* is given by :

$$PC(S) = \frac{\text{number of correct predictions in state } S}{\text{total number of residues predicted in state } S} \times 100. \tag{2.1}$$

The percent of Overall Accuracy (OA) which is the total percentage of all correct predictions often denoted by Q_3 in all the three structures is,

$$OA = \frac{P_\alpha + P_\beta + P_c}{N} \tag{2.2}$$

where N is the total number of residues and P_S is the total number of correct predictions of type S . The correlation coefficients for each state S were obtained as follows:

$$C_S = \frac{(p_S\ n_S) - (u_S\ o_S)}{\sqrt{(n_S + u_S)\,(n_S + o_S)\,(p_S + u_S)\,(p_S + o_S)}} \tag{2.3}$$

where:

- p_S refers to the number of correctly predicted residues in state S.
- n_S is the number of residues, which are not part of state S and have correctly been identified not to belong to state S.

- o_S is the number of residues which are not part of state S and have incorrectly been predicted to belong to state S.
- u_S is the number of residues that belong to state S which have been missed during classification and therefore not classified to any structure.

Table 2.3 gives the performance results for the protein secondary structure prediction reported in [22].

Table 2.3: Protein secondary structure prediction using Neural Networks

Quality index	Training set (48 proteins)	Test set (12 Proteins)
Percent correct (three state total)	68.5	63.2
Correlation coefficients		
C_α	0.49	0.41
C_β	0.47	0.32
C_c	0.43	0.36
Percent correct of predicted		
$PC\,(\alpha)$	65.3	59.2
$PC\,(\beta)$	63.4	53.3
$PC\,(c)$	71.1	66.9

From the NN approach, each state has a prediction accuracy of more than 50% with the coils having the highest prediction. A similar pattern is observed on the test set though it has dropped significantly compared to the accuracy on the training set. Prediction of alphas and betas is not too different on the training set but significant differences can be seen on the two classes of the test set. A similar approach by Qian and Sejnowski [38] which uses a different output coding $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \alpha$, $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \beta$ and $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = coil$ can be used.

Many researchers have applied Neural Networks to solve the problems of secondary structure prediction. Some of them include [9], [27], [45], and as mentioned above [38].

2.4.2.2 Support Vector Machines

Support Vector Machines [13] are machine learning algorithms that are used for classification and regression problems. Given a two class problem, the Support Vector Machines approach can be summarised in the following way:

- First, find a linear decision boundary that separates the data into two classes.
- If the boundary does not separate the data successfully, the input vectors are mapped in higher dimensional space using a non-linear function φ where it is hoped that the separation will be achieved.
- Kernel methods are used to enable mappings to higher dimensional space.

For multi-class protein structure prediction by Hua and Sun [24], the data have 3 classes, α, β, and c. In this approach, the following six binary classifiers are created: $H/\sim H$, $E/\sim E$, $C/\sim C$, H/E, E/C, and C/H. $H/\sim H$ refers to secondary structures which are helices against those, which are different from the helices. However, H/C distinguishes between secondary structure types H and C. The dataset used in this approach is the $RS126$. It contains 126 proteins chains that share less than 25% identity. The data is available from `http://antheprot-pbil.ibcp.fr/`. For an objective comparison with other methods, the multiple sequence profiles taken from the HSSP database available at `http://www.sander.embl-heidelberg.da/hssp/` are used. OA is obtained in the same manner as in (2.2). Table 2.4 gives the results of multi class Support Vector Machines with a window length of 17 residues.

Table 2.4: Protein secondary structure prediction with Support Vector Machines

Quality index	% Accuracy
Overall accuracy	73.5
$H/\sim H$	79.36
$E/\sim E$	79.15
$C/\sim C$	67.10
H/E	72.02
E/C	72.10
C/H	74.66

The results of multi class SVM show that $H/\sim H$ has the highest accuracy while $C/\sim C$ has the lowest prediction accuracy. Hua and Sun combined the binary classifiers to construct tertiary classifiers. One of the ways this was achieved was to use the outputs of the binary classifiers as inputs to the Neural Networks. They created a network with an input layer of size 6, a hidden layer and 3 output units of H, E, or C. SVM like the other methods discussed earlier, predict secondary structure of proteins from their primary sequence.

2.4.2.3 Chou and Fasman Method

The Chou Fasman Method [12] is based on a set of predictive rules and a table of predictive values called conformational parameters. These parameters, denoted by P_α and P_β, are the quantitative measures of potential helical and sheet segments in a protein. The method relies in predicting for a single residue. The following prediction rules are used to determine alpha and sheet regions in a protein:

- Helix regions are those in which 4 out of 6 segments with residues that are helices exist. For any segment of six residues or longer in a protein with $P(\beta) \geq 1.03$ and $P(\alpha) > P(\beta)$ is predicted as helical.
- Similarly a segment of five or more residues in a protein with $P(\beta) \geq 1.05$ where $P(\beta) > P(\alpha)$ and terminating at the right place with the right boundaries, is predicted as $\beta - sheet$.
- Residues that are neither $\alpha - helix$ nor $\beta - sheet$ are regarded as $turns$.

In this approach, the 20 amino acids are classified into three categories: formers, breakers, and those which are indifferent to the helical and β regions. This classification is performed before calculating P_α and P_β for any protein segment.

In [12], 19 proteins were evaluated to determine the performance of the Chou Fasman Method in predicting their secondary structures. Conformational parameters were derived based on 15 proteins and later tested on the last 4 proteins. The following measures are used to assess the performance of the approach in predicting secondary structure of proteins.

% of residues correctly predicted in each conformational state k :

$$\%_k = \frac{(n_k - \text{number incorrect})}{n_k} \times 100 \tag{2.4}$$

where k represents the α, β or coil regions in the structure of proteins determined by X-ray analysis, n_k is the total residues in state k while the number incorrect is the number of k residues missed in prediction. The % of total residues in the proteins that are predicted correctly is given by:

$$\%_N = \frac{(N - n)\,(N - \text{number incorrect})}{N} \times 100 \tag{2.5}$$

where N is the total residues in a protein.

The quality of prediction of a given type of secondary structure is given as

$$Q_k = \frac{\%_k + \%_{nk}}{2}. \tag{2.6}$$

The $\%_k$ is the same as the one in (2.4). $\%_{nk}$ represents the percentage of correctly predicted residues not incorporated in the conformational state and is given by:

$$\%_{nk} = \frac{(n_{nk} - \text{number incorrect})}{n_{nk}} \tag{2.7}$$

and $n_{nk} = N - n_k$. The number incorrect prediction in this case refers to the residues that are over predicted in state k. The overall performance of the approach on 15 proteins and 4 for testing are shown in Table 2.5.

Table 2.5: Protein secondary structure prediction from Chou Fasman Method

Number of proteins	$(n_\alpha/n_\beta/N)$	(α_m/α_0)	(β_m/β_0)	Total income	$\%_\alpha$	Q_α	$\%_\beta$	Q_β	$\%_N$	$\%_{\alpha+n\alpha}$
15	(922/442/2473)	171/159	65/224	560	81	86	85	87	77	87
4	(246/173/760)	60/44	22/99	192	76	84	89	86	75	86

In Table 2.5, Q_α and Q_β are obtained with (2.6). The correct predictions in each conformational state $\%_\beta$ can be computed with (2.4). $\%_N$ is obtained in a similar way as in (2.5). The following indices are also used to achieve results in Table 2.5:

1. $(n_\alpha/n_\beta/N)$ - refers to the number of helical residues, $\beta - sheet$ residues and the total residues for the 15 proteins.
2. (α_m/α_0) - are respectively the number of helical residues missed in prediction and over predicted.
3. (β_m/β_0) - are also respectively the number of $\beta - sheet$ residues missed in prediction and over predicted
4. $\%_{\alpha+n\alpha} = \frac{(N-\alpha_m-\alpha_0)}{N} \times 100$ - refers to the residues predicted correctly when the β conformation is neglected in the prediction.
5. Total incorrect $= (\alpha_m + \alpha_0 + \beta_m + \beta_0)$-

   (number of incorrect residues counted twice in α_m and β_0 or β_m and α_0)

From this approach, the percent of total residues from the 15 proteins that are correctly predicted is 77%. The approach also achieves an accuracy of 75% on the 4 proteins used for testing. It is interesting to mention that though Chou and Fasman record these high accuracies, the accuracies drop when different proteins are used. For example, Kabsch and Sander [27] used the data set from [26] and their results based on Chou and Fasman method show the prediction accuracy of about 50% for three state prediction.

2.4.2.4 GOR V

The GOR method [29] is one of the earlier methods of secondary structure prediction from their primary sequence. Different versions were used for this purpose: GOR I,II,III, and IV. All these versions are web based and predictions are made directly from them. The difference in the methods is in the database used, predicted secondary structures and also what the

method is based on. Table 2.6 gives the details of the previous methods and the database used for this algorithm.

Table 2.6: GOR Method of protein secondary structure prediction

Type	Number of proteins	Number of residues	Conformations	Algorithm is based on:
GOR I	26	4500	H,E,C and turns T	Singlet frequency information within the window
GOR II	75	12757	H,E,C and turns T	Same as in GOR I
GOR III	75	12757	H,E,C	Information about the frequencies of pairs (doublets) of residuals
GOR IV	267	63566	H,E,C	Information theory and Bayesian statistics
GOR V	513	84107	H,E,C	Information theory, Bayesian statistics and evolutionary information of a residue

The GOR V [48] algorithm is implemented by combining information theory, Bayesian statistics, and evolutionary information. It is made available online from `http://gor.bb.iastate.edu/`. PSI-BLAST is used to include the evolutionary information of sequence and also to generate multiple alignments of which only sequence alignments are used. The sequences for this method were taken from a database constructed from the Cuff and Barton database [15]. To ensure sequence dissimilarities, aligned sequences with $z - scores < 5$ are considered dissimilar.

Prediction of secondary structure of a protein can be performed in the following way:

- Provide the sequence to the server.

- At each residue position, GOR V server that was trained on 513 proteins calculates the helix, sheet, and coil probabilities, and structural states having highest probabilities are used to make an initial prediction.
- The following post initial prediction applies:
  - helices shorter than five residues are converted to coil.
  - sheets shorter than two residues are converted to coil.
- The user receives as output the secondary structure prediction for the input sequence and the probabilities for each secondary state element at each position.
- Prediction results are automatically sent to the user through email and they are also shown in the web browser, which should stay open during the run.

The GOR V server uses OA obtained with (2.2) and Segment Overlap Measure (SOV) to assess prediction accuracy [15]. While prediction for a sequence with 100 amino acids can take approximately one minute, more time is spent on PSI_BLAST alignments. The results obtained are based on full jack-knife procedure. For this method, during prediction a sequence whose structure is already predicted is removed from the database. The results [29] of this approach are given in Table 2.7.

Table 2.7: GOR V for secondary structure prediction

Quality index	% Accuracy
Overall accuracy	73.5
SOV	70.8
Percent correct of predicted	
H	73.7
E	70.0
C	74.2

Unlike Neural Networks, GOR V predicts coils better.

2.4.2.5 Nearest Neighbor Method

The approach [46] is to use the secondary structures of the known proteins to determine the structure of the new target protein. For the test residue, prediction for an amino acid residue in the centre of a sequence window is performed. Here is a summary of the steps that can achieve this:

- Identify sequences of known structures.
- A window of length n is slid through known sequences to create a list of short sequences. Comparisons of test window sequences are made with n residue windows from the database. Sequences chosen should have some window similarities with the target protein.
- The short sequences chosen are the nearest neighbours.
- Secondary structures of the short segments for all the windows are recorded. This is done for each central amino acid.
- Frequencies of the middle amino acid in each of the matching fragments are then used to predict the secondary structure of the middle amino acid residue.

The decision as to whether the window of a sequence is similar to the window of a target protein is achieved by the use of scores. The method of scores involves creating some environment classes for residues with known tertiary structures. Local structural features i.e. secondary structures are used to achieve the groupings. Thus, for each alignment of the residues belonging to a particular class, a score is assigned.

The data used for the application of Nearest Neighbor Method with Multiple Sequence Alignment is the RS126 data set. The alignments of homologous proteins for RS126 data set were also obtained. The prediction of these three structures was based on the jack-knife procedure. Overall accuracy OA, is 70.9%. The mean score value is used to achieve this accuracy. An improvement on the accuracy to 71.3% was due to the inclusion of information about the deletions in aligned sequences. These are predicted as coil. The outputs of the nearest neighbours were combined and used as inputs into the Neural Network. These increased the accuracy by 0.9% to 71.8%. Further improvement to 72.2% of the multiple sequence alignment approach was achieved through exclusion of short α and β.

The following rules were applied:

1. Helices of residue length one or two are converted to coil.
2. A strand of residue length one is converted to coil.
3. Strand segments of length two surrounded by alpha residues are converted to alpha.

Though different methods have different effect on the prediction accuracy, the changes in accuracy are based on 70.9%. There are some variations in the window sizes and in the number of nearest neighbours. The following performance measure Score was used:

$$Score\left(R_i, E_j\right) = log_{10}\left(\frac{P\left\langle R_i \mid E_j\right\rangle}{P\left(R_i\right)}\right) \tag{2.8}$$

where $P\left(R_i \mid E_j\right)$ is the probability of finding residue i in environment j, and $P\left(R_i\right)$ is the probability of finding residue i in any environment.

Mean score value is the average scores of the homologous fragments of a test window sequence compared to a particular fragment database. The overall accuracy, Q_3 is calculated in the same manner as in (2.2). The results of these three approaches are given in Table 2.8.

Table 2.8: Protein secondary structure prediction with Nearest Neighbor and Multiple Sequence Alignments.

Scheme	Window size	Number of Nearest Neighbors	$Q_3\%$
MSA	17	50	71.3
Jury MSA	$11, 17, 23$	$50, 100$	71.8
Jury MSA filter	$11, 17, 23$	$50, 100$	72.2

For jury MSA and Jury MSA filter schemes, window size 11,17, 23 means that the different schemes were combined together and different window sizes were used. Prediction accuracy achieved is 71.8% and 72.2% respectively for the schemes.

2.4.2.6 Neural Networks using Frequency Profiles

In this approach [44], similar proteins are grouped together, and their aligned sequences presented to the network for prediction. The advantage is that multiple sequence alignment contains more information about proteins than single sequences. The approach is as follows:

- Multiple sequence alignments are used as inputs to the neural network.
- During training, the data base of protein families aligned to proteins of known structure is used.
- For prediction, the data base is scanned for all homologues of the protein to be predicted and the family profile of amino acid frequencies at each alignment position is fed into the neural network.
- Each type of secondary structure has an equal proportion of 33% of the states during training. This ensures balanced training as the proportions are not based on the data base.

The first network has three layers: input layer, hidden and output layer which has three output neurons where $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ codes for α. The outputs of the first network are used as inputs into the second network, which are propagated through the network to obtain new output comprising of the three states.

The data consisting of 126 proteins, in which no two sequences have more than 25% of identical residues, were used for the prediction. The data are from the database in [21]. Three quality indices were used: OA the percent correct for the three states (2.2), the percentage of correct predictions for each state (2.1) and correlation coefficients (2.3). Table 2.9 gives the results of the performance of frequency profiles in predicting the secondary structure of the proteins.

Table 2.9: Protein secondary structure prediction with NN and Frequency Profiles

Quality index	Training set (%)	Test set (%)
Percent correct (three state total)		69.7
Percent correct of predicted		
H	70	72
E	64	57
C	72	73

Frequency profiles have improved the performance on unseen data compared to simple NN. Unlike in NN, the prediction accuracy of coils and alphas is about the same, above 70%. Though the prediction of betas has improved, it is still lowest of all the classes. The next section describes a method which predicts tertiary structure from primary sequence.

2.4.3 Tertiary structure prediction

There are several approaches that attempt to reduce the gap between known amino acid sequences (primary structure) and tertiary structures. Homology, a knowledge-based approach, will be considered.

2.4.3.1 Homology/Comparative Modeling

Homology modeling [33], otherwise known as *Comparative modeling,* is a knowledge-based approach to protein structure prediction which attempts is to relate a sequence of new protein to the database of proteins of known tertiary structures. The idea is to look at the sequence (primary structure) of the protein of interest and identify at least one tertiary structure that displays essentially similar primary sequence as the protein to be modelled. Thereafter predictions of the three-dimensional structure of the target population will be based on the three dimensional structure of the close match. The steps to achieve such modelling are as follows:

- Search for proteins whose 3D structures are known and whose primary sequences are related to the new sequence whose tertiary structure we wish to determine.
- Pick those structures that will be used as templates. These are the possible sequences close to the sequences of the new protein.

- Align the template sequences with the target sequence.
- Build the model for the target sequence given its alignment with the template structures.
- Evaluate the model using a variety of principles. However, the whole process from template selection to model building can be performed repeatedly until a desired model is achieved.

There exist different databases to choose template proteins from. Selection of a valid model to be used for structure prediction can also be made based on several methods. Table 2.10 gives an example of the programs and servers that can be used to choose the required model.

Table 2.10: Programs and servers used for model selection in Comparative Modeling

Stage	Method	Type
Database	ProteinDataBank	Server
Template Search	BLAST	Server
Sequence Alignment	BLAST	Server
Modeling	MODELER	Program
Model Selection	AQUA	Program

2.4.3.2 Effectiveness of Comparative Modeling Methods

The accuracy of comparative modeling methods in predicting tertiary structure of proteins depends on the sequence similarities of the proteins that are used as templates and the target protein. Proteins with more than 50% of sequence similarity with proteins in the database, high accuracy predictions can be achieved. For similarities of 30-50% in sequences between target protein and protein with known structure, medium accuracy will be achieved. If the sequence similarities are below 30%, then prediction will be very poor [23].

Chapter 3

Neural Networks

3.1 Introduction

This chapter provides an introduction to Neural Networks. It is arranged in 4 sections. Section 3.2 gives a basic theory of Neural Networks. Section 3.3 discusses Multilayer Perceptrons. It also states an important theorem in Neural Networks: the universal approximation property. Section 3.4 looks at different learning approaches that can be used to achieve minimum error and better performance on the unseen data. The Resilient Backpropagation Algorithm and Early Stopping will be discussed in this section.

3.2 Basic Theory of Neural Networks

3.2.1 Biological inspiration

Artificial Neural Networks receive their motivation from biological systems because they are based on the idea of imitating the human brain. The neuron is the basic unit of the brain that processes and transmits information [16]. A neuron has the following structural components: synapses, dendrites, cell body, and axon. Neurons have their connections through the synapses. Synapses communicate and process information through the dendrites into the cell body. Each dendrite may have many synapses attached to it. The signals are accumulated in the cell body and when their quantity reaches a certain level (usually referred to as the threshold), then the new signal is released and carried to other neurons through the axons, [43]. See Figure 3.1.

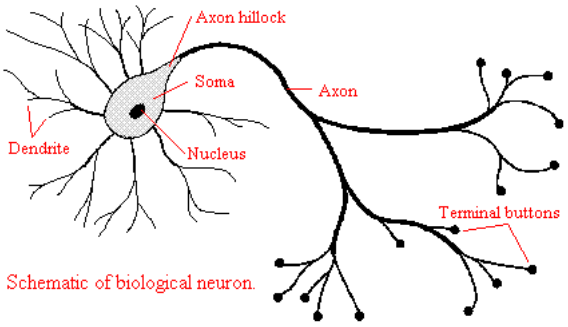

Figure 3.1: Biological Neuron
(from http://www.virtualventures.ca/~neil/neural/neuron-a.html *15-01-2008 20:00 pm*)

3.2.2 What are Neural Networks?

Haykin [20], describes Neural Network (NN) as follows:

> "A massively parallel processor that has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects: knowledge is acquired by the network through a learning process. Interneuron connection strengths known as synaptic weights are used to store the knowledge."

In [41], Neural Networks are discussed in relation to other methods for classification such as Classification trees, Nearest Neighbour, Logistic discriminant and Linear discriminant methods. From a statistical point of view, two-class classification problems can be performed with logistic discrimination. This is a very important result in classification because Logistic discriminant "can be fitted by maximum likelihood, which corresponds to fitting a neural network without a hidden layer" [41]. Tests were done using synthetic datasets with two variables to show the relationship between the classification methods. The results from this analysis revealed small differences in the error rates during prediction based on different classification methods.

From the discussions in [41], it is difficult to distinguish between Neural Networks and other modern statistical methods. However, it is imperative to note that for the test set of synthetic

datasets, the two populations are normally distributed. This implies that the results of Neural Networks were also obtained under the same assumptions as other methods. This might have a possible influence on the results and perhaps if there were no statistical model assumptions, Neural Networks would have performed better. The motivation could be that Neural Networks do not need statistical model assumptions to operate.

3.2.3 Why use Neural Networks rather than statistical approaches for prediction?

Neural Networks have applications in speech and pattern recognition, time series and in recent years, predicting the secondary structure of proteins. Literature shows their performance and success in predicting the secondary structure of proteins from their amino acid sequences (primary sequence) with prediction accuracy of 64.3% from earlier approaches [38] to 70% and better based on multiple sequence alignment rather than single sequences [44]. Neural Networks have properties which distinguish them from other machine-learning and statistical techniques. They are adaptive systems that build a model that finds relationships between inputs and outputs. The input-output mapping is established through changing the parameters of the network to obtain the desired output. They can approximate any function to any prescribed precision. Unlike most statistical approaches, Neural Networks are not based on the assumption of a particular probability model [11].

3.2.4 A model of a Neuron

Suppose we have an input vector $x = (x_1, x_2, x_3, ..., x_n) \in R^n$ of real numbers. The inputs are sent through the connections to the accumulator and the connections may carry some weight vector $w = (w_1, w_2, w_3, ..., w_n) \in R^n$. These are applied to the inputs x_i by multiplying the input with its corresponding weight w_i. The products are then added and the information sent to the accumulator is of the form

$$\langle w, x \rangle = \sum_{i=1}^{n} w_i \cdot x_i.$$

The accumulated sum of the inputs is compared to a value called the threshold such that if

greater than the threshold, then the output is 1 and if less, it is 0. The threshold adds a charge known as the bias to the sum of the signals. The bias is denoted as b. The output produced is given as

$$y = f\left[\langle w, x\rangle + b\right] \tag{3.1}$$

where f is an activation function and y is the output signal. An example of an activation function is the Heaviside function and it is given by

$$f(x) = \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{for } x \leq 0 \end{cases} . \tag{3.2}$$

The bias can be included in the neuron in which case, the input x_0 has a value of $+1$ attached and multiplied to give a threshold value $w_0 = b$ and added to the sum to produce equivalent output [2].

$$y = f\left[\langle w, x\rangle\right] = f\left(\sum_{i=0}^{n} w_i \cdot x_i\right). \tag{3.3}$$

The mapping from the inputs $x_0, x_1, x_2, x_3, ..., x_n$ to the output y is shown in Figure 3.2.

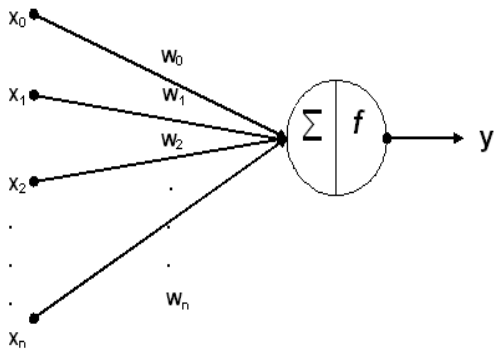

Figure 3.2: Basic neuron model

From Figure 3.2, a step function f is applied to $\sum$ which represents the sum obtained from the products of weights and biases for the neuron. This model represents a Perceptron**.** A perceptron is a neural network with binary outputs**,** i.e. as in a Heaviside function, where the output has values of 0 or 1. For a set of points, a perceptron is used in classification. Consider a situation in which we have inputs $x = (x_1, x_2)$ belonging to classes A and B. This gives rise to the form of classification illustrated in Figure 3.3. A point that lies above the line belongs to class 1 and a point below the line belongs to class 2.

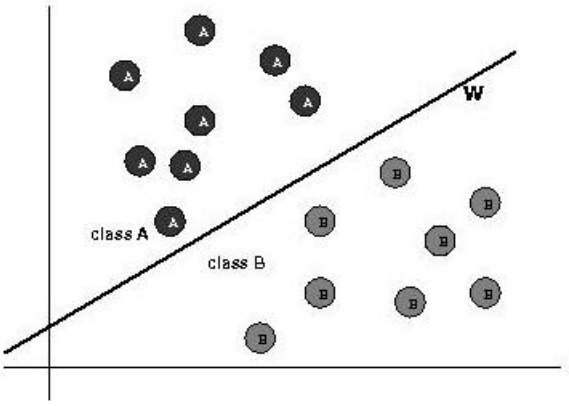

Figure 3.3: Classification to illustrate the perceptron.

The decision boundary W is generated by the perceptron. If the boundary does not give a correct classification the weights of the perceptron are changed until the correct output is achieved. The success of this method depends on whether classification gets the learning task (inputs and matching targets) right or not. If the perceptron gets the classification right, no adjustment is required. However, if it gets the classification wrong, then the weights are changed to get it right [2].

3.2.5 A layer of Neurons

One layer of neurons

Generally, there will be more than one neuron in a network. For such networks, inputs are connected to the neurons and then the output is obtained. Consider a layer with two neurons. Figure 3.4 shows such a network.

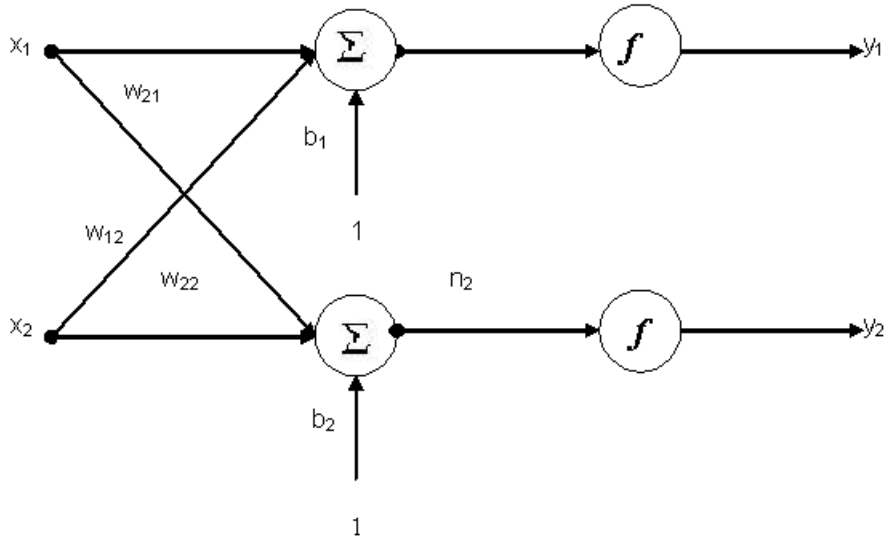

Figure 3.4: A network with a layer of neurons

This network has two neurons in a layer. A typical network will have more than two neurons for each layer. The weights of connections of neurons 1 and 2 are respectively denoted as follows;

$$(w_{11}, w_{12}), (w_{21}, w_{22}).$$

By convention, the indices of the weights are:

1. First index = relates to a neuron
2. Second index = relates to the input

For the net outputs $n_1$and n_2

$$\begin{aligned} n_1 &= w_{11}x_1 + w_{12}x_2 + b_1 \\ n_2 &= w_{21}x_1 + w_{22}x_2 + b_2 \end{aligned} \tag{3.4}$$

and using the matrix multiplication, we can write this in matrix-vector form as:

$$\begin{pmatrix} n_1 \\ n_2 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

The matrix

$$W = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}$$

is called the weight matrix. To the net output, a transfer function f is applied componentwise to obtain the following output:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = f \begin{pmatrix} n_1 \\ n_2 \end{pmatrix} = f(\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}). \tag{3.5}$$

To generalize to the case of a neural network with n inputs $(x_1, x_2, ..., x_n)$ and m neurons and m outputs $(y_{1,}, y_2, ..., y_m)$, the weight matrix is

$$\begin{pmatrix} w_{11} & w_{12} & . & . & . & w_{1n} \\ w_{21} & w_{22} & . & . & . & w_{2n} \\ . & & & & & . \\ . & & & & & . \\ . & & & & & . \\ w_{m1} & w_{m2} & . & . & . & w_{mn} \end{pmatrix}$$

and the bias vector $(b_1, b_2, ..., b_m)'$. The net output becomes

$$\begin{pmatrix} n_1 \\ n_2 \\ . \\ . \\ . \\ n_m \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & . & . & . & w_{1n} \\ w_{21} & w_{22} & . & . & . & w_{2n} \\ . & & & & & . \\ . & & & & & . \\ . & & & & & . \\ w_{m1} & w_{m2} & . & . & . & w_{mn} \end{pmatrix} . \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ . \\ . \\ . \\ b_m \end{pmatrix} . \tag{3.6}$$

Component wise evaluation of the transfer function yields

$$\begin{pmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_m \end{pmatrix} = f(\begin{pmatrix} w_{11} & w_{12} & . & . & . & w_{1n} \\ w_{21} & w_{22} & . & . & . & w_{2n} \\ . & & & & & . \\ . & & & & & . \\ . & & & & & . \\ w_{m1} & w_{m2} & . & . & . & w_{mn} \end{pmatrix} . \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ . \\ . \\ . \\ b_m \end{pmatrix}). \tag{3.7}$$

Therefore, from a given list of inputs $(x_1, x_2, ..., x_n)' \in R^n$ and the output $(y_1, y_2, ..., y_m)' \in R^m$ is computed. With the notation $x_0 = 1$ and $w_{0k} = b_k$ this equation becomes:

$$\begin{pmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_m \end{pmatrix} = f(\begin{pmatrix} w_{01} & w_{02} & . & . & . & w_{0n} \\ w_{11} & w_{12} & . & . & . & w_{1n} \\ . & & & & & . \\ . & & & & & . \\ . & & & & & . \\ w_{m1} & w_{m2} & . & . & . & w_{mn} \end{pmatrix} . \begin{pmatrix} 1 \\ x_1 \\ . \\ . \\ . \\ x_n \end{pmatrix}). \tag{3.8}$$

$F : R^n \to R^m$ given by (3.7) or (3.8) is called the input-output mapping of the neural network. It depends on the weights and biases as parameters [43].

More than one layer of Neurons

A neural network can have a single output, e.g. a perceptron or more than one output, such as in Figure 3.2 on page 29. Networks may be designed to have more than one layer, in which case we will have the input layer, hidden layers and the output layer. The hidden layers are the layers between the input and output layer. The case of one hidden layer is shown in Figure 3.5.

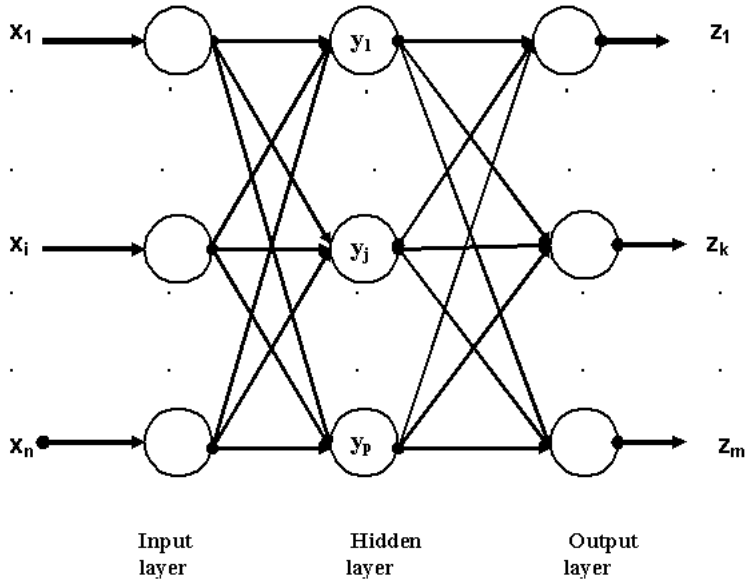

Figure 3.5: A network with one hidden layer of neurons

The input-mapping $F : R^n \rightarrow R^m$ of a neural network with a layer of neurons can be extended to the case in which there is a layer of neurons between the inputs and the outputs. Let us change the notation a little. The inputs are $(x_1, x_2, ..., x_n)' \in R^n$ as before, but the notation for the outputs $(y_1, y_2, ..., y_m) \in R^m$ will change to $(y_1, y_2, ..., y_p) \in R^p$. Therefore, (3.8) is rewritten as:

$$\begin{pmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_p \end{pmatrix} = f(\begin{pmatrix} w_{01} & w_{02} & . & . & . & w_{0n} \\ w_{11} & w_{12} & . & . & . & w_{1n} \\ . & & & & & . \\ . & & & & & . \\ . & & & & & . \\ w_{p1} & w_{p2} & . & . & . & w_{pn} \end{pmatrix} . \begin{pmatrix} 1 \\ x_1 \\ . \\ . \\ . \\ x_n \end{pmatrix}). \tag{3.9}$$

If we denote the weight matrix in (3.9) as W, then (3.9) can also be written as $y = f(W \cdot x)$. To compute the output $(z_1, z_2, ..., z_m)' \in R^m$ for a network with 1 hidden layer, the output $(y_1, y_2, ..., y_p) \in R^p$ from the previous layer is used as an input into the next layer. Therefore, it follows that

$$\begin{pmatrix} z_1 \\ z_2 \\ . \\ . \\ . \\ z_m \end{pmatrix} = f(\begin{pmatrix} v_{01} & v_{02} & . & . & . & v_{0p} \\ v_{11} & v_{12} & . & . & . & v_{1p} \\ . & & & & & . \\ . & & & & & . \\ . & & & & & . \\ v_{m1} & v_{m2} & . & . & . & v_{mp} \end{pmatrix} . \begin{pmatrix} 1 \\ y_1 \\ . \\ . \\ . \\ y_p \end{pmatrix}) \tag{3.10}$$

and if V denotes the weight matrix in (3.10), then $z = f\langle V \cdot y \rangle$. We can therefore summarise that $F(x) = f\langle V \cdot y \rangle = f\langle V \cdot f\langle W \cdot x \rangle \rangle$. Hence, the input-output mapping of a network with 1 hidden layer is a function $F : R^n \to R^m$.

3.3 Multilayer Perceptrons

Networks to be considered are Feed Forward Neural Networks (FFNN). For this type of network, the inputs have connections to other layers in the network. The output of each layer is the input to the subsequent layer. Information can be sent only in one direction: the forward direction. This means that for a FFNN, each layer has connection only to the subsequent layer. Feed forward networks with sigmoid transfer functions are called Multilayer Perceptrons (MLPs). The major difference between the multilayer perceptrons and the single layer

perceptron is in their activation functions. Perceptron use step functions for activation. They also have direct connections of inputs to outputs, while multilayer perceptrons use sigmoid functions and have input connections that do not go directly to the output layer but go instead through hidden layers [3]. Figure 3.5 shows an example of MLPs with p hidden neurons.

3.3.1 Linear inseparability

The perceptron has wide applications for linearly separable problems. By linearly separable problems, we mean those in which there exists a hyperplane that separates the inputs into classes. The perceptron finds weights and biases to achieve such classification. An example of linearly separable problems is also shown in Figure 3.3 on page 30.

Problems that are not linearly separable are called linearly inseparable. An example is the XOR problem (Figure 3.6). In this problem, it is not possible to achieve correct classification using linear approaches. Suppose that we have the inputs $x = (x_1, x_2)$ belonging to two classes of $c = \{c_1, c_2\}$ as illustrated. In this example, vectors $x = (0, 0)$and $x = (1, 1)$ belong to c_1 while $x = (0, 1)$ and $(0, 1)$ belong to c_2. However, not all the points can be classified correctly by a perceptron [3].

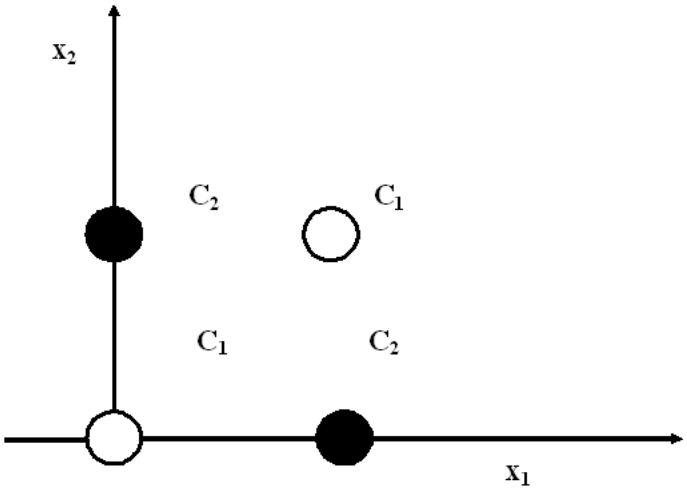

Figure 3.6: XOR problem

The immediate approach to the XOR problem would be to create more perceptrons such that each of them finds sections of inputs that are linearly separable. A class of an output can be determined and if classification is successful, there is no need to adjust. However, in case of a misclassification the result is that the weights cannot be adjusted to give the correct output because of the hard limit function that is used. The result is that, the network cannot determine by how much to adjust the weights to get the classification right. A correct approach would be to modify the activation function to give details of the weights so that learning can take place. An example for such an activation function is the sigmoid function [2]. The sigmoid functions are obtained by evaluating (3.11) below [43]:

$$f(x) = \frac{1}{1 + e^{\beta x}}, \quad \beta > 0. \tag{3.11}$$

The graph of the Sigmoid function is shown in Figure 3.7.

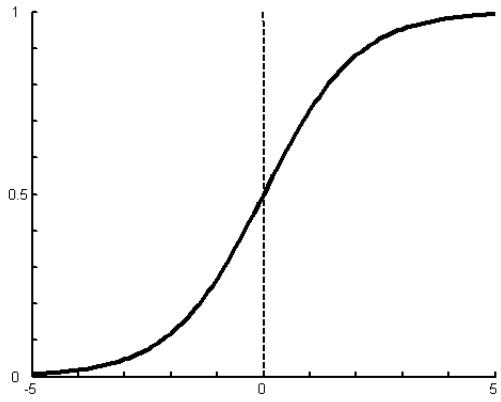

Figure 3.7: Sigmoid Function

3.3.2 Universal Approximation Property

Multilayer perceptrons are very successful and widely used. One of the reasons is that they are universal approximators of continuous functions [23].

THEOREM 2.1. *Given any measurable* $F : R^n \rightarrow R^m$, *there exists a feed-forward network with one hidden layer which can approximate that function well to any desired accuracy.*

This means that for FFNN, one hidden layer is sufficient to approximate any function with prescribed precision. This theorem is, however not constructive because it does not give a hint of how many hidden neurons we need and how the weights and biases have to be chosen. Therefore, learning strategies are needed to determine those parameters. Section 2.4 outlines some of the algorithms used during learning to determine the parameters of the network.

3.4 Learning Algorithms for MLPs

3.4.1 Supervised learning

Feed forward networks can be trained by a technique known as Supervised Learning. In this technique, input samples are presented to the network together with a set of desired responses at the output layer. The differences (errors) between the desired and actual response for each node in the output layer, are found. Weight changes in the network are determined according to a certain learning algorithm. The objective is to create the input-output model with correct mapping such that for unseen inputs, their outputs can be predicted successfully [43]. Learning, or training is a process in which the parameters of the network are changed to make differences between the actual output and the desired output as small as possible [2].

3.4.2 Backpropagation

3.4.2.1 Steepest descent to minimize error function

Suppose we have the following learning task:

Table 3.1: A learning task for the network

x^1	x^2	...	x^i	...	x^N
t^1	t^2	...	t^k	...	t^N

with inputs $x_i \in \mathbb{R}^n$ and targets $t_i \in \mathbb{R}^m$ for $i = 1, 2, ..., n$. A vector x^i for the inputs has the following components: $(x_1^i, x_2^i, ...x_k^i, ...x_n^i)^T$. The objective of learning is to achieve minimum error on the training samples. This error is the difference between the desired output t_k and the actual output z_k. It is termed the training error and is defined as

$$E = \frac{1}{2}\sum_{k=1}^{m}(t_k - z_k)^2 = \frac{1}{2} \parallel t - z \parallel^2, \tag{3.12}$$

where t and z are target and network output vector of length m. E is the error function [16].

Gradient descent or steepest descent [3] is a training algorithm in which the weights of the network are updated to yield minimum error on the observed samples. At the initial stage, the weight vector $w^{(0)}$ is chosen randomly and is updated repeatedly as the error decreases in the direction in which the error is greatly reduced. Therefore, it is evaluated at $w^{(\tau)}$:

$$\Delta w^{(\tau)} = -\eta \nabla E \mid_{w^{(\tau)}} \tag{3.13}$$

where η is called the learning rate and determines the extent in the change of weights. τ represents a step of the weight update and ∇E represents the gradient or the change in error [3]. Gradient descent has two approaches: batch version and pattern-based methods. In batch version, the updates are stored and accumulated and the change of parameters becomes effective only after a whole pass through the learning task is done. The weight update is done

according to (3.13). The Pattern-based method is the one in which the weight update is made after presentation of each sample. The weight update for this approach is given as

$$\Delta w^{(\tau)} = -\eta \nabla E^n \mid_{w^{(\tau)}} \tag{3.14}$$

where n is the position of the sample that is being presented. For example, $\Delta w^{(\tau)} = -\eta \nabla E^{(6)} \mid_{w^{(\tau)}}$ is the weight update when the 6^{th} sample is presented. For both approaches, an appropriate choice of the learning rate needs to be made. There are a few problems associated with descent strategies and some of the examples are [3]:

- Local minima in the error function surface.
- Insufficient search direction and a need to choose a suitable value of the learning rate parameter η.
- For a step size that is too high, the gradient moves around the minimum but does not converge to the minimum.
- Slow convergence in case of small step size.

3.4.2.2 Backpropagation for NN with one hidden layer

The derivations of backpropagation for Neural Network with one hidden layer are based on work by [16].

For the learning task in Table 3.1, the weights are initialised in a random manner and changed in the direction that reduces error. The change in weights is given as

$$\Delta w = -\eta \frac{\partial E}{\partial w}, \tag{3.15}$$

where η is the learning rate that indicates the relative size of change in weights. Consider Figure 3.8, a network with three layers:

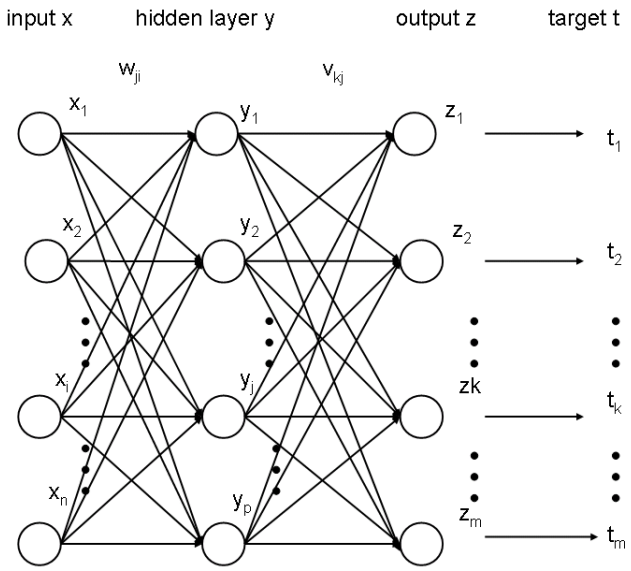

Figure 3.8: A network with three layers

Figure 3.8 shows the input layer, hidden layer and output layer. From this graph, consider the hidden neuron j which is given in Figure 3.9.

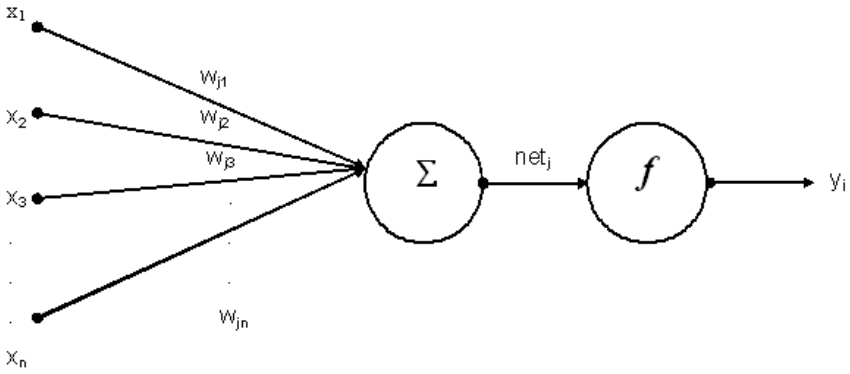

Figure 3.9: Hidden neuron j

From Figure 3.8 and 3.9 the following network components are defined:

w_{j0}: bias to neuron j.

v_{k0}: bias to neuron k.

y_j: the output obtained after activation function has been applied to net_j at neuron j.

z_m: the output obtained after activation function has been applied to net_k at neuron k.

net_j: weighted sum of the inputs plus a bias at neuron j.

net_k: weighted sum of inputs plus a bias at neuron k.

The output result for neuron j from the input units $x^i = (x_1, x_2, ..., x_n)$ is given as:

$$\begin{aligned} net_j &= \sum_{i=1}^{n} w_{ji} x_i + w_{j0} \\ &= \sum_{i=0}^{n} w_{ji} x_i \\ &= \langle w^j, x \rangle \end{aligned} \tag{3.16}$$

where subscript i denote an index for the input layer, j is an index for the unit of the hidden layer and n is the number of input units. w_{ji} refers to the input-hidden weights at the hidden unit j. The output of neuron j obtained by applying the activation function f on the net output net_j is obtained here as:

$$y_j = f(net_j). \tag{3.17}$$

Similarly, consider output neuron k in Figure 3.10.

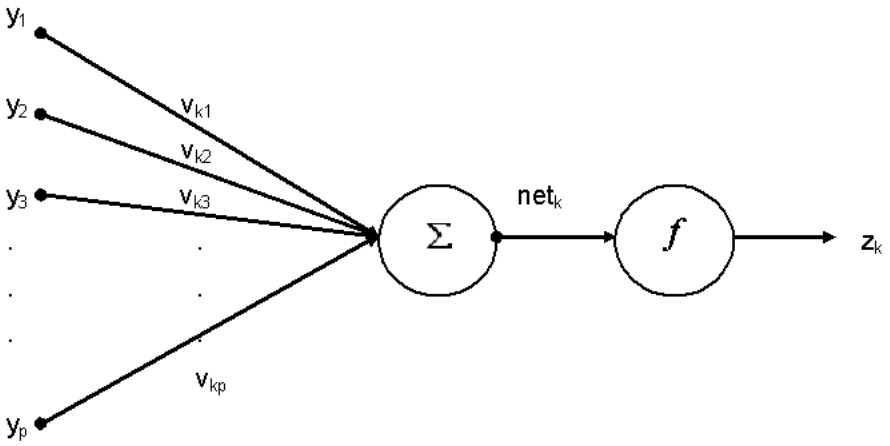

Figure 3.10: Output neuron k

For the hidden units, the outputs to the neuron k of the output layer are derived from the following:

$$\begin{aligned} net_k &= \sum_{j=1}^{p} v_{kj} y_j + v_{k0} \\ &= \sum_{j=0}^{p} v_{kj} y_j \\ &= \left\langle v^k, y \right\rangle \end{aligned} \tag{3.18}$$

where subscript k denotes the index unit of the output layer and p is the number of the hidden units.

$$z_k = f(net_k). \tag{3.19}$$

In both cases, the bias unit has values $x_0 = 1$ and $y_0 = 1$. Suppose that we want to determine the weights for the hidden-output layer v_{kj}. Chain rule differentiation is used.

$$\frac{\partial E}{\partial net_k} = \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial v_{kj}} = -\delta_k \frac{\partial net_k}{\partial v_{kj}} \tag{3.20}$$

and δ_k is defined to be:

$$\begin{aligned} \delta_k &= -\frac{\partial E}{\partial net_k} \\ &= -\frac{\partial E}{\partial net_k} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k) \end{aligned} \tag{3.21}$$

and

$$\frac{\delta net_k}{\delta v_{kj}} = y_j. \tag{3.22}$$

These results give a weight update for the hidden-output weights:

$$\Delta v_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j. \tag{3.23}$$

To obtain the update for the weights in the input-hidden layer, we also apply the chain rule and calculate:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}. \tag{3.24}$$

From (3.17), it follows that

$$\frac{\partial y_j}{\partial net_j} = f'(net_j) \tag{3.25}$$

and also from (3.16) that

$$\frac{\partial net_j}{\partial w_{ji}} = x_i. \tag{3.26}$$

To determine $\frac{\partial E}{\partial y_j}$, we write the error function E in the following way:

$$E = \frac{1}{2}\sum_{k=1}^{m}(t_k - z_k)^2 = \frac{1}{2}\left((t_1 - z_1)^2 + (t_2 - z_2)^2 + ... + (t_m - z_m)^2\right). \tag{3.27}$$

If we consider neuron l, the output z_l can be expressed as follows:

$$z_l = f(net_l) = f(v_{l1}y_1 + v_{l2}y_2 + .. + v_{lj}y_j ... + v_{lp}y_p) \qquad for\ l = 1, 2, ...m, \tag{3.28}$$

therefore,

$$\begin{aligned}
\frac{\partial E}{\partial y_j} &= \frac{\partial E}{\partial z_1}\frac{\partial z_1}{\partial net_1}\frac{\partial net_1}{\partial y_j} + \frac{\partial E}{\partial z_2}\frac{\partial z_2}{\partial net_2}\frac{\partial net_2}{\partial y_j} + ... + \frac{\partial E}{\partial z_m}\frac{\partial z_m}{\partial net_1}\frac{\partial net_m}{\partial y_j} \\
&= (t_1 - z_1)f'(net_1)v_{1j} + (t_2 - z_2)f'(net_2)v_{2j} + ... + (t_m - z_m)f'(net_m)v_{mj} \\
&= \sum_{k=1}^{m}(t_k - z_k)f'(net_k)v_{kj}.
\end{aligned} \tag{3.29}$$

If we define

$$\delta_j \equiv f'(net_j)\sum_{k=1}^{m} v_{kj}\delta_k, \tag{3.30}$$

then the learning rule for the weight update for the input-hidden layers is given as

$$\Delta w_{ji} = \eta x_i \delta_j = \eta \underbrace{\sum_{k=1}^{m} v_{kj}\delta_k f'(net_j)}_{\delta_j} x_i. \tag{3.31}$$

3.4.2.3 Resilient Backpropagation

The training algorithm used in this thesis for Neural Networks is Resilient Backpropagation [40]. The algorithm has two passes through the network; the forward and backward pass. For the forward pass, during training, a sample is presented to the network as input. For each layer, the output from the previous layer is used as an input to the next hidden layer until the output layer is reached and the output is produced. The output response is then compared to the known target output. Based on the value of the error, the connection weights are adjusted. In the backward pass weights are adapted to ensure that the minimum error between the targets and the actual outputs is achieved [20].

The algorithm depends on the initial update values and the step size that has a default setting. The update values determine the size of the weight steps while the maximum step size has to be provided to avoid weights that are too large for the network. The approach provides faster convergence to the minimum because of the few parameters required to obtain optimal convergence times [40].

For the Resilient Backpropagation algorithm, the size of the weight change is determined by the update-value $\triangle_{ij}^{(t)}$ given as,

$$\triangle w_{ij}^{(t)} = \begin{cases} -\triangle_{ij}^{(t)}, & if\ \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ +\triangle_{ij}^{(t)}, & if\ \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ 0, & else \end{cases}, \tag{3.32}$$

where $\frac{\partial E}{\partial w_{ij}}^{(t)}$denotes the summed gradient information over all patterns of the training set. The new update-values $\triangle_{ij}(t)$ are determined by

$$\triangle_{ij}^{(t)} = \begin{cases} \eta^{+} * \triangle_{ij}^{(t-1)}, & if\ \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^{-} * \triangle_{ij}^{(t-1)}, & if\ \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \triangle_{ij}^{(t-1)}, & else \end{cases}, \tag{3.33}$$

where $0 < \eta^{-} < 1 < \eta^{+}$.

The decrease and increase factors, i.e. $\eta^{-}$and η^{+} respectively, are fixed. In this way, search in

parameter space is made easier because of few freely adjustable parameters. The factors are set to $\eta^- = 0.5$ while $\eta^+ = 1.2$. The initial update values $\triangle_{ij}$ are set to $\triangle_0 = 0.1$. The maximum weight step, determined by the size of the update value is limited. This is important since it prevents the weights from becoming too large. To avoid this situation, the upper bound defined by $\triangle_{max}$ is set to 50.0 and the minimum step size is fixed to $\triangle_{min} = 1e^{-6}$.

ALGORITHM 2.1.*Resilient Backpropagation algorithm* [40]

For this algorithm, the minimum (maximum) gives the minimum (maximum) of two numbers. The sign operator returns $+1$ if the argument is positive, -1 if the argument is negative and 0 otherwise.

$\forall i, j : \triangle_{ij}(t) = \triangle_0$

$\forall i, j : \frac{\partial E}{\partial w_{ij}}(t-1) = 0$

Repeat

Compute Gradient $\frac{\partial E}{\partial w}(t)$

For all weights and biases{

if $\left(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) > 0\right)$then {

$\triangle_{ij}(t) = \text{minimum}\,(\triangle_{ij}(t-1) * \eta^+, \triangle_{max})$

$\triangle w_{ij}(t) = -\text{sign}\left(\frac{\partial E}{\partial w_{ij}}(t)\right) * \triangle_{ij}(t)$

$\triangle_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t)$

$\frac{\partial E}{\partial w_{ij}}(t-1) = \frac{\partial E}{\partial w_{ij}}(t)$

}

else if $\left(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) < 0\right)$ then {

$\triangle(t) = \text{maximum}\,(\triangle_{ij}(t-1) * \eta^-, \triangle_{min})$

$\frac{\partial E}{\partial w_{ij}}(t-1) = 0$

}

else if $\left(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) = 0\right)$ then {

$\triangle w_{ij}(t) = -\text{sign}\left(\frac{\partial E}{\partial w_{ij}}(t)\right) * \triangle_{ij}(t)$

$\triangle_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t)$

$\frac{\partial E}{\partial w_{ij}}(t-1) = \frac{\partial E}{\partial w_{ij}}(t)$

}

}

Until (converged)

3.4.3 Early stopping approach

A problem of learning with NN is overtraining. The reason is that NN adapts too strongly to the training set and predicts unseen samples poorly. One approach that can be used to achieve better performance on unseen data is early stopping. In this technique, the available data is divided into three subsets: training set, validation set and the test set. The training set is used to compute and update the parameters of the network. During training, the training error decreases as the number of iterations increases. The error on the validation set is also monitored. It decreases as well up to a certain number of iterations but then will increase with further training. Training will stop after a certain specified number of iterations i.e. 500 epochs. And the weights and biases that occur at the minimum of validation error are the parameters of the network. The error measures on training set and validation set are shown in Figure 3.11 .

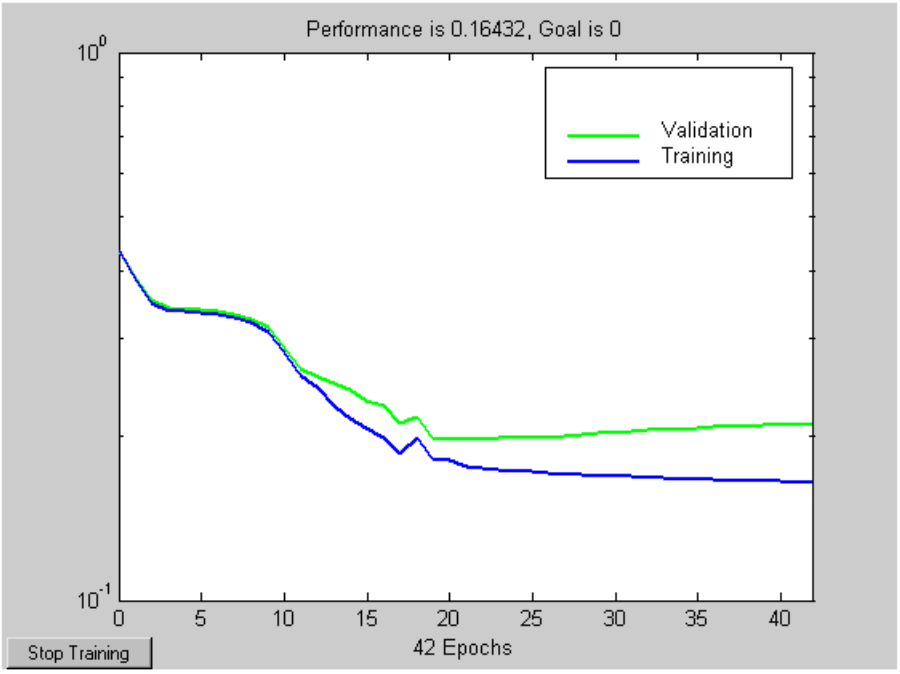

Figure 3.11: Training and validation set

From Figure 3.11, the error on the training set and validation set decreased until at epoch 19. After 19 epochs, the validation error started to increase and training stopped after 42 epochs. Up to that point, the validation error was increasing. The parameters of the network in this case will be those that occur at 19 epochs where the validation error is at minimum.

To obtain a good estimate of the accuracy of the classifier, we use a separate data, the test set. This should be independent from data used for training and validation [43].

Chapter 4

Support Vector Machines

4.1 Introduction

Chapter 4 gives an overview of Support Vector Machines (SVM). Section 4.2 introduces linearly separable two class problems. The optimal separating hyperplane for such problems is considered. Section 4.3 deals with techniques to handle linearly inseparable two class problems. Section 4.4 discusses non-linear Support Vector Machines. Section 4.5 states the universal approximation property of Support Vector Machines.

4.2 Classification of Linearly Separable Two-Class Problems

4.2.1 Overview

Support Vector Machines are machine-learning algorithms that were developed by Vapnik and co-workers [15]. They constructed a separating hyperplane with the objective of separating the data into different classes. The separating hyperplane should have maximum distance to the nearest data point in each class.

4.2.2 Primal problem for linearly separable data

This section considers support vector classification for linearly separable data. The basic ideas in implementing the algorithm for linearly separable class problems are from [18]. Suppose we

have the following set of training vectors:

$$D = \{(x_1, y_1), ...(x_l, y_l)\}, \quad x_i \in \mathbb{R}^n, \quad y_i \in \{-1, 1\} \tag{4.1}$$

$y_i \in \{-1, 1\}$ implies that there are two classes: positive sample points (belonging to class 1) and negative sample points (belonging to class 2). If we assume that there exists a hyperplane $\mathcal{H}$ with normal vector w and bias b satisfying the relationship

$$\langle w, x\rangle + b = 0. \tag{4.2}$$

For our purpose, linearly separable means that we can find a pair (w, b) such that [7] :

$$\begin{aligned} &\langle w, x_i\rangle + b \geq 1, \ \forall x_i \in \text{class1} \\ &\langle w, x_i\rangle + b \leq -1, \ \forall x_i \in \text{class2}. \end{aligned} \tag{4.3}$$

The distance $d(w, b; x)$ of a point x from a hyperplane (w, b) is [18]:

$$d(w, b; x) = \frac{|\langle w, x_i\rangle + b|}{\|w\|} \tag{4.4}$$

where w is the vector orthogonal to the hyperplane and b controls the distance to the origin [13] and the Euclidean length $\|w\|$ is defined as $\|w\| = \langle w, w\rangle$. Combining the set of inequalities in (4.3) results in the inequality

$$y_i \left[\langle w, x_i\rangle + b\right] \geq 1, \quad i = 1, 2, ...l. \tag{4.5}$$

Therefore, a separating hyperplane must satisfy this constraint. The maximum distance of a point to the hyperplane is called the margin. It is maximised subject to the same conditions as in (4.5). The margin, denoted by M, is obtained by [18]:

$$
\begin{aligned}
M(w,b) &= \min_{x_i:y_i=-1} d(w,b:x_i) \; + \; \min_{x_i:y_i=1} d(w,b;x_i) \\
&= \min_{x_i:y_i=-1} \frac{|\langle w,x_i\rangle + b|}{\|w\|} + \min_{x_i:y_i=1} \frac{|\langle w,x_i\rangle + b|}{\|w\|} \\
&= \frac{1}{\|w\|}\left(\min_{x_i:y_i=-1} |\langle w,x_i\rangle + b| + \min_{x_i:y_i=1} |\langle w,x_i\rangle + b|\right) \qquad (4.6) \\
&= \frac{2}{\|w\|}.
\end{aligned}
$$

Figure 4.1 shows linearly separable two-class problems with different margins. In Figure 4.1 (a), the distance between two classes of data is small and therefore small margin. However, in Figure 4.1 (b), the distance between the two classes is larger and hence a large margin. A large margin is more preferred because with large margin, few possibilities to separate the data exist.

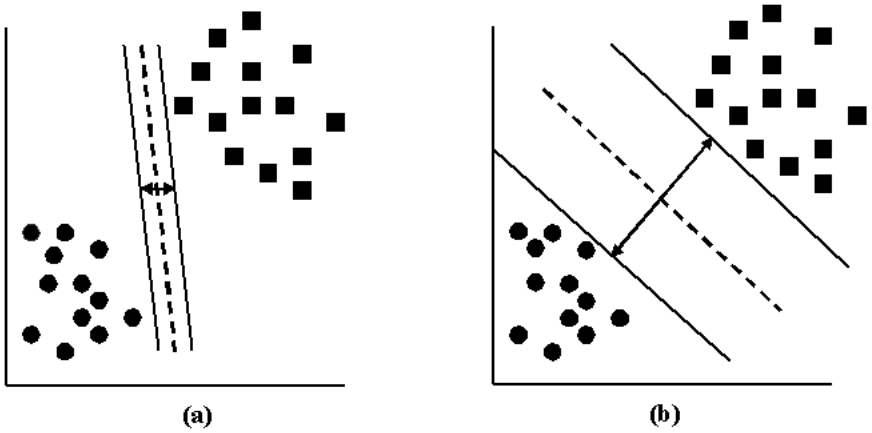

Figure 4.1: (a) Separating hyperplane with small margin (b) Separating hyperplane with large margin

The hyperplane that optimally separates the data is the one that minimises

$$\Phi(w) = \frac{1}{2} \|w\|^2 . \tag{4.7}$$

The result in (4.7) also comes from the fact that $\|w\| = \min$ if and only if $\frac{2}{\|w\|} = \max$. It is a Quadratic Programming (QP) problem. It finds (w, b) that minimises $\Phi(w) = \frac{1}{2} \|w\|^2$ under the constraint $y_i [\langle w, x_i \rangle + b] \geq 1$. This can be written as follows:

$$\begin{aligned} &\text{minimize } \Phi(w) = \frac{1}{2} \|w\|^2 \\ &\text{subject to } y_i [\langle w, x_i \rangle + b] \geq 1, \quad \forall i = 1, 2, ..., l. \end{aligned} \tag{4.8}$$

The advantage in this QP problem is that the solution is unique [18].

As we have seen in this section, maximising the margin leads to the quadratic programming problem in (4.8). The next section discusses the dual form of this QP problem. Dual formulation of the problem is important in SVM for two reasons. The main reason is that the constraints are represented in Lagrange form, so it is much easier to handle. Secondly, in the formulation of the dual problems, training data will only appear in the form of dot products between vectors. This is important since it will help us to generalise to the non-linear cases [7].

4.2.3 Dual optimization problem for linearly separable data

The constraint optimization is dealt with by introducing the Lagrange multipliers:

$$\alpha_i \geq 0, \quad i = 1, 2, ...l \tag{4.9}$$

The constraints in (4.5) and the multipliers are both multiplied together and the result subtracted from the objective function of the primal problem to yield the Lagrangian,

$$\Phi(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{l} \alpha_i \left(y_i \left[\langle w, x_i \rangle + b \right] - 1 \right). \tag{4.10}$$

The Lagrangian has to be minimised with respect to w and b, and maximised with respect to $\alpha_i \geq 0$ because for the optimal solution $\alpha_i' s$ has to be as large as possible. The classical Lagrangian duality permits the transformation of the primal problems in (4.8) into a dual problem. This is known as the Wolfe Dual Problem [31].

The dual problem is given as:

$$\begin{gathered} \max_{\alpha} W(\alpha) = \max_{\alpha} \left(\min_{w,b} \Phi(w, b, \alpha) \right) \\ \text{subject to } y_i \left(w, x_i + b \right) - 1 \geq 0 \\ \text{and } \alpha_i \geq 0. \end{gathered} \tag{4.11}$$

The minimum with respect to w and b of the Lagrangian, Φ, is given by

$$\begin{aligned} \frac{\partial \Phi}{\partial b} = 0 \quad &\Rightarrow \quad \sum_{i=1}^{l} \alpha_i y_i = 0 \\ \frac{\partial \Phi}{\partial w} = 0 \quad &\Rightarrow \quad w = \sum_{i=1}^{l} \alpha_i y_i x_i. \end{aligned} \tag{4.12}$$

Substituting (4.12) into (4.10) gives the following dual problem [31]:

$$\begin{aligned} W_D(\alpha) &= \frac{1}{2}\left(\left\{\sum_{i=1}^{l}\alpha_i y_i x_i\right\}.\left\{\sum_{j=1}^{l}\alpha_j y_j x_j\right\}\right) - \sum_{i=1}^{l}\alpha_i\left\{y^i\left(\left(\sum_{j=1}^{l}\alpha_j y_j x_j\right), x_i + b\right) - 1\right\} \\ &= \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle - \sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle - b\sum_{i=1}^{l}\alpha_i y_i + \sum_{i=1}^{l}\alpha_i \\ &= \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle . \end{aligned} \tag{4.13}$$

Hence the dual problem is simplified into the following:

$$\begin{aligned} &\text{maximise } \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle \\ &\text{subject to } \alpha_i \geq 0,\ i = 1, ...l \\ &\text{and } \sum_{i=1}^{l}\alpha_i y_i = 0. \end{aligned} \tag{4.14}$$

Therefore, it is important to note that the Wolf Dual Problem is equivalent to the original optimization in that the α_i obtained in both cases are identical [31]. Moreover, the solution to our dual problem under the constraints in (4.14) occurs at the same values w and b as our primal problem but under the constraints in (4.5) ; and the objective function is the same. For α_i^* that maximises $W_D(\alpha)$ subject to the constraints in (4.14), the parameters of the optimal separating hyperplane are given by:

$$w^* = \sum_{i=1}^{l}\alpha_i^* y_i x_i. \tag{4.15}$$

To obtain the bias b^*, Karush Kuhn Tucker (KKT) complementary conditions are used [10]. The KKT condition that holds at the solution of the primal problem are:

$$\alpha_i \{y_i (\langle w, x_i \rangle + b) - 1\} = 0 \quad i = 1, 2, ..., l. \tag{4.16}$$

The $x_i's$ that lie on the margin of the hyperplane where $\alpha_i \neq 0$ in (4.16) are called Support Vectors. Note that these occur where (4.5) satisfies the equality sign. Support Vectors are very important during training because they are samples for which, if the rest of the data are removed, will still define the same decision boundary [18]. Figure 4.2 shows the separating hyperplanes for a two class problem. Support vectors are indicated by circles. They lie on the hyperplanes H_1 and H_2.

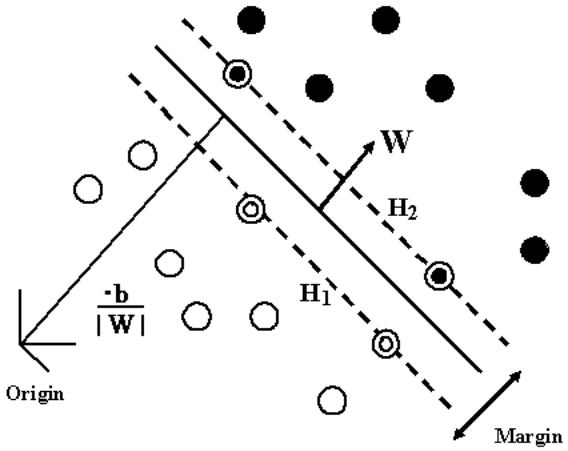

Figure 4.2: Linearly separable data with Support Vectors (circled).

Therefore, for a support vector x_j, the bias b^* [31] is obtained from:

$$b^* = y_i - \langle w^*, x_j \rangle . \tag{4.17}$$

For linearly separable two-class problems, the hyperplane is defined by a few sample points called Support Vectors. In practice, one takes the arithmetic mean on all support vectors. For real problems, data may not always be linearly separable. The next section discusses the approach to handling non-linear two-class problems.

4.3 Classification of Nonlinearly Separable Two-Class Problems

4.3.1 Linear inseparability

Not all two class problems are linearly separable. The algorithm for linearly separable data will not yield a separating hyperplane classifying the samples without error where data are linearly inseparable. An example of an inseparable two-class problem is shown in Figure 4.3 and also in Figure 3.6 for the XOR problem.

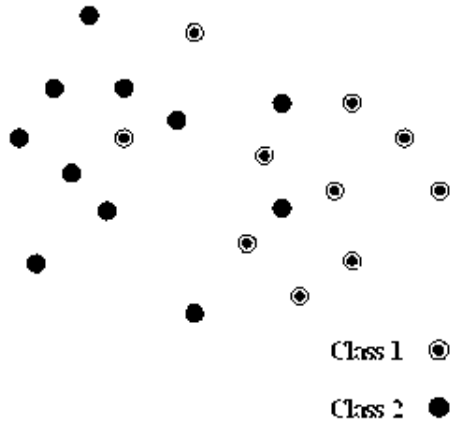

Figure 4.3: Linearly inseparable two-class problems

The objective is to find a separating hyperplane that allows some error of misclassification the data. An approach is to introduce "slack variables" to allow for the violation of equations (4.5) and (4.10). The slacks are:

$$\xi_i \geq 0 \quad i = 1, ..., l. \tag{4.18}$$

In order to relax the constraints, we modify them to:

$$y_i \left[\langle w, x^i \rangle + b \right] \geq 1 - \xi_i \;\; i, = 1, ..., l \tag{4.19}$$

where $\xi_i \geq 0$. The "slack variables" are interpreted as follows (see Figure 4.4):

- if $\xi_i = 0$, then the sample x_i is on the correct side of the hyperplane and outside the maximal margin. Such a sample does not influence the location of the separating hyperplane.
- if $0 \leq \xi_i \leq 1$, then the sample x_i is still on the correct side of the hyperplane, but it falls inside the margin. Such a sample will be classified correctly, though it may be difficult to do so, (Figure 4.4(a)).
- if $1 \leq \xi_i$ then $C(1 - \xi_i) \leq 0$, i.e. sample x_i falls on the wrong side of the hyperplane. Such a sample will be misclassified (Figure 4.4(b)).

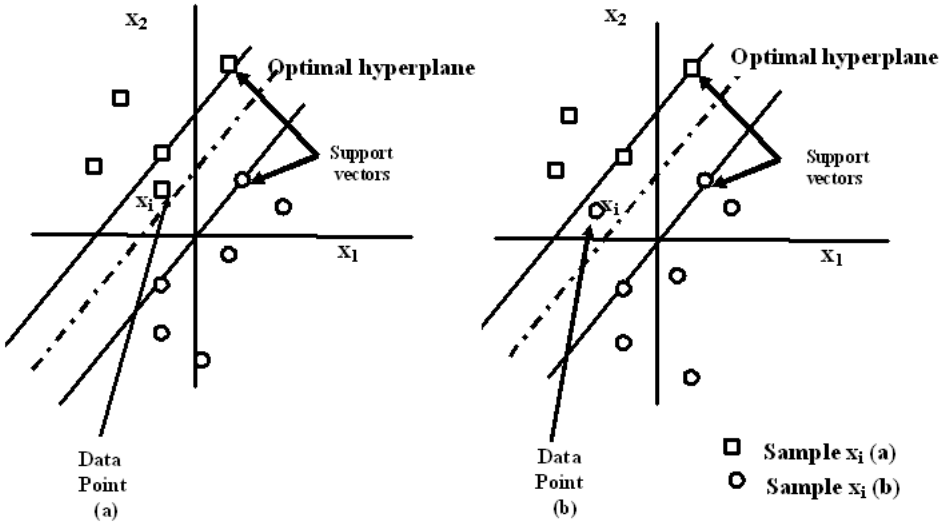

Figure 4.4: The effect of "slack variables" for linearly inseparable two-class problems

4.3.2 Primal problem for linearly inseparable data

The generalised optimal separating hyperplane is determined by $w, b,$ and ξ_i for $i = 1, 2, ..., l$ which maximises the function

$$\begin{gathered}\Phi(w, b, \xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i \\ \text{subject to } y_i\left[\langle w, x_i\rangle + b\right] \geq 1 - \xi_i \\ \text{and } \xi_i \geq 0 \; i, = 1, ..., l. \end{gathered} \tag{4.20}$$

The degree to which violations of the original constraints should be penalised is determined by the parameter C. This parameter represents a trade off between low errors and the desirability of a large maximum margin of the hyperplane [28]. The solution to the optimization problem in (4.20) occurs at the saddle point of the Lagrangian,

$$\Phi(w, b, \alpha, \xi, \beta) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i\left(y_i\left[\langle w, x_i\rangle + b\right] - 1 + \xi_i\right) - \sum_{i=1}^{l}\beta_i\xi_i \tag{4.21}$$

where β_i and α_i are the Lagrange multipliers for the problem [18].

4.3.3 The dual problem for linearly inseparable data

The Lagrangian has to be minimised with respect to w, b and maximised with respect to α, β. The transformed dual problem is given in the form:

$$\max_{\alpha} W_D(\alpha, \beta) = \max_{\alpha,\beta}\left(\min_{w,b,\xi}\Phi(w, b, \alpha, \xi, \beta)\right). \tag{4.22}$$

The minimum with respect to w, b and ξ of the Lagrangian Φ occurs at:

$$\begin{aligned}
\frac{\partial \Phi}{\partial b} = 0 \quad &\Rightarrow \quad \sum_{i=1}^{l} \alpha_i y_i = 0 \\
\frac{\partial \Phi}{\partial w} = 0 \quad &\Rightarrow \quad w = \sum_{i=1}^{l} \alpha_i y_i x_i \\
\frac{\partial \Phi}{\partial \xi} = 0 \quad &\Rightarrow \quad \alpha_i + \beta_i = C.
\end{aligned} \tag{4.23}$$

Substitution of equations (4.12), (4.23) into (4.21), leads to a dual problem that is obtained in the following way:

$$\begin{aligned}
W_D(\alpha, \beta) &= \frac{1}{2}\left(\left\{\sum_{i=1}^{l} \alpha_i y_i x_i\right\} . \left\{\sum_{j=1}^{l} \alpha_j y_j x_j\right\}\right) + (\alpha_i + \beta_i) \sum_{i=1}^{l} \xi_i \\
&\quad - \sum_{i=1}^{l} \alpha_i \left\{ y^i \left(\left(\sum_{j=1}^{l} \alpha_j y_j x_j \right), x_i + b \right) - 1 + \xi_i \right\} - \sum_{j=1}^{l} \beta_i \xi_i \\
&= \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle - \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + \sum_{i=1}^{l} \alpha_i \xi_i + \sum_{i=1}^{l} \beta_i \xi_i \\
&\quad - b \sum_{i=1}^{l} \alpha_i y_i + \sum_{i=1}^{l} \alpha_i - \sum_{i=1}^{l} \alpha_i \xi_i - \sum_{i=1}^{l} \beta_i \xi_i \\
&= -\frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + \sum_{i=1}^{l} \alpha_i.
\end{aligned} \tag{4.24}$$

The dual problem is therefore defined as:

$$\begin{gathered}
W_D(\alpha, \beta) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\
\text{subject to } 0 \leq \alpha_i \leq C \quad i = 1, 2, ..., l \\
\text{and } \sum_{j=1}^{l} \alpha_j y_j = 0.
\end{gathered} \tag{4.25}$$

The constraints for the linearly inseparable case are still the same as in the separable case i.e. $\alpha_i \geq 0$. The dual problem is constrained by $0 \leq \alpha_i \leq C$ because we require that $\alpha_i \leq C$ from (4.23). Introducing slack variables ξ_i limits the size of Lagrange multipliers because of the upper bound C. It does not also guarantee linear separability. The section that follows describes techniques to handle non-linear classification problems.

4.4 Non-linear Support Vector Machines: Feature Maps and Kernels

In the case of nonlinearly separable two class problems, slack variables were introduced in the objective function. However, for data that are too "noisy", separability may not be achieved. An approach to overcome the problem can be traced back to the work of Vapnik and Cortes [15]. In their approach, data are mapped into higher dimensional space where it is hoped that data they can be linearly separated.

4.4.1 Feature Space

Suppose we have the input vector $x = (x_1, ..., x_n)'$. A mapping of the input vector into a feature space $\mathcal{F}$, is performed through

$$\varphi(x) = \varphi(x_1, x_2, ..., x_n), \tag{4.26}$$

where x denotes training data in input space. These mappings can also be defined as $\varphi : \mathbb{R}^n \rightarrow \mathcal{F}$ where $\mathcal{F}$ is an inner product space. An example of mapping an input vector into higher dimensional space is shown in Figure 4.5. It shows a feature mapping from a two dimensional input space to a two dimensional feature space. In this case, the linear separation is not achieved in the input space but is in the feature space.

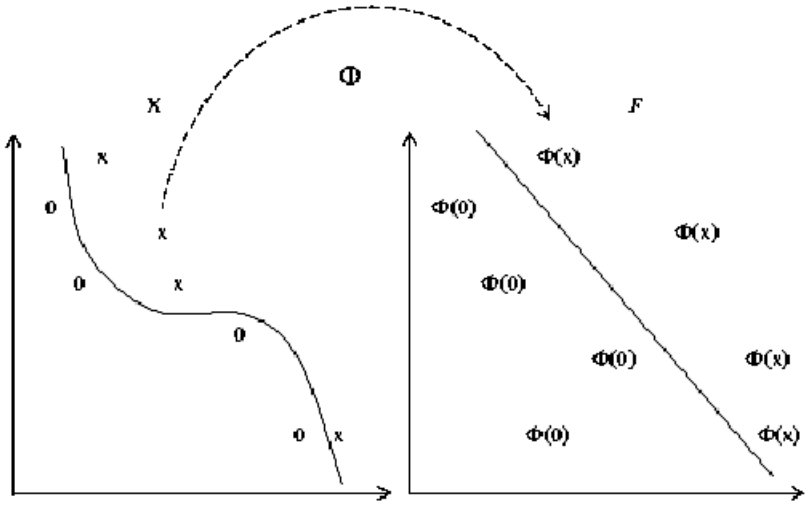

Figure 4.5: Feature mapping in classification

In two-class problems, with the input vector x, the objective is to determine to which of the two classes each sample point belongs. Therefore, after transforming the vector to a feature (separating) space i.e. $x \mapsto \varphi(x)$, classification will depend on the sign of the function

$$f(x) = \langle w, \varphi(x) \rangle + b. \tag{4.27}$$

As has already been established from (4.12), the weight vector w has a new representation of the form

$$w = \sum_{i=1}^{l} \alpha_i y_i \varphi(x_i). \tag{4.28}$$

It is important to note that the weight vector is now presented in the new space where the x_i have been transformed into $\varphi(x_i)$. For f to be classified,

$$\begin{aligned} f(x) &= \langle w, \varphi(x) \rangle + b \\ &= \sum_{i=1}^{l} \alpha_i y_i \langle \varphi(x_i), \varphi(x_j) \rangle + b. \end{aligned} \tag{4.29}$$

It is also important to note that for the solution of the dual problem in feature space, only inner products occur in the dot products of the feature points [7]. An example where this occurs is in (4.13). Therefore, the dual problem becomes:

$$\begin{gathered} W_D(\alpha, \beta) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1} \alpha_i \alpha_j y_i y_j \langle \varphi(x_i), \varphi(x_j) \rangle \\ \text{subject to } 0 \leq \alpha_i \leq C \quad i = 1, 2, ..., l \\ \text{and } \sum_{j=1}^{l} \alpha_j y_j = 0. \end{gathered} \tag{4.30}$$

4.4.2 Kernels

It has been established that in non-linear separable class problems, a feature mapping to higher dimensional space may help to achieve linearity, which might not otherwise have been possible in the input space. However, for this to happen, the input vector has to be transformed. And since the input arguments$\langle x_i, x_j \rangle$ can be expressed in a form of inner products $\langle \varphi(x_i), \varphi(x_j) \rangle$, it follows that direct computations in the feature space can be performed without the mapping φ. A function that performs such computations is called a Kernel [49]. A kernel is a function K, such that:

$$K(x_i, x_j) = \langle \varphi(x_i), \varphi(x_j) \rangle \tag{4.31}$$

where φ is a mapping from an input space to feature space $\mathcal{F}$ and is of the form

$$\varphi : \begin{cases} R^n & \longrightarrow \mathcal{F} \\ x & \mapsto \varphi(x) \end{cases} . \tag{4.32}$$

A feature map can be performed in two steps. It involves a non-linear transformation of data into the feature space and then performs linear classification in the feature space [14]. Kernels can be used to map data into a feature space. Training can take place in that space and does not require knowledge of the feature map [15]. The relationship in (4.31) that exists between kernels and inner products is important because it implies that knowledge of the feature map φ is irrelevant in performing a mapping to the feature space. Since the inner products appear in (4.30), applying a kernel function gives rise to the following QP [18]:

$$\begin{gathered} \max W(\alpha_i) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j K\left(x_i, x_j\right) \\ \text{subject to } \alpha_i \geq 0,\ i = 1, ..., l \\ \text{and } \sum_{i=1}^{l} \alpha_i y_i = 0. \end{gathered} \tag{4.33}$$

For the optimal hyperplane, weights are obtained by solving:

$$w^* = \sum_{i=1}^{l} y_i \alpha_i^* \varphi(x_i) \tag{4.34}$$

and the bias for non-zero α_i is obtained from [31]:

$$\begin{aligned} b^* &= y_i - \langle w^*, \varphi(x_i) \rangle \\ &= y_i - \left\langle \left(\sum_{j=1}^{l} \alpha_j^* y_j \varphi(x_j) \right), \varphi(x_i) \right\rangle \\ &= y_i - \sum_{j=1}^{l} \alpha_j^* y_j K\left(x_i, x_j\right) \end{aligned} \tag{4.35}$$

for some i with $0 < \alpha_i < C$.

4.4.3 Examples of kernels

The following are the commonly used kernels [7]:

- Polynomial $K(x_i, x_j) = (\langle x_i, x_j \rangle + c)^d$.
- Radial Basis Function $K(x_i, x_j) = e^{\frac{-\|x_i - x_j\|^2}{2\sigma^2}}$.
- Sigmoid $K(x_i, x_j) = tanh(\kappa \langle x_i, x_j \rangle + \Theta)$.

d, σ, κ and Θ are parameters of the kernels.

4.5 Universal Approximation Property

It has been determined in the previous sections that SVM can create a separating hyperplane with maximum margin to ensure good generalisation. From this concept a Support Vector Machine algorithm was developed to handle linearly separable classification problems. For non-linear separable classification problems, data can be presented in feature space by the use of kernels and in this space it is hoped that linearity will be achieved. According to [19], this has led to an important universal property of SVMs which states that:

THEOREM 3.1. *Universal Approximation Property*

For any (measurable) function $F : \mathbb{R}^n \mapsto \{-1, 1\}$ there exists a Support Vector Machine (with Gaussian kernel) which can approximate the function to any prescribed precision in probability.

This means that Support Vector Machine with Gaussian kernel is sufficient to approximate any function to any desired accuracy.

Chapter 5

Estimating the Accuracy of a Classifier

5.1 Introduction

This chapter discusses methods to estimate the accuracy of a classifier. A detailed discussion of the probability of a correct classification is given in Section 5.2. Test Set Approach and Estimates for Overall Accuracy and Standard Error are given in detail in Section 5.3. A review of Cross-validation as accuracy estimation method is given in Section 5.4. For these sections, we derive the estimate for the probability of making a correct classification. These derivations are largely taken from Chapter 11 in [6] and [30].

5.2 Probability of Correct Classification: Test Set Approach

Suppose we have the following learning task:

x^1	x^2	...	x^N
t_1	t_2	...	t_N

with $x^i \in \mathcal{F} \subset R^n$ and $t^i \in C = \{w_1, w_2, ..., w_c\}$. $\mathcal{F}$ is the feature space and C is a set of class labels for the inputs $x^1, x^2, ..., x^N$. Also assume we have a probability distribution on $\mathcal{F} \times C$. For any $A \subset \mathcal{F}$ and any $w_i \in C$, then the probability

$$P(A, w) = P\left(x \in A, w = w_i\right) \tag{5.1}$$

is known. If we have a classifier $\Phi : \mathcal{F} \mapsto C$, then we define its correct probability by

$$P(correct) = P\left(\Phi(x) = t\right) \tag{5.2}$$

for a randomly selected sample $(x, t) \in \mathcal{F} \times C$. By correct probability, we refer to the probability of getting a correct classification. More precisely,

$$P(correct) = P\left(\{(x, t) \in \mathcal{F} \times C \mid \Phi(x) = t\}\right). \tag{5.3}$$

In general, this correct probability can only be estimated. Define the random variable

$$Z((x, t)) = \begin{cases} 1 & if\ \Phi(x) = t \\ 0 & else \end{cases} \tag{5.4}$$

The expected value of Z becomes

$$E(Z) = 1 \cdot P(\Phi(x) = t) + 0 \cdot P(\Phi(x) \neq t) = P(\Phi(x) = t) = P(correct). \tag{5.5}$$

Hence we can use the sample mean of a random variable $Z_1, Z_2, ..Z_N$ with $Z_i = Z(x^i, t_i)$ as an estimator for $P(correct)$:

$$\hat{P} = \hat{P}(correct) = \bar{Z} = \frac{1}{N}\sum_{i=1}^{N} Z((x^i, t_i)) = \text{proportion of correct classifications, in the data.} \tag{5.6}$$

So we randomly sample N instances from $\mathcal{F} \times C$ and take its arithmetic mean of the values $Z_i = Z((x^i, t_i))$. It is well known that this estimator is unbiased:

$$E(\hat{P}) = E\left(\frac{1}{N}\sum_{i=1}^{N} Z_i\right) = \frac{1}{N}\sum_{i=1}^{N} E(Z_i) = \frac{1}{N} N E(Z) = P(correct). \tag{5.7}$$

To determine the standard error of our estimator, consider each (x^i, t_i) as a Bernoulli trial where a success is a correct classification, i.e. $\Phi(x^i) = t_i$, and the success probability is $P = P(correct)$. Then

$$Y = \sum_{i=1}^{N} Z((x^i, t_i)) \tag{5.8}$$

is the number of successes and has therefore a binomial distribution, $Y \sim b(N, P(correct))$. Therefore, the standard error is

$$SE(\hat{P}) = \sigma_{\hat{p}} = \sqrt{var(\hat{P})} = \sqrt{var(\frac{1}{N}\sum_{i=1}^{N} Z((x^i, t_i))} = \sqrt{\frac{1}{N^2} var(Y)} = \sqrt{\frac{P(1-P)}{N}}. \tag{5.9}$$

If the sample is large enough, we can replace the unknown P by our estimate $\hat{P}$ in order to estimate this standard error. We can use this estimate of the standard error and obtain confidence intervals for the correct probability based on our point estimate $\hat{P}$: If the sample size, N, is large enough, by the Central Limit Theorem [52],

$$\frac{\hat{P} - P}{\sigma_{\hat{p}}} \sim N(0, 1). \tag{5.10}$$

Therefore, given the confidence level $1 - \alpha$, we determine $z_{\alpha/2}$ such that

$$P\left(-z_{\alpha/2} \leq \frac{\hat{P}-P}{\sigma_{\hat{p}}} \leq z_{\alpha/2}\right) = 1-\alpha \tag{5.11}$$

and solve for $P = P(correct)$:

$$P\left(\hat{P} - z_{\alpha/2}\sigma_{\hat{p}} \leq P(correct) \leq \hat{P} - z_{\alpha/2}\sigma_{\hat{p}}\right) = 1-\alpha. \tag{5.12}$$

Hence with a probability $1-\alpha$, the true probability is in the interval

$$[\hat{P} - z_{\alpha/2}\sigma_{\hat{p}}, \hat{P} + z_{\alpha/2}\sigma_{\hat{p}}]. \tag{5.13}$$

Note here that if we consider the sample variance, s^2, of the numbers $Z_i = Z((x^i, t_i)) \in \{0,1\}$, then

$$\begin{aligned} s^2 &= \frac{1}{N}\sum_{i=1}^{N}\left(Z_i - \frac{1}{N}\sum_{i=1}^{N}Z_i\right)^2 = \frac{1}{N}\sum_{i=1}^{N}Z_i^2 - \bar{Z}^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}Z_i - \bar{Z}^2 = \bar{Z}(1-\bar{Z}). \end{aligned} \tag{5.14}$$

Hence this sample variance equals

$$s^2 = \hat{P}(1-\hat{P}) \tag{5.15}$$

and the standard error of our estimator for the error probability can be written as

$$SE(\hat{P}) = \sqrt{\frac{s^2}{N}}. \tag{5.16}$$

Classifiers are learnt from data in a way that the overall accuracy of the data is maximised (or, which is the same, the error probability estimate is minimized). Hence, if we estimate $P(correct)$ by the overall accuracy on the training data, OAtrain, then this estimator is overly optimistic. For this reason, the test set or cross-validation is used to obtain overall accuracy, a good estimate for $P(correct)$. Let a learning task $\mathcal{L}$ be randomly sampled from $\mathcal{F}\times C$. The learning task is used to construct a classifier $\Phi_{\mathcal{L}} : \mathcal{F} \to \mathcal{C}$. The learning task is therefore called the training set as we train the classifier on that set. A second set T, called the test set, is used to estimate the $P(correct)$. Because the number of available data is restricted, getting an independent data set is often difficult because new samples cannot be sampled easily. Therefore, a natural approach would be to divide the data set D and set aside some of it as the test set and pretend that it is our unseen data. The split of the data set D into training set $\mathcal{L}$ and a test set $T = D \setminus \mathcal{L}$:

$$D = \mathcal{L} \cup (D \setminus \mathcal{L}) = \mathcal{L} \cup T. \tag{5.17}$$

A 2/3-1/3 split is often used or just 70% of all data as training data and 30%, the remaining data, as test set. Random selection of the training data $\mathcal{L}$ from the data set D is used to achieve independence of $\mathcal{L}$ and T. The problem that often occurs is that the proportions of classes in the data set may not be the same as the priors of the classes. This occurs when one class has a very small prior probability. If there are too few samples of the classes in our training set, the classifier will not learn this class well. In that case more samples of the class have to be used for training. Therefore, prior probabilities have to be given careful consideration when estimating the correct probability. We write:

$$P(correct) = \sum_{j=1}^{c} P(w_j) P\left(\Phi_{\mathcal{L}}(x) = w_j \mid w_j\right). \tag{5.18}$$

We estimate

$$P\left(\Phi_{\mathcal{L}}(x) = w_j \mid w_j\right) = \frac{N_{jj}}{N} \tag{5.19}$$

where N_{jj} is the number of all w_j test samples that are classified as w_j. Therefore the error matrix with these values is given by:

$$\begin{pmatrix} N_{11} & N_{12} & . & . & . & N_{1c} \\ N_{21} & N_{22} & & & & N_{2c} \\ . & & & & & \\ . & & & & & \\ . & & & & & \\ N_{c1} & & & & & N_{cc} \end{pmatrix} \tag{5.20}$$

where the diagonal elements represent numbers of correct classifications. If we substitute 5.19 in 5.18 we obtain the estimate

$$\hat{P}(correct) = \sum_{j=1}^{c} P(w_j) \frac{N_{jj}}{N_j}. \tag{5.21}$$

Note that if the priors are reflected as proportions of the class labels in the test set, i.e. $P(w_j) = \frac{N_j}{N}$, then $\hat{P}(correct)$ is the proportion of the correct classified samples. This means that given the priors, from (4.19), $\hat{P}(correct)$ is

$$\begin{aligned} \hat{P}(correct) &= \sum_{j=1}^{c} P(w_j) \frac{N_{jj}}{N_j} \\ &= \sum_{j=1}^{c} \frac{N_j}{N} \cdot \frac{N_{jj}}{N_j} \\ &= \frac{1}{N} \sum_{j=1}^{c} N_{jj} \end{aligned} \tag{5.22}$$

which is the proportion of correct samples. To show that our estimator is unbiased, define the random variable Z as before by

$$Z((x,t)) = \begin{cases} 1 & if\ \Phi_{\mathcal{L}}(x) = t \\ 0 & else \end{cases}. \tag{5.23}$$

Therefore,

$$\sum_{j=1}^{N} Z_j = N_{11} + N_{22} + ... + N_{cc} \tag{5.24}$$

and hence, $\hat{P}(correct) = \frac{\sum N_{jj}}{N} = OA$. OA refers to Overall Accuracy. Then

$$E(Z|w_j) = 1 \cdot P(Z = 1|w_j) + 0 \cdot P(Z = 0|w_j) = P(\Phi_{\mathcal{L}}(x) = w_j|w_j). \tag{5.25}$$

We then have

$$\begin{aligned} P(correct) &= \sum_{j=1}^{c} P(w_j) P\left(\Phi_{\mathcal{L}}(x) = w_j \mid w_j\right) \\ &= \sum_{j=1}^{c} P(w_j) E(Z|w_j). \end{aligned} \tag{5.26}$$

As ubiased estimator for $E(Z|w_j)$ we use the sample mean of the w_j samples:

$$\hat{P}_j = \frac{1}{N_j} \sum_{t_i = w_j} Z(x^i, t_i) = \frac{1}{N_j} \cdot N_{jj} \tag{5.27}$$

where N_{ij} is the number of correct classified samples w_j in the test set. Hence the estimator

$$\begin{aligned}\hat{P} &= \sum_{j=1}^{c} P(w_j)\frac{N_{jj}}{N_j} \\ &= \sum_{j=1}^{c} P(w_j)\hat{P}_j \end{aligned} \tag{5.28}$$

is unbiased for $P(correct)$.

A similar approach to the one in the previous section can be used to estimate the standard error much more simply. Define for each class w_j the random variable

$$Y_j = \sum_{t_i=w_j} Z((x^i, t_i)) \tag{5.29}$$

which counts the number of correct classifications of w_j samples. So again we have binomial random variables where success is a correct sample and the probability of success shall be denoted by $P_j = P(correct|w_j)$.We can then write

$$\hat{P}_j = \sum_{j=1}^{c} P(w_j)\frac{1}{N_j}Y_j. \tag{5.30}$$

We will assume that the random variables $Y_1, Y_2, ..., Y_c$ are independent (though this is not always true). With $var(Y_j) = N_j P_j(1 - P_j)$, the variance of our estimator becomes

$$var(\hat{P}) = \sum_{j=1}^{c}((P(w_j))^2\frac{1}{N_j^2}var(Y_j) = \sum_{j=1}^{c}((P(w_j))^2\frac{1}{N_j}P_j(1 - P_j). \tag{5.31}$$

If we estimate the "success probability" P_j by $\hat{P}_j = \frac{N_{jj}}{N_j}$, then we obtain the formula

$$var(\hat{P}) \approx \sum_{j=1}^{c}((P(w_j))^2\frac{1}{N_j}\hat{P}_j(1 - \hat{P}_j) = \sum_{j=1}^{c}((P(w_j))^2\frac{N_{jj}}{N_j^2}\left(1 - \frac{N_{jj}}{N_j}\right). \tag{5.32}$$

The standard error of our estimator thus becomes

$$SE(\hat{P}) = \sqrt{\sum_{j=1}^{c}((P(w_j))^2 \frac{1}{N_j}\hat{P}_j(1-\hat{P}_j)}. \tag{5.33}$$

Consider the sample variance s_j of the values $Z_i = Z((x^i, t_i))$ with $t_i = w_j$ (i.e. we consider w_j samples only). Then, again for $Z_i \in \{0, 1\}$,

$$s_j^2 = \sum_{t_i=w_j}(Z_i^j - \hat{P}_j)^2 = \frac{1}{N_j}\sum_{t_i=w_j} Z_i^2 - \hat{P}_j^2 = \frac{1}{N_j}\sum_{t_i=w_j} Z_i - \hat{P}_j^2 = \hat{P}_j(1-\hat{P}_j). \tag{5.34}$$

In other words: the variance of $\hat{P}_j$ is the sample variance s_j^2 of the values $Z_i = Z((x^i, t_i))$ with $t_i = w_j$. Therefore we can also write for the standard error of $\hat{P}$:

$$SE(\hat{P}) = \sqrt{\sum_{j=1}^{c}(P(w_j))^2 \frac{s_j^2}{N_j}}. \tag{5.35}$$

5.3 Probability of Correct Classification: Cross-validation Approach

The aim of our work is to compare the performance of Neural Networks and Support Vector Machines in predicting the secondary structure of proteins from their primary sequences. Model selection is important in our study because the best model to be chosen in each method is the one that performs well on unseen data. Estimating the accuracy of a classifier in this case is essential because it will assist us in choosing the best model from the other models for each method. An approach that is used in an attempt to choose the best model with better accuracy, is Cross-Validation [30].

In cross-validation, the data set D is randomly divided into K equally sized subsets (=folds): $D_1, D_2, ..., D_K$ such that

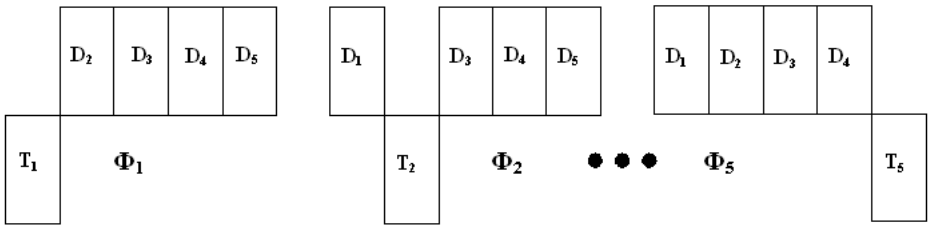

Figure 5.1: 5-fold Cross-validation

$$\bigcup_{k=1}^{k} D_k = D \text{ and } D_k \cap D_l = \emptyset \quad \text{for } k \neq l \tag{5.36}$$

The idea is to train K classifiers $\Phi_1, \Phi_2, ...\Phi_K$. The classifier Φ_K is trained on the set $\mathcal{L}_k = D \setminus D_k$ and its error rate is estimated on the test set $T_k = D_k$. Figure 5.1 shows an example of 5-fold cross-validation.

The upper parts in this graph are used for training the classifiers while the lower part is used as a test set for estimating the correct probability. The disjoint union of the test sets is the whole data set and therefore, each sample is used once as a test sample.

The basic assumption in cross-validation [30] is: If $\Phi : \mathcal{F} \mapsto \mathcal{C}$ is a classifier trained on the whole data set D and $\Phi_k : \mathcal{F} \mapsto \mathcal{C}$ are the classifiers trained on $\mathcal{L}_k = D \backslash D_k$, then

$$P(correct; \Phi) \approx P(correct; \Phi_k) \tag{5.37}$$

for all $k = 1, 2, ..., K$.

In practice different number of folds are often suggested e.g. K=10 or K=20. If K=size of the data set, cross-validation is also called the Leave-one-out method. Let us get an estimate of the correct probability of the classifier Φ based on our basic assumption. To this end, we denote

$N_{jj}^{(k)} =$ number of w_j samples in $T_k = D_k$ classified as w_j by Φ_k;

$N_{jj} = \sum_{k=1}^{K} N_{jj}^{(k)} =$ total number of w_j samples in D classified as w_j;

$N^{(k)} = \frac{N}{K}$ number of samples in T_k;

$N_j^{(k)} =$ number of w_j samples in $T_k = D_k$;

$N_j = \sum_{k=1}^{K} N_j^{(k)} =$ number of w_j samples in D

The estimate of $\hat{P}(correct) = OA$ for a cross-validation approach is derived in a similar manner as when derived for the test approach. There are some differences in the formulas used. For the test set approach, we relied on the training data $\mathcal{L}$, to make derivations while in cross-validation approach, the OA derived depend on the folds D_k**.** If we define again (for the classifier Φ trained on the whole of D) a random variable Z,

$$Z((x,t)) = \begin{cases} 1 & if\ \Phi(x) = t \\ 0 & else \end{cases} \tag{5.38}$$

then $E(Z \mid w_j) = P(correct \mid w_j)$ and:

$$P(correct) = \sum_{j=1}^{c} P(w_j)P(correct \mid w_j) = \sum_{j=1}^{c} P(w_j)E(Z \mid w_j). \tag{5.39}$$

An unbiased estimator for $E(z \mid w_j)$ is the sample mean,

$$\hat{P}_j = \frac{1}{N_j} \sum_{(x,t)\in D, t=w_j} Z((x,t)) = \frac{N_{jj}}{N_j} \tag{5.40}$$

so that we obtain an unbiased estimator for $P(correct)$

$$\hat{P} = \hat{P}(correct) = \sum_{j=1}^{c} P(w_j)\frac{N_{jj}}{N_j}. \tag{5.41}$$

Note that with

$$\frac{N_j^{(k)}}{N^{(k)}} \approx \frac{N_j}{N} \tag{5.42}$$

it is implied that the proportions of the classes are maintained. This is approximately true if we randomly sample the data of D in the folds. If this condition is enforced by sampling exactly the proportion of each class into folds, then we speak of "stratification" and

$$N^{(k)} \approx \frac{N}{K} \tag{5.43}$$

then

$$\frac{1}{N_j} = \frac{1}{K(N_j^{(k)})}. \tag{5.44}$$

Using the fact that the whole data set D is the disjoint union of the folds D_k, we can thus write:

$$\begin{aligned} \hat{P}_j &= \frac{1}{N_j} \sum_{k=1}^{K} \sum_{(x,t)\in D_k; t=w_j} Z((x,t)) \\ &= \sum_{k=1}^{K} \frac{1}{K(N_j^{(k)})} \sum_{(x,t)\in D_k; t=w_j} Z((x,t)) \\ &= \frac{1}{K} \sum_{k=1}^{K} \frac{1}{N_j^{(k)}} N_{jj}^{(k)}. \end{aligned} \tag{5.45}$$

If we denote by $\hat{P}_j^{(k)} = \frac{1}{N_j^{(k)}} N_{jj}^{(k)}$ as the proportion of classified w_j samples in T_k by Φ_k, then

$$\hat{P}_j = \frac{1}{K} \sum_{k=1}^{K} \hat{P}_j^{(k)} \tag{5.46}$$

is the average of the proportions of correctly classified w_j samples by the Φ_k on their test sets T_k. From this it follows furthermore that

$$\hat{P}(correct) = \sum_{j=1}^{c} P(w_j)\hat{P}_j = \frac{1}{K}\sum_{k=1}^{K}\underbrace{\sum_{j=1}^{c} P(w_j)\hat{P}_j^{(k)}}_{=\hat{P}(correct;\Phi_k)}. \tag{5.47}$$

So we obtain the result that

$$P(correct) = \frac{1}{K}\sum_{k=1}^{K}\hat{P}(correct;\Phi_k) \tag{5.48}$$

i.e the estimate for the correct probability of the classifier Φ is the average of the estimates for the correct probabilities of the Φ_k. Also note that if $P(w_j) = \frac{N_j}{N}$ for all classes $w_1, w_2, ..., w_c$ then $\hat{P}(correct)$ becomes just the proportion of correct classified samples in D by all classifiers Φ_k on their test sets. $\hat{P}(correct)$ can then be derived from the priors using (5.47) and (5.44) with $N_j = K(N_j^{(k)})$ in the following way:

$$\begin{aligned}
\hat{P}(correct) &= \sum_{j=1}^{c} P(w_j)\hat{P}_j \\
&= \sum_{j=1}^{c} \cdot\frac{N_j}{N}\cdot\frac{1}{K}\sum_{k=1}^{K}\frac{1}{N_j^{(k)}}N_{jj}^{(k)} \\
&= \sum_{j=1}^{c}\frac{K(N_j^{(k)})}{N}\cdot\frac{1}{K}\sum_{k=1}^{K}\frac{1}{N_j^{(k)}}N_{jj}^{(k)} \\
&= \frac{1}{N}\sum_{j=1}^{c}\cdot\sum_{k=1}^{K}N_{jj}^{(k)} \\
&= \frac{1}{N}\sum_{j=1}^{c}N_{jj}.
\end{aligned} \tag{5.49}$$

The derivation for an estimate of the standard error of $\hat{P}(correct)$ is similar as in the test set

approach. If we define a random variable $Y_j = \sum_{(x,t)\in D;t=w_j} Z((x,t))$, to be the number of correct w_j samples and $Y_j \sim b(N_j, P(correct|w_j))$, then we can write

$$var(\hat{P}) = var\left(\sum_{j=1}^{c} P(w_j)\frac{1}{N_j}\sum_{(x,t)\in D;t=w_j} Z((x,t))\right). \tag{5.50}$$

The $var(\hat{P})$ can be rewritten in the following way:

$$var(\hat{P}) = \sum_{j=1}^{c}(P(w_j))^2\frac{1}{N_j^2}N_jP_j(1-P_j). \tag{5.51}$$

We can estimate the P_j with the $\hat{P}_j^{(k)}$ of the classifiers Φ_k by

$$\hat{P}_j = \frac{1}{K}\sum_{k=1}^{K}\hat{P}_j^{(k)}. \tag{5.52}$$

Similarly, the result of the estimate of the standard error $\hat{P}$ is

$$SE(\hat{P}) = \sqrt{var(\hat{P})} = \sqrt{\sum_{j=1}^{c}((P(w_j))^2\frac{\hat{P}_j(1-\hat{P}_j)}{N_j}}.$$

This estimate of the standard error can be used to determine a 2-sided confidence intervals for $\hat{P}$:

$$1-\alpha = P\left((P(correct) \in \left[\hat{P} - z_{\alpha/2}SE(\hat{P}), \hat{P} + z_{\alpha/2}SE(\hat{P})\right]\right) \tag{5.53}$$

The above formula of the standard error $\hat{P}$ is quite complicated. In order to obtain a simpler estimate, we can also use a similar idea as in the last section. Write

$$\hat{P} = \hat{P}(correct) = \sum_{j=1}^{c} P(w_j)\hat{P}_j = \sum_{j=1}^{c} P(w_j)\frac{1}{N_j} \sum_{(x,t)\in D; t=w_j} Z((x,t)). \tag{5.54}$$

Then the variance becomes

$$var(\hat{P}) = \sum_{j=1}^{c} P(w_j)^2 \frac{1}{N_j^2} N_j var(Z \mid w_j). \tag{5.55}$$

We again estimate the variance $var(Z \mid w_j)$ by the sample variance s_j^2 of the samples $Z((x,t))$ with $t = w_j$ and the one as in (5.15). Hence we can again estimate the standard error by

$$SE(\hat{P}) = \sqrt{\sum_{j=1}^{c} P(w_j)^2 \frac{1}{N_j}\hat{P}_j(1-\hat{P}_j)} = \sqrt{\sum_{j=1}^{c} P(w_j)^2 \frac{s_j^2}{N_j}}. \tag{5.56}$$

Chapter 6

Data and Methods

6.1 Introduction

This chapter discusses the data and methods used in this study for comparing Neural Networks and Support Vector Machines. It also considers some performance measures that enable the comparison of both the Support Vector Machines (SVM) and Neural Networks (NN). Section 6.2 gives a description of the data and its composition. Input coding and secondary structure coding are described in Section 6.3. Section 6.4 discusses the NN and SVM approach, their formulation and evaluation of results. A brief overview of the software used in the analysis is presented in Section 6.5.

6.2 Software

The software used for the experiments is Matlab Version 7.4.0.287 (R2007a). The Neural Networks toolbox Version 5.0.2 (R2007a) is used for Neural Networks. For Support Vector Machines, Matlab codes from Schwaighofer, A. (2002) are used. They are available from `http://ida.first.fraunhofer.de/~anton/software.html`. The computer that was used to perform the experiments for model selection is an Intel(R) Core(TM) 2CPU6300@1.86GHz . Other programs used in this study can be found in Appendix 3. A summary of what each code is intended to achieve can also be found in Appendix 3. The codes are created to assist pre-processing of the amino acid sequences and the secondary structures into real numbers. Some

create inputs and outputs for Neural Network and Support Vector Machine implementation. For either a NN or SVM, there are codes used to achieve training. Also, there are codes that deal specifically with classification of amino acid to their secondary structures. Another code is implemented to assess prediction accuracy of both methods.

6.3 The Dataset

The data are comprised of 62 proteins (Appendix 4). These proteins have also been used in the work of [20] with the objective of predicting the secondary structure of proteins with Neural Networks. Sixty of the proteins are available at `http:antheprot-pbil.ibcp.fr/` in the Kabsch & Sander database. Two of the proteins were not defined-namely Rubredoxin 2RXN and Cytochrome c550. Comparisons of the 62 proteins as defined in the Dictionary of Secondary Structure Prediction (DSSP) table [26] and from the Kabsch and Sander database were made. The two missing proteins were located in the RCSB Protein Databank. However, in the databank, the proteins had new identifiers: Rubredoxin 2RXN and Cytochrome c550 were replaced by Rubredoxin 5RXN and Cytochrome 155c respectively.

6.3.1 The form of the data

The data are structured in rows by protein name, primary and secondary structure. An example of a protein Avian polypeptide is:

>Avian polypeptide

`GPSQPTYPGDDAPVEDLIRFYDNLQQYLNVVTRHRY*`

`CCCCCCCCCTTSCHHHHHHHHHHHHHHHHHHHHTTCCC*`

The primary structure is a sequence of amino acids, which are represented by a 1 letter code as described in Chapter 1 (refer to Table 2.1). The secondary structures are made of 8 classes : H, B, E, G, I, T, S and rest marked a dash $(-)$. They are the same size as the primary sequence. The next section describes how the input coding is performed and how the secondary structure is reduced from 8 classes to 3 classes.

6.4 Data Coding

6.4.1 Pre-processing

Feature extraction [14] is a form of pre-processing in which the original variables are transformed into new inputs for classification. This initial process is important in protein structure prediction as the primary sequences of the data are presented as single letter code. It is therefore important to transform them into numbers. Different procedures can be adopted for this purpose, however, for the purpose of the present study, orthogonal coding will be used to convert the letters into numbers.

6.4.2 Input coding system

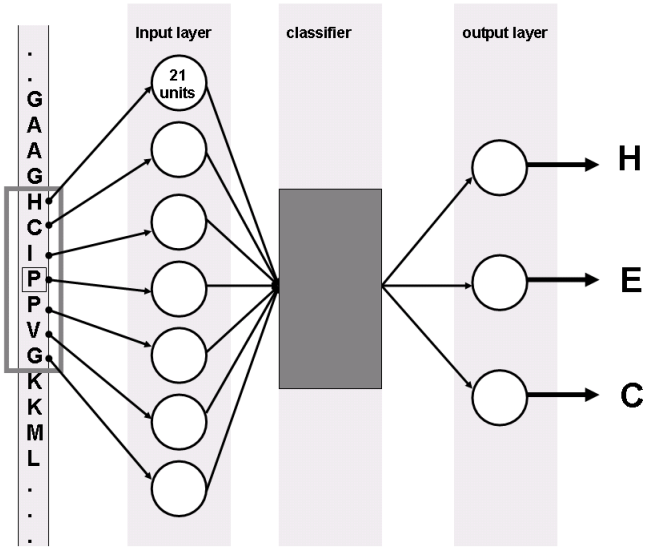

Figure 6.1: Input and output coding for protein secondary structure prediction

Figure 6.1 gives a network structure for a general classifier. The primary sequences are used as inputs to the network. To determine these inputs, a similar coding scheme as used by Holley and Karplus [22] has been adopted. To read the inputs into the network, a network encodes a moving window through the primary sequences. An illustration of a moving window is shown in Figure 6.2. A window of size 13 is chosen for the purpose of this study. This is chosen

based on the previous study by Qian & Sejnowski [38], where size 13 yielded best results. The reason being that too large or too a small window size reduced the performance on the test set. For that window, prediction is made for the amino acid residue that is in the centre. A binary encoding scheme is therefore used to assign numerical values to letters. For each binary vector, there are 21 positions: 20 positions for the letters of amino acids and 21^{st} for a null input denoted by $*$. For the window size of 13, the input pattern contains 13 input residues. Each residue will be assigned 1 depending on its position while the other positions will be assigned 0's. However, prediction will only be made for the central residues. In that way, there will be 21×13 input groups for which 13 of them will have values 1 and the rest 0. For the window length of 13 obtained from $2n + 1$ for $n = 6$, the dimension of the samples is $(2n + 1) \times 21 = 273$. To illustrate the idea of a moving window, suppose that a protein has the following primary sequence:

```
...KLNTDETGACPQACYA
```

Figure 6.2 shows an example of a moving window of length 7. Suppose that each window is treated as a training pattern in predicting for the central residue. From the graph, the first window `'TDEPGAC'` is the first pattern and is used to predict for the residue `'P'`. However, to predict for the next residue `'G'`, the window slides to the next group `'DEPGACP'`. The window slides to the next group until the last group `'CPQACYA'` is reached and prediction is made for the central residue `'A'`.

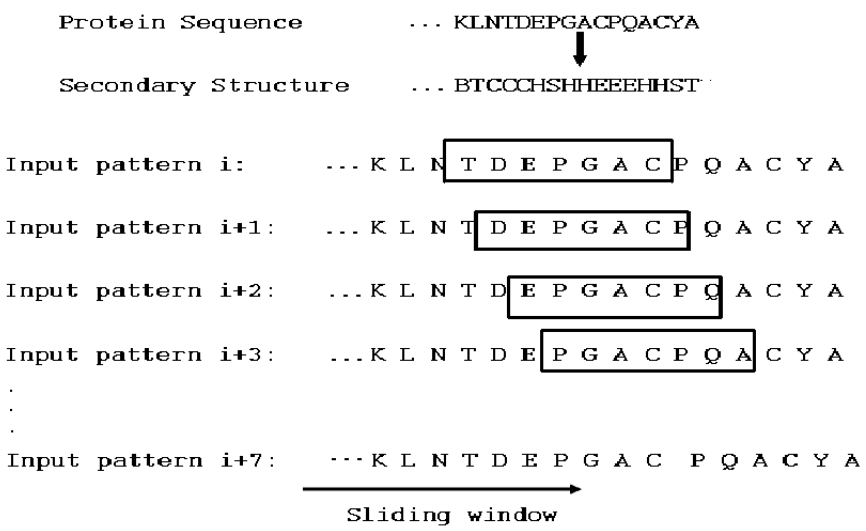

Figure 6.2: A sliding window of length 7

The orthogonal coding of the 20 amino acids in Chapter 2 is shown in Table 6.1. The assignment of the amino acids to their positions is determined with the function **orthcoding.m** available from Appendix 3.

Table 6.1: Orthogonal coding of 20 amino acids

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

Therefore, the orthogonal coding for a sequence `KLNTDETGACPQACYA`, is given in Table 6.2. From this table a unique binary vector is assigned for each residue.

Table 6.2: Orthogonal coding of an amino acid

K	L	N	T	D	E	T	G	A	C	P	Q	A	C	Y	A
0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1
0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

6.4.3 Secondary structure encoding

There are various methods that can be used to perform secondary structure assignment. These include DSSP [26], DEFINE [39] and STRIDE [17]. The assignments of secondary structure in this study are based on the DSSP method. It classifies the secondary structures into 8 classes. The 8 secondary structures are reduced to the known 3 structures: $\alpha - helices$, $\beta - strand$ and coils - residues neither helices nor strand. The same mapping is used by Cuff and Barton [14] and Hua and Sun [24]. Table 6.3 shows the 8 secondary structural classes defined by DSSP, their standard abbreviations and the mappings to three structural classes. *Rest* $(-)$ defines secondary structures which do not belong to any DSSP defined categories.

Table 6.3: Secondary structure assignment

DSSP class	Abbreviation	3 state classes
$\alpha - helix$	H	H
$3_{10} helix$	G	H
$\beta - strand$	E	E
$isolated\ \beta - ridge$	B	E
$\pi - helix$	I	C
$Turn$	T	C
$Bend$	S	C
$Rest$	$-$	C

Target coding differs between Neural Networks and Support Vector Machines. In Neural Networks, the following target coding for the output neurons is used; for example, in alpha (class 1) versus no alpha (class 2), the targets have the following coding $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ to denote class 1, i.e. alpha, and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ to denote class 2, i.e. no alpha. For Support Vector Machines however, target coding is different: $+1$ represents target belonging to class 1, i.e. alpha while -1 denotes targets belonging to class 2, i.e. no alpha.

Previous sections described the form of data that will be used during analysis to compare the performance of NN and SVM in predicting the secondary structure of proteins. These sections also gave a discussion of how the inputs and targets are to be created. The next section (6.5) outlines formulation of both NN and SVM and describes how the results are going to be evaluated.

6.5 Neural Networks and Support Vector Machines

6.5.1 Formulation

A detailed discussion of Neural Networks and Support Vector Machines has already been given in the previous chapters. This section focuses on formulating a method for objective comparison between the two approaches. To achieve this, the following two-class problems are created; the One-Against-All classifiers $H/\sim H$, $E/\sim E$, $C/\sim C$ and the One-against-One

classifiers H/E, E/C and C/H. We denote a binary classification problem for classes S and T by S/T. If T is the union of classes different from S, then write $S/\sim S$ for S/T. Therefore, as an example, a classifier $H/\sim H$ means that we consider the classification task of classifying $\alpha's$ vs. no $\alpha's$ and H/E means that we try to separate $\alpha's$ from $\beta's$. The data comprises 10766 samples and the secondary structure composition is given in Table 6.4.

Table 6.4: Total number of secondary structure states for the dataset

structure	Total number	Percentage
H	3047	28.3
E	2288	21.3
C	5431	50.4

For One-Against-All classifiers, all the 10766 samples are used in formulating Neural Networks and Support Vector Machines, while for the One-against-One classifiers, samples differ based on the classifier under consideration. For example, in H/E, only two classes are considered: alphas and betas. This means that coils are excluded from the data set. Table 6.5 gives the total number of samples for all the One-against-One classifiers.

Table 6.5: Total number of samples for each classifier

Binary Classifiers	Total samples
H/E	5535
E/C	7719
C/H	8478

10-fold cross-validation is used to estimate how well the model obtained will perform on unseen data. We train 10 classifiers $\Phi_1, \Phi_2, ..., \Phi_{10}$ for NN and SVM each time leaving out one of the subsets from training. This ensures that all samples are used only once for both training and testing. 500 cycles for training are used in NN. The early stopping approach (ES), and without early stopping approach (WES) will be considered for Neural Networks. The variable net.trainParam.max_fail=20 in ES will be used to ensure that when training stops, the parameters of the network obtained at the minimum of the validation error are returned. However, in WES training continues for 500 cycles. In order to be able to compare the performance of ES and WES in NN, a similar split of the folds will be used. The only difference is

in the manner in which the parameters of the network will be chosen. In ES they will depend on validation error, while in WES, they will depend on training error.

For both approaches, experiments are performed 10 times each and averaged to avoid bias in the prediction accuracies that can occur depending on which proteins are used as the test set each time an experiment is performed. 5% of training data is used as validation set even if validation set is not used at all. Weights of the connections from the input layer to the output layer can be initially assigned random values, e.g. (-0.1,0.1). Different numbers of hidden neurons in the hidden layers are used: e.g. 0,1,2,10,15,20 and 40. It is of particular interest to see how 0 or 1 hidden neurons will affect the performance of Neural Networks. This is because the universal approximation theorem of multilayer perceptron states that only one hidden layer is sufficient for any feed-forward network to approximate a function with desired accuracy. This theorem is reflected in the results of the study by Holley and Karplus [22]. Their study reveals that a network with 0 hidden units has prediction accuracy of 62.3% while the one with 20 hidden units achieves prediction accuracy of 59.3% on the test set. Similarly, Qian and Sejnowski [38] reported that a network with 0 hidden units reached prediction accuracy of 62.5% while a network with 40 hidden units reached 62.7%. Resilient Backpropagation will be used for training Neural Networks.

In Support Vector Machines, the following Gaussian Kernel is used:

$$K\left(x_i, x_j\right) = exp\left(-\gamma \parallel x_i - x_j \parallel^2\right).$$

SVM has two parameters: the kernel γ and the cost parameters C. For this study, a kernel parameter of $\gamma = 0.1$ will be used and is fixed for all experiments. Hua and Sun [24] used this parameter in their study. Their cost parameter was set to 1.5 to construct the classifiers. Since we are interested in the error estimates for accuracy in SVM, the cost parameter will be varied to ensure better parameters for the model. The cost parameters used are $0.1, 0.3, 0.5, 0.7, 0.9, 1$ and 5.

6.5.2 Evaluation of results

To enable a comparison of both the Support Vector Machines and Neural Networks, similar measures have to be used. The Overall Accuracy (OA) is used for this purpose. It is obtained using $\hat{P}(correct)$ from 5.49. It is the measure of the accuracy of a classifier. It gives the proportion of correctly classified samples. The standard error values, $SE(\hat{P})$ for these accuracies are obtained with 5.56.

Chapter 7

Results

7.1 Introduction

The objective of the study is to compare the performance of Neural Networks and Support Vector Machines. The two approaches were applied on 62 globular proteins described in Chapter 6. This chapter gives the results of the performance of Neural Networks (NN) and Support Vector Machines (SVM) in predicting the secondary structure of proteins from their primary sequences. A detailed summary of the results and the notation can be found in the Appendix 1 for Neural Networks and Appendix 2 for Support Vector Machines. Section 7.2 gives the results of NN performance in predicting the secondary structure of the 62 proteins. It also includes a discussion of the effect of number of samples on the time required to train a learning task. Section 7.3 presents the results from SVM. The effect of variations in cost parameter on prediction accuracy and the relationship between the values of cost parameters and time required for training are also discussed. Comparisons of the performance of both NN and SVM are discussed in Section 7.4.

7.2 Data Partition

For the two methods, the same partitioning of the data into training set, validation set and test set was used. 10-fold cross validation as described in Chapter 6 was used; the data was randomly divided into 10 parts, one used as a test set and the rest for training. However,

for training, a further partitioning was done. 5% of the training data was excluded from the training data and used as validation set. The window size was fixed at 13.

7.3 Results: Neural Networks

In neural network approaches, networks with no hidden neurons (0 hidden neurons), or with one hidden layer with 1, 2, 10, 15, 20, 40 hidden neurons were used. As we used two-class problems (binary classifiers), the networks had two output neurons to learn the targets, i.e. $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ for class 1 and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ for class 2. Six binary classifiers $H/\sim H$, $E/\sim E$, $C/\sim C$, H/E, E/C and C/H were created. For example, $H/\sim H$ is a binary classifier, which interprets class 1 as alphas and class 2 as structures that are not alphas. Training was performed for a maximum of 500 epochs (= passes through the leaning task). The Matlab parameter max_fail was used to achieve early stopping in Neural Networks. It was set to 20. The Resilient Backpropagation [40] was used for training.

A detailed summary of results showing the overall accuracy estimate on the training set (OAtrain), overall accuracy estimate of the test set (OAtest) obtained from different network architectures, time taken during training (Time), the best classifier for each classification task (NNopt), the standard error of the test set (SEtest) and the optimal accuracies, is given in Appendix 1. They are available for both early stopping (ES) and without early stopping (WES). NNopt is obtained in the following way:

- For each classifier, a network with no hidden layer, 1,2,10,15,20 or 40 hidden neurons in the hidden layer is considered. The notation NN0hu, NN1hu, NN2hu, NN10hu, NN20hu and NN40hu respective of the hidden neurons is used. *hu* refers to the number of hidden units.
- Overall accuracy for each hidden neurons is observed and recorded.
- The highest of all the accuracies either OAtrain or OAtest reported for each hidden neuron is denoted under NNopt. The shortest time it took to train each classifier also falls under this category as Time. The SEtest refers to the standard error of the architecture with better prediction accuracy. OAtrain or OAtest that fall under NNopt are also referred to as Optimal classifiers.

NNopt is simply the highest accuracy occurring in all the hidden neurons for each classifier. It also includes the smallest amount of time required during training and also the smallest standard error of the entire learning task. Table 7.1 gives the results for all the six binary classifiers based on NNopt.

Table 7.1: NN: Optimal classifiers for OAtrain and OAtest in ES and WES

	ES		WES	
Quantity	OAtrain	OAtest	OAtrain	OAtest
$H/\sim H$	79.58	74.66	99.36	74.63
$E/\sim E$	83.01	79.94	99.44	79.68
$C/\sim C$	73.98	68.65	99.26	68.82
H/E	80.85	72.46	99.73	72.37
E/C	82.36	76.85	99.57	76.16
C/H	79.60	73.11	99.50	73.27

In Table 7.1 the prediction accuracies for both ES and WES are different for the training set for each binary classifier. However, for the two approaches, the overall accuracy is not different on the test sets. For each binary classifier, the results on the training set and test set vary significantly. In ES, $E/\sim E$ has achieved the highest accuracy of about 83% on the training set while the highest overall accuracy of about 80% on the test set is also observed for that classifier. However, WES achieves an overall accuracy of about 100% for OAtrain, which also occurs in all the classifiers, and about 80% on OAtest. The lowest overall accuracy occurs in $C/\sim C$, with the accuracy of about 69% on the test set in both ES and WES.

7.3.1 Effect of the number of neurons in the hidden layers on prediction accuracy

ES and WES

Different network architectures were used to look at the effect of the number of hidden neurons on the overall accuracy of the network. For WES, there exists a positive relationship between the prediction accuracy and the number of hidden neurons. For an increase in the number of hidden neurons, OAtrain also increases. ES also shows a positive relationship between the number of hidden neurons and OAtrain. Figure 7.1 gives as an example H/E a classifier that shows positive relationship between the hidden neurons and overall accuracy for OAtrain in

WES. From this graph, OAtrain has the lowest value at 0 hidden neurons. The highest values occur at 40 hidden neurons. Between 0-10 hidden neurons, OAtrain is increasing. However, beyond 10 hidden neurons, OAtrain is constant (about 100%) up at 40 hidden neurons.

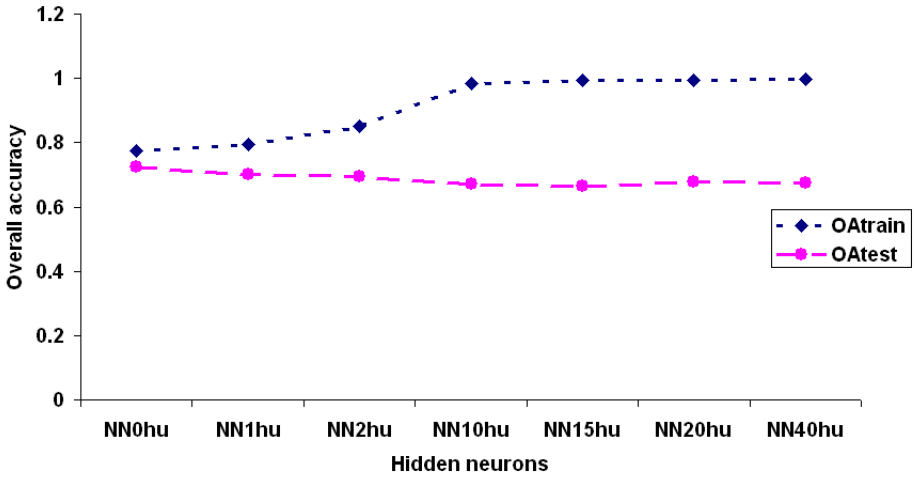

Figure 7.1: Number of hidden neurons vs. accuracy (H/E) WES

$C/\sim C$ is an example of a classifier showing a different pattern. The graph of this classifier is given in Figure 7.2. In this graph, the overall accuracy at 0 hidden neurons (no hidden neurons) is higher than when the network has 1 hidden neuron. But beyond 1 hidden neurons, the overall accuracies increase with increasing number of hidden neurons.

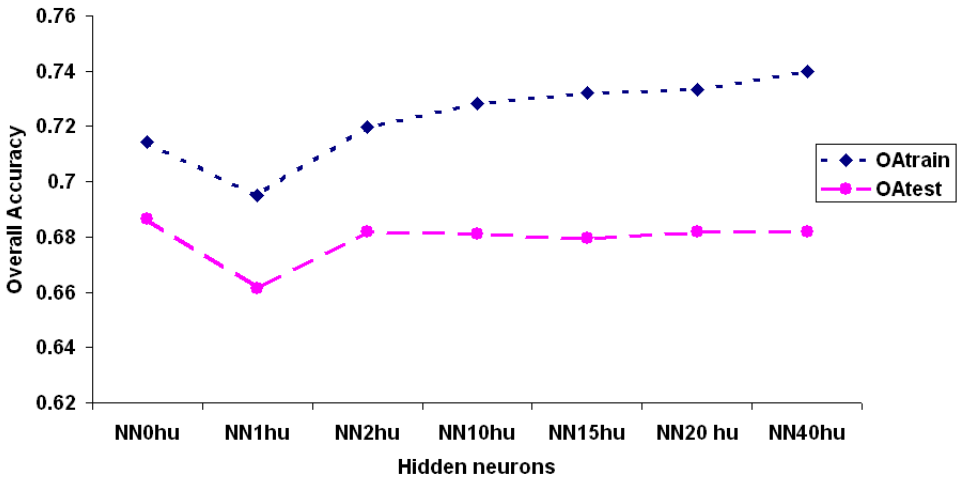

Figure 7.2: Number of hidden neurons vs. accuracy coils vs coils ($C/\sim C$) with ES

The OAtest is almost constant with an increase in the number of hidden neurons for ES. For WES, OAtest decreases with an increase in the number of hidden neurons. Again for ES, Figure 7.2 shows that OAtest has highest values at 0 hidden neurons. However, between 0 and 1 hidden neurons, OAtest decreases. Another increase in OAtest is observed between 1 and 2 hidden neurons, and remains constant over all hidden neurons beyond 2 hidden neurons. But in WES, OAtest is constant between 0 and 2 hidden neurons and then starts to decrease from 2 hidden neurons. The conclusion is that with the WES approach, OAtest decreases with an increase in the number of hidden neurons, while in ES 0 hidden neuron is sufficient to train a classifier because it provides the highest value of prediction accuracy.

H/E does not show a similar pattern as other classifiers for all the hidden neurons. Though the same behaviour with other classifiers is observed at 0,1 and 2 hidden neurons, the overall accuracy does not increase with an increase in the number of hidden neurons beyond 2 hidden neurons. It is almost constant between 2-40 hidden neurons. The plot of overall accuracy for this classifier is shown in Figure 7.3.

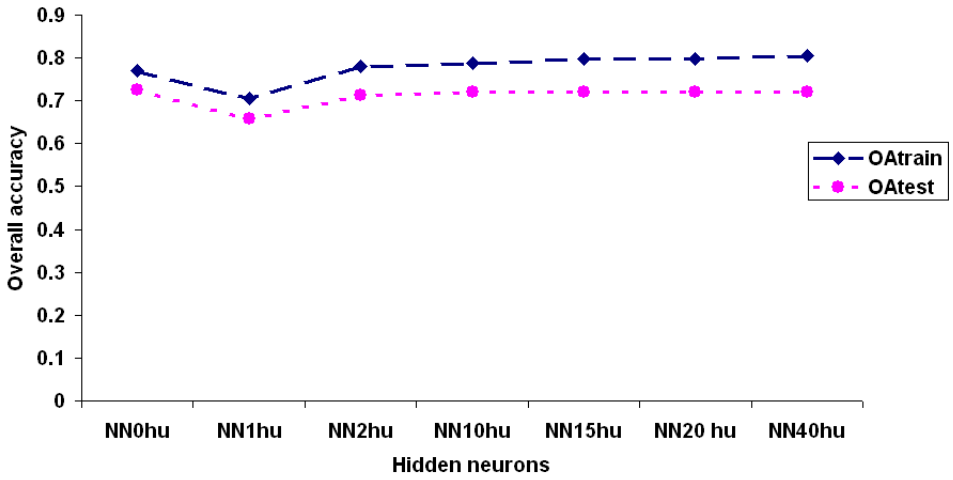

Figure 7.3: Number of hidden neurons vs. accuracy (H/E) with ES

OAtest for H/E however shows a pattern similar to all those of other classifiers.

7.3.2 Training time vs. prediction accuracy in Neural Networks

There are differences in the time required to train the network for ES and WES. For all the binary classifiers, less training was required for ES than in WES. For WES, there is a positive relationship between time required during training and the number of hidden neurons. For example, 0 hidden neurons require about 3 min to train, while a network with 40 hidden neurons requires about 25 min. Figure 7.4 gives an example of a graph showing the relationship between training time and number of hidden neurons when WES is used.

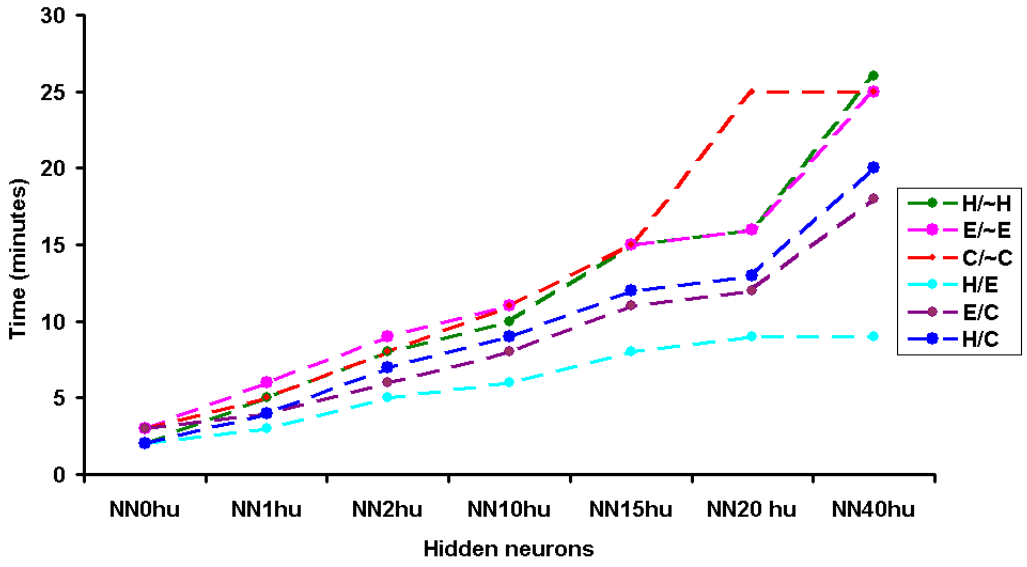

Figure 7.4: Training time vs. hidden neurons in WES

From 7.4, a network with no hidden neurons, requires less time for training, while a network with 40 hidden neurons requires longer. There is a huge increase in time between 20-40 hidden neurons. This is due to the complexity of networks when more hidden neurons are added. A different pattern is observed when ES is used. All ES training occurs within 5 minutes, and training time is constant over most of the network architectures. Small differences in times are recorded over all hidden neurons for One-Against-All classifiers. Small differences in time are also observed over all hidden neurons for One-Against-One classifiers. The relationship between time and hidden neurons in ES is not consistent. This can be shown in Figure 7.5.

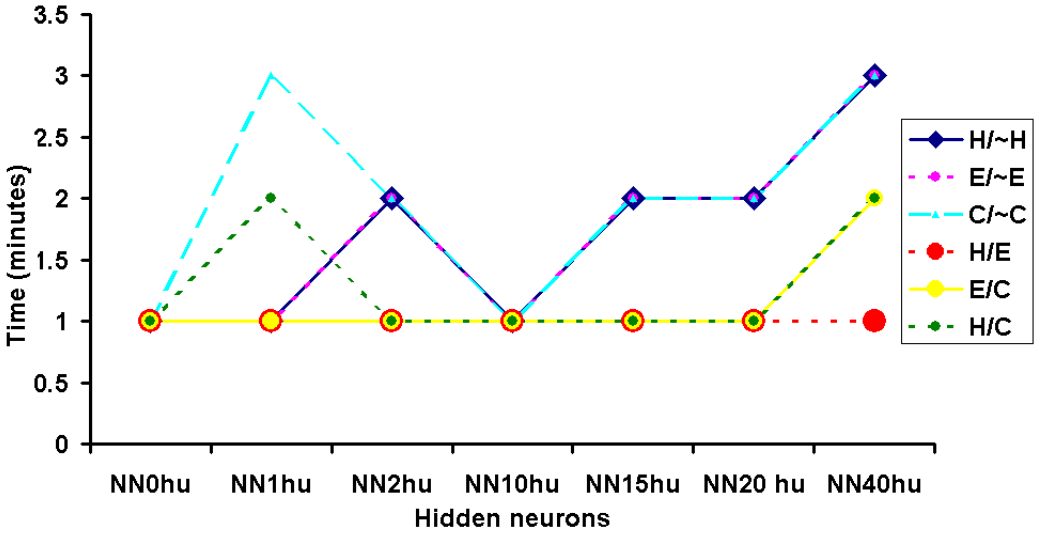

Figure 7.5: Training time vs. hidden neurons in ES

The conclusion is that for ES experiments, the important part of training takes place early and late overtraining will occur if training is not stopped.

7.3.3 Training time vs. total number of samples

For One-Against-All classifiers, 10766 samples are presented to the network for training. However, for One-Against-One classifiers, a different number of samples are presented to the network. This has an effect on the time required to train the classifiers. Table 7.2 shows the total number of samples used to train the network and the time required to train the network when ES and WES approaches are used.

Table 7.2: Training time vs. total samples for ES and WES

Binary Classifier	Total Samples	Time (ES)	Time (WES)
$H/\sim H$	10766	3	26
$E/\sim E$	10766	3	25
$C/\sim C$	10766	3	25
H/E	5535	1	9
E/C	7719	2	18
H/C	8478	2	20

Table 7.2 shows that time required to train the One-Against-All classifiers is almost the same in ES. Also, the time required to train classifiers in WES is almost the same. However, among the One-Against-One classifiers there are differences in the time required to train the classifiers. These differences can be observed in WES. H/E requires about 9 minutes to train, while E/C and H/C require about 18 minutes and 20 minutes respectively. This huge difference is attributed to the number of samples presented to the network. The samples from H/E make just over half of the samples used in One-Against-One classifiers. Therefore, it is to be expected that less time will be required. Also, it has fewer samples than E/C (2184) and H/C (2943) thus the same argument applies here. The difference in the number of samples between E/C and H/C is 759 samples. So, it is reasonable that there are small differences in the amount of time required during training. Thus training time increases as the number of samples increases.

7.4 Results: Support Vector Machines

In SVM, again 6 binary classifiers $H/\sim H$, $E/\sim E$, $C/\sim C$, H/E, E/C and C/H were constructed as in NN. Kernel parameter γ was fixed for $\gamma = 0.1$ [24]. The cost parameter C was however varied over the following values: 0.1,0.3,0.5,0.7,0.9,1,5. For a summary of results detailing the best classifier for each classification task SVMopt, OAtrain, OAtest, SEtest, time taken during training for each different value of the cost parameters (Time) see Appendix 2. The results of SVM application in predicting the secondary structure of proteins based on SVMopt are given in Table 7.3.

Table 7.3: SVM: Optimal classifiers for OAtrain and OAtest

Quantity	OAtrain	OAtest
$H/\sim H$	1	75.35
$E/\sim E$	1	80.15
$C/\sim C$	1	68.81
H/E	1	74.65
E/C	1	75.94
C/H	1	73.63

For all the binary classifiers, all the predictions on the training sets are correct (100% correct predictions). However, OAtest results vary significantly among some classifiers. The prediction accuracies are in the range 68-80%. The best prediction accuracy is recorded for $E/\sim E$ while $C/\sim C$ has the lowest accuracy compared to all the classifiers. The highest accuracy of 80% for OAtest is observed for the One-Against-All approach while the highest accuracy of about 76% is observed in One-Against-One approach on OAtest.

7.4.1 Effect of variations of cost parameters on prediction accuracy

The general observation on OAtrain is that the prediction accuracies increase with an increase in the values of the cost parameters. Minimum accuracies occur at $C = 0.1$ while the highest overall accuracy occurs at $C = 5$. For this parameter, OAtrain achieves 100% of correct predictions for each classifier. Though total correct prediction is achieved during training, the OAtest is constant for all the parameters. This means that increasing the value of the cost parameter has no effect on the prediction accuracy. Figure 7.6 shows the effect of variations in C on the overall accuracy for the H/E classifier.

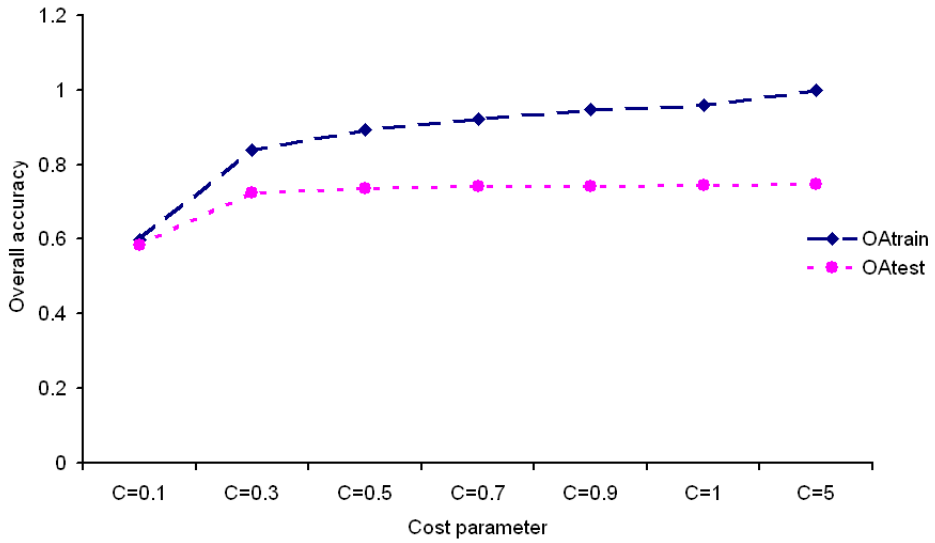

Figure 7.6: Cost parameters vs. accuracy (H/E)

From Figure 7.6, it can be seen that there is a rapid increase in OAtrain between 0.1and 0.3. But beyond $C = 0.3$, the OAtrain continues to increase steadily until it reaches 1. However, the rate of increase is not the same with larger values of the cost as observed for small values of the cost. OAtest is almost constant with increase in the value of cost. This is remarkable as no overtraining occurs.

7.4.2 Training time vs. cost parameters in Support Vector Machines

Figure 7.7 shows different cost parameters and the time taken to train the six binary classifiers. Based on this graph, it takes more time to train the One-Against-All classifiers than it took to train the One-Against-One binary classifiers. $C/\sim C$ took the longest time to train while H/E took the shortest time. Positive relationship exists between time and the cost parameters because an increase in the number of cost parameters increases the time required to train the classifiers. The longest time it to train a classifier was about 57 minutes, and that occurred at $C = 5$ for $C/\sim C$. The shortest time to train a classifier was about 13 minutes, which occurred at $C = 0.1$ for H/E. Furthermore, training time increases rapidly between $C = 1$

and $C = 5$ for all the classifiers.

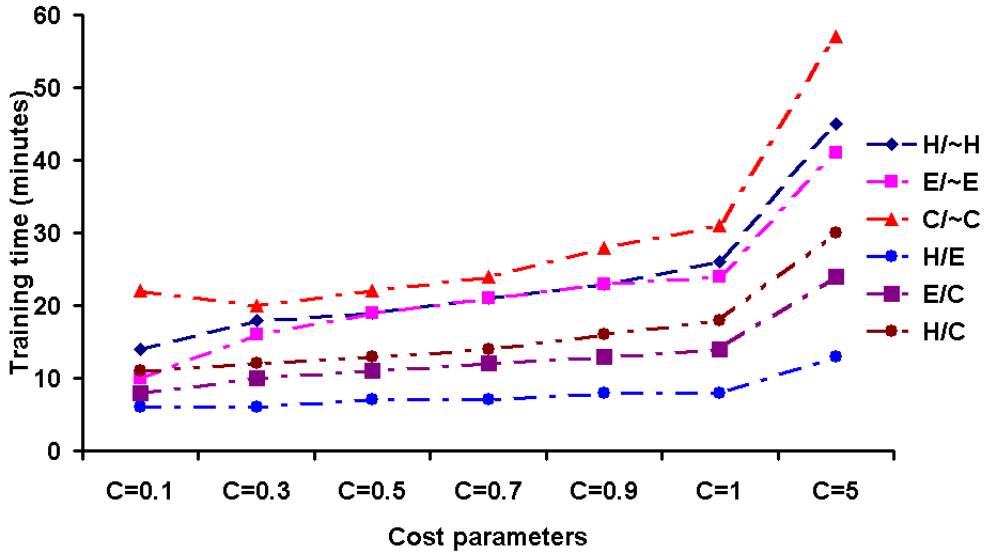

Figure 7.7: Training time vs. cost parameters in Support Vector Machines

As in Neural Networks, differences in the time required to train some classifiers at $C = 5$ can also be attributed to the different number of samples presented to each classifier. It has been established that One-Against-All classifiers take longer time during training than One-Against-One binary classifiers. Let us recall that One-Against-All classifiers use 10766 samples for each learning task while for One-Against-One, samples differ: for H/E there are 5535 samples, 7719 samples for E/C and H/C has 8478 samples (refer to Table 7.2). The total number of samples for each classifier (for each learning task) is related to training time and it can be expected that H/E will take the shortest time to train while, H/C will take the longest time to train for One-Against-One classifiers. At the same time, it can be expected that when the total number of samples used to create a learning task is about the same, time required during training should be about the same.

7.5 Comparisons of NN and SVM

The objective of the study is to compare the performances of Support Vector Machines and Neural Networks in predicting the secondary structure of proteins. This Chapter presents and discuss the results achieved by the research. This is done such that we obtain a good estimate for the probability of correctly predicting unseen data. The best classifiers denoted by NNopt and SVMopt for each classification task will be used to compare their performance. This is done by looking at the highest prediction accuracies in all hidden layers for each binary classifier in NN and in different values of the cost parameters in SVM. The accuracies for both approaches are given in Table 7.4.

Table 7.4: Comparison of Neural Networks and Support Vector Machines

Binary classifiers	NN % accuracy	SVM % accuracy
$H/\sim H$	74.63	75.35
$E/\sim E$	79.68	80.15
$C/\sim C$	68.82	68.81
H/E	72.37	74.65
E/C	76.16	75.94
C/H	73.27	73.63

The results in Table 7.4 show the performance of both NN and SVM in predicting the secondary structure of proteins. Generally, there are no big differences in the overall accuracies of the classifiers for NN and SVM. For the One-against-All approaches, SVM achieved the highest prediction accuracy of about 80%. But in One-Against-One approaches, NN achieved the highest accuracy of about 76%. The highest prediction accuracies were observed for $E/\sim E$ in the two methods while the lowest overall accuracy was observed for $C/\sim C$ in both methods.

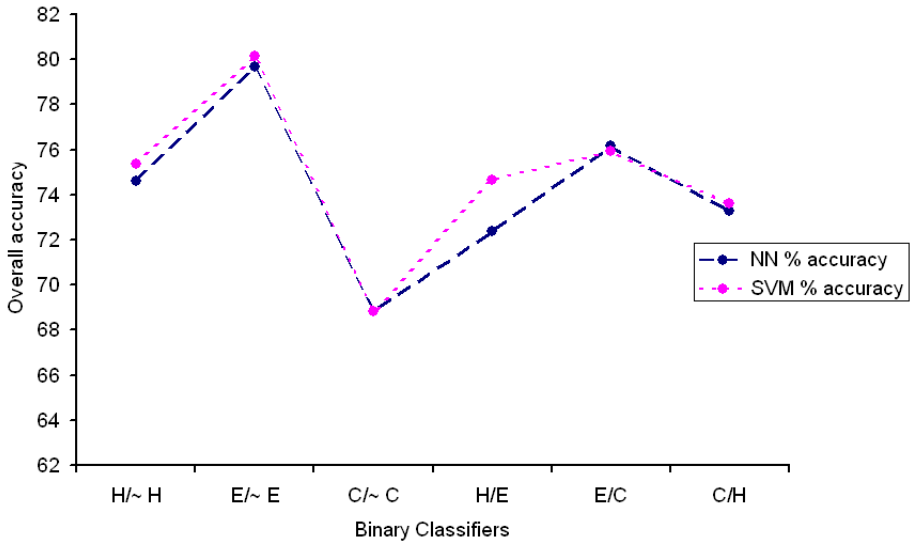

Figure 7.8: Comparisons of NN and SVM

Figure 7.8 is created to reveal if there are any differences in the performance of NN and SVM. The results in Table 7.4 are used to achieve this. From this graph, it is apparent that accuracies for SVM are higher than in NN on all binary classifiers. However, the difference in the values is very small. For H/E, the difference is about 3% between the SVM and NN. Numerically, 76.16% and 75.94% are different. However, we must ask whether it is possible to distinguish between the performances of NN and SVM approaches from the practical point of view based on these accuracies.

To determine whether the prediction accuracies are really different from each other, heuristic confidence intervals based on the point estimate $\hat{P}$ were created for each binary classifier as derived in Section 4.4. They were obtained using:

$$1 - \alpha = P\left((P(error) \in \left[\hat{P} - z_{\alpha/2}SE(\hat{P}), \hat{P} + z_{\alpha/2}SE(\hat{P})\right]\right). \tag{7.1}$$

OAtrain and OAtest from Table 7.4 for each method were used as point estimates. The standard errors (the lowest standard error for different hidden neurons or for different cost

parameters in each classifier), which can also be found under NNopt and SVMopt, were used. All the results are based on 95% confidence interval obtained by:

$$95\% \text{ interval } : z_{\alpha/2} = z_{0.025} = 1.96. \tag{7.2}$$

It is important to mention that OAtest is obtained as a sample mean (of the $Z_1's$) and that we appeal to the Central Limit Theorem for using $z_{0.025} = 1.96$. The results of the confidence intervals (CI) are given in Table 7.5.

Table 7.5: Standard error estimates and confidence intervals for Point Estimates

Binary classifiers	NN SEtest	NN CI	SVM SEtest	SVM CI
$H/\sim H$	0.0042	(0.7381,0.7545)	0.0028	(0.7480,0.7590)
$E/\sim E$	0.0039	(0.7892,0.8044)	0.0026	(0.7964,0.8066)
$C/\sim C$	0.0045	(0.6794,0.6970)	0.0030	(0.6822,0.6940)
H/E	0.0061	(0.7117,0.7357)	0.0040	(0.7387,0.7543)
E/C	0.0045	(0.7528,0.7704)	0.0033	(0.7529,0.7659)
C/H	0.0048	(0.7233,0.7421)	0.0032	(0.7300,0.7426)

Table 7.5 shows the confidence intervals of each classifier for Neural Networks and Support Vector Machines. It also has the standard errors of OAtest from each approach. From this Table, Neural Networks classifier $H/\sim H$ has the $SE(\hat{P}) = 0.0042$ and $\hat{P} = 0.7463$ (refer to Table 7.4). Therefore, to calculate CI for this classifier at 95% level:

$CI = [0.7463 - 1.96(0.0042), 0.7463 + 1.96(0.0042)]$

$= (0.7381, 0.7545)$

Also for Support Vector Machines, the classifier $H/\sim H$ gave the following CI based on $\hat{P} = 0.7535$ and $SE(\hat{P}) = 0.0028$:

$CI = [0.7535 - 1.96(0.0028), 0.7535 + 1.96(0.0028)]$

$= (0.7480, 0.7590)$

The interpretation is that if we were to repeatedly perform experiments for a fairly large number of times, from the same population and calculate a confidence interval for the population parameter each time, about 95% of the confidence intervals would contain the true value of the

population parameter. The plots of confidence intervals for SVM and NN for each classifier were constructed. Some of the confidence intervals overlap - i.e. $H/\sim H$, $E/\sim E$ and C/H. An example is given in Figure 7.9 for the $H/\sim H$ classifier.

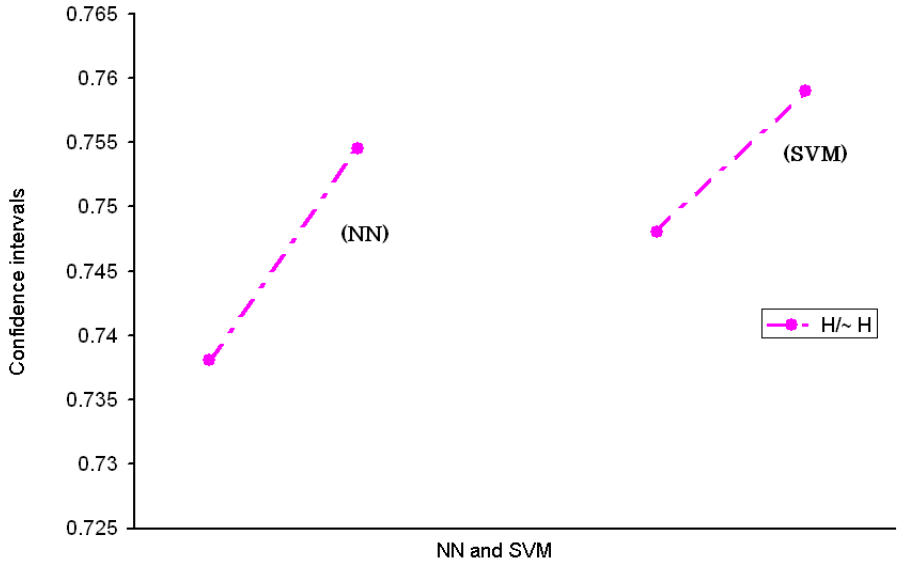

Figure 7.9: Overlapping confidence intervals ($H/\sim H$)

For these classifiers, it means that we cannot distinguish between the true accuracies based on our experiments, as both true accuracies (say, the one from NN and the other from SVM) could be identical. This is then interpreted from a statistical point of view to mean that the classifiers may have the same accuracies. There are some which do not overlap but one interval can be found within the other, e.g. $C/\sim C$, and E/C. This is also an overlap. However, for H/E, the intervals are distinct. An example in which confidence intervals are distinct is given in Figure 7.10. There are slight differences between the upper limit of NN and the lower limit of SVM.

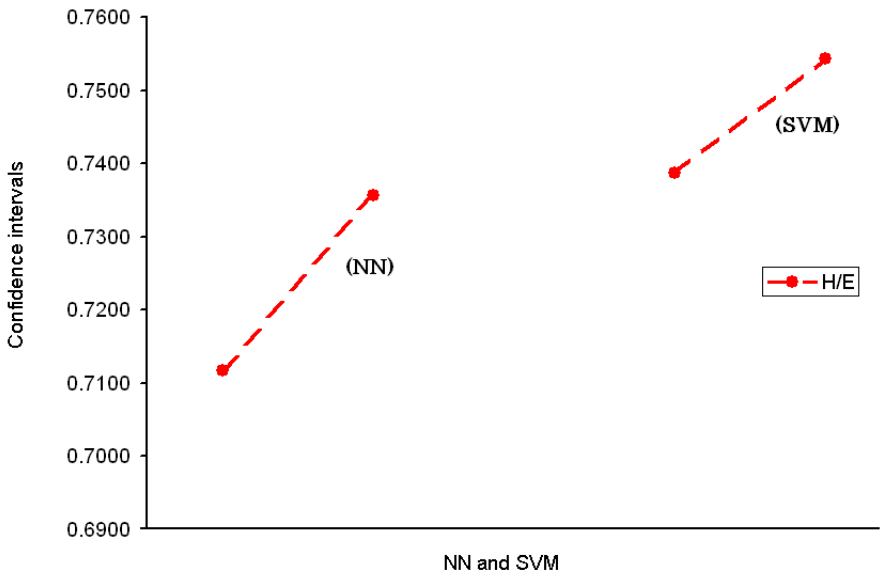

Figure 7.10: Non-overlapping confidence intervals (H/E)

For H/E, there is a difference of about 2.28% in the accuracies for both methods. This means that based on our heuristic 95% confidence intervals, we can conclude that SVM performs better on this problem. However, from a practical point of view it is doubtful if a biochemist using both methods for protein prediction would find SVM superior to NN: both point estimates are very similar.

Chapter 8

Conclusions

8.1 Introduction

Chapter 7 presented the results obtained from comparing the performance of Neural Networks and Support Vector Machines in predicting secondary structure of proteins from their secondary structures. Section 8.2 gives a conclusion and discussion of the overall research investigation. The contributions of the thesis are indicated and future research is identified in Section 8.3.

8.2 Conclusions and discussion

Protein structure prediction is an important step towards predicting the tertiary structure of proteins (and Quartenary structure). The reason is that knowing the tertiary structure of proteins can help to determine their functions. The main aim of this thesis was to compare performance of Support Vector Machines and Neural Networks in predicting the secondary structure of proteins from their amino acid sequences. The following conclusions were derived:

- For Neural Networks, there is a dependence of prediction accuracy on the number of hidden neurons. Furthermore, either a simple architecture with no hidden layer, or one hidden layer with 1 hidden neuron is more sufficient than architecture with e.g. 40 hidden neurons. Also, early stopping guarantees a good classifier that is independent of the number of hidden neurons.

- Although in Support Vector Machines there seems to occur over-adaptation to the training set (the training samples are learnt completely), it is interesting that the overall accuracy ($P(correct)$) on the test data is unaffected.
- Neural Networks with early stopping or with simple architecture are far superior to Support Vector Machines with respect to training time. Support Vector Machines are also demanding in terms of memory, whereas Neural Networks can be applied on less powerful computers.
- In respect of the overall accuracy estimates for the test data, Support Vector Machines are slightly superior in almost all classification tasks. However, from a statistical point of view, they are not different. Except for the alpha vs. beta (H/E) classifier, confidence intervals do overlap for most of the binary classifiers, suggesting that the true accuracies obtained from both methods can be identical. However, from a practical point of view, there seems to be no real difference between the two methods.
- We conclude from this that reported differences in the literature for Support Vector Machines and Neural Networks are not due to the methods but are influenced by different designs of the classifiers. Therefore, some of the variables that may influence differences in the results include different input coding systems, window length, cross validation methods, the parameters used to create a learning task and even the structural class assignments.

Support Vector Machines results were compared with the results of multiclass Support Vector Machines based on the work of Hua and Sun [24]. Both approaches have some similar characteristics. The same coding scheme is used and similar structural class assignments of 8 classes to 3 classes are also used. Both approaches created six binary classifiers. The results are obtained from the same window length of 13 and the prediction accuracies are obtained in the same way as in 5.49. The difference between the two approaches is in cross-validation. Hua and Sun's approach uses a 7-fold cross-validation, while the approach in the present study uses 10. Also Hua and Sun use multiple sequences (profiles) rather than single sequences to create inputs, while we use only single sequences. Our approach uses 62 proteins from the Kabsch and Sander database `http://antheprot-pbil.ibcp.fr/`, while the Hua and Sun use the RS126 dataset. The same kernel parameter $\gamma = 0.1$ is used in both approaches. However, our cost parameter is varied over the range [0.1,5] while Hua and Sun use C=1.5, which they find to be optimal. The results of these two approaches are shown in Table 8.1.

Table 8.1: Comparison of SVM and Hua and Sun's Approach

Binary Classifiers	Accuracy (%)	Accuracy (%) Hua and Sun
$H/\sim H$	73.74	79.74
$E/\sim E$	80.32	80.4
$C/\sim C$	68.31	69.77
H/E	71.75	76.24
E/C	73.72	73.76
C/H	75.35	76.31

From the results in Table 8.1 it is apparent that there are some small differences in the prediction accuracies of all the binary classifiers. The exception is in classifier $H/\sim H$ because big differences are observed for overall accuracy between the two approaches. In both methods, the highest predictions are observed for $E/\sim E$ with the accuracy of approximately 80%. $C/\sim C$ again have the lowest prediction accuracies. In general, Hua and Sun's approach achieves better results than own method.

8.3 Future work

There are further developments, which can be incorporated to improve the performance of all the classifiers. The literature shows that increasing the size of the training data set for example improves the performance significantly [24]. Another factor is the window size. Choosing an appropriate window length also helps to improve the performance. This can be seen in the work of Qian and Sejnowski [38]. The percentage of correctly classified samples range from $53.9, (57.7,$ and $60.5)$ for the window size $1, (3,$ and $5,$ respectively) to an improved $61.9, (62.3, 62.1, 62.7, 62.2, 61.5,$ and $61.6)$ for the window sizes $7, (9, 11, 13, 15, 17,$ and $21,$ respectively).

The results of this thesis show that Neural Networks and Support Vector Machines have similar prediction accuracies. Of major interest could therefore be to compare the effect of different input coding schemes (other than orthogonal coding) on the prediction accuracies.

Appendix : Results

Introduction

Appendix 1 gives a detailed summary of the results obtained from a neural network with different architectures when early stopping is used and also when it is not used. Appendix 2 shows the results obtained using Support Vector Machines. The cost parameter is varied over a range of numbers, while the kernel parameter is fixed at $\gamma = 0.1$. Appendix 3 gives the general Matlab codes that are used for both Neural Networks and Support Vector Machines implementation. The dataset used to achieve the results is given in Appendix 4.

Appendix 1: PSSP with NN

The following notation will be used in the presentation of results in NN. Note that *hu* refers to the hidden units

NN opt: The best classifier for each classification task.

NN 0hu: no hidden layer

NN 1hu: 1 unit in hidden layer

NN 2hu: 2 units in hidden layer

NN 10hu: 10 units in hidden layer

NN 15hu: 15 units in hidden layer

NN 20hu: 20 units in hidden layer

NN 40hu: 40 units in hidden layer

OAtrain: Overall accuracy obtained from training data

OAtest: Overall accuracy obtained from testing data

SEtest: the standard error of OAtest

All the results were obtained by averaging 10 experiments for each classifier in NN. The results are presented as follows:

$H/\sim H$

Results of Neural Networks with early stopping: max fail = 20.

Table 8.2: NN-ES: alpha vs. no alpha

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20hu	NN40hu
OAtrain	0.7958	0.7710	0.7632	0.7735	0.7847	0.7859	0.7884	0.7958
OAtest	0.7466	0.7466	0.7362	0.7392	0.7435	0.7438	0.7460	0.7458
SEtest	0.0042	0.0042	0.0042	0.0045	0.0042	0.0042	0.0042	0.0042
Time	~1min	~1min	~1 min	~2min	~1min	~2min	~2min	~3min

Results of Neural Networks without early stopping.

Table 8.3: NN-WES: alpha vs. no alpha

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20hu	NN40hu
OAtrain	0.9936	0.7723	0.7725	0.804	0.9595	0.9786	0.9853	0.9936
OAtest	0.7463	0.7463	0.7405	0.7329	0.7006	0.6982	0.6968	0.6990
SEtest	0.0042	0.0042	0.0042	0.0043	0.0044	0.0044	0.0044	0.0044
Time	~2min	~2min	~5min	~8min	~10min	~15min	~16min	~26min

$E/\sim E$

Results of Neural Networks with early stopping max fail = 20.

Table 8.4: NN-ES:beta vs. no beta

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20hu	NN40hu
OAtrain	0.8301	0.8160	0.8137	0.8214	0.8276	0.8274	0.8296	0.8301
OAtest	0.7994	0.7994	0.7902	0.7935	0.7964	0.7972	0.7965	0.7959
SEtest	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039
Time	~1min	~1min	~1min	~2min	~1min	~2min	~2min	~3min

Results of Neural Networks without early stopping.

Table 8.5: NN-WES: beta vs. no beta

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20hu	NN40hu
OAtrain	0.9944	0.8178	0.8256	0.8529	0.9672	0.9821	0.9884	0.9944
OAtest	0.7968	0.7968	0.7932	0.7805	0.7452	0.7419	0.7414	0.7401
SEtest	0.0039	0.0039	0.0039	0.0040	0.0042	0.0042	0.0042	0.0042
Time	~3min	~3min	~6min	~9min	~11min	~15min	~16min	~25min

$C/\sim C$

Results of Neural Networks with early stopping max fail = 20.

Table 8.6: NN-ES: coil vs. no coil

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20 hu	NN40hu
OAtrain	0.7398	0.7145	0.6951	0.7196	0.7283	0.7321	0.7333	0.7398
OAtest	0.6865	0.6865	0.6615	0.6817	0.6811	0.6796	0.6820	0.6820
SEtest	~0.0045	0.0045	0.0045	0.0045	0.0045	0.0045	0.0045	0.0045
Time	~1min	~1min	~3min	~2min	~1min	~2min	~2min	~3min

Results of Neural Networks without early stopping.

Table 8.7: NN-WES: coil vs. no coil

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20hu	NN40hu
OAtrain	0.9926	0.7190	0.7191	0.7497	0.9306	0.9658	0.9797	0.9926
OAtest	0.6882	0.6882	0.6865	0.6717	0.6274	0.6259	0.6230	0.6194
SEtest	0.0045	0.0045	0.0045	0.0045	0.0047	0.0047	0.0047	0.0047
Time	~3min	~3min	~5min	~8min	~11min	~15min	~25min	~25min

H/E

Results of Neural Networks with early stopping max fail = 20.

Table 8.8: NN-ES: alpha vs. beta

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20 hu	NN40hu
OAtrain	0.8045	0.7698	0.7059	0.7795	0.7876	0.7988	0.7986	0.8045
OAtest	0.7246	0.7246	0.6573	0.7125	0.7204	0.7216	0.7204	0.7208
SEtest	0.0061	0.0061	0.0063	0.0062	0.0061	0.0061	0.0061	0.0061
Time	~1min	~1min	~1min	~1min	~1min	~1min	~1min	1min

Results of Neural Networks without early stopping.

Table 8.9: NN-WES: alpha vs. beta

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20hu	NN40hu
OAtrain	0.9973	0.7755	0.7951	0.8521	0.9848	0.9923	0.9948	0.9973
OAtest	0.7237	0.7237	0.7012	0.6943	0.6700	0.6662	0.6767	0.6750
SEtest	0.0061	0.0061	0.0062	0.0063	0.0064	0.0065	0.0064	0.0064
Time	~2min	~2min	~3min	~5min	~6min	8min	~9min	~9min

E/C

Results of Neural Networks with early stopping max fail = 20.

Table 8.10: NN-ES: beta vs. coil

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20 hu	NN40hu
OAtrain	0.8236	0.7926	0.7842	0.7990	0.8077	0.8102	0.8114	0.8236
OAtest	0.7685	0.7610	0.7433	0.7554	0.7560	0.7565	0.7578	0.7585
SEtest	0.0049	0.0049	0.0049	0.0049	0.0049	0.0049	0.0049	0.0049
Time	~1min	~1min	~1min	~1min	~1min	~1min	~1min	~2min

Results of Neural Networks without early stopping.

Table 8.11: NN-WES: beta vs. coil

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20hu	NN40hu
OAtrain	0.9957	0.7964	0.8044	0.8469	0.9740	0.9862	0.9911	0.9957
OAtest	0.7616	0.7616	0.7521	0.7401	0.7023	0.6979	0.6966	0.7015
SEtest	0.0049	0.0049	0.0049	0.0050	0.0054	0.0052	0.0052	0.0052
Time	~3min	~3min	~4min	~6min	~8min	~11min	~12min	~18min

H/C

Results of Neural Networks with early stopping max fail = 20.

Table 8.12: NN-ES: alpha vs. coil

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20 hu	NN40hu
OAtrain	0.7960	0.7616	0.7427	0.7607	0.7803	0.7825	0.7840	0.7960
OAtest	0.7311	0.7311	0.7012	0.7243	0.7260	0.7259	0.7265	0.7270
SEtest	0.0048	0.0048	0.0049	0.0048	0.0048	0.0048	0.0048	0.0048
Time	~1min	~1min	~2min	~1min	~1min	~1min	~1min	~2min

Results of Neural Networks without early stopping.

Table 8.13: NN-WES: alpha vs. coil

experiment	NNopt	NN0hu	NN1hu	NN2hu	NN10hu	NN15hu	NN20hu	NN40hu
OAtrain	0.9950	0.7644	0.7716	0.8109	0.9674	0.9838	0.9897	0.9950
OAtest	0.7327	0.7327	0.7270	0.7095	0.6756	0.6743	0.6752	0.6769
SEtest	0.0048	0.0048	0.0048	0.0049	0.0051	0.0051	0.0051	0.0051
Time	~2min	~2min	~4min	~7min	~9min	~12min	~13min	~20min

Appendix 2: PSSP with SVM

The following notation will be used in the presentation of results in SVM. The kernel parameter is constant at $\gamma = 0.1$ Cost parameter C varies and the following values are used to obtain the results: 0.1,0.3,0.5,0.7,0.9,1, and 5.

SVMopt: The best classifier for each classification task.

OAtrain: Overall accuracy during training for each cost parameter.

OAtest: Overall accuracy during testing for each cost parameter.

SE: the standard error of OAtest

All the results were obtained by averaging 10 experiments for each cost parameter in SVM. The results are presented as follows:

$H/\sim H$

Table 8.14: SVM: alpha vs alpha

Experiment	SVMopt	C=0.1	C=0.3	C=0.5	C=0.7	C=0.9	C=1	C=5
OAtrain	1	0.7170	0.7199	0.7911	0.8742	0.9261	0.9434	1
OAtest	0.7535	0.7170	0.7172	0.7284	0.7417	0.7509	0.7531	0.7535
SEtest	0.0028	0.0029	0.0029	0.0029	0.0029	0.0028	0.0028	0.0028
Time	~14min	~14min	~18min	~19min	~21min	~23min	~26min	45min

$E/\sim E$

Table 8.15: SVM:beta vs beta

Experiment	SVMopt	C=0.1	C=0.3	C=0.5	C=0.7	C=0.9	C=1	C=5
OAtrain	1	0.7874	0.7875	0.7919	0.8339	0.8997	0.9261	1
OAtest	0.8015	0.7875	0.7875	0.7876	0.7898	0.7952	0.7979	0.8015
SEtest	0.0026	0.0029	0.0027	0.0026	0.0026	0.0026	0.0026	0.0026
Time	~10min	~10min	~16min	19min	~21min	~23min	~24min	~41min

$C/\sim C$

Table 8.16: SVM:coil vs coil

Experiment	SVMopt	C=0.1	C=0.3	C=0.5	C=0.7	C=0.9	C=1	C=5
OAtrain	1	0.7420	0.8066	0.8549	0.8969	0.9327	0.9476	1
OAtest	0.6881	0.6732	0.6849	0.6873	0.6873	0.6881	0.6865	0.6744
SEtest	0.0030	0.0030	0.0030	0.0030	0.0030	0.0030	0.0030	0.0030
Time	~20min	~22min	~20min	~22min	~24min	~28min	~31min	~57 min

H/E

Table 8.17: SVM:alpha vs beta

Experiment	SVMopt	C=0.1	C=0.3	C=0.5	C=0.7	C=0.9	C=1	C=5
OAtrain	1	0.5978	0.8383	0.8922	0.9236	0.9492	0.9606	1
OAtest	0.7465	0.5848	0.7249	0.7368	0.7414	0.7430	0.7449	0.7465
SEtest	0.0040	0.0045	0.0041	0.0040	0.0040	0.0040	0.0040	0.0040
Time	~6min	~6min	6min	~7min	~7min	~8min	~8min	~13min

E/C

Table 8.18: SVM: beta vs coil

Experiment	SVMopt	C=0.1	C=0.3	C=0.5	C=0.7	C=0.9	C=1	C=5
OAtrain	1	0.7040	0.7235	0.8199	0.8940	0.9319	0.9461	1
OAtest	0.7594	0.7036	0.7077	0.7288	0.7472	0.7558	0.7579	0.7594
SEtest	0.0033	0.0035	0.0035	0.0034	0.0033	0.0033	0.0033	0.0033
Time	~8min	~8min	~10min	~11min	~12min	~13min	~14min	~24min

H/C

Table 8.19: SVM: alpha vs coil

Experiment	SVMopt	C=0.1	C=0.3	C=0.5	C=0.7	C=0.9	C=1	C=5
OAtrain	1	0.6418	0.7835	0.8658	0.9109	0.9411	0.9529	1
OAtest	0.7363	0.6406	0.7012	0.7253	0.7331	0.7350	0.7363	0.7329
SEtest	0.0032	0.0035	0.0033	0.0033	0.0032	0.0032	0.0032	0.0032
Time	~11min	~11min	~12min	~13min	~14min	~16min	18min	30min

Appendix 3: Matlab CODES

Summary of Matlab codes

The Matlab codes used to achieve the results used to compare the performance of Neural Networks and Support Vector Machines in predicting secondary structures of proteins from their primary sequences can be classified into 3 categories; The general functions applicable to both Neural Networks and Support Vector Machines, Neural Networks codes and Support Vector Machine codes. The window size to be used with the codes is 13 and the data used is shown in Appendix 4.

1. function [m] = error_matrixKS(targ,class)

   - This function creates a matrix to show both correct and incorrect classifications during prediction. The code is intended for a three class problem in which case we end up with a 3×3 matrix. In our case, we have created binary classifiers. We therefore have 2×2 error matrices where diagonal elements give correct classification, i.e. residues that are classified as alpha when they are in fact alphas and the off diagonal elements give incorrect classifications i.e. residues classified as alphas though they are not classifiers.

2. function out = orthcoding(letter)

   - The function out = orthcoding(letter) is used to convert letters of the 20 amino acid residues into letters. Also it returns the orthogonal coding for each residue. The inputs to the function are the letters for each sequence and the outputs are the unit vectors (orthogonal coding) for each residue.

3. function targ = target_coding(letter)

   - Secondary structures are reduced from 8 to 3 classes with the DSSP conversion. This function does the same work: it maps the 8 structures to 3 structures of $H, E,$ and, C. Thereafter it returns the targets of the network as vectors: $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$

for class 1, $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ for class 2 and $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ for class 3.

4. function [inputs,targets] = kabsch_sander_orth_coding(s,window_size);

   - This function creates inputs and targets of the neural network from the amino acid sequences and their corresponding secondary structures respectively. It uses the codes from orthcoding and target_coding to create this parameters. For our data, there are 273×10766 inputs and 3×10766 targets for a window size of 13. The window size can be varied. s refers to the data of 62 proteins.

5. nn_cross_val_two_class_one_against_one.m

   - The aim of this script is to produce the prediction overall accuracy on training data and on test data. Standard error estimates on the test data and the time it takes to train a Neural Network will also be monitored.
   - For this script data is read and a learning task is created. Inputs and targets created with function [inputs,targets] = kabsch_sander_orth_coding(s,window_size) are used. Since there are 3 classes for the targets, classes are merged to create two class problems and in that way the following binary classifiers, can be created: $H/\sim H$, $E/\sim E$, $C/\sim C$,H/E,E/C and C/H. Suppose we want to create a binary classifier $H/\sim H$. The first thing that we do to achieve this is to identify targets of class 1,2 and 3 and merge classes 2 and 3 together to for non-alpha class. However, for the classifier H/E, in this case we do not merge the classes, instead we remove the inputs and targets corresponding to class 3 which is coils. They will have the form: $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ to denote class 1, i.e. alpha, and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ for class 2, i.e. no alpha. Therefore, the result is a learning task with two classes H and E.
   - A network is chosen and for our work, we are interested in the following network architectures: a network with no hidden layer, 1 layer, 2,10,20 and 40 units in the hidden layer. The training algorithm used is Resilient Backpropagation. In Matlab this can be identified as trainrp. One of the training parameters used for trainrp is max_fail. This is set to 500 when no early stopping is employed and set to 20 to perform early stopping.

- 10-fold cross validation is used for the network. 1 fold is used as test data and the rest for training. However, for the training data, a further split of 5% is used as validation set.

6. function out_code = protein_code_svm(n);

   - This code is implemented to find targets for Support Vector Machines. It works in a similar way as function targ = target_coding(letter) for Neural Networks. However, for Support Vector Machines, the targets are not vectors but rather $+1$ represents targets belonging to class 1, i.e. alpha while -1 denotes targets belonging to class 2, i.e. no alpha.

7. function [inputs,target_arrayK_H] = kabsch_sander_svm(s,window_size);

   - This function creates inputs and targets of the Support Vector Machines from the amino acid sequences and their corresponding secondary structures respectively. It uses the codes from orthcoding and target_coding to create this parameters. For our data, there are 273×10766 inputs and 1×10766 targets for a window size of 13. The window size can be varied. s refers to the data of 62 proteins.

8. svm_cross_val_two_class1.m

   - The aim of this script is to produce the prediction overall accuracy on training data and on test data using Support Vector Machines. Standard error estimates on the test data and the time it takes to train a Support Vector Machine will also be monitored.
   - For this script data is read and a learning task is created. Inputs and targets created with [inputs,target_arrayK_H] = kabsch_sander_svm(s,window_size) are used; Since there are 3 classes for the targets, the merge is performed in a similar way as in Neural Networks and targets are represented by $+1$ for class 1 and -1 for class 2. In that way we can create our binary classifiers. For Support Vector Machines, the kernel use is the Gaussian Kernel and is denoted by $rbf1$. For this rbf, the kernel parameter (rbf width) is kept constant at $\gamma = 0.1$ and the cost parameters used are: 0.1,0.3,0.5,0.7,0.9,1 and 5.

9. function net = svm(nin, kernel, kernelpar, C, use2norm, qpsolver, qpsize);

   - This function is used to create a Support Vector Machine Classifier. The structure NET has the following fields:

   Basic SVM parameters:

   'type' = 'svm'

   'nin' = number of input dimensions

   'kernel' = kernel function i.e. rbf1

   'kernelpar' = parameters for the kernel function i.e. $\gamma = 0.1$.

   'c' = C Upper bound for the coefficients NET.alpha during training, i.e. $C = 0.1, 0.3, 0.5, 0.7, 0.9, 1$ and 5.

   However, the other parameters are optional and were not used with this function.

10. function [Y, Y1] = svmfwd(net, X)

    - svmfwd is a function used for forward propagation through svm. For a data structure NET obtained from function net = svm(nin, kernel, kernelpar, C, use2norm, qpsolver, qpsize); the matrix of vectors X is used as input into the Support Vector Machine and the matrix of outputs Y are computed.
    - Y is a column vector with one entry for each input vector in X. Y(i) is the SVM output for input vector X(i,:), it is +1, if X(i,:) is classified as belonging to class 1 and -1, if X(i,:) is classified as belonging to class 2.

11. function K = svmkernel(net, X1, X2)

    - This function computes svm kernel function. The Support Vector Machine structure NET must contain 2 field NET.kernel and NET.kernelpar, selecting the kernel function and its parameters. K will be a matrix and K(i,j) is the result of the kernel function for inputs X1(i,:) and X2(j,:).

12. function net = svmtrain(net, X, Y, alpha0, dodisplay)

    - svmtrain - train a svm using decomposition algorithm . The structure net, has the following parameters:

'X' = training data X

'Y' = target values.

'NET' = the net structure indicated here was obtained with the function in 9.

    - The other parameters in the function were not used.

General codes that are used for both NN and SVM implementation are given below:

```
function [m] = error_matrixKS(targ,class)
% Misclassifications
% inputs:  classifications on helix, strand (sheet) or coil
% outputs:  samples classified into correct classes and wrong classes, i.e.  samples that
are helices
% yet classified as coil.
% This is a general code for a three class problem and can be extended to more than
% three classes or less, i.e.  two classes.


helix_index = find(targ == 1);
sheet_index = find(targ == 2);
coil_index = find(targ == 3);


helix_helix = length(find(class(helix_index) == 1));
helix_sheet = length(find(class(helix_index) == 2)); % a12 is where targets are of class
1 but
                      % classification is of class 2
helix_coil = length(find(class(helix_index) == 3));


sheet_helix = length(find(class(sheet_index) == 1));
sheet_sheet = length(find(class(sheet_index) == 2));
sheet_coil = length(find(class(sheet_index) == 3));


coil_helix = length(find(class(coil_index) == 1));
coil_sheet = length(find(class(coil_index) == 2));
coil_coil = length(find(class(coil_index) == 3));
```

```
m = [helix_helix, helix_sheet, helix_coil;

sheet_helix, sheet_sheet, heet_coil;

coil_helix, coil_sheet, coil_coil];
```

```
function out = orthcoding(letter)


% the letters are the coding of the 20 amino acids
% returns the orthogonal coding of each letter.
% inputs:  the sequence of amino acids
% output:  the unit vector of each amino acid
out = zeros(21,1);
if letter == 'A'
out(1) = 1;
elseif letter == 'C'
out(2) = 1;
elseif letter == 'D'
out(3) = 1;
elseif letter == 'E'
out(4) = 1;
elseif letter == 'F'
out(5) = 1;
elseif letter == 'G'
out(6) = 1;
elseif letter == 'H'
out(7) = 1;
elseif letter == 'I'
out(8) = 1;
elseif letter == 'K'
out(9) = 1;
elseif letter == 'L'
out(10) = 1;
```

```
elseif letter == 'M'
out(11) = 1;
elseif letter == 'N'
out(12) = 1;
elseif letter == 'P'
out(13) = 1;
elseif letter == 'Q'
out(14) = 1;
elseif letter == 'R'
out(15) = 1;
elseif letter == 'S'
out(16) = 1;
elseif letter == 'T'
out(17) = 1;
elseif letter == 'V'
out(18) = 1;
elseif letter == 'W'
out(19) = 1;
elseif letter == 'Y'
out(20) = 1;
else
out(21) = 1;
end
```

```
function [inputs,targets] = kabsch_sander_orth_coding(s,window_size);


% creates some inputs and targets for the network
% Usage:  [inputs,target_arrayK_H] = kabsch_sander_nn(s)
%
% Parameters:  inputs - network inputs window size*21*length(s)/3
%       targets - network targets (1,0,0),(0,1,0),(0,0,1)
%       s - structure with character arrays of data


start = ones(1,(window_size-1)/2)*42; % It creates a window in which the first amino acid
can be predicted for.
start = char(start);% i.e *******AFGTV******** for a protein AFGTV since we predict for
         the centrally located amino acid G.
index_prime = 2:3:length(s); % 2,5,8,...


inputs = []; % array is empty


for seq = 1:  length(index_prime)


seq8 = s{index_prime(seq)};
seq8 = [start,seq8,start];
% start with s{index_prime(1)} = s{2}
% then s{index_prime(2)} = s{5} ...


for i=1:length(seq8)-window_size+1
  window = seq8(i:i+window_size-1);
  for k=1:length(window)
```

```
   result8(k,:)  = orthcoding(window(k))';
  end
  input_i8 = reshape(result8',window_size*21,1);
  input8(:,i) = input_i8; % input array for seq8 (current sequence)
end


inputs = [inputs,input8]; % hang input8 at the end to input array
clear input8 % clear the variable input8 (may have different number of columns each time)
end


index_secondary = 3:3:length(s); % 3,6,9,...


targets = []; % array is empty


for seqt = 1:length(index_secondary)
target_sec = s{index_secondary(seqt)}; % start with s{index_prime(1)} = s{3}
                  % then s{index_prime(2)} = s{6} ...


  for i=1:length(target_sec)
  target_sec_output(i,:)  = target_coding(target_sec(i))';
    % target coding in which a vector of three elements is determined
end
targets = [targets,target_sec_output']; % allows to add each residue without overwriting
the others
clear target_sec_output
end
return
```

```
function targ = target_coding(letter)


% The function gives the orthogonal coding of the secondary structure of the

% amino acids.  The structures have been reduced from 8 to 3 classes through the following mapping:

% H,G to H

% B,E to E

% the rest i.e.  C,S,T to C


targ = zeros(3,1);

if letter == 'H'

targ(1) = 1;

elseif letter == 'G' % if we have G and H then we have a helix which is H

targ(1) = 1;

elseif letter == 'B'

targ(2) = 1;

elseif letter == 'E' % If we have B and E then we have sheet which is E

targ(2) = 1

else

targ(3) = 1; % If we have anything other than H and B for instance I,S,T then we have coil

end
```

```
% nn_cross_val_two_class_one_against_one.m


% nn_cross_val_two_class_one_against_one is used to perform 10-fold cross-validation

% for feed-forward neural network with a hidden layer with different hidden units and
the following output coding:  (1,0) for alpha, (0,1) for non-alpha.

% where we have a binary classifier i.e.  alpha against beta, coils are excluded from
the data.

% Reading in the data


%clear all

close all


diary F:\nn_pract1_beta_against_no_betas_kabch_exp1.txt % specifies the path in which
each result is stored.


disp('************************************************************')

disp('************************************************************')


text = ['experiment starts:  ',num2str(fix(clock))];


disp(text)

disp('')

data_file = 'D:\Thesis - experiments_november_files\Kabsch_and_Sander_edit_new1.txt'

s = importdata(data_file);


window_size = 13 % can be varied but must be an odd number

K = 10 % number of folds in K-fold cross-validation
```

```
merge_classes = [1,3] % classes which are identified, tested against the missing one
single_class = setdiff(1:3,merge_classes)
%*********************************************************************
%network parameters
epoch_number = 500 % 200 epochs for training
%method = 'trainscg' % method for training (change this here, e.g.
%method = 'traingd'; %for gradient descent
method = 'trainrp' %for resilient gradient descent
%method = 'trainlm'; % not possible:  out of memory!!!!!!!!!
%method = 'traingda'; % steepest descent with adaptive learning rate
%method = 'traingdx';


architecture = [1,2]
trans_functions = {'logsig','logsig'}
max_fail = 20 % epochs for non-improvement on validation set
% use max_fail = 20 for early stopping
% max_fail = 500:  no early stopping
p = 5 % percentage of the validation set from the training set


%___________________________________
%rand_thesisKS = randperm(10766);
%___________________________________
%Parameters:
% inputs - inputs for the network
% targets - targets for the network
% inputstrain - Training inputs
% target_arrayK_Htrain - Training targets
```

```
% inputstest - Testing inputs
% target_arrayK_Htest - Testing targets


% Creating a learning task


disp('create inputs and targets...')
[inputs,targets] = kabsch_sander_orth_coding(s,window_size);
[n,m] = size(targets)


targets2 = zeros(n-1,m);
targets2(1,:)  = targets(single_class,:);
targets2(2,:)  = sum(targets(merge_classes,:),1);
targets = targets2;


%-------------------------------------------------------------------------
% creating inputs and targets for binary classifier for either one class
% or another i.e.  alpha or beta
% classes are not merged in this case
% the new inputs and targets replaces the ones obtained from above.
% this can be done for different binary classifiers.
% the following commands can replace the ones that have been obtained through merging.
[n,m] = size(targets)
coil_index = find(targets(3,:)==1); % to remove targets that are coil from the targets
alpha_beta_index = setdiff(1:m,coil_index);
inputs = inputs(:,alpha_beta_index); % new inputs excluding those whose targets are coil
targets = targets([1,2],alpha_beta_index); % new targets with no coils
[n,m] = size(targets)
```

```
%_____________________________________________________________________


rand_thesisKS = randperm(m);

m = length(rand_thesisKS)

%_____________________________________________________________________
% The objective is to perform early stopping on the training set.  The data

% is divided into training set, validation set and test set.


fold_number = round(m/K); % number of samples per fold


for k = 1:K

disp('_____________________________________________________________')

disp(['fold ',num2str(k),' in progress'])

%test_index{k} = rand_thesisKS((k-1)*1077+1:min([(1077)*k,10766])); % avoid the case k=10

test_index{k} = rand_thesisKS((k-1)*fold_number+1:min([(fold_number)*k,m])); % avoid the
case k=10

train_index{k} = setdiff(rand_thesisKS,test_index{k});


rand_index = randperm(length(train_index{k})); % permute the training

                    % index in order to randomly select

                    % p percent as

                    % validation index


val_index{k} = train_index{k}(rand_index(1:round(p*length(train_index{k})/100)));

new_train_index{k} = setdiff(train_index{k},val_index{k}); % remaining samples as training
set
```

```
inputs_val = inputs(:,val_index{k}); % inputs
inputs_train = inputs(:,new_train_index{k});
inputs_test = inputs(:,test_index{k});

targets_train = targets(:,new_train_index{k}); % targets
targets_val = targets(:,val_index{k});
targets_test = targets(:,test_index{k});

val.P = inputs_val;
val.T = targets_val;

mmx = minmax(inputs);
% network formulation and training

net = newff(mmx,architecture,trans_functions,method);
net.trainParam.epochs = epoch_number;
net.trainParam.max_fail = max_fail; % 5 is Matlab's default
net = init(net);

net = train(net,inputs_train,targets_train,[],[],val); % used in early stopping
% net = train(net,inputs_train,targets_train)    % used in without early stopping
ytrain = sim(net,inputs_train);

[mtrain,class_train] = max(ytrain,[],1); % classification
[ttrain,targ_train] = max(targets_train,[],1); % create targets with labels 1,2,3
alphas_train = length(find(targ_train == 1));
betas_train = length(find(targ_train == 2));
```

```
coils_train = length(find(targ_train == 3));
percent_train = [alphas_train,betas_train,coils_train]/(alphas_train+betas_train+coils_train)


var_train(k) = var(class_train ~= targ_train,1);
N_train(k) = length(targ_train); % number of samples in fold


% create error matrix for the overall accuracy of the training data
error_matrix_train = error_matrixKS(targ_train,class_train)
% overall accuracy
OAtrain(k) = sum(diag(error_matrix_train))/sum(sum(error_matrix_train))


% class accuracies (producer's and user's accuracy)
for class=1:3 % 3 classes
  UAtrain(class) = error_matrix_train(class,class)/sum(error_matrix_train(:,class));
  PAtrain(class) = error_matrix_train(class,class)/sum(error_matrix_train(class,:));
end
PAtrain_matrix(:,k) = PAtrain';
UAtrain_matrix(:,k) = UAtrain';
PAtrain % display producer's accuracy
UAtrain % display user's accuracy
%-------------------------------------------------------------------
% simulation on validation data
yval = sim(net,inputs_val);
[mval,class_val] = max(yval,[],1);
[tval,targ_val] = max(targets_val,[],1);
alphas_val = length(find(targ_val == 1));
betas_val = length(find(targ_val == 2));
```

```
coils_val = length(find(targ_val == 3));
percent_val = [alphas_val,betas_val,coils_val]/(alphas_val+betas_val+coils_val)


var_val(k) = var(class_val ~= targ_val,1);
N_val(k) = length(targ_val); % number of samples in fold
% create error matrix for the overall accuracy of the validation data
error_matrix_val = error_matrixKS(targ_val,class_val)
% overall accuracy
OAval(k) = sum(diag(error_matrix_val))/sum(sum(error_matrix_val))
% class accuracies
for class=1:3 % 3 classes
  UAval(class) = error_matrix_val(class,class)/sum(error_matrix_val(:,class));
  PAval(class) = error_matrix_val(class,class)/sum(error_matrix_val(class,:));
end
PAval_matrix(:,k) = PAval';
UAval_matrix(:,k) = UAval';
PAval % display producer' accuracy
UAval % display user's accuracy
%-------------------------------------------------------------------
% simulation on test data


ytest = sim(net,inputs_test);
[mtest,class_test] = max(ytest,[],1);
[ttest,targ_test] = max(targets_test,[],1);
alphas_test = length(find(targ_test == 1));
betas_test = length(find(targ_test == 2));
coils_test = length(find(targ_test == 3));
```

```
percent_test = [alphas_test,betas_test,coils_test]/(alphas_test+betas_test+coils_test)


var_test(k) = var(class_test ~= targ_test,1);
N_test(k) = length(targ_test); % number of samples in fold
% create error matrix for the overall accuracy of the test data
error_matrix_test = error_matrixKS(targ_test,class_test)
% overall accuracy
OAtest(k) = sum(diag(error_matrix_test))/sum(sum(error_matrix_test))
% class accuracies
for class=1:3 % 3 classes
  UAtest(class) = error_matrix_test(class,class)/sum(error_matrix_test(:,class));
  PAtest(class) = error_matrix_test(class,class)/sum(error_matrix_test(class,:));
end
PAtest_matrix(:,k) = PAtest';
UAtest_matrix(:,k) = UAtest';
PAtest % display producer' accuracy
UAtest % display user's accuracy
end


OA_est_train = mean(OAtrain)
OA_est_val = mean(OAval)
OA_est_test = mean(OAtest)


SE_train1 = sqrt(sum(N_train.*var_train)/m/m)
SE_val1 = sqrt(sum(N_val.*var_val)/m/m)
SE_test1 = sqrt(sum(N_test.*var_test)/m/m)
```

```
PA_est = mean(PAtest_matrix,2)'

UA_est = mean(UAtest_matrix,2)'


text = ['experiment ends:  ',num2str(fix(clock))]

diary off

%fclose(fid); %close the logfile
```

Note:

For a classifier One-Against-One i.e. H/E in which case we have alphas or betas, the targets and inputs are created slightly differently from the ones created with **nn_cross_val_two_class_one_against_one.m**. The reason is that inputs and targets that are coils, such as in this case, are excluded and the new inputs whose targets can only be either alpha or beta are used to obtain the results.

A Matlab codes for implementing SVM is freely available at:
`http://ida.first.fraunhofer.de/~anton/softwar.html`. These codes are used to obtain the results in Chapter 7, Appendix 1 . The are as follows:

```
%Support Vector Machine toolbox
% Version 2.51, January 2002
%
% SVM functions:
% svm - create a support vector machine classifier
% svmfwd - forward propagation through svm
% svmkernel - compute svm kernel function
% svmtrain - train a svm using decomposition algorithm
%
% Files for PRLOQO code:
% loqo.c (C) by Steve Gunn
% pr_loqo.c (C) by Alex Smola
% pr_loqo.h (C) by Alex Smola
%
```

The Support Vector Machine implementation is given below:

```
function out_code = protein_code_svm(n);


% It converts the secondary structure outputs to 3 classes of labels 1,2,3.
% n is the output from the targets of training and test data.


if n == 'H'
out_code = 1;
elseif n== 'G' % if we have G and H then we have a helix which is H
out_code = 1;
elseif n == 'B'
out_code = 2;
elseif n =='E' % If we have B and E then we have sheet which is E
out_code = 2;
else
out_code = 3; % If we have anything other than H and B for instance C,S,T then it is coil
end
```

```
function [inputs,target_arrayK_H] = kabsch_sander_svm(s,window_size);

% creates inputs and targets for the svm approach

start = ones(1,(window_size-1)/2)*42; %
start = char(start);

index_prime = 2:3:length(s); % 2,5,8,...

inputs = []; % array is empty
for seq = 1:  length(index_prime)

seq8 = s{index_prime(seq)};
seq8 = [start,seq8,start];
% start with s{index_prime(1)} = s{2}
% then s{index_prime(2)} = s{5} ...

for i=1:length(seq8)-window_size+1
   window = seq8(i:i+window_size-1);
  for k=1:length(window)
    result8(k,:)  = orthcoding(window(k))';
  end
  input_i8 = reshape(result8',window_size*21,1);
  input8(:,i) = input_i8; % input array for seq8 (current sequence)
end

inputs = [inputs,input8]; % hang input8 at the end to input array
clear input8 % clear the variable input8 (may have different number of columns each time)
```

```
end


index_secondary = 3:3:length(s); % 3,6,9,...


target_arrayK_H = []; % array is empty


for seqt = 1:length(index_secondary)
target_sec = s{index_secondary(seqt)}; % start with s{index_prime(1)} = s{3}
% then s{index_prime(2)} = s{6} ...
for i=1:length(target_sec)
  target_sec_output(i,:)  = protein_code_svm(target_sec(i))';
end
target_arrayK_H = [target_arrayK_H,target_sec_output'];
clear target_sec_output
end
```

```
% svm_cross_val_two_class1.m


% classify the Protein data (3 classes) using svm

% select optimal cost parameter C on the first fold (otherwise too time

% consuming)

% Reading in the data


clear all

close all


diary F:\svm_pract1_alpha_against_no_alphas_kabsch_exp6.txt % a path in which the results
are stored


disp('*********************************************************')

disp('*********************************************************')

text = ['experiment starts:  ',num2str(fix(clock))];


disp(text)

disp('')


data_file = 'D:\Thesis - experiments_november_files\Kabsch_and_Sander_edit_new1.txt'


s = importdata(data_file); % load data file


window_size = 13 % can be varied but must be an odd number
```

```
K = 10 % K folds for cross-validationAll the results were obtained by averaging 10 experiments
for each classifier in NN or for each cost parameter in SVM.

p = 5 % percentage used for validation set

merge_classes = [2,3] % classes which are identified, tested against the missing one

single_class = setdiff(1:3,merge_classes)

%method = 'linear'

%method = 'poly'

method = 'rbf1'

% determine optimal parameters using validation set

Q = 0.1 % width for rbf

C = [0.1] % weight factor for the sum of slacks

bestC = C(1); % in case of no optimization


disp('create inputs and targets...')

[inputs,targets] = kabsch_sander_svm(s,window_size); % target_array in this case is a
1*10766 data

[n,m] = size(targets)


index_single = find(targets == single_class);

index_merge = setdiff(1:m,index_single);


targets(index_single) = 1; % binary targets for svm

targets(index_merge) = -1;

%_____________________________________________________________________________

% creating inputs and targets for binary classifier for either one class

% or another i.e.  alpha or beta

% classes are not merged in this case

% the new inputs and targets replaces the ones obtained from above.
```

```
% this can be done for different binary classifiers.
% The following commands can replace the ones that have been obtained through merging.
% coil_index = find(targets(1,:)==3);
% alpha_beta_index = setdiff(1:m,coil_index);
% inputs = inputs(:,alpha_beta_index);
% targets = targets(:,alpha_beta_index);
% [n,m] = size(inputs)
%_____________________________________________________________________________
[d,N] = size(inputs) % d = dimension = windowize*21


rand_thesisKS = randperm(m);
m = length(rand_thesisKS)


fold_number = round(m/K); % number of samples per fold


for k = 1:K
disp('_____________________________________________________________')
disp(['fold ',num2str(k),' in progress'])
disp(['time:  ',num2str(fix(clock))]) % time


%test_index{k} = rand_thesisKS((k-1)*1077+1:min((1077)*k,10766)); % avoid the case k =
10
test_index{k} = rand_thesisKS((k-1)*fold_number+1:min([(fold_number)*k,m])); % avoid the
case k=10
train_index{k} = setdiff(rand_thesisKS,test_index{k});


rand_index = randperm(length(train_index{k})); % permute the training index in order
```

```
% to randomly select
% p percent as validation index
val_index{k} = train_index{k}(rand_index(1:round(p*length(train_index{k})/100)));
new_train_index{k} = setdiff(train_index{k},val_index{k}); % remaining samples as training
set
inputs_val = inputs(:,val_index{k})'; % each sample must be a row
inputs_train = inputs(:,new_train_index{k})';
inputs_test = inputs(:,test_index{k})';
targets_train = targets(:,new_train_index{k})'; % targets
targets_val = targets(:,val_index{k})';
targets_test = targets(:,test_index{k})';


% determine optimal parameters on first fold (otherwise it takes too
% long)
if k == 0 % no optimization of C
%_____________________________________________________________________
   clear OAcheck
   clear SE_check
  for cc = 1:length(C)
    disp(['try ',num2str(C(cc)),'...'])
    check_index = new_train_index{k}(1:5:end); % 20 percent for parameter selection
    inputs_check = inputs(:,check_index)';
    targets_check = targets(:,check_index)';
    net = svm(d,method,Q,C(cc)); % create a svm
    net = svmtrain(net,inputs_check,targets_check)
    pred_val = svmfwd(net,inputs_val);
    index1 = find(pred_val == 1);
    index2 = find(pred_val == -1);
```

```
   pred_val(index1) = 1;
   pred_val(index2) = 2;
   index1 = find(targets_val == 1);
   index2 = find(targets_val == -1);
   targets_val(index1) = 1;
   targets_val(index2) = 2;
   alphas_val = length(find(targ_val == 1));
   betas_val = length(find(targ_val == 2));
   coils_val = length(find(targ_val == 3));
percent_val = [alphas_val,betas_val,coils_val]/(alphas_val+betas_val+coils_val)
   check_val = find(pred_val~=targets_val);
   pred_val = pred_val';
   targ_val = targets_val';
   var_val = var(pred_val ~= targ_val,1);
% creates an error matrix of dimension 3x3
   error_matrix_val = error_matrixKS(targ_val,pred_val);
% overall accuracy
    OAcheck(cc) = sum(diag(error_matrix_val))/sum(sum(error_matrix_val)) % proportion
of
misclassifications.
   SE_check(cc) = sqrt(var_val/length(targ_val))
%--------------------------------------------------------------------
  end
% determine optimal architecture
  opt_index = find(OAcheck == max(OAcheck)); % lowest error rate decides
  opt = opt_index(1); % OSE rule
% for 1SE rule:  two additional lines of code
%if one_se_rule == 1
```

```
  opt_index = find(OAcheck >= OAcheck(opt)-SE_check(opt));
  opt = opt_index(1)
%end
  bestC = C(opt)
end
%-------------------------------------------------------------------------
%learner = svm(d,method,bestQ,bestC);
%learner = svm(d,method,Q(qq),Carray_train,0,'loqo');
if k>1
   alpha0 = net.alpha;
   net = svm(d,method,Q,bestC,alpha0);
else
   net = svm(d,method,Q,bestC);
end


warning off;
net = svmtrain(net,inputs_train,targets_train);
%-------------------------------------------------------------------
% training set
pred_train = svmfwd(net,inputs_train);
index1 = find(pred_train == 1);
index2 = find(pred_train == -1);
pred_train(index1) = 1;
pred_train(index2) = 2;
targ_train = targets_train;
index1 = find(targ_train == 1);
index2 = find(targ_train == -1);
```

```
N_train(k) = length(targ_train); % number of samples in fold
error_matrix_train = error_matrixKS(targ_train,pred_train)
% overall accuracy
OAtrain(k) = sum(diag(error_matrix_train))/sum(sum(error_matrix_train))
% class accuracies (producer's and user's accuracy)
for class=1:3 % 3 classes
targ_train(index1) = 1;
targ_train(index2) = 2;
alphas_train = length(find(targ_train == 1));
betas_train = length(find(targ_train == 2));
coils_train = length(find(targ_train == 3));
percent_train = [alphas_train,betas_train,coils_train]/(alphas_train+betas_train+coils_train)
prot = find(pred_train~=targ_train);
var_train(k) = var(pred_train ~= targ_train,1);
N_train(k) = length(targ_train); % number of samples in fold
error_matrix_train = error_matrixKS(targ_train,pred_train)
% overall accuracy
OAtrain(k) = sum(diag(error_matrix_train))/sum(sum(error_matrix_train))
% class accuracies (producer's and user's accuracy)
for class=1:3 % 3 classes
  UAtrain(class) = error_matrix_train(class,class)/sum(error_matrix_train(:,class));
  PAtrain(class) = error_matrix_train(class,class)/sum(error_matrix_train(class,:));
end
PAtrain_matrix(:,k) = PAtrain';
UAtrain_matrix(:,k) = UAtrain';
PAtrain % display producer's accuracy
UAtrain % display user's accuracy
%_____________________________________________________________________
```

```
%validation set
pred_val = svmfwd(net,inputs_val);
index1 = find(pred_val == 1);
index2 = find(pred_val == -1);
pred_val(index1) = 1;
pred_val(index2) = 2;
targ_val = targets_val;
index1 = find(targ_val == 1);
index2 = find(targ_val == -1);
targ_val(index1) = 1;
targ_val(index2) = 2;
alphas_val = length(find(targ_val == 1));
betas_val = length(find(targ_val == 2));
coils_val = length(find(targ_val == 3));
percent_val = [alphas_val,betas_val,coils_val]/(alphas_val+betas_val+coils_val)
prot = find(pred_val~=targ_val);
var_val(k) = var(pred_val ~= targ_val,1);
N_val(k) = length(targ_val); % number of samples in fold
error_matrix_val = error_matrixKS(targ_val,pred_val)
% overall accuracy
OAval(k) = sum(diag(error_matrix_val))/sum(sum(error_matrix_val))
% class accuracies (producer's and user's accuracy)
for class=1:3 % 3 classes
  UAval(class) = error_matrix_val(class,class)/sum(error_matrix_val(:,class));
  PAval(class) = error_matrix_val(class,class)/sum(error_matrix_val(class,:));
end
PAval_matrix(:,k) = PAval';
UAval_matrix(:,k) = UAval';
```

```
PAval % display producer's accuracy

UAval % display user's accuracy

%_____________________________________________________________________

%test set

pred_test = svmfwd(net,inputs_test);

index1 = find(pred_test == 1);

index2 = find(pred_test == -1);

pred_test(index1) = 1;

pred_test(index2) = 2;

targ_test = targets_test;

index1 = find(targ_test == 1);

index2 = find(targ_test == -1);

targ_test(index1) = 1;

targ_test(index2) = 2;

alphas_test = length(find(targ_test == 1));

betas_test = length(find(targ_test == 2));

coils_test = length(find(targ_test == 3));

percent_test = [alphas_test,betas_test,coils_test]/(alphas_test+betas_test+coils_test)

%post processing of test data

prot = find(pred_test~=targ_test);

var_test(k) = var(pred_test ~= targ_test,1);

N_test(k) = length(targ_test); % number of samples in fold

error_matrix_test = error_matrixKS(targ_test,pred_test)

% overall accuracy

OAtest(k) = sum(diag(error_matrix_test))/sum(sum(error_matrix_test))

% class accuracies (producer's and user's accuracy)

for class=1:3 % 3 classes

 UAtest(class) = error_matrix_test(class,class)/sum(error_matrix_test(:,class));
```

```
 PAtest(class) = error_matrix_test(class,class)/sum(error_matrix_test(class,:));
end
PAtest_matrix(:,k) = PAtest';
UAtest_matrix(:,k) = UAtest';
PAtest % display producer's accuracy
UAtest % display user's accuracy
%fix(clock)
end
% cross-validation
% need to work out accuracies according to cross validation and standard
% error
OA_est_train = mean(OAtrain)
OA_est_val = mean(OAval)
OA_est_test = mean(OAtest)
SE_train = sqrt(sum(N_train.*var_train)/m/m)
SE_val = sqrt(sum(N_val.*var_val)/m/m)
SE_test = sqrt(sum(N_test.*var_test)/m/m)
PA_est = mean(PAtest_matrix,2)'
UA_est = mean(UAtest_matrix,2)'
text = ['experiment ends:  ',num2str(fix(clock))]
diary off
```

Appendix 4: Dataset

62 proteins

1.Acprotease

GVGTVPMTDYGNDVEYYGQVTIGTPGKSFNLNFDTGSSNLWVGSVQCQASGCKGGRDKFNPSDGSTFKATGYDASIGYGDGSASGVLGY
DTVQVGGIDVTGGPQIQLAQRLGGGGFPGDNDGLLGLGFDTLSITPQSSTNAFQDVSAQGKVIQPVFVVYLAASNISDGDFTMPGWIDN
KYGGTLLNTNIDAGEGYWALNVTGATADSTYLGAIFQAILDTGTSLLILPDEAAVGNLVGFAGAQDAALGGFVIACTSAGFKSIPWSIY
SAIFEIITALGNAEDDSGCTSGIGASSLGEAILGDQFLKQQYVVFDRDNGIRLAPVA

CTTCCCCCBCTTSSCBBCCEECSSSCCEECCEEESSCCCBEECBSCCCSSSSCSSCCCBCTTTCSSCCCCCCBCCCCCSSCCCEEBCCC
CCCBSSSCBCCSCTTCEEEEECSSSSSSSSCSSEEECSCSSSCSSSSSCCCHHHHHHHSSSCSSSCCCCEEECTTTCEEECCSSCCCTT
SBCSCCCCBCCCCSSSSCCEECCCCCSSSSCCCCSCEECCCTTSSSEECCSSTTTGGGTTSCCCCCSSSSCCCBCSCTTSCCCCCCBSS
SCBCCCSSSSCCCCSSSCCCCCCCCCSSSSBEECHHHHTTBCCEEETTTEEECCBBC

2.Aproteinase

AASGVATNTPTANDEEYITPVTIGGTTLNLNFDTGSADLWVFSTELPASQQSGHSVYNPSATGKELSGYTWSISYGDGSSASGNVFTDS
VTVGGVTAHGQAVQAAQQISAQFQQDTNNDGLLGLAFSSINTVQPQSQTTFFDTVKSSLAQPLFAVALKHQQPGVYDFGFIDSSKYTGS
LTYTGVDNSQGFWSFNVDSYTAGSQSGDGFSGIADTGTTLLLLDDSVVSQYYSQVSGAQQDSNAGGYVFDCSSSVPDFSVSISGYTATV
PGSLINYGPSGNGSTCLGGIQSNSGIGFLIFGDIFLKSQYVVFDSDGPQLGFAPQA

CBCEEEEEEECTTSCCEEEEEEETTEEEEEEEESSCCCEEECBSSSCHHHHTTSCCBCHHHHCEEEEEEEEEECTTSCEEEEEEEEEE
EEETTEEEEEEEEECSEECHHHHHCTTCSEEEECSCGGGCCBSSSCCCCHHHHHTTTBSSSEEEEECCSSSCEEEEESCCCGGGBSSC
CEEEECCGGGTSCBCCEEEEEETTEEECCCCEEECTTCSSEEECHHHHHHHHTTCSSEEEETTTTEEEECTTSCCCCEEEEETTEEEEE
CGGGGEEEEETTTTEEEESEEECCSCSSEEECHHHHTTBCEEEETTTTEEEECBBC

3.Actinidin

LPSYVDWRSAGAVVDIKSQGECGGCWAFSAIATVEGINKITSGSLISLSEQELIDCGRTQNTRGCDGGYITDGFQFIINDGGINTEENY
PYTAQDGDCDVALQDQKYVTIDTYENVPYNNEWALQTAVTYQPVSVALDAAGDAFKQYASGIFTGPCGTAVDHAIVIVGYGTEGGVDYW
IVKNSWDTTWGEEGYMRILRNVGGAGTCGIATMPSYPVKY

CCSCEEGGGGTCCCCCCBCTTSCCHHHHHHHHHHHHHHHHHHSCCCCBCHHHHHHHCCBTTBCGGGCCCHHHHHHHHHHHTCBCBTTTS
CCCSSCCCCCHHHHHCCCBCCCEEEECCTTCHHHHHHHHHHSCEEEEECCCSHHHHHCCSSEECCCCCSSCCEEEEEEEEEEETTEEEE
EEECSBCTTSTBTTEEEEECCCCTTCGGGTTSSCEEEECC

4.Alpha Lytic protease

ANIVGGIEYSINNASLCSVGFSVTRGATKGFVTAGHCGTVNATARIGGAVVGTFAARVFPGNDRAWVSLTSAQTLLPRVANGSSFVTVR
GSTEAAVGAAVCRSGRTTGYQCGTITAKNVTANYAEGAVRGLTQGNACMGRGDSGGSWITSAGQAQGVMSGGNVQSNGNNCGIPASQRS
SLFERLQPILSQYGLSLVTG

CCCCBTCEEEESSSSEEECCCEEEETTEEEEECCSSSSCTTCEEEETTEEEEEEEEEECSBCCEEEEEECSSSCCCCEEEETTEEEECC
BCCCCCTTCCCEEEETTTEEEECCEEEEEEEEECSSSEEEEEEEECSCCBTTCTTCEEECTTCBCCEEEEEECCCTTSBSSSSCGGGCC
EEEEESHHHHHHHTCEECCC

5.Arabinose binding protein

ENLKLGFLVKQPEEPWFQTEWKFADKAGKDLGFEVIKIAVPDGEKTLNAIDSLAASGAKGFVICTPDPKLGSAIVAKARGYDMKVIAVD
DQFVNAKGKPMDTVPLVMMAATKIGERQGQELYKEMQKRGWDVKESAVMAITANELDTARRRTTGSMDALKAAGFPEKQIYQVPTKSND
IPGAFDAANSMLVQHPEVKHWLIVGMNDSTVLGGVRATEGQGFKAADIIGIGI

CCCCCCEEECCCSSTTHHHHHHHHHHHHHHHSSSCCCEEECCSHHHHHHHHHHHHHTCCCCCCBCCSCSSCTTHHHHHHHHHCCCCCBCS
SCCCSTTCCCCCSSCCCCBCSHHHHHHHHHHHHHHHHHHHHTCCSTTCCEEECCCCSSGGGHHHHHHHHHHHHHHSCCSSSCEECCCSSSS
SHHHHHHHHHHHHHCCSCCSCEEECSSSTTTHHHHHHHHSSSCCSTTCCCCEE

6.Avian polypeptide

GPSQPTYPGDDAPVEDLIRFYDNLQQYLNVVTRHRY

CCCCCCCCCTTSCHHHHHHHHHHHHHHHHHHHTTCCC

7.Azurin

SVDIQGNDQMQFNTNAITVDKSCKQFTVNLSHPGNLPKNVMGHNWVLSTAADMQGVVTDGMASGLDKDYLKPDDSRVIAHTKLIGSGE
KDSVTFDVSKLKEGEQYMFFCTFPGHSALMKGTLTLK

CEEEECCSSSCCSCSSEEECSSSSEEEEEEECCSSCCTTTSCBCCEEEETTTTHHHHHHHHHHHHHHHSSCSSCTTCSBCCCCBCTTC
CEEEEEESSSCCSSCCEEEECCSTTTTTTSEEEEEEC

8.Bence Jones dimer M

EGDIQMTQSPSSLSASVGDRVTITCQASQDIIKYLNWYQQTPGKAPKLLIYEASNLQAGVPSRFSGSGSGTDYTFTISSLQPEDIATY
YCQQYQSLPYTFGQGTKLQI

TCCCEEEECSEEEECTTCCEEEEEEESSCCTTCEEEEEECTTSCCEEEEETTTEECTTCCTTEEEEEETTEEEEEESSCCGGGCSEEE
EEECSSSSCCBCCCEEEEEC

9.Beta trypsin

IVGGYTCGANTVPYQVSLNSGYHFCGGSLINSQWVVSAAHCYKSGIQVRLGEDNINVVEGNEQFISASKSIVHPSYNSNTLNNDIMLI
KLKSAASLNSRVASISLPTSCASAGTQCLISGWGNTKSSGTSYPDVLKCLKAPILSNSSCKSAYPGQITSNMFCAGYLQGGKDSCQGD
SGGPVVCSGKLQGIVSWGSGCAQKNKPGVYTKVCNYVSWIKQTIASN

CCSCEECCTTSSTTEEEEESSSEEEEEEEEETTEEEECHHHCCSSCEEEESCSSTTSCCSCCEEEEEEEEEECTTCCTTTCTTCCEEE
EESSCCCCCSSSCCCCCCSSCCCTTCEEEEEESCCCCSSSCCCSSCCEEEEEECCCHHHHHHHSTTTCCTTEEEESCTTSSCCCCTTC
TTCEEEETTEEEEEEECSSSSCTTCCEEEEEHHHHHHHHHHHHHHC

10.Calcium parvalbumin B

AFAGVLNDADIAAALEACKAADSFNHKAFFAKVGLTSKSADDVKKAFAIIDQDKSGFIEEDELKLFLQNFKADARALTDGETKTFLKA
GDSDGDGKIGVDEFTALVKA

CCTTCSCHHHHHHHHHHTCSSSCCCHHHHHHHTTCTTSCHHHHHHHHHHHCSSCCSEEEHHHHHTSSTTTSTTCCCCCHHHHHHHHHH
CCTTCSSEEEHHHHHHHHHC

11.Carbonic anhydrase C

HWGYGKHDGPEHWHKDFPIAKGERQSPVDIDTHTAKYDPSLKPLSVSYDQATSLRILNDGHAFNVEFDDSEDKAVLKGGPLDGTYRL
IQFHFHWGSLDGEGSQHTVDKKKYAAELHLVHWNTKYGDFGKAVQQPDGLAVLGIFLKVGSAKPGLQKVVDVLDSIKTKGKSADFTN
FDPRGLLPESLDYWTYPGSLTTPPLLECVTWIVLKEPISVSSEQVLKFRKLNFDGEGEPEELMVDNWRPAQPLKNRQIKASF

CCCSSTTSSGGGCTTSCGGGGCSSCSSCEECTTTSCCCTTSCCEEEECTTCCCCEEEECSSCEEEECCCSSSSSEEEETTCSSCEEE
EEEEEEECSSTTCCCSSEETTBCCSEEEEEEEEEGGGSSHHHHTTSTTSEEEEEEEEEEESCCHHHHHHHGGGGTTSSSSCEEECCC
CCTTTTSCSCCCCEECEECCSSSSCCSCEEEEECSSCEEEEHHHHHHHTTCBSSCTTCSCCBCCCCCCCCCCCCSCCCEECC

12.Carboxypeptidase A

RSTNTFNYATYHTLDEIYDFMDLLVAQHPELVSKLQIGRSYEGRPIYVLKFSTGGSNRPAIWIDLGIHSRQWITQATGVWFAKKFTE
NYGQNPSFTAILDSMDIFLEIVTNPNGFAFTHSENRLWRKTRSVTSSSLCVGVDANRNWDAGFGKAGASSSPCSETYHGKYANSEVE
VKSIVDFVKNHGNFKAFLSIHSYSQLLLYPYGYTTQSIPDKTELNQVAKSAVAALKSLYGTSYKYGSIITTIYQASGGSIDWSYNQG
IKYSFTFELRDTGRYGFLLPASQIIPTAQETWLGVLTIMEHTVNNGY

CCSTTCCCSSCCCHHHHHHHHHHHHHHSSTTTEEEEEEEECTTSCEEEEEEECSCCSSCCEEEEECCSSTTCTHHHHHHHHHHHHHHH
HBTTBHHHHHHHTTSEEEEESCSSHHHHHHHHHTCTTCCSCCCCCSSCSCCCCCGGGSSSSSTTSSSSBSCTTSTTBCCSSTTCSHH
HHHHHHHHHHHCCEEEEEEEECSSCEEEESCSSCCCCCTTHHHHHHHHHHHHHHHTTSSSCCCCEEEEHHHHSCCCCSCHHHHHHHHT
CSCEEEEECSCSSSSTTCCCGGGHHHHHHHHHHHHHHHHHHHHTCCC

13.Concanavalin A

ADTIVAVELDTYPNTDIGDPSYPHIGIDIKSVRSKKTAKWNMQDGKVGTAHIIYNSVDKRLSAVVSYPNADATSVSYDVDLNDVLPE
WVRVGLSASTGLYKETNTLISWSFTSKLKSNSTHQTDALHFMFNQFSKDQKDLILEGDATTGTDGNLELTRVSSNGSPEGSSVGRAL
FYAPVHIWESSAATVSFEATFAFLIKSPDSHPADGIAFFISNIDSSIPSGSTGRLLGLFPDAN

CCCEEEEEEECSCCTTTTCCSSCEEEEEESSSSCSSEEECCCCTTCEEEEEEEEETTTCEEEEEEECTTSCCEEEEEECCCTTTSCC
EEEEEEEEECSSSCCCCEESCEEEEEEECCTTTCCCEEEEEEESCCCSCCSSEEEEESCEECSSSSEECSCCCSSSCCCSSCEEEEE
ESSCEECCCTTCSCEEEEEEEEEECCCSSSSCCCCEEEEEECSSCCCCSSCCSSSTTTSCSCC

14.CytB5

AVKYYTLEQIEKHNNSKSTWLILHYKVYDLTKFLEEHPGGEEVLREQAGGDATEDFEDVGHSTDARELSKTFIIGELHPDDRSKI

CCCEECHHHHTTCEETTEEEEEETTEEEECTTTHHHCTTCSHHHHHTTTEECHHHHHHTTCCHHHHHHHHHHEEEEECHHHHHHC

15.CytochromeB562

ADLEDDMQTLNDNLKVIEKABBZKANDAALVKMRAAALNAQKATPPKLEDNSQPMKDFRHGFDILVEGIDDALKLANEGKVKEAQAA
AEQLKTTRNAYHQKYR

CCHHHHHHHHHHHHHHTTTSCCSTTTTHHHHHHHHHHHHHHGGGSCCTTTTTCCSSTTHHHHHHHHHHHHHHHHHHHHHHTCSHHHHHH
HHHHHHHHHHHHHSCC

16.Cyrochrome c550

NEGDAAKGEKFNKCKACHMIQAPDGTDIKGGKTGPNLYGVVGRKIASEEGFKYGEGILEVAEKNPDLTWTEANLIEYVTDPKPLVKK
MTDDKGAKTKMTFKMGKNQADVVAFLAQDDPDAXXXXXXXXXXXXX

CCCCSHHHHHHTTTTTTEECCCSSSCSSCCCSSEEECSSCTTSCTTCSSSSCCCHHHHHHHHHCSCCCCCSHHHHHHHHSSTTTTTT
TSCCSCCCCSCCCCCCSCHHHHHHHHHHHSCCSCSSCSCCSTTCCC

17.Cyto C

GDVAKGKKTFVQKCAQCHTVENGGKHKVGPNLWGLFGRKTGQAEGYSYTDANKSKGIVWNNDTMLEYLENPKKYIPGTKMIFAGIKK
KGERQDLVAYLKSATS

CCHHHHHHHHHHHTTTTCCCSTTCCCCSSCCCTTCTTSBTTCCTTCCCCHHHHHSCCBCSHHHHHHHHHCHHHHSTTCCCCCCCCCC
HHHHHHHHHHHHHHHC

18.CytoC551

EDPEVLFKNKGCVACHAIDTKMVGPAYKDVAAKFAGQAGAEAELAQRIKNGSQGVWGPIPMPPNAVSDDEAQTLAKWVLSQK

CCHHHHHHHTTGGGTCCSSSCSSSCCHHHHHHHHTTCTTHHHHHHHHHHHCBCSSSSSSCBCCCSCCHHHHHHHHHHHHTCC

19.Deoxyhemoglobin

LSADQISTVQASFDKVKGDPVGILYAVFKADPSIMAKFTQFAGKDLESIKGTAPFETHANRIVGFFSKIIGELPNIEADVNTFVASH
KPRGVTHDQLNNFRAGFVSYMKAHTDFAGAEAAWGATLDTFFGMIFSKM

CCHHHHHHHHHHHHTTTTCHHHHHHHHHHHCHHHHTTCTTTTTSCHHHHTTSHHHHHHHHHHHHHHHHHHHTTTCCHHHHHHHHHHH
GGGTCCHHHHHHHHHHHHHHHHHHSCGGGGHHHHHHHHHHHHHHHHHC

20.Ferredoxin PA

AYVINDSCIACGACKPECPVNIIQGSIYAIDADSCIDCGSCASVCPVGAPNPED

CCEECTTCCCCCTTGGGCTTCCBCSSSCCBCTTTCCCCCHHHHHCSSCCEESCC

21.Feroxin SP

ATYKVTLIDEAEGINETIDCDDDTYILDAAEEAGLDLPYSCRAGACSTCAGTITSGTIDQSDQSFLDDDQIEAGYVLTCVAYPTSDC
TIKTHQEEGLY

CCCSCCCCCTTCCSTTCCSTTCCCSCCCSSSSCCCCSCCTTCSSSSSTTCCCCCCSSCCCSCCCCSSSCSCCCSCCCSTTCCCCTTC
CSSCSSSCCCC

22.Ferricytochrome C2

EGDAAAGEKVSKKCLACHTFDQGGANKVGPNLFGVFENTAAHKDNYAYSESYTEMKAKGLTWTEANLAAYVKNPKAFVLEKSGDPKA
KSKMTFKLTKDDEIENVIAYLKTLK

CTHHHHHHHHHTTTCTTBCCSTTCCCTTSCCCTTCTTCBTTCCTTSCCCTTTSCSSSSCCBCCSHHHHHHTTSSSTTTTTTSCCSSC
CCSCCCCCCCHHHHHHHHHHHHHHCC

23.Flavodoxin

MKIVYWSGTGNTEKMAELIAKGIIESGKDVNTINVSDVNIDELLNEDILILGCSAMGDEVLEESEFEPFIEEISTKISGKKVALFGS
YGWGDGKWMRDFEERMNGYGCVVVETPLIVQNEPDEAEQDCIEFGKKIANI

CEEEEECSSSHHHHHHHHHHHHHHHHTTCCCEEEEGGGCCSTTTTTCSEEEEEECCBTTTBCCTTTHHHHHHHHTTTCTTCEEEEEEE
ESSSCSHHHHHHHHHHHHTTCEECSCCEEEESSCGGGHHHHHHHHHHHHTC

24.Folate reductase

TAFLWAQNRNGLIGKDGHLPWHLPDDLHYFRAQTVGKIMVVGRRTYESFPKRPLPERTNVVLTHQEDYQAQGAVVVHDVAAVFAYAK
QHLDQELVIAGGAQIFTAFKDDVDTLLVTRLAGSFEGDTKMIPLNWDDFTKVSSRTVEDTNPALTHTYEVWQKKA

CEEEEEEETTBCSBSSSSCSSCCHHHHHHHHHHHHTTSEEECCHHHHTTSSSSSCSSSEEECCCCCSSCCCTTBCCCSCHHHHHHHTT
SSCSCCEEECSCHHHHHHHHTCCSEEEEEEESSCCCCSBCCCCCCCSCSCEEEEEEECCSSTTTCEEEEEEECCC

25.Gamma trypsin A

CGVPAIQPVLSVNGEEAVPGSWPWQVSLQDKTGFHFCGGSLINENWVVTAAHCGVTTSDVVVAGEFDQGSSSEKIQKLKIAKVFKNS
KYNSLTINNDITLLKLSTAASFSQTVSAVCLPSASDDFAAGTTCVTTGWGLTRYTPDRLQQASLPLLSNTNCKKYWGTKIKDAMICA
GASGVSSCMGDSGGPLVCKKNGAWTLVGIVSWGSSTCSTSTPGVYARVTALVNWVQQTLAAN

CCCSSCCCCCSBTCEECCTTSSTTEEEEECTTCCEEEEEEEEETTEEEECGGGCCCTTSEEEESCSCTTCSSSCCEEEEEEEEEECT
TCBTTTTBSCCEEEEESSCCCCCSSCCCCBCCCTTCCCCTTCEEEEEESSCSCCCCSSCEEEEEECCCHHHHTTTSGGGCCTTEEEE
ECSSCBCCTTCTTCEEEEEETTEEEEEEEEECCTTCCTTSEEEEEEGGGTHHHHHHHHHHC

26.Glucagon

HSQGTFTSDYSKYLDSRRAQDFVQWLMNT

CCSCSCHHHHTTTHHHHHHHHHHTTSCCC

27.Hemoglobin

VLSAADKTNVKAAWSKVGGHAGEYGAEALERMFLGFPTTKTYFPHFDLSHGSAQVKAHGKKVGDALTLAVGHLDDLPGALSDLSNL
HAHKLRVDPVNFKLLSHCLLSTLAVHLPNDFTPAVHASLDKFLSSVSTVLTSKYRVQLSGEEKAAVLALWDKVNEEEVGGEALGRL
LVVYPWTQRFFDSFGDLSNPGAVMGNPKVKAHGKKVLHSFGEGVHHLDNLKGTFAALSELHCDKLHVDPENFRLLGNVLVVVLARH
FGKDFTPELQASYQKVVAGVANALAHKYH

CCCHHHHHHHHHHHHHHHGGGHHHHHHHHHHHHHHHHCGGGGGGCTTSCCSTTCHHHHHHHHHHHHHHHHHHTTTTCHHHHTHHHHHH
HHHTSCCCTHHHHHHHHHHHHHHHHHHCTTTCCHHHHHHHHHHHHHHHHHHHTTTTCCCCCHHHHHHHHHHHTTCCHHHHHHHHHHHH
HHHSGGGGGGGGGGCCCSSHHHHHTCHHHHHHHHHHHHHHHHHHTTGGGHHHHSHHHHHHHHTTSCCCTHHHHHHHHHHHHHHHHH
HGGGSCHHHHHHHHHHHHHHHHHHSSCC

28.Hipip

SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFMQADAAGATDEWKGCQLFPGKLINVNGWCASWTLKAG

CCCTTBCCTTCHHHHHHTCBSSGGGSSHHHHCCSSSCGGGCCGGGBTTEETTSTTCCSSEEEETTSTTCEEETTCCCTTCCBCCC

29.Insulin

GIVEQCCTSICSLYQLENYCNFVNQHLCGSHLVEALYLVCGERGFFYTPKA

CHHHHHHHSCBCHHHHGGGBCCCCBCCCTHHHHHHHHHHHHGGGCEEECCCC

30.Lactate dehydrydrogenase

ATLKDKLIGHLATSQEPRSYNKITVVGCDAVGMADAISVLMKDLADEVALVDVMEDKLKGEMMDLQHGSLFLHTAKIVSGKDYSVS
AGSKLVVITAGARQQEGESRLNLVQRNVNIFKFIIPNIVKHSPDCIILVVSNPVDVLTYVAWKLSGLPMHRIIGSGCNLDSARFRY
LMGERLGVHSCSGVGWVIGQHGDSVPSVWSGMWNALKELHPELGTNKDKQDWKKLHKDVVDSAYEVIKLKGYTSWAIGLSVADLAE
TIMKNLCRVHPVSTMVKDFYGIKDNVFLSLPCVLNDHGISNIVKMKLKPNEEQQLQKSATTLWDIQKDLKF

CCTHHHHSCCCTTCCCCCCSSEEEEECCSHHHHHHHHHHHHHTTCCSEEEEECSCHHHHHHHHHHHHTTGGGSCCCEEEEESSSGGG
CSCSEEEECCCCCCCSSCCHHHHHHHHHHHHHHHHHHHHHHHCTTCCCEECSSCHHHHHHHHHHHHCCCTTSCBCCTTTTHHHHHTT
TTTTTTTCCTTTSCCEECBSSSTTCCEETTTCCBTTBCHHHHHCCCSSSSSGGGGTHHHHTSTTSHHHHBSCCCHHHHHHHHTHHH
HHHTTCCEEECCEEECTTSTTCCSSCEEECCEEEBTTBCCCBCCCCCCHHHHHHHHHHHHHHHHHCCCSCC

31.Lambda Fab

XSVLTQPPSVSGAPGQRVTISCTGSSSNIGAGNHVKWYQQLPGTAPKLLIFHNNARFSVSKSGSSATLAITGLQAEDEADYYCQSY
DRSLRVFGGGTKLTVLRQPKAAPSVTLFPPSSEELQANKATLVCLISDFYPGAVTVAWKADSSPVKAGVETTTPSKQSNNKYAASS
YLSLTPEQWKSHKSYSCQVTHEGSTVEKTVAPTECSXVQLEQSGPGLVRPSQTLSLTCTVSGSTFSNDYYTWVRQPPGRGLEWIGY
VFYHGTSDTDTPLRSRVTMLVNTSKNQFSLRLSSVTAADTAVYYCARDLIAGCIDVWGQGSLVTVSSASTKGPSVFPLAPSSKSTS
GGTAALGCLVKDYFPEPVTVSWNSGALTSGVHTFPAVLQSSGLYSLSSVVTVPSSSLGTQTYICNVNHKPSNTKVDKKVEPKSC

CCSCBCCSEEEECTTSCCEEEEECCTTTTTSSCCEEEEEECSSSSCEECCCSSCSSEEEEEETTEEEEEECSCCSTTCEEEEEEEE
ETTEEEEEEEEEEEESCCCCCCCEEECCCCCTTTTTTTCCEEEEEEECCSSSCCEEEEECSSSBCCTTEEECCCEECSTTCEEEEE
EEECCSCSSCCCSCEEEEEESSSCEEEEEECSSCCCCCEEEEECCSEECTTSCEEEEEEEESSCTTTCEEEEEEECTTCCCEEEEE
EETTSCEEECSSSTTSEEEECCSSSSCEEEEECSCCGGGCEEEEEEEECSSSCCCCEEEEEEEEECSSCCBCCBCCBCCSCCSSCS
TTSCCEEEEEEEEBSSCCEEEETTTSCCTTCCBCCCEECSSSCEEEEEEECCSSSTTSSSCCEEEEEEGGGTEEEEEECCCSCC

32.Lyzozyme Egg

KVFGRCELAAAMKRHGLDNYRGYSLGNWVCAAKFESNFNTQATNRNTDGSTDYGILQINSRWWCNDGRTPGSRNLCNIPCSALLSS
DITASVNCAKKIVSDGDGMNAWVAWRNRCKGTDVQAWIRGCRL

CBCCHHHHHHHHHHTTCTTBTTBCTHHHHHHHHHHTSSBTTCEECCSSSCCEETTTTEETTTTEECSSCTTCCCTTCEEGGGGGSS
SCHHHHHHHHHHHHSSSGGGGSHHHHHHTTTSCGGGSSTTCCC

33.LyzoT4

MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAKSELDKAIGRNCNGVITKDEAEKLFNQDVDAAVRGILRNAKLKP
VYDSLDAVRRCALINMVFQMGETGVAGFTNSLRMLQQKRWDEAAVNLAKSRWYNQTPNRAKRVITTFRTGTWDAYKNL

CCSHHHHSTTTCCEEEEEEETTSCEEEETTEEEESCSCSSHHHHHHHHHTTSCCTTBCCHHHHHHHHHHHHHHHHHHHHTSTTTHH
HHHHSCHHHHHHHHHHHHHHCHHHHTTCHHHHHHHHTTCCTTTSSSSTTSHHHHHSHHHHHHHHHHHHSSCCSSSCC

34.Methemoglobin

PIVDTGSVAPLSAAEKTKIRSAWAPVYSDYETSGVDILVKFFTSTPAAEEFFPKFKGLTTADELKKSADVRWHAERIIDAVDDAVA
SMDDTEKMSSMKDLSGKHAKSFEVDPEYFKVLAAVIADTVAAGDAGFEKLLRMICILLRSAY

CCCSSSCCCCCCHHHHHHHHHTTHHHHHTTTHHHHHHHHHHHHHCTTTTTTCGGGTTCCSSSTTTSCHHHHHHHHHHHHHHHHHHH
HSSSCSTTGGGHHHHHHHHTTTCCCGGGHHHHHHHHHHHHCCSCCSHHHHHHHHHHHHGGGC

35.Myoglobin

VLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRFKHLKTEAEMKASEDLKKHGVTVLTALGAILKKKGHHEAEL
KPLAQSHATKHKIPIKYLEFISEAIIHVLHSRHPGNFGADAQGAMNKALELFRKDIAAKYKELGYQG

CCCHHHHHHHHHHHHHHTTSHHHHHHHHHHHHHHHCHHHHHTCHHHHTCCSHHHHHHCHHHHHHHHHHHHHHHHHTTTTCCHHHH
HHHHHHHHHTTCCCHHHHHHHHHHHHHHHHHHHCTTTTSHHHHHHHHHHHHHHHHHHHHHHHHHTCCC

36.Nuclease

ATSTKKLHKEPATLIKAIDGDTVKLMYKGQPMTFRLLLVDTPETKHPKKGVEKYGPEASAFTKKMVENAKKIEVEFNKGQRTDKYG
RGLAYIYADGKMVNEALVRQGLAKVAYVYKPNNTHEQHLRKSEAQAKKEKLNIWSE

CCSSSSCCCEEEEEEEECSTTEEEEECSSCEEEEEETTEECCCSSCSTTSSCTTHHHHHHHHHHHHHTCSCEEEEECSSCCBCTTS
CEEEEEEETTEEHHHHHHHTTSCEECCCCTTCCTTHHHHHHHHHHHHHHTTCGGGGC

37.Papain

IPEYVDWRQKGAVTPVKNQGSCGSCWAFSAVVTIEGIIKIRTGNLNQYSEQELLDCDRRSYGCNGGYPWSALQLVAQYGIHYRNTY
PYEGVQRYCRSREKGPYAAKTDGVRQVQPYNQGALLYSIANQPVSVVLQAAGKDFQLYRGGIFVGPCGNKVDHAVAAVGYGPNYIL
IKNSWGTGWGENGYIRIKRGTGNSYGVCGLYTSSFYPVKN

CCSCEESTTTTCCCCCCCCCSSCCHHHHHHHHHHHHHHHHHHCCCCCBCHHHHHHHCTTTTSTTCCCHHHHHHHHHHSCBCBSSSS
CCCSSCCCCCHHHHCSCSBCCSCEEEECSSCHHHHHHHHHHSCEEEEECCSCSSTTTCSSSEECCSCCSCCCEEEEEEEECSSEEE
EECSSCSSSTTTSEEEEECCSSCSSCGGGTTSCEEEECCC

38.Phophoglycerate mutase

PKLVLVRHGQSEWNEKNLFTGWVDVKLSAKGQQEAARAGELLKEKGVNVLVDYTSKLSRAIQTANIALEKADRLWIPVNRSWRLNE
RHYGDLQGKDKAQTLKKFGEEKFNTYRRSFDVPPPPIDASSPFSQKGDERYKYVDPNVLPETESLALVIDRLLPYWQDVIAKLVGK
TSMIAAHGNSLRGLVKHLEGISDADIAKLNIPPGTILVFELDENLKPSKPSYYLDPEA

CCCEECBCCCCSSHHHHSCCCTTCCCCCHHHHHHHHHHHHHHHHHHSCCCSEEEEESCHHHHHHHHHHHHHHTCCCSCCEEEESSSCC
CCCCSCCSCCSSHHHHHTCSHHHHHSTTSSSCCSCCCCSSSSSCCCCSSTTTTSCSSCSCSSCCTTTHHHHHHHHHHHTHHHHCSS
CEEECCCTTHHHHHHHHHSSCCSSSSSSCCCCBSSCEECCCCTTTCCCSCCBCSCTTC

39.Phospholipase A2

ALWQFNGMIKCKIPSSEPLLDFNNYGCYCGLGGSGTPVDDLDRCCQTHDNCYKQAKKLDSCKVLVDNPYTNNYSYSCSNNEITCSS
ENNACEAFICNCDRNAAICFSKVPYNKEHKNLDKKNC

CHHHHHHHHHHHCTTCCHHHHTTTBTTTBSSCCCSSCSSHHHHHHHHHHHHHHHHHTTCHHHHHTTCCTTTCCCCEEEETTEEEECT
TCCHHHHHHHHHHHHHHHHHHHTSCCCGGGBTCCGGGC

40.Plastocyanin

IDVLLGADDGSLAFVPSEFSISPGEKIVFKNNAGFPHNIVFDEDSIPSGVDASKISMSEEDLLNAKGETFEVALSNKGEYSFYCSP
HQGAGMVGKVTVN

CEEEESCTTCCSCEESSEEEECTTCEEEEEECSSCCBCCEECTTSSCTTCCHHHHSCCTTCCBCSTTCEEEEECCSCEEEEEECGG
GTTTTCEEEEEEC

41.Ribonuclease S

KETAAAKFERQHMDSSTSAASSSNYCNQMMKSRNLTKDRCKPVNTFVHESLADVQAVCSQKNVACKNGQTNCYQSYSTMSITDCRE
TGSSKYPNCAYKTTQANKHIIVACEGNPYVPVHFDASV

CCCHHHHHHHHHHBCCCSSCCCCCCCHHHHHHHHHTTSSSSSCCSEEEEECSCHHHHHGGGGSEEEEETTTEEEEEECSSCEEEEEEEE
CTTCBTTBCCEEEEEEEECEEEEECSSSCCEEEEEEEC

42.Rubredoxin

MKKYTCTVCGYIYDPEDGDPDDGVNPGTDFKDIPDDWVCPLCGVGKDEFEEVEE

CCCEEETTTCCEECTTTCBGGGTBCTTCCGGGSCTTCBCTTTCCBGGGEEECCC

43.Sgriseus proteinase

IAGGEAITTGGSRCSLGFNVSVNGVAHALTAGHCTNISASWSIGTRTGTSFPNNDYGIIRHSNPAAANGRVYLYNGSYQDITTAGN
AFVGQAVQRSGSTTGLRSGSVTGLNATVNYGSSGIVYGMIQTNVCAQPGDSGGSLFAGSTALGLTSGGSGNCRTGGTTFYQPVTEA
LSAYGATVL

CCSSSEECSSSCCEECCCEEEETTEEEEEECHHHHSSCSCBTTEEEEECCCSBSCCEEEEESSTTSCCSEEECSSSCEEECCCCCC
CCTTCEEEEECSSSCEEEEEEEEEEEEECGGGCEEEEEEEESCCCCTTCBTCEEEETTEECCCEEEESSBTTTBCCEEECCHHHH
HHHTTCEEC

44.Subtilisin BPN

AQSVPYGVSQIKAPALHSQGYTGSNVKVAVIDSGIDSSHPDLKVAGGASMVPSETPNFQDDNSHGTHVAGTVAALNNSIGVLGVAP
SSALYAVKVLGDAGSGQYSWIINGIEWAIANNMDVINMSLGGPSGSAALKAAVDKAVASGVVVVAAAGNEGSTGSSSTVGYPGKYP
SVIAVGAVDSSNQRASFSSVGPELDVMAPGVSIQSTLPGNKYGAYNGTSMASPHVAGAAALILSKHPNWTNTQVRSSLQNTTTKLG
DSFYYGKGLINVQAAAQ

CCCCCHHHHHHTHHHHHTTTCSCTTCEEEEEESCCCTTCTTCCCCEEEECCTTCCCSSCCSSSHHHHHHHHHHCCSSSSSCCCSST
TCEEEEEECEETTTEECHHHHHHHHHHHHHHTTCSEEEECCCBSCCCHHHHHHHHHHHHHHTCEEEEECCSCCCSTTSCCCCBTTTST
TSEEEEEECTTCCBCTTSCCCTTCCEEEECSSEEEEETTTEEEEECCHHHHHHHHHHHHHHHHHHSTTSCHHHHHHHHHTTSBCCS
CHHHHTTCBCCHHHHHC

45.Thermolysin

ITGTSTVGVGRGVLGDQKNINTTYSTYYYLQDNTRGDGIFTYDAKYRTTLPGSLWADADNQFFASYDAPAVDAHYYAGVTYDYYKN
VHNRLSYDGNNAAIRSSVHYSQGYNNAFWNGSEMVYGDGDGQTFIPLSGGIDVVAHELTHAVTDYTAGLIYQNESGAINEAISDIF
GTLVEFYANKNPDWEIGEDVYTPGISGDSLRSMSDPAKYGDPDHYSKRYTGTQDNGGVHINSGIINKAAYLISQGGTHYGVSVVGI
GRDKLGKIFYRALTQYLTPTSNFSQLRAAAVQSATDLYGSTSQEVASVKQAFDAVGVK

CCCCEEEEEEECSSSCEEEEEEECSSSEECBCCSSTTCEEEEECTTCSCSSCEECEESSSEECSGGGTTTSSTTTTTHHHHHHHTT
TSCCSTTTTSCCCEEEEEEETTTCCCCEECSSSEEEECCCSSSBCCGGGCHHHHHHHHHHHHHHHHTTCCCSSHHHHHHHHHHHHHH
HHHHHHHHTSSCCSEESCSSBCSSCSSCCSEESSCGGGTTCCCSGGGCCCSSSHHHHTTTTTHHHHHHHHHHHTCEESSSCEECCC
CSTTTHHHHHHHHTTTCCTTCCHHHHHHHHHHHHHHHTTCTTCHHHHHHHHHHHHHTCC

46.Tosyl elastase

VVGGTEAQRNSWPSQISLQYRSGSSWAHTCGGTLIRQNWVMTAAHCVDRELTFRVVVGEHNLNQNNGTEQYVGVQKIVVHPYWNTD
DVAAGYDIALLRLAQSVTLNSYVQLGVLPRAGTILANNSPCYITGWGLTRTNGQLAQTLQQAYLPTVDYAICSSSSYWGSTVKNSM
VCAGGDGVRSGCQGDSGGPLHCLVNGQYAVHGVTSFVSRLGCNVTRKPTVFTRVSAYISWINNVIASN

CBTCEECCTTTCTTEEEEEEEETTEEEEEEEEEEEETTEEECSGGGCSCCCEEEEESCSBTTSCCSCCEEEEEEEEEECTTCCTT
CGGGCCCCEEEEESSCCCCBTTBCCCCCCCTTCCCCTTCCEEEEESCBSSTTCCBCSBCEEEECCEECHHHHTSTTTTGGGSCTTE
EEECCSSSSBCCTTCTTCEEEEEETTEEEEEEEEEECBTTBSSBTTBCEEEEEGGGSHHHHHHHHHTC

47.Triosephosphate isomerase

APRKFFVGGNWKMNGKRKSLGELIHTLDGAKLSADTEVVCGAPSIYLDFARQKLDAKIGVAAQNCYKVPKGAFTGEISPAMIKDIG
AAWVILGHSERRHVFGESDELIGQKVAHALAEGLGVIACIGEKLDEREAGITEKVVFQETKAIADNVKDWSKVVLAYEPVWAIGTG
KTATPQQAQEVHEKLRGWLKTHVSDAVAVQSRIIYGGSVTGGACKELASQHDVDGFLVGGASLKPEFVDIINAKH

CCCCCEEEEECCBCCCHHHHHHHHHHHHHHCCCCSSCEEEEECCTTHHHHHHHHSCTTEEEEEECCCSSSSBSCSSCCCHHHHHHHT
CCEEEECCHHHHHHHCCCHHHHHHHHHHHHHHTTCCEEEEEEECHHHHHHTTHHHHHHHHHHHHHHHCCCCTTEEEEEEEGGGSSSS
SCCCHHHHHHHHHHHHHHHHHHHHCHHHHHHSEEEECSCCCTTHHHHHHTSTTCCEEEESGGGGSTHHHHHHTCCC

48.Trypsin inhibitor

RPDFCLEPPYTGPCKARIIRYFYNAKAGLCQTFVYGGCRAKRNNFKSAEDCMRTCGGA

CCGGGGSCCCCCSCCCCEEEEEEETTTTEEEEEEECSSSCCSSCBSSHHHHHHHHSCC

49.Adenylate kinase

MEEKLKKSKIIFVVGGPGSGKGTQCEKIVQKYGYTHLSTGDLLRAEVSSGSARGKMLSEIMEKGQLVPLETVLDMLRDAMVAKVDT
SKGFLIDGYPREVKQGEEFERKIGQPTLLLYVDAGPETMTKRLLKRGETSGRVDDNEETIKKRLETYYKATEPVIAFYEKRGIVRK
VNAEGSVDDVFSQVCTHLDTLK

CHHHHHHSCEEEEECCSSTTTTHHHHHHHHHTCCEEEEHHHHHHHHHHHSCCHHHHHHHHHHHSSCCCCHHHHHHHHHHHHHHTTT
CSCEEEESCCSSSSHHHHHHHHSCCCSEEEEECCCHHHHHHHHHHHTTTTCCCCCCHHHHHHHHHHHHHHHTTHHHHHHHHSCEEE
ECCCSCHHHHHHHHHHHHHHC

50.Alcohol dehydrogenase

STAGKVIKCKAAVLWEEKKPFSIEEVEVAPPKAHEVRIKMVATGICRSDDHVVSGTLVTPLPVIAGHEAAGIVESIGEGVTTVRPG
DKVIPLFTPQCGKCRVCKHPEGNFCLKNDLSMPRGTMQDGTSRFTCRGKPIHHFLGTSTFSQYTVVDEISVAKIDAASPLEKVCLI
GCGFSTGYGSAVKVAKVTQGSTCAVFGLGGVGLSVIMGCKAAGAARIIGVDINKDKFAKAKEVGATECVNPQDYKKPIQEVLTEMS
NGGVDFSFEVIGRLDTMVTALSCCQEAYGVSVIVGVPPDSQNLSMNPMLLLSGRTWKGAIFGGFKSKDSVPKLVADFMAKKFALDP
LITHVLPFEKINEGFDLLRSGESIRTILTF

CCSSSCEEEEEEEBCSTTSCCEEEEEEECCCCTTEEEEECCEEECCHHHHHHHTTSSCCCSSBCCCCCBCCEEEEECTTCCSCCTT
CBEEECSSCCCSSSTTTSSTTCCCCSSCSSSSCCCSCTTSCCSEEETTEECBCCTTTCCSBSEEEEEGGGEEECCSSCCTTTGGGG
GTHHHHHHHHHHTTTCCCTTCCEEEECCSHHHHHHHHHHHHHHSCCCEEEECSCGGGHHHHHHHTCSEEECTTTSSSCHHHHHHHTT
TSCBSEEEECSCCHHHHHHHHHTBCTTTCEEEECCCCTTCCCCCCCTHHHHTTCEEEECCGGGCCHHHHHHHHHHHHHTSCCCCTT
TEEEEEETTTHHHHHHHHHTSCCCEEEEEC

51.Cobratoxin

IRCFITPDITSKDCPNGHVCYTKTWCDAFCSIRGKRVDLGCAATCPTVKTGVDIQCCSTDNCNPFPTRKRP

CCCCCSSSSSCCCCCTTSCEEEEEECCTTHHHHCCEEEEECCSSCCCCCSSCEEEEECCTTCCSCCSTTCC

52.Crambin

TTCCPSIVARSNFNVCRLPGTPEAICATYTGCIIIPGATCPGDYAN

CEECSSHHHHHHHHHHHTTTCCHHHHHHHHSCEECSSSCCCGGGCC

53.Gluthatione reductase

VASYDYLVIGGGSGGLASARRAAELGARAAVVESHKLGGTCVNVGCVPKKVMWNTAVHSEFMHDHAVYGFPSCEGFFNWRVIKEKR
DAYVSRLNAIYQNNLTKSHIEIIRGHAAFTSDPKPTIEVSGKKYTAPHILIATGGMPSTPHESQIPGASLGITSDGFFQLEELPGR
SVIVGAGYIAVEMAGILSALGSKTSLMIRHDNVLRSFDSMISTNCTEELENAGVEVLKFSQVKEVKKTLSGLEVSMVTAVPGRLPV
MTMIPDVDCLLWAIGRVPNTKDLSLNKLGIQTDDKGHIIVDEFQNTNVKGIYAVGDVCGKALLTPVAIAAGRKLAHRLFEYKEDSK
LDYNNIPTVVFSHPPIGTVGLTEDEAIHKYGIENVKTYSTSFTPMYHAVTKRKTKCVMKMVCANKEEKVVGIHMQGLGCDEMLQGF
AVAVKMGATKADFDNTVAIHPTSSEELVTLR

CCCCSCCEESCBTTHHHHHHHHHHTTCCEEEEESSCTBHHHHHHSHHHHHHHHHHHHHHHHSSCGGGSCCCCCCCCCCHHHHHHHH
HHHHHHHHHHHHHHHHHHTTEEEEESCCCBCSSSSSCBCSTTCCBCCSSEEECCCCEECCCCSTTSSSCTTCEEHHHHTTCSSCCSS
EEEECCSHHHHHHHHHHHHHTTCCCEEECSSSSSCTTSCHHHHHHHHHHHHTTSSCCEETTEEEEEEETTSSSCEEEEEECSSSSCCE
EEECCSSCEEEECSCEECSCTTSCTTTTTCCBCTTSCBCCCSSSBCSSSSEEECSTTSSSCCCHHHHHHHHHHHHHHHHSCCTTCC
CCSSSCCEEECCSSCEEEEECCHHHHHHHSCSSSEEEEEEEECCGGGSSCSSCCCEEEEEEEETTTTEEEEEEEESTTHHHHHHHH
HHHHTTTCBHHHHHTSCCCSSSSGGGGSSCC

54.Glyceraldehyde dehydrogenase

SKIGIDGFGRIGRLVLRAALSCGAQVVAVNDPFIALEYMVYMFKYDSTHGVFKGEVKMEDGALVVDGKKITVFNEMKPENIPWSKA
GAEYIVESTGVFTTIEKASAHFKGGAKKVVISAPSADAPMFVCGVNLEKYSKDMTVVSNASCTTNCLAPVAKVLHENFEIVEGLMT
TVHAVTATQKTVDGPSAKDWRGGRGAAQNIIPSSTGAAKAVGKVIPELDGKLTGMAFRVPTPDVSVVDLTVRLGKECSYDDIKAAM
KTASEGPLQGFLGYTEDDVVSSDFIGDNRSSIFDAKAGIQLSKTFVKVVSWYDNEFGYSQRVIDLLKHMQKVDSA

CCEEEECTTTHHHHHHHHHHTTTCCCCCEECTTSCHHHHHHHHHCBTTTBCCCSCEEEETTEEEETTEEECEECCSSTTTSCCTTT
TCCEEEECCSSCCSHHHHHHHHHHSCSEEEEESCCSSSCCBCTTTTTTSCCTTCSEEEECCHHHHHHHHHHHHHHHHHCCBCCCEE
EEECCCTTCCSBCCCCCTTCTTCCBSSSCCEEEEECHHHHHHHHSSSSSSSCCEEEEECSCCSCCEEEEEEECTTCCCHHHHHHHH
HHHHHTTTTTSEEEECSCCCTTCCSSSCCSSEEESSTTCEEETTEEEEECCCCHHHHHHHHHHHHHHHHHHHHTC

55.Leghemoglobin

GALTESQAALVKSSWEEFNANIPKHTHRFFILVLEIAPAAKDLFSFLKGTSEVPQNNPELQAHAGKVFKLVYEAAIQLEVTGVVAS
DATLKNLGSVHVSKGVADAHFPVVKEAILKTIKEVVGAKWSEELNSAWTIAYDELAIVIKKEMDDAA

CTTCHHHHHHHHHHHHHHHHTTHHHHHHHHHHHHHHHHCGGGGGGCGGGCTTCCCCSSCHHHHHHHHHHHHHHHHHHHHHHSSCCC
CHHHHHHHHHHHHTTCCGGGHHHHHHHHHHHHHHHHHGGGCCHHHHHHHHHHHHHHHHHHHHHHTC

56.Mellitin

GIGAVLKVLTTGLPALISWIKRKRQQ

CHHHHHHHHHTTHHHHHHHHHHHHHC

57.Neurotoxin

RICFNQHSSQPQTTKTCSPGESSCYHKQWSDFRGTIIERGCGCPTVKPGIKLSCCESEVCNN

CEEECCSTTCSCCEEECCTTCCCEEEEEEEETTEEEEEEESCCCCCSSCCEEEECSTTTTC

58.Ovomucoid

LAAVSVDCSEYPKPACPKDYRPVCGSDNKTYSNKCNFCNAVVESNGTLTLNHFGKC

CCCEEECCTTCCCSSCCCCCCCEEETTSCEESSHHHHHHHHHHTTTSCCEEEESCC

59.Prealbumin

CPLMVKVLDAVRGSPAINVAVHVFRKAADDTWEPFASGKTSESGELHGLTTEEQFVEGIYKVEIDTKSYWKALGISPFHEHAEVV
FTANDSGPRRYTIAALLSPYSYSTTAVVT

CCEEEEEEETTTTEECCSCEEEEEEECTTSSEEEEEEEECCTTSEECCSCCTTTSCCEEEEEEECHHHHHHHHTCCCSCSCEEEE
EEECSSSCCCEEEEEEEETTEEEEEEEC

60.Rhodanese

VHQVLYRALVSTKWLAESVRAGKVGPGLRVLDASWYSPGTREARKEYLERHVPGASFFDIEECRDKASPYEVMLPSEAGFADYVG
SLGISNDTHVVVYNGDDLGSFYAPRVWWMFRVFGHRTVSVLNGGFRNWLKEGHPVTSEPSRPEPAIFKATLNRSLLKTYEQVLEN
LESKRFQLVDSRAQGRYLGTQPEPDAVGLDSGHIRGSVNMPFMDFLTENGFEKSPEELRAMFEAKKVDLTKPLIATCRKGVTACH
IALAAYLCGKPDVAIYDGSWFEWFHRAPPETWVSQGKG

CCCCCCCCEECHHHHHHHHHHTCBTTTEEEEECCCCCTTTCCHHHHHTTSBCTTCEECCTTSSSCTTSSSSCCCCCHHHHHHHHG
GGSCCTTSEEEECCCSTTSCSSHHHHHHHHHHTCCCCEEEETTHHHHHHHHTCCCBCCCCCCCCCCCCCCCCTTSEECTTHHHHH
HHHCCSEEEECSCHHHHHTCSCCSSCSSCCCCBCTTCEECCTTTTBCTTSCBCCHHHHHHHHHHTTCCTTSCEEEECSSSSTTHH
HHHHHHHHTCTTCEEETTTHHHHHHHSCGGGSBCSSCC

61.Str subtilisin inhibitor

YAPSALVLTVGKGVSATTAAPERAVTLTCAPGPSGTHPAAGSACADLAAVGGDLNALTRGEDVMCPMVYDPVLLTVDGVWQGKRV
SYERVFSNECEMNAHGSSVFAF

CCCCEEEEEECBCSSTTSCCCSEEEEEECSSCCEESSTTHHHHHHHHHHHTSCSTTSCCCSSCCCCCCCBCBCCEEEEEETTEEE
EEECCBSBHHHHHHTTSSSSCC

62.Super oxide dismutase

ATKAVCVLKGDGPVQGTIHFEAKGDTVVVTGSITGLTEGDHGFHVHQFGDNTQGCTSAGPHFNPLSKKHGGPKDEERHVGDLGNV
TADKNGVAIVDIVDPLISLSGEYSIIGRTMVVHEKPDDLGRGGNEESTKTGNAGSRLACGVIGIAK

CCSEEEEEBCSSSCEEEEEEEEETTEEEEEEEEESCCSEEEEEEEESCCCCTTGGGTTCSBCCSSCCCCCCTTCSSCCTTEEEEE
EEBTTTBEEEEEEESSCBSSSTTBCTTSEEEEESSCCCCSSCCSTTTTTSCSCCCEEEEEECEECC

Bibliography

[1] Bajorath, J., Stenkamp, R., & Aruffo, A. (1993). *Knowledge-Based Model Building of Proteins: Concepts and Examples.* Protein Science, 2: pp. 1798- 1810.

[2] Beale, R. and Jackson, T. (1998). *Neural Computing: An Introduction.* Institute of Physics Publishing, Bristol and Philadelphia.

[3] Bishop, C.M. (1995). *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford.

[4] Boser, B.E., Guyon, I.M., & Vapnik, V.N. (1992). *A Training Algorithm for Optimal Margin Classifiers.* In: Proceedings of the 5th ACM Workshop on Computational Learning Theory, pp. 144-152, ACM Press, New York.

[5] Branden, C. & Tooze, J. (1991). *Introduction to Protein Structure.* Garland Publishing Company, New York.

[6] Breiman, L., Friedman, J.H., & Olshen, R.A. (1984). *Classification and Regression Trees.* CRC Press.

[7] Burges, C.J.C. (1998). *A Tutorial on Support Vector Machines for Pattern Recognition.* Data Mining and Knowledge Discovery, 2(2): pp. 121-167.

[8] Campbell, C. and Cristianini, N. (1999). *Simple Learning Algorithms for Training Support Vector Machines.* Technical Report CIG-TR-KA, University of Bristol, Engineering Mathematics, Computational Intelligence Group.

[9] Chandonia, J.M. & Karplus, M. (1995). *Neural Networks for Secondary Structure and Structural Class Prediction.* Protein Sci. 4: pp. 275-285.

[10] Chen, P.H., C.J. Lin and Schölkopf, B. (2005). *A Tutorial on v-Support Vector Machines.* Applied Stochastic Models in Business and Industry 21(2): pp. 111-136.

[11] Cheng, B. & Titterington, D.M. (1994). *Neural Networks: A review from a Statistical Perspective.* Statistical Science, 9(1): pp. 2-30.

[12] Chou, P.Y. & Fasman, G.D. (1974). *Prediction of Protein Conformation*. Biochemistry, 13(2): pp. 222-245.

[13] Cristianini, N., & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and other Kernel based learning methods*. Cambridge University Press, Cambridge.

[14] Cuff, J.A. & Barton, G.J. (1999). *Evaluation and Improvement of Multiple Sequence Methods for Protein Secondary Structure Prediction*. Proteins: Structure, Function and Genetics 34: pp. 508-519.

[15] Cortes, C., & Vapnik, V. (1995). *Support Vector Networks*. Machine Learning, 20: pp. 273-297.

[16] Duda, R.O., Hart, P.E. & Stork, D.E. (2000). *Pattern Classification*. John Wiley & Sons, Canada.

[17] Frishman, D. & Argos, P. (1995). *Knowledge-Based Protein Secondary Structure Assignment*. Proteins: Struct. Function. and Genet. 23: pp. 566-579.

[18] Gunn, S. (1998). *Support Vector Machines for Classification and Regression*. Technical Report, ISIS, Department of Electronics and Computer Science, University of Southampton.

[19] Hammer, B. & Gersmann, K. (2001). *A Note on the Universal Approximation Capability of Support Vector Machines*. Neural Processing Letters, 17: pp. 43-53.

[20] Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. Macmillan, New York.

[21] Hobohm, U., Scharf M., Schneider R. & Sander C. (1992). *Selection of Representative Protein Data Sets*. Protein Sci. 1: pp. 409-417.

[22] Holley, H.L. & Karplus, M. (1989). *Protein Secondary Structure Prediction with a Neural Network*. Proc. Nat. Acad. Sci., U.S.A. 86: pp. 152-156.

[23] Hornik, K., Stinchcombe, M. & White, H. (1989). *Multilayer Feedforward Networks Are Universal Approximators*. Neural Networks, 2: pp. 359-366.

[24] Hua, S. & Sun, Z. (2001). *A Novel Method of Protein Secondary Structure Prediction with High Segment Overlap Measure: Support Vector Machine Approach*. J. Mol. Biol. 308: pp. 397-407.

[25] Jacobson, M. and Sali, A. (2004). *Comparative Protein Structure Modeling and its Applications to Drug Discovery*. Annu. Rep. Med. Chem. 39: pp. 259-276.

[26] Kabsch, W. & Sander, C. (1983). *Dictionary of Protein Secondary Structure: Pattern Recognition of Hydrogen Bonded and Geometrical Features*. Biopolymers, 22: pp. 2577-2637.

[27] Kabsch, W. & Sander, C. (1983). *How Good are Predictions of Protein Secondary Structure?* FEBS lett, 155(2): pp. 179-182.

[28] Kim, H. & Park, H. (2003). *Protein Secondary Structure Prediction Based on an Improved Support Vector Machine Approach*. Protein Engineering. 16(8): pp. 553-560.

[29] Kloczkowski, A., Ting, K.L., Jernigan, R.L., & Garnier, J. (2002). *Combining GOR V Algorithm with Evolutionary Information for Protein Secondary Structure Prediction from Amino Acid Sequence*. Proteins: Struct. Funct. Genet. 49: pp. 154-166.

[30] Kohavi, R. (1995). *A Study of Cross-Validation and Bootstrap for Accuracy and Model Selection*. Comp.Sci. Dept. Stanford University, Standford.

[31] Kroon, S. & Omlin, C.W. (2004). *Getting to Grips with Support Vector Machines: Theory*. South African Statistist. J. 38: pp. 93-114.

[32] Lesk, A.M. (2004). *Introduction to Protein Science Architecture, Function and Genomics*. Oxford University Press, Univer. of Cambridge.

[33] Marti-Renom, M.A., Stuart, A.C., Fiser, A., Sanchez, R., Melo, F. & Sali, A. (2000). *Comparative Protein Structure Modeling of Genes and Genomes*. Annu. Rev. Biophys. Biomol. Struct. 29: pp. 291-325.

[34] Mount, D.W. (2001). *Bioinformatics: Sequences and Genome Analysis*. Cold Spring Harbour, New York: Cold Spring Habour Laboratory Press.

[35] Nguyen, M.N. & Rajapakse, J.C (2003). *Multi-Class Support Vector Machines for Protein Secondary Structure Prediction*. Genome Informatics 14: pp. 218-227.

[36] Noble, W.S. (2006). *What is a Support Vector Machine?* Nature Biotechnology, 24: pp. 1565-1567.

[37] Osuna, E.E., Freund, R. & Girosi, F. (1997). *Support Vector Machines: Training and Applications*. AI Memo 1602, MIT.

[38] Qian, N. & Sejnowski, T.J. (1988). *Predicting the Secondary Structure of Globular Proteins Using Neural Network Models*. J. Mol. Biol. 202: pp. 865-884.

[39] Richard, F. M. & Kundrot, C.E. (1988). *Indentification of Structural Motifs from Protein Coordinate Data: Secondary Structure and First-Level Supersecondary Structure*. Proteins: Struct. Funct. Genet. 3: pp. 71-84.

[40] Riedmiller, M. (1994). *Rprop-Description and Implementation Details*. Technical Report, University of Karlsruhe, Germany.

[41] Ripley, B.D. (1994). *Neural Networks and Related Methods for Classification*. J. Roy. Statist. Soc. Ser. B 56: pp. 409-456.

[42] Ripley, B.D. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge.

[43] Rojas, R. (1996). *Neural Networks: A Systematic Introduction*. Springer. Germany.

[44] Rost, B. & Sander, C. (1993). *Improved Prediction of Protein Secondary Structure by Use of Sequence Profiles and Neural Networks*. Proc. Natl. Acad. Sci. 90: pp. 7558-7562.

[45] Rost, B. & Sander, C. (1993). *Prediction of Protein Secondary Structure at Better than 70% Accuracy*. J.Mol. Biol. 232: pp. 584-599.

[46] Salamov, A.A. & Solovyev, V.V. (1995). *Prediction of Protein Secondary Structure by Combining Nearest- Neighbor Algorithms and Multiple Squence Alignments*. Communication. J. Mol. Biol. 247: pp. 11-15.

[47] Scholkopf, B. (1999). *Support Vector Learning*. Microsoft Reasearch. England.

[48] Sen, T.Z., Jernigan, R.L., Garnier, J. & Kloczkowski, A. (2005). *GOR V Server for Protein Secondary Structure Prediction. Bioinformatic Applications Note*. 21(11): pp 2787-2788.

[49] Shawe- Taylor, J. & Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. University Press, Cambridge.

[50] Tramontano, A. (2004). *Protein Structure Prediction: Concepts and Applications*. Wiley-VCH, Weinheim.

[51] Voet, D., Voet, G. & Pratt, W.C. (2006). *Fundamentals of biochemistry: life at the molecular level*. New York : Wiley.

[52] Wackerly, D.D., Mendenhall, W., & Schaeffer, R. (2002). *Mathematical Statistics with Applications*. Duxbury Press, USA.

[53] Whitford, D. (2005). *Proteins Structure and Function*. John Wiley & Sons Ltd, England.

Some additional information relating to the toolbox, databases and other material used can be obtained from the following sources:

- Number of tertiary structures: `http://www.rcsb.org/pdb/` *accessed 5-02-2008*
- Number of protein sequences: `http://www.ebi.ac.uk/trembl/` *accessed 5-02-2008*
- Kabsch & Sander database (1983): `http://www.anteprot-pbil.ibcp.fr/` *accessed 26-03-2006*
- SVM Matlab codes: Schwaighofer, A. (2002). Version 2.51, January, available from `http://ida.first.fraunhofer.de/~anton/software.html` *accessed 16-11-2006.*
- Mathworks: `http://www.mathworks.com` *accessed 15-10-2007*
- HSSP database: `www.sander.embl-heidelberg.da/hssp/`
- Cristianini, N. (2001). Support Vector Machines Tutorial. `http://www.support-vectpr.net/tutorial.html` *accessed 12-04-2005*