

# Prediction of Protein Secondary Structure using Binary Classification Trees, Naive Bayes Classifiers and the Logistic Regression Classifier

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in the

DEPARTMENT OF STATISTICS

of

RHODES UNIVERSITY

by

Ahmed Abdelkarim Eldud Omer

January 2015



# Abstract

The secondary structure of proteins is predicted using various binary classifiers. The data are adopted from the RS126 database. The original data consists of protein primary and secondary structure sequences. The original data is encoded using alphabetic letters. These data are encoded into unary vectors comprising ones and zeros only. Different binary classifiers, namely the naive Bayes, logistic regression and classification trees using hold-out and 5-fold cross validation are trained using the encoded data. For each of the classifiers three classification tasks are considered, namely helix against not helix (H/ $\sim$ H), sheet against not sheet (S/ $\sim$ S) and coil against not coil (C/ $\sim$ C). The performance of these binary classifiers are compared using the overall accuracy in predicting the protein secondary structure for various window sizes.

Our result indicate that hold-out cross validation achieved higher accuracy than 5-fold cross validation. The Naive Bayes classifier, using 5-fold cross validation achieved, the lowest accuracy for predicting helix against not helix. The classification tree classifiers, using 5-fold cross validation, achieved the lowest accuracies for both coil against not coil and sheet against not sheet classifications. The logistic regression classifier accuracy is dependent on the window size; there is a positive relationship between the accuracy and window size. The logistic regression classifier approach achieved the highest accuracy when compared to the classification tree and Naive Bayes classifiers for each classification task; predicting helix against not helix with accuracy 77.74%, for sheet against not sheet with accuracy 81.22% and for coil against not coil with accuracy 73.39%. It is noted that it is easier to compare classifiers if the classification process could be completely facilitated in R. Alternatively, it would be easier to assess these logistic regression classifiers if SPSS had a function to determine the accuracy of the logistic regression classifier.

Keywords: Classification tree, Naive Bayes, logistic regression, hold-out, 5-fold cross validation, protein secondary structure prediction



# Contents

<b>Abstract</b>	<b>i</b>
<b>List of Tables</b>	<b>vi</b>
<b>List of Figures</b>	<b>viii</b>
<b>1 Outline</b>	<b>1</b>
1.1 Framework . . . . .	1
1.2 Research Background . . . . .	1
1.3 Goal of the Research . . . . .	2
1.4 Introduction . . . . .	2
<b>2 Classification</b>	<b>3</b>
2.1 Introduction . . . . .	3
2.2 Classification . . . . .	3
2.3 Measurement of Classifier Accuracy . . . . .	4
2.4 Measurement of Binary Classifier Accuracy . . . . .	6
2.5 Cross Validation . . . . .	9
2.5.1 $k$ -fold Cross Validation . . . . .	10
2.5.2 Hold-Out Cross Validation . . . . .	11
2.5.3 Leave-One-Out Cross Validation . . . . .	12
<b>3 Classification Using the Naive Bayes and Logistic Regression Classifiers</b>	<b>15</b>
3.1 Introduction . . . . .	15
3.2 Classifiers Based on Bayes Rule . . . . .	15

3.2.1	Estimating the Prior Probability . . . . .	17
3.2.2	Estimating the Likelihood . . . . .	17
3.3	Naive Bayes Classifier . . . . .	18
3.3.1	Estimating the Maximum Likelihood for Naive Bayes Classifier . . . . .	19
3.4	Logistic Regression . . . . .	20
3.4.1	The Logistic Regression Model . . . . .	20
3.4.2	Estimation of the Logistic Regression Model Parameters . . . . .	23
3.4.2.1	The Newton Raphson Method . . . . .	25
3.4.3	Using Logistic Regression as a Classifier . . . . .	28
<b>4</b>	<b>Classification Trees</b>	<b>31</b>
4.1	Introduction . . . . .	31
4.2	Binary Classification Trees . . . . .	31
4.3	Impurity Functions . . . . .	32
4.3.1	Node Impurity . . . . .	32
4.3.2	Tree Impurity . . . . .	33
4.4	Splitting Rules . . . . .	34
4.4.1	Gini Index . . . . .	34
4.4.2	Entropy Index . . . . .	38
4.5	Splitting Procedure . . . . .	39
4.5.1	Maximum Tree . . . . .	40
4.6	Pruning a Tree . . . . .	41
4.6.1	Cost Complexity Pruning . . . . .	41
4.7	Cross Validation . . . . .	45
<b>5</b>	<b>Prediction of the Protein Secondary Structure</b>	<b>47</b>
5.1	Introduction . . . . .	47
5.2	Proteins and Amino Acids . . . . .	47
5.3	Protein Structure and Function . . . . .	49
5.4	Secondary Structure Prediction . . . . .	50
5.5	The RS126 Data Set . . . . .	50

5.6	Prediction of Protein Secondary Structure using Classification Trees . . . . .	55
5.6.1	Hold-Out Classification . . . . .	55
5.6.1.1	Data Set Partition and Application . . . . .	55
5.6.1.2	Classification Tree Accuracy Using Hold-Out Cross Validation . . . . .	55
5.6.2	5-fold Cross Validation . . . . .	56
5.6.2.1	Data Set Partition and Application . . . . .	56
5.6.2.2	Classification Tree Accuracy Using 5-fold Cross Validation . . . . .	57
5.7	Prediction of Protein Secondary Structure using Naive Bayes Classifiers . . . . .	58
5.7.1	Hold out classification . . . . .	58
5.7.1.1	Data Set Partition and Application . . . . .	58
5.7.1.2	Naive Bayes Classifier Accuracy Using Hold-Out Cross Validation . . . . .	58
5.7.2	5-fold Cross Validation . . . . .	60
5.7.2.1	Data Set Partition and Application . . . . .	60
5.7.2.2	Naive Bayes Classifier Accuracy using 5-fold Cross Validation . . . . .	60
5.8	Prediction of Protein Secondary Structure using Logistic Regression Classifiers . . . . .	61
5.8.1	Hold-Out Cross Validation . . . . .	61
5.8.1.1	Data Set Partition and Application . . . . .	61
5.8.1.2	Logistic Regression Classifier Accuracy . . . . .	62
5.8.2	5-fold Cross Validation . . . . .	63
5.8.2.1	Data Set Partition and Application . . . . .	63
5.8.2.2	Logistic Regression Classifier Accuracy Using 5-fold Cross Validation . . . . .	63
5.9	Comparison of the Naive Bayes, Classification Tree and Logistic Regression Classifiers . . . . .	65
5.9.1	Helix against not Helix . . . . .	65
5.9.2	Coil against not Coil . . . . .	67
5.9.3	Sheet against not Sheet . . . . .	70

## 6 Conclusion 75

6.1	Introduction . . . . .	75
6.2	Conclusions and Discussion . . . . .	75
6.3	Future Study . . . . .	77

Bibliography	78
Appendix A Classification Trees: Hold-Out Cross Validation	83
Appendix B Classification Trees: 5-fold Cross Validation	87
Appendix C Naive Bayes: Hold-Out Cross Validation	93
Appendix D Naive Bayes: 5-fold Cross Validation	97
Appendix E Logistic Regression: Hold-Out Cross Validation	101
Appendix F Logistic Regression: 5-fold Cross Validation	103
Appendix G Confidence Interval: R Code	105



# List of Tables

2.1	Possible outcomes of the classification of a binary variable. . . . .	6
2.2	A confusion matrix for the classification of $n$ binary observations. . . . .	7
2.3	10-fold cross validation (iteration process). . . . .	11
2.4	Hold-out cross validation. . . . .	12
2.5	Leave one out cross validation. . . . .	13
4.1	Classification table: The binary split of $Y$ for variable $x_j$ . . . . .	31
4.2	The Gini split and goodness of the split in the non-terminal nodes in the tree depicted in figure 4.4.2. . . . .	38
4.3	The Entropy split and goodness of the split in the non-terminal nodes in the tree depicted in figure 4.4.2. . . . .	39
5.1	Amino acid abbreviations. . . . .	48
5.2	The actinoxanthin (1acx.concise) entry in the RS126 data set. . . . .	51
5.3	The amino acid encoding matrix. . . . .	52
5.4	The secondary structure encoding. . . . .	52
5.5	The first four observations of the actinoxanthin (1acx.concise) sequence as considered in this study. . . . .	53
5.6	Encoding of the actinoxanthin (1acx.concise) entry for a window of size 3. . . . .	53
5.7	Test accuracy (%) of the classification trees. . . . .	56
5.8	Test accuracy (%) of the classification trees using 5-fold cross validation. . . . .	57
5.9	Test accuracy (%) of the Naive Bayes classifier. . . . .	59
5.10	Test accuracy (%) of the Naive Bayes classifiers using 5-fold cross validation. . . . .	60
5.11	Test accuracy (%) of the logistic regression classifier. . . . .	63
5.12	Test accuracy (%) of the logistic regression classifier using 5-fold cross validation. . . . .	64

5.13	Test accuracy (%), for all window sizes, when predicting helix against not helix for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation. . . . .	65
5.14	Summary of the test accuracy (%), amongst all window sizes, when predicting helix against not helix for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation. . . . .	66
5.15	Test accuracy (%), for all window sizes, when predicting coil against not coil for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation. . . . .	68
5.16	Summary of the test accuracy (%), amongst all window sizes, when predicting coil against not coil for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation. . . . .	68
5.17	Test accuracy (%), for all window sizes, when predicting sheet against not sheet for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation. . . . .	71
5.18	Summary of the test accuracy (%), amongst all window sizes, when predicting sheet against not sheet for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation. . . . .	71
6.1	Comparison of the effect of change in window size on logistic regression and support vector machine Hua and Sun (2001). . . . .	76
6.2	Comparison of the logistic regression result with support vector machine using result of Hua and Sun (2001)and Tsilo (2009). . . . .	77

# List of Figures

3.4.1 Odds, logit and probability . . . . .	22
4.2.1 A figure depicting the root node, non-terminal and terminal nodes of a classification tree. . . . .	32
4.4.1 Binary split the node $t_1$ . . . . .	36
4.4.2 An example of a binary tree. . . . .	37
4.6.1 The maximum tree, $T_0$ . . . . .	42
4.6.2 Sub-tree $T_1$ . . . . .	43
4.6.3 Sub-tree $T_3$ . . . . .	43
4.6.4 Sub-tree $T_1$ . . . . .	44
4.6.5 Sub-tree $T_2$ . . . . .	45
4.6.6 Sub-tree $T_4$ . . . . .	45
5.2.1 An example of a protein amino acid structure. . . . .	48
5.2.2 The four levels of protein structure (From Boundless, 2014). . . . .	49
5.5.1 Creating the data set for a window of size 3 (From Baxter and Jäger, 2011). . .	54
5.6.1 Test accuracy (%) of the classification trees. . . . .	55
5.6.2 Test accuracy (%) of the classification trees using 5-fold cross validation. . . .	57
5.7.1 Test accuracy (%) of the Naive Bayes classifier. . . . .	59
5.7.2 Test accuracy (%) of the Naive Bayes classifier using 5-fold cross validation. . .	61
5.8.1 Test accuracy (%) of the logistic regression classifier. . . . .	62
5.8.2 Test accuracy (%) of the logistic regression classifier using 5-fold cross validation.	64
5.9.1 Comparison of the test accuracies (%) for predicting helix against not helix for each of the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation. . . . .	66

5.9.2 Heuristic confidence intervals: Comparison of the test accuracies (%) for predicting helix against not helix for each of the Naive Bayes, classification tree and logistic regression classifiers using 5-fold cross validation. . . . . 67

5.9.3 Comparison of the test accuracies (%) for predicting coil against not coil for each of the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation. . . . . 69

5.9.4 Heuristic confidence intervals: Comparison of the test accuracies (%) for predicting coil against not coil for each of the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation. . . . . 70

5.9.5 Comparison of the test accuracies (%) for predicting sheet against not sheet for each of the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation. . . . . 72

5.9.6 Heuristic confidence intervals: Comparison of the test accuracies (%) for predicting sheet against not sheet for each of the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation. . . . . 73

# Acknowledgments

I would like to thank my supervisor, Mr Jeremy Baxter, for the patient guidance, encouragement and advice he has provided throughout my time as his student. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly. I would like to thank Professor Sarah Radloff, for giving me a chance to be a student at Rhodes University, and also for her patient guidance and unwavering support through out my study.

I would also like to thank all the members of the Statistics Department at Rhodes University and also those at the University of the Western Cape for all the support they gave me through out my study.

I would also like to acknowledge the support I got from my friends.

Finally, I must express my very profound gratitude to my family especially my mother, brothers (Nasreldein, Mohamed and Ibrahim) and sisters for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.



# Chapter 1

## Outline

### 1.1 Framework

This study considers how a researcher might predict the secondary structure of proteins based on the proteins primary structure. Statistical classification, measures of classifier accuracy and cross validation are introduced and discussed in chapter 2. Chapter 3 discusses how Bayes rule can be used to construct the Naive Bayes classifier and how logistic regression can be utilised to construct a classifier. Classification trees are defined and discussed in chapter 4. Chapter 5 describes the structure of proteins and how the data set used in this study was processed for supervised classification. The results of the various classifiers, using hold-out and 5-fold cross validation, are reported and discussed. These classifiers are compared in the context of this particular data set. Chapter 6 compares these results to those of other researchers. Appendices A to G contain the relevant R code used in this study.

### 1.2 Research Background

Protein secondary structure prediction is the prediction of the secondary structure of a protein based on the primary structure that is from the linear sequence of amino acids. The prediction of the secondary structure depends on the amino acids sequence. An aim of theoretical chemistry and bioinformatics is to predict the sequence of the protein structure from the primary structure (Zhang and Rajapakse, 2009). Some of the computationally based methods that can be used to predict the secondary predictions include Naive Bayes, logistic regression classifier, classification trees, neural networks, support vector machines and nearest neighbor methods (Singh *et al.*, 2008).

## 1.3 Goal of the Research

The aim of this study is to predict the secondary structure of the proteins using three classification approaches, namely the classification tree, Naive Bayes and logistic regression classifiers. This study assesses the performance of these classifiers using hold-out and 5-fold cross validation. To achieve these objectives, R and SPSS are used to perform the calculations relevant to this study.

## 1.4 Introduction

The objective of this study is to train classification tree, Naive Bayes and logistic regression classifiers based on a sequence of protein primary structure in order to predict the proteins secondary structure. The data used in this study consist of 126 proteins available in the Rost and Sander database (Rost and Sander, 1993) available from <http://www.anteprot-pbil.ibcp.fr/>. As discussed in section 5.5, this data set contains the proteins name and the primary and secondary structure sequence of each protein.

The data processing is done in steps: The first step performed is pre-processing. The data is presented as letters and the purpose of pre-processing is to convert the letters into numbers. To achieve this, the orthogonal coding scheme (Holley and Karplus, 1989) is used. The second step is to assign the secondary structure. Secondary structures are classified into 8 categories, namely  $\alpha$ -helix (H),  $3_{10}$ -helix (G),  $\pi$ -helix (I),  $\beta$ -strand (E), isolated  $\beta$ -bridge (B), turn (T), bend (S) and rest (-) where the last category is for unclassified structures. Section 5.5 indicates how these structures are reduced to 3 categories, namely helix (H), sheet (E) and coil (C).

SPSS and R, statistical software, are used to fit and assess the performance or accuracy of the various classifiers used in this study. Hold-out and 5-fold cross validation are used to estimate the performance of each of the classifiers, namely the classification tree (section 5.6), Naive Bayes (section 5.7) and logistic regression (section 5.8) classifiers. The classification tree, Naive Bayes and logistic regression classifiers are compared in section 5.9 using the overall accuracy measure as defined and discussed in section 2.4. These results are compared to those of other researchers in chapter 6.



# Chapter 2

## Classification

### 2.1 Introduction

Section 2.2 introduces classification in a mathematical context. Section 2.3 defines and discusses various measures of classifier accuracy. Section 2.4 discusses these measures for binary classifiers. The cross validation approach to assessing classifier accuracy is discussed in section 2.5.

### 2.2 Classification

Classification is an important technique that is used in statistics. Classification is sometimes known as statistical pattern recognition or discrimination (Breiman *et al.*, 1984). When modeling the relationship between the response variable denoted by  $y$ , and the predictor variable denoted by  $\mathbf{x} = (x_0, x_1, x_2, \dots, x_p)'$ , where  $\mathbf{x} \in \mathbb{R}^{p+1}$ , the predictors may either be continuous or discrete random variables. If the response variable is continuous the modeling process is termed regression modeling. If the response variable is categorical the modeling process termed classification modeling (Izenman, 2009; Han and Micheline, 2006). In classification the response variable is labeled as belonging to one of  $L$  classes. There is often no natural ordering to these classes, but they are labeled as  $1, 2, \dots, L$ , where  $L$  is the number of classes. When there are more than two classes, that is when  $L > 2$ , the modeling is called multi-class classification.

This report focuses on binary variable classification, that is where there are two classes. The class labels are commonly taken to be  $y \in \{0, 1\}$ . It is assumed that the response variable is influenced by the associated predictor variables,  $\mathbf{x}$ . The classification is a conditional distribution where the response variable is binary, where  $p(y = 1 \mid \mathbf{x}) = 1 - p(y = 0 \mid \mathbf{x})$ , and hence follows a Bernoulli distribution with parameter  $\beta$ , which denotes the probability of 1:

$$p(y \mid \beta) = \beta^y (1 - \beta)^{1-y} \text{ where } y = 0, 1$$

The response or dependent variable is affected by the predictor or independent or feature or attribute variables,  $\mathbf{x} = (x_0, x_1, x_2, \dots, x_p)'$  where  $p$  is the number of predictor or independent variables in the data set. Classification models can be used to predict the class of unknown observations. The goal is to build a model and use this model to predict which category a new subject or object belongs to. Thus the purpose of classification is to build a model which can be used for predicting the class label for an observation based on the values of the attributes or independent variables.

## 2.3 Measurement of Classifier Accuracy

There are several accuracy measures of classifier accuracy for instance specificity, sensitivity, misclassification and accuracy rate (Zaki and Meira Jr, 2014). Accuracy measures are designed to focus on specific aspects of a classifiers accuracy, for example the overall classification accuracy (Labatut and Cherifi, 2012; Foody, 2002).

Denote the true or observed class of an observation as  $y_i$  and the associated observations of the independent or predictor variables as  $\mathbf{x}_i \in \mathbb{R}^{p+1}$ . Consider a classifier, denoted by  $M$ , which is simply a function or rule that assigns to  $\mathbf{x}_i$  a class label denoted by  $\hat{y}_i$ , that is

$$M : \mathbf{x}_i \mapsto \hat{y}_i = f(\mathbf{x}_i)$$

Let  $I$  denote an indicator function that has value 1 when the argument is true and 0 otherwise. For each observation  $(\mathbf{x}_i, y_i)$  and associated predicted class label,  $\hat{y}_i$ , where  $i = 1, \dots, n$ , an indicator function can be used to denote a misclassification as follows

$$I(y_i \neq \hat{y}_i) = \begin{cases} 1 & \text{if } \hat{y}_i \neq y_i \\ 0 & \text{if } \hat{y}_i = y_i \end{cases}$$

or a correct classification as

$$I(y_i = \hat{y}_i) = \begin{cases} 1 & \text{if } \hat{y}_i = y_i \\ 0 & \text{if } \hat{y}_i \neq y_i \end{cases}$$

The error rate (Zaki and Meira Jr, 2014, page 603) or misclassification error rate is the fraction of incorrect predictions for the classifier over a data set. The error rate is defined in terms of the indicator function as

$$\text{Error rate} = \frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i)$$

The error rate is an estimate of the probability of misclassification and hence the lower the error rate the better the classifier. The accuracy of a classifier (Zaki and Meira Jr, 2014, page 603) is the fraction of correct predictions over a data set and is defined in terms of the indicator function as

$$\begin{aligned}\text{Accuracy} &= \frac{1}{n} \sum_{i=1}^n I(y_i = \hat{y}_i) \\ &= 1 - \text{Error rate}\end{aligned}$$

The lower the misclassification error rate the higher the accuracy. Accuracy estimates the probability of a correct prediction and hence the higher the accuracy the better the classifier. Classifiers with smaller misclassification error rates, or equivalently higher accuracy, are preferred (Zaki and Meira Jr, 2014, page 602).

The error rate and the accuracy rate are global or overall measures that do not explicitly consider the classes that contribute to the error. This more detailed information can be assessed or measured by tabulating the class specific agreement and disagreement between the true or observed labels and the predicted labels. Accuracy can thus be assessed or measured using a contingency table which is often termed the confusion or error matrix (Sammut and Webb, 2011; Zaki and Meira Jr, 2014, page 604). Consider a set of  $n$  observations of the predictor variables,  $\mathbf{x}_i$ , with true or observed class labels,  $y_i$ , a classifier  $M$  with associated predicted class labels  $\hat{y}_i$  where there are  $L$  classes,  $l \in \{0, \dots, L\}$ . Denote the observed data as  $\mathbf{X} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$  and the associated predicted data as  $\mathbf{X}^* = \{(\mathbf{x}_1, \hat{y}_1), (\mathbf{x}_2, \hat{y}_2), \dots, (\mathbf{x}_n, \hat{y}_n)\}$ .

Partition or group the observed data according to the class labels, that is partition the  $n$  observations into  $L$  classes denoted as  $D = \{D_1, D_2, \dots, D_L\}$  where  $D_l = \{\mathbf{x}_i, y_i = l\}$ . Let  $R = \{R_1, R_2, \dots, R_L\}$  denote the set of grouped or partitioned data according to the predicted class labels, where  $R_l = \{\mathbf{x}_i, \hat{y}_i = l\}$ . Let  $d_l = |D_l|$  denote the size of the true class  $l$  and  $r_l = |R_l|$  denote the size of the observed class  $l$ . Cross tabulate  $R$  and  $D$  into a  $L$  by  $L$  cross-tabulation table, where the entries in the table, denoted by  $n_{jk}$ , are given by

$$n_{jk} = |R_j \cap D_k| = |\{\hat{y}_i = j \cap y_i = k, \mathbf{x}_i \in D\}|$$

where  $j$  and  $k$  denote the class labels (Zaki and Meira Jr, 2014, page 604). Thus if there are  $L$  classes then the confusion matrix is an  $L$  by  $L$  matrix where the columns denote the observed or true class label and the rows represent the predicted or hypothesized class label (Fawcett, 2006). A confusion matrix therefore represents the cross count between the predicted class labels and the actual observed class labels.

## 2.4 Measurement of Binary Classifier Accuracy

Consider a binary classifier which maps each instance or observation,  $y_i$ , to one and only one of two classes, labeled either positive or negative, denoted by  $\hat{y}_i \in \{0, 1\}$ . As shown in table 2.1, for each observation there are four possible outcomes (Fawcett, 2006):

- If the observation is positive and it is classified or predicted as positive it is termed a true positive;
- If the observation is positive and it is classified or predicted as negative, it is termed a false negative;
- If the observation is negative and it is classified or predicted as negative, it is termed a true negative;
- If the observation is negative and it is classified or predicted as positive, it is termed a false positive.

In this context the confusion matrix is 2 by 2 matrix,  $\begin{bmatrix} n_{00} & n_{01} \\ n_{10} & n_{11} \end{bmatrix}$ , for a set of observations as represented in table 2.2. Each entry in the matrix, denoted by  $n_{jk}$ , where  $j = 0, 1$  and  $k = 0, 1$ , indicates the total number of observations of class  $k$  which were assigned to or labeled by the classifier to class  $j$ .  $n_{++}$ , or just  $n$ , denotes the total number of the observations.  $n_{00}$  denotes the number of true positives (TP),  $n_{11}$  denotes the number of true negatives (TN),  $n_{01}$  denotes the number of false positives (FP) and  $n_{10}$  denotes the number of false negatives (FN).  $n_{j+}$  represents the total number of observations predicted in class  $j$ ,  $n_{+k}$  represents the total number of the observations observed to be in class  $k$ . These frequencies can be expressed in terms of the indicator notation as follows

$$\begin{aligned} n_{00} &= \frac{1}{n} \sum_{i=1}^n I(y_i = \hat{y}_i = 0) & n_{11} &= \frac{1}{n} \sum_{i=1}^n I(y_i = \hat{y}_i = 1) \\ n_{10} &= \frac{1}{n} \sum_{i=1}^n I(\hat{y}_i = 1, y_i = 0) & n_{01} &= \frac{1}{n} \sum_{i=1}^n I(\hat{y}_i = 0, y_i = 1) \end{aligned}$$

where the indicator function  $I$  has value 1 when its argument is true and 0 otherwise.

Table 2.1: Possible outcomes of the classification of a binary variable.

Predicted Class $\hat{y}_i$	Observed Class $y_i$		
		Positive: $y_i = 0$	Negative: $y_i = 1$
	Positive: $\hat{y}_i = 0$	True Positive (TP)	False Positive (FP)
	Negative: $\hat{y}_i = 1$	False Negative (FN)	True Negative (TN)
Total		Positives (P)	Negatives (N)

Table 2.2: A confusion matrix for the classification of  $n$  binary observations.

Predicted Class $\hat{y}_i$	Observed Class $y_i$			Total
		Positive: $y_i = 0$	Negative: $y_i = 1$	
	Positive: $\hat{y}_i = 0$	$n_{00}$	$n_{01}$	$n_{0+} = n_{00} + n_{01}$
	Negative: $\hat{y}_i = 1$	$n_{10}$	$n_{11}$	$n_{1+} = n_{10} + n_{11}$
	Total	$n_{+0} = n_{00} + n_{10}$	$n_{+1} = n_{01} + n_{11}$	$n = n_{++}$

Various accuracy metrics for binary classification utilize the confusion matrix. The error rate or misclassification error rate or inaccuracy is defined as the probability of an incorrect classification and is estimated (Han and Micheline, 2006; Zaki and Meira Jr, 2014) as

$$\begin{aligned}
\text{Error rate} &= \frac{\text{False positives (FP)} + \text{False negatives (FN)}}{\text{Total observations}} \\
&= \frac{n_{01} + n_{10}}{n_{++}} = \frac{n_{01} + n_{10}}{n} \\
&= \frac{1}{n} \sum_{i=1}^n I(\hat{y}_i \neq y_i) \\
&= \frac{1}{n} \sum_{i=1}^n I(\hat{y}_i = l, y_i \neq l, l = \{0, 1\}) \\
\therefore \text{Misclassification error rate} &= 1 - \text{Accuracy}.
\end{aligned}$$

For a binary classifier the accuracy is defined as the probability of correct classification and is estimated (Fawcett, 2006) as

$$\begin{aligned}
\text{Accuracy} &= \frac{n_{00} + n_{11}}{n_{++}} = \frac{n_{00} + n_{11}}{n} \\
&= \frac{1}{n} \sum_{i=1}^n I(\hat{y}_i = y_i) \\
&= \frac{1}{n} \sum_{i=1}^n I(\hat{y}_i = l, y_i = l, l \in \{0, 1\}).
\end{aligned}$$

The true positive rate, sensitivity or hit rate or recall, denoted as tp rate, of a binary classifier is estimated (Fawcett, 2006) as

$$\begin{aligned}
\text{Sensitivity or tp rate} &= \frac{\text{Positives correctly classified (TP)}}{\text{Total positives (P)}} \\
&= \frac{n_{00}}{n_{00} + n_{10}} = \frac{n_{00}}{n_{+0}}.
\end{aligned}$$

The false positive rate or false alarm rate, denoted as fp rate, of a binary classifier is estimated (Fawcett, 2006) as

$$\begin{aligned}\text{fp rate} &= \frac{\text{Negatives incorrectly classified (FP)}}{\text{Total negatives (N)}} \\ &= \frac{n_{01}}{n_{01} + n_{11}} = \frac{n_{01}}{n_{+1}}.\end{aligned}$$

For a binary classifier the true negative rate or specificity is estimated (Fawcett, 2006) as

$$\begin{aligned}\text{Specificity} &= \frac{\text{True negatives (TN)}}{\text{False positives (FP) + True negatives (TN)}} \\ &= \frac{n_{11}}{n_{01} + n_{11}} = \frac{n_{11}}{n_{+1}}\end{aligned}$$

For a binary classifier the positive predicted value or precision is an estimate of the probability of correctly classifying the positive outcomes and is defined (Fawcett, 2006) as

$$\begin{aligned}\text{Precision} &= \frac{\text{True positives (TP)}}{\text{True positives (TP) + False positives (FP)}} \\ &= \frac{n_{00}}{n_{00} + n_{01}} = \frac{n_{00}}{n_{0+}}\end{aligned}$$

There is an interchange between sensitivity and specificity: predicting all positive will give the outcome of 100% sensitivity but 0% specificity and vice-versa. Specificity is inversely related to the false positive rate since

$$\begin{aligned}1 - \text{Specificity} &= 1 - \frac{n_{11}}{n_{+1}} \\ &= \frac{n_{+1} - n_{11}}{n_{+1}} = \frac{(n_{01} + n_{11}) - n_{11}}{n_{+1}} = \frac{n_{01}}{n_{+1}} \\ &= \text{fp rate}\end{aligned}$$

Accuracy can be determined from the specificity and sensitivity as follows

$$\begin{aligned}\text{Accuracy} &= \text{Sensitivity} \left( \frac{n_{0+}}{n_{++}} \right) + \text{Specificity} \left( \frac{n_{1+}}{n_{++}} \right) \\ &= \frac{n_{00}}{n_{+0}} \times \frac{n_{+0}}{n_{++}} + \frac{n_{11}}{n_{+1}} \times \frac{n_{+1}}{n_{++}} \\ &= \frac{n_{00}}{n_{++}} + \frac{n_{11}}{n_{++}} = \frac{n_{00} + n_{11}}{n}\end{aligned}$$

In addition to the measures defined above a number of other measures that characterize the performance of a classifier can be found in the literature (Altman and Bland, 1994; Kuhn, 2008;

Zaki and Meira Jr, 2014), for example

$$\begin{aligned}
\text{F-measure} &= \frac{2}{\left(\frac{1}{\text{precision}}\right) + \left(\frac{1}{\text{sensitivity}}\right)} \\
\text{Prevalence} &= \frac{\text{TP} + \text{FN}}{n} \\
\text{Positive Predicted Values} &= \frac{\text{sensitivity} * \text{prevalence}}{\text{sensitivity} * \text{prevalence} + (1 - \text{sensitivity}) (1 - \text{prevalence})} \\
\text{Negative Predicted Values} &= \frac{\text{sensitivity} * (1 - \text{prevalence})}{\text{sensitivity} (1 - \text{prevalence}) + (\text{specificity}) (1 - \text{prevalence})} \\
\text{Detection rate} &= \frac{\text{TP}}{n} \\
\text{Detection prevalence} &= \frac{(\text{TP} + \text{FP})}{n} \\
\text{Balanced accuracy} &= \frac{\text{sensitivity} + \text{specificity}}{2}
\end{aligned}$$

All of these measures are available in the *caret* (Kuhn, 2014) package in R (R Core Team, 2014).

## 2.5 Cross Validation

Cross validation is a method of evaluating and comparing learning algorithms or classifiers by dividing data set  $D$  into two sets, a training data set denoted by  $D_t$  and a validation data set denoted by  $D_v$  (Gareth *et al.*, 2013, page 176). The training data set,  $D_t$ , is used to train or build the classifier, that is the training data is used to estimate the various model parameters. The validation data set,  $D_v$ , is used to evaluate the model or classifier (Refaeilzadeh *et al.*, 2009). Each observation in the data set  $D$  has the same chance of being selected in either of the training or validation data sets. It is assumed that the data are independent, that is the training and validation data came from the same data set which has the same distribution (Burman *et al.*, 1994; Gelman and Wang, 2013). When the response variable,  $y$ , is quantitative the regression model is evaluated using a score function, namely the mean square error (MSE) or the adjusted  $R$  squared statistic (assuming there are multiple independent variables) (Gareth *et al.*, 2013). If the response variable,  $y$ , is qualitative or categorical the classification model is evaluated using a score function which typically is an estimate of the misclassification error (Gareth *et al.*, 2013).

Consider a binary qualitative response variable,  $y_i$ , where the two categories are denoted by  $l = 0, 1$  and a classifier,  $M$ , trained using the training data  $D_t$ . The model accuracy is estimated using the validation data  $D_t$ , thus the accuracy rate or the misclassification rate are estimated

as

$$\begin{aligned}
\text{Accuracy}_v &= \text{Average} \{I_{D_v}(y_i = l, \hat{y}_i = l, l \in \{0, 1\})\} \\
&= \frac{1}{n_v} \sum_{i=1}^{n_v} I(y_i = \hat{y}_i, \mathbf{x}_i \in D_v, n_v = |D_v|) \\
&= A_v \\
\text{Misclassification}_v &= \text{Average} \{I_{D_v}(y_i = l, \hat{y}_i \neq l, l \in \{0, 1\})\} \\
&= \frac{1}{n_v} \sum_{i=1}^{n_v} I(y_i \neq \hat{y}_i, \mathbf{x}_i \in D_v, n_v = |D_v|) \\
&= MC_v
\end{aligned}$$

where  $n_v$  denotes the number of observations in the validation set,  $D_v$ .

The accuracy evaluated using the validation set has been denoted as  $A_v$  and misclassification error evaluated using the validation set as  $MC_v$ . There are several different approaches to cross validation, for example  $k$ -fold cross validation, hold out cross validation and leave one out cross validation (LOOCV) as discussed in the following sections.

### 2.5.1 $k$ -fold Cross Validation

Randomly partition the data set  $D$  into  $k$  approximately equal sized sets or folds.  $k - 1$  folds, that contain  $\frac{(k-1)n}{k}$  observations of the data set  $D$ , are used as the training data to build the model. The remaining fold, that contains  $\frac{n}{k}$  observations, is used as the validation data set to evaluate the model. This process is repeated  $k$  times, where for each iteration a different fold is used as the validation set. The score function using  $k$ -fold cross validation is defined as

$$\begin{aligned}
\text{Accuracy}_{k\text{-fold CV}} &= \text{Average}_{k \text{ folds}} \{A_v^{kj}\} = \frac{1}{k} \sum_{j=1}^k A_v^j \quad (2.5.1) \\
&= \frac{1}{k} \sum_{j=1}^k \left\{ \text{Average}_i \left\{ I_{D_{j\text{th validation set}}}(y_i = \hat{y}_i) \right\} \right\} \\
&= \frac{1}{k} \sum_{j=1}^k \left\{ \frac{1}{n_j} \sum_{i=1}^{n_j} I(y_i = \hat{y}_i, \mathbf{x}_i \in D_j, n_j = |D_j|) \right\}
\end{aligned}$$

where  $A_v^j$  is the accuracy calculated using the  $j^{\text{th}}$  validation set for the classifier trained using training data  $D = D/D_j$ , that is

$$A_v^j = \frac{1}{n_j} \sum_{i=1}^{n_j} I(y_i = \hat{y}_i, \mathbf{x}_i \in D_j, n_j = |D_j|)$$



$$\begin{aligned}
\text{Misclassification}_{k\text{-fold CV}} &= \text{Average}_{k \text{ folds}} \{MC_v^{kj}\} = \frac{1}{k} \sum_{j=1}^k MC_v^j \\
&= \frac{1}{k} \sum_{j=1}^k \left\{ \text{Average}_i \left\{ I_{D_j \text{th validation set}} (y_i \neq \hat{y}_i) \right\} \right\} \\
&= \frac{1}{k} \sum_{j=1}^k \left\{ \frac{1}{n_j} \sum_{i=1}^{n_j} I(y_i = \hat{y}_i, \mathbf{x}_i \in D_j, n_j \neq |D_j|) \right\}
\end{aligned}$$

where  $MC_v^j$  is the misclassification error of the  $j^{\text{th}}$  validation set, for the classifier trained using training data  $D = D/D_j$ .

As an example consider performing 10-fold cross-validation. Start by randomly dividing the data set  $D$  into 10 folds or sets of approximately equal size, denoted as  $D_1, D_2, \dots, D_{10}$ . Consider this as a process with 10 iterations or steps. In iteration 1 train the classifier using training data comprised as  $D_t = \{D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, D_{10}\}$ . Calculate the accuracy rate,  $A_v^1$ , and misclassification error rate,  $MC_v^1$ , using the data in fold  $D_1$  as the validation data. In iteration 2 train the classifier using training data comprised as  $D_t = \{D_1, D_3, D_4, D_5, D_6, D_7, D_8, D_9, D_{10}\}$ . Calculate the accuracy rate,  $A_v^2$ , and misclassification error rate,  $MC_v^2$ , using the data in fold  $D_2$  as the validation data. Continue the process, as shown in table 2.3 until iteration 10 is completed. The accuracy and misclassification rates are then calculated as

$$\begin{aligned}
\text{Accuracy}_{10\text{-fold CV}} &= \frac{1}{10} \sum_{j=1}^{10} A_v^j \\
\text{Misclassification}_{k\text{-fold CV}} &= \frac{1}{10} \sum_{j=1}^{10} MC_v^j.
\end{aligned}$$

Table 2.3: 10-fold cross validation (iteration process).

Iteration	Training data: $D_t$	Validation Data: $D_v$	Accuracy	Misclassification
1	$D_t = \{D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, D_{10}\}$	$D_1$	$A_v^1$	$MC_v^1$
2	$D_t = \{D_1, D_3, D_4, D_5, D_6, D_7, D_8, D_9, D_{10}\}$	$D_2$	$A_v^2$	$MC_v^2$
3	$D_t = \{D_1, D_2, D_4, D_5, D_6, D_7, D_8, D_9, D_{10}\}$	$D_3$	$A_v^3$	$MC_v^3$
	$\vdots$	$\vdots$		
9	$D_t = \{D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_{10}\}$	$D_9$	$A_v^9$	$MC_v^9$
10	$D_t = \{D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9\}$	$D_{10}$	$A_v^{10}$	$MC_v^{10}$

### 2.5.2 Hold-Out Cross Validation

Hold out cross validation is a variation of the  $k$ -fold cross validation method where  $k = 2$  folds are used. The data set  $D$  is randomly divided into two parts, the training data  $D_t$  and

the validation data  $D_v$ . As shown in table 2.4 the data set  $D$  is thus partitioned into two approximately equal sized sets, denoted as  $D_1$  and  $D_2$ . In iteration 1 the first fold,  $D_1$ , is used as the training data to build the classifier while the second fold,  $D_2$ , is used to validate. The process is repeated except the data sets are swapped, that is in iteration 2 the second fold,  $D_2$ , is used to train while the first fold,  $D_1$ , is used to evaluate the model. The score function for the data set  $D$  using hold-out cross validation is defined as  $\text{Accuracy}_{\text{hold out}} = \frac{1}{2} \sum_{j=1}^2 A_v^j$  and  $\text{Misclassification}_{\text{hold out}} = \frac{1}{2} \sum_{j=1}^2 MC_v^j$ .

Table 2.4: Hold-out cross validation.

Training data set $D_t$	$\Rightarrow$	Model	$\Rightarrow$	Validation data set $D_v$	$\Rightarrow$	Accuracy & Misclassification
$D_1$	$\Rightarrow$	First	$\Rightarrow$	$D_2$	$\Rightarrow$	Result: $A_v^1, M_c^1$
$D_2$	$\Rightarrow$	Second	$\Rightarrow$	$D_1$	$\Rightarrow$	Result: $A_v^2, M_c^2$

### 2.5.3 Leave-One-Out Cross Validation

Leave one out cross validation (LOOCV) is a special case of  $k$ -fold cross where  $k = n$ , the number of observations in the data set  $D$ . Partition the data set  $D$  into  $n$  equal folds  $\frac{D}{n} = D_1, D_2, D_3, \dots, D_n$  with one observation per fold. Each  $D_i$  represents a row observation of  $(y_i, \mathbf{x}_i)$  where  $i = 1, 2, 3, \dots, n$ . The training data set contains  $k - 1$  folds or  $n - 1$  observations which are used to fit the model while the validation data set, which contains only one fold namely the single observation  $D_i$ , is used to evaluate the model.

As demonstrated in table 2.5, in iteration  $i$  the training data are  $D_t = \{D_j \text{ where } j \neq i\}$  and validation data are  $D_v = \{D_j\} = \{\mathbf{x}_j\}$ . The classifier is evaluated using this single observation and hence

$$A_v^j = I(y_j = \hat{y}_j, \mathbf{x}_i \in D_j) \quad MC_v^j = I(y_j \neq \hat{y}_j, \mathbf{x}_i \in D_j)$$

The misclassification and accuracy estimate for leave one out cross validation is the average of the  $n$  estimates, that is

$$\begin{aligned} \text{Accuracy}_{\text{LOOCV}} &= \frac{1}{n} \sum_{j=1}^n A_v^j \\ &= \frac{1}{n} \sum_{j=1}^n I(y_j = \hat{y}_j, \mathbf{x}_j \in D_j) \end{aligned}$$

therefore

$$\begin{aligned}
\text{Misclassification}_{\text{LOOCV}} &= \frac{1}{n} \sum_{j=1}^n MC_v^j \\
&= \frac{1}{n} \sum_{j=1}^n I(y_j \neq \hat{y}_j, \mathbf{x}_j \in D_j)
\end{aligned}$$

Table 2.5: Leave one out cross validation.

Iteration	Training data: $D_t$	Validation Data: $D_v$	Accuracy	Misclassification
1	$D_t = \{D_2, D_3, D_4, \dots, D_{n-1}, D_n\}$	$D_1$	$A_v^1$	$MC_v^1$
2	$D_t = \{D_1, D_3, D_4, \dots, D_{n-1}, D_n\}$	$D_2$	$A_v^2$	$MC_v^2$
3	$D_t = \{D_1, D_2, D_4, \dots, D_{n-1}, D_n\}$	$D_3$	$A_v^3$	$MC_v^3$
$\vdots$	$\vdots$	$\vdots$		
$n-1$	$D_t = \{D_1, D_2, D_3, D_4, \dots, D_n\}$	$D_{n-1}$	$A_v^{n-1}$	$MC_v^{n-1}$
$n$	$D_t = \{D_1, D_2, D_3, D_4, \dots, D_{n-1}\}$	$D_n$	$A_v^n$	$MC_v^n$



## Chapter 3

# Classification Using the Naive Bayes and Logistic Regression Classifiers

### 3.1 Introduction

This chapter provides an introduction to the Naive Bayes and logistic regression classifiers for binary classification. Section 3.2 discusses the basic theory of Bayes rule and section 3.3 discusses the Naive Bayes classifier in detail. Section 3.4 discusses the logistic regression classifier, including discussion on how to estimate the logistic regression model parameters.

### 3.2 Classifiers Based on Bayes Rule

The Bayes rule classifier uses Bayes theorem to predict the new class  $l$ ,  $l = 0, 1$  for the observed  $\mathbf{x}$ . It estimates the posterior probability  $P(\mathbf{Y} = l \mid \mathbf{X} = \mathbf{x})$  for each class  $l$ . Observations are classified to the class that has the highest probability (Zaki and Meira Jr, 2014, page 467; Gareth *et al.*, 2013, page 38), thus the predicted class for observation  $\mathbf{x}$  is given as

$$\hat{\mathbf{y}} = \underset{l}{\operatorname{argmax}} \{p(\mathbf{Y} = l \mid \mathbf{X} = \mathbf{x})\} \quad (3.2.1)$$

Bayes theorem can be derived from the basic probability concept by using the conditional probability rule

$$\begin{aligned} p(\mathbf{X} = \mathbf{x}, Y = l) &= p(Y = l \mid \mathbf{X} = \mathbf{x})p(\mathbf{X} = \mathbf{x}) \\ &= p(\mathbf{X} = \mathbf{x} \mid Y = l)p(Y = l) \end{aligned} \quad (3.2.2)$$

equating both sides yields

$$p(Y = l | \mathbf{X} = \mathbf{x})p(\mathbf{X} = \mathbf{x}) = p(\mathbf{X} = \mathbf{x} | Y = l)p(Y = l)$$

and thus

$$p(Y = l | \mathbf{X} = \mathbf{x}) = \frac{p(\mathbf{X} = \mathbf{x} | Y = l)p(Y = l)}{p(\mathbf{X} = \mathbf{x})} \quad (3.2.3)$$

$p(\mathbf{X} = \mathbf{x})$  is the probability of observing  $\mathbf{X} = \mathbf{x}$  from any class  $l$ ,  $l = 0, 1$  given as

$$\begin{aligned} p(\mathbf{X} = \mathbf{x}) &= p((\mathbf{X} = \mathbf{x}, Y = 0) \cup (\mathbf{X} = \mathbf{x}, Y = 1)) \\ &= p(\mathbf{X} = \mathbf{x}, Y = 0) + p(\mathbf{X} = \mathbf{x}, Y = 1) \end{aligned}$$

From equation 3.2.2

$$p(\mathbf{X} = \mathbf{x}) = p(\mathbf{X} = \mathbf{x} | Y = 0)p(Y = 0) + p(\mathbf{X} = \mathbf{x} | Y = 1)p(Y = 1) \quad (3.2.4)$$

Substituting equation 3.2.4 into the denominator of equation 3.2.3 yields Bayes theorem which gives the posterior probability in term of the likelihood and the prior probability.  $p(Y = l | \mathbf{X} = \mathbf{x})$  is posterior probability of the observation in class  $l$  given the distribution of  $\mathbf{X} = \mathbf{x}$ ,  $p(\mathbf{X} = \mathbf{x} | Y = l)$  is the likelihood function of observing  $\mathbf{X}$  assuming  $\mathbf{Y}$  belongs to class  $l$  and  $p(Y = l)$  is the prior probability of class  $l$ .

$$\begin{aligned} p(Y = l | \mathbf{X} = \mathbf{x}) &= \frac{p(\mathbf{X} = \mathbf{x} | Y = l)p(Y = l)}{p(\mathbf{X} = \mathbf{x} | Y = 0)p(Y = 0) + p(\mathbf{X} = \mathbf{x} | Y = 1)p(Y = 1)} \\ &= \frac{p(\mathbf{X} = \mathbf{x} | Y = l)p(Y = l)}{\sum_{l=0}^1 p(\mathbf{X} = \mathbf{x} | Y = l)p(Y = l)} \end{aligned} \quad (3.2.5)$$

where  $p(\mathbf{x})$  is the marginal probability. Bayes rule in equation 3.2.1 can be rewritten as a posterior probability

$$\begin{aligned} \hat{\mathbf{y}} &= \operatorname{argmax}_l \{p(\mathbf{Y} = l | \mathbf{X} = \mathbf{x})\} \\ &= \operatorname{argmax}_l \left\{ \frac{p(\mathbf{X} = \mathbf{x} | Y = l)p(Y = l)}{\sum_{l=0}^1 p(\mathbf{X} = \mathbf{x} | Y = l)p(Y = l)} \right\} \\ &= \operatorname{argmax}_l \{p(\mathbf{X} = \mathbf{x} | Y = l)p(Y = l)\} \end{aligned}$$

since the  $p(\mathbf{X} = \mathbf{x}) = \sum_{l=0}^1 p(\mathbf{X} = \mathbf{x} \mid Y = l)p(Y = l)$  is fixed for a particular  $\mathbf{x}$ . Thus the predicted class depends on the likelihood of the relevant class and prior probability of that class as follows (Zaki and Meira Jr, 2014, page 468)

$$\hat{\mathbf{y}} = \underset{l \in \{0,1\}}{\operatorname{argmax}} \{p(\mathbf{X} = \mathbf{x} \mid Y = l)p(Y = l)\}$$

### 3.2.1 Estimating the Prior Probability

The likelihood,  $p(\mathbf{X} = \mathbf{x} \mid Y = l)$ , and the prior class probabilities,  $p(Y = l)$ , can be estimated from the training data set  $D$  (Zaki and Meira Jr, 2014, page 468). Divide the training data set  $D$  by the number of the classes  $l$ . Let  $D_l$  represent the subset of the training data set  $D$  labeled as the class  $l$ ,  $l = 0, 1$ , that is

$$\begin{aligned} D &= \{\mathbf{X} \in \mathbb{R}^p \text{ which have class label } Y = l, l = 0, 1\} \\ D_0 &= \{\mathbf{X} \in \mathbb{R}^p \text{ which have class label } Y = 0\} \\ D_1 &= \{\mathbf{X} \in \mathbb{R}^p \text{ which have class label } Y = 1\} \end{aligned}$$

The training data set has  $n$  observations and each subset  $D_l$  has  $n_l$  observations. The prior probability for class  $l$  can be estimated as

$$p(Y = l) = \frac{n_l}{n} \quad \text{for } l = 0, 1$$

### 3.2.2 Estimating the Likelihood

The likelihood function,  $p(\mathbf{X} = \mathbf{x} \mid Y = l)$  is estimated from the joint probability of all  $\mathbf{X} = \mathbf{x}$  over  $p$  dimensions, that is seek

$$\max \{L(\beta)\} = \max \{p(\mathbf{X} \in \mathbb{R}^p \mid Y = l)\}$$

If all the features or attributes are numeric either a parametric, for example using the multivariate normal distribution, or a non-parametric approach can be followed (Zaki and Meira Jr, 2014, page 468). If the features or attributes are categorical then a categorical approach, using for example the multivariate Bernoulli distribution, can be utilised (Zaki and Meira Jr, 2014, page 471). Often however there is not sufficient data to reliably estimate the joint probability density or mass function, especially for high-dimensional data (Zaki and Meira Jr, 2014, page 473). An approach to overcome this is to reduce the set of parameters, as described next section.

### 3.3 Naive Bayes Classifier

The Naive Bayes classifier is based on Bayes Rule. It assumes that the attributes,  $\mathbf{X} \in \mathbb{R}^p$ , are conditionally independent of the response,  $Y$  (Bishop, 2006; McCallum and Nigam, 1998; Lewis, 1998). The training dataset  $D$  consists of  $n$  observations of  $p$  variables, the binary response variable denotes the class label,  $Y = l$  where  $l = 0, 1$ . The Naive Bayes classifier uses Bayes theorem directly to predict the class for a new test instance, given  $\mathbf{X} = \mathbf{x}$ . Applying the independent assumption of the Naive Bayes classifier, the general property of probabilities and the chain rule to equation 3.2.2 as follows:

$$\begin{aligned} p(\mathbf{X} = \mathbf{x} \mid Y = l) &= p(\mathbf{X} \in \mathbb{R}^p \mid Y = l) \\ &= p(X_1 = x_1, X_2 = x_2, \dots, X_p = x_p \mid Y = l) \end{aligned} \quad (3.3.1)$$

$$\begin{aligned} &= p(X_1 = x_1 \mid Y = l) \times p(X_{p-1} = x_{p-1}, \dots, X_p = x_p \mid Y = l, X_1 = x_1) \\ &= p(X_1 = x_1 \mid Y = l) \times \dots \times p(X_p = x_p \mid Y = l, X_1 = x_1, \dots, X_{p-1} = x_{p-1}) \\ &= p(X_1 = x_1 \mid Y = l) \times p(X_2 = x_2 \mid Y = l) \times \dots \times p(X_p = x_p \mid Y = l) \\ &= \prod_{j=1}^p p(X_j = x_j \mid Y = l) \end{aligned} \quad (3.3.2)$$

We derive Naive Bayes classifier from Bayes rules by applying the equation 3.3.2 into equation 3.2.5 as follows

$$p(Y = l \mid \mathbf{X} = \mathbf{x}) = \frac{p(Y = l) \prod_{j=1}^p p(X_j = x_j \mid Y = l)}{\sum_{l=0}^1 p(Y = l) \prod_{j=1}^p p(X_j = x_j \mid Y = l)} \quad (3.3.3)$$

Equation 3.3.3 is the fundamental equation for the Naive Bayes classifier. Shown above is how to calculate the probability that the response is of class  $l$  when the observed attributes have values  $\mathbf{x}$ .  $p(Y = l)$  and  $p(X_j = x_j \mid Y = l)$  are estimated from the training data set. The Naive Bayes classification rule is thus:

$$\hat{\mathbf{y}} = \underset{l \in \{0, 1\}}{\operatorname{argmax}} \left\{ \frac{p(Y = l) \prod_{j=1}^p p(X_j = x_j \mid Y = l)}{\sum_{l=0}^1 p(Y = l) \prod_{j=1}^p p(X_j = x_j \mid Y = l)} \right\} \quad (3.3.4)$$

In practice we are only interested in the numerator of equation 3.3.4, since the denominator does not depend on  $Y$  as the values of the features  $\mathbf{X}$  are given and hence the denominator is effectively constant (Han *et al.*, 2012, page 352). The Naive Bayes classifier is thus defined as



follows

$$\hat{y} = \underset{l \in \{0, 1\}}{\operatorname{argmax}} \left\{ p(Y = l) \prod_{j=1}^p p(X_j = x_j \mid Y = l) \right\}$$

### 3.3.1 Estimating the Maximum Likelihood for Naive Bayes Classifier

We seek the maximum likelihood estimates for the Naive Bayes classifier, that is we seek estimates to maximize

$$L(\beta) = p(Y = l) \prod_{j=1}^p p(X_j = x_j \mid Y = l) \quad (3.3.5)$$

From equation 3.2.2 we can rewrite equation 3.3.5 as follows

$$\begin{aligned} \log \{L(\beta)\} &= \log \left\{ p(Y = l) \prod_{j=1}^p p(\mathbf{X} \in \mathbb{R}^p \mid Y = l) \right\} \\ l(\beta) &= \log \{p(Y = l)\} + \log \left\{ \prod_{j=1}^p p(\mathbf{X} \in \mathbb{R}^p \mid Y = l) \right\} \\ &= \log \{p(Y = l)\} + \sum_{j=1}^p \log \{p(\mathbf{X} \in \mathbb{R}^p \mid Y = l)\} \end{aligned}$$

For the maximum likelihood estimation we seek the parameter values that maximize  $l(\beta)$ . This leads to the following points:

- $p(Y = l) \geq 0$  for all  $l = 0, 1$ ;
- For all  $Y$  and  $\mathbf{X}$ ,  $p(\mathbf{X} = \mathbf{x} \mid Y = l) \geq 0$ ;
- For all  $j = 1, 2, \dots, p$  and  $l = 0, 1$  and  $\sum_{j,l} p(\mathbf{X} = \mathbf{x} \mid Y = l) = 1$ .

We utilize Lagrange multipliers to estimate the parameters  $p(Y = l)$  and  $p(\mathbf{X} \in \mathbb{R}^p \mid Y = l)$  from the training data set  $D$ :

$$p(Y = l) = \frac{\sum_{i=1}^n (Y_i = l)}{n} = \frac{\text{count}(Y_i)}{n}$$

where  $i = 1, 2, \dots, n$  and  $l = 0, 1$ .  $\sum_i (Y_i = l) = \text{count}(Y_i)$  is simply the number of times that the label  $Y_i = l$  is seen in the training data set.

Similarly, the maximum likelihood estimates for the  $p(\mathbf{X} = x \mid Y = l)$  parameters for all  $l = 0, 1$ , for all  $i = 1, 2, \dots, n$  and for all  $j = 1, 2, \dots, p$  take the following form

$$\begin{aligned} p(\mathbf{X} = x \mid Y = l) &= \frac{\sum_{i=1}^n (Y_i = l, X_{ij} = x_{ij})}{\sum_{i=1}^n (Y_i = l)} \\ &= \frac{\text{count}_i(\mathbf{X} \in \mathbb{R}^p, Y = l)}{\text{count}(Y_i = l)} \end{aligned}$$

where  $\text{count}_i(Y = l, X = x) = \sum_{i=1}^n (Y_i = l, X_{ij} = x_{ij})$  we simply count the number of times label  $Y = l$  is seen in conjunction with  $x_j$  taking value  $x$ ; count the number of times the label  $Y = l$  is seen in total; then take the ratio of these two terms.

It must be noted that the parameter estimation or learning described above is fairly general. The only special requirement is the Naive Bayes assumption which assumes conditional independence of features. This makes it a Naive Bayes classifier (Zhu, 2010).

## 3.4 Logistic Regression

Binary logistic regression is an approach to learning the function of the posterior probability  $p(Y = l \mid \mathbf{X} = \mathbf{x})$ . The aim is to determine the target variable  $y$  which is binary,  $y \in (0, 1)$ . The target variable is influenced by the associated predictor variables  $\mathbf{x}$ ,  $\mathbf{x} = [x_1, \dots, x_p]' \in \mathbb{R}^p$ . The predictor variables can be quantitative or qualitative (nominal or ordinal). Logistic regression models can be used to predict the probability that  $y$  belongs to a particular class (Hastie *et al.*, 2009, page 95).

### 3.4.1 The Logistic Regression Model

Logistic regression is the logit or the natural logarithm of the odds ratio (Peng and So, 2002). It models the relationship between the target variable  $y$  denote by  $p(Y = l \mid \mathbf{X} = \mathbf{x})$ , as a linear function of predictor random variables denoted by  $g(\mathbf{x})$

$$\begin{aligned} g(\mathbf{x}) &= \boldsymbol{\beta}'\mathbf{x} \\ &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p \end{aligned}$$

where  $\boldsymbol{\beta}_{(p+1) \times 1}$  is a vector of the unknown parameters  $\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, \dots, \beta_p]'$  and  $\mathbf{x}_{(p+1) \times 1}$  is a vector of the predictor variables,  $\mathbf{x} = [1, x_1, x_2, \dots, x_p]'$ . Logistic regression models the

relationship between the probability of an observation being in class  $l$  and the linear function of  $g(x)$ . The probability that  $y$  is in the class 1 given the data, denoted by  $\pi_1$ , is modeled as

$$\begin{aligned}
 \pi_1 &= p(Y = 1 \mid g(\mathbf{x})) \\
 &= \frac{\exp(g(\mathbf{x}))}{1 + \exp(g(\mathbf{x}))} \\
 &= \frac{1}{\exp(-g(\mathbf{x}))(1 + \exp(g(\mathbf{x})))} \\
 &= \frac{1}{\exp(-g(\mathbf{x})) \exp(g(\mathbf{x})) + \exp(-g(\mathbf{x}))} \\
 &= \frac{1}{1 + \exp(-g(\mathbf{x}))}
 \end{aligned} \tag{3.4.1}$$

The sum of the total probability is equal to one, since there are two class

$$p(Y = 1 \mid g(\mathbf{x})) + p(Y = 0 \mid g(\mathbf{x})) = 1$$

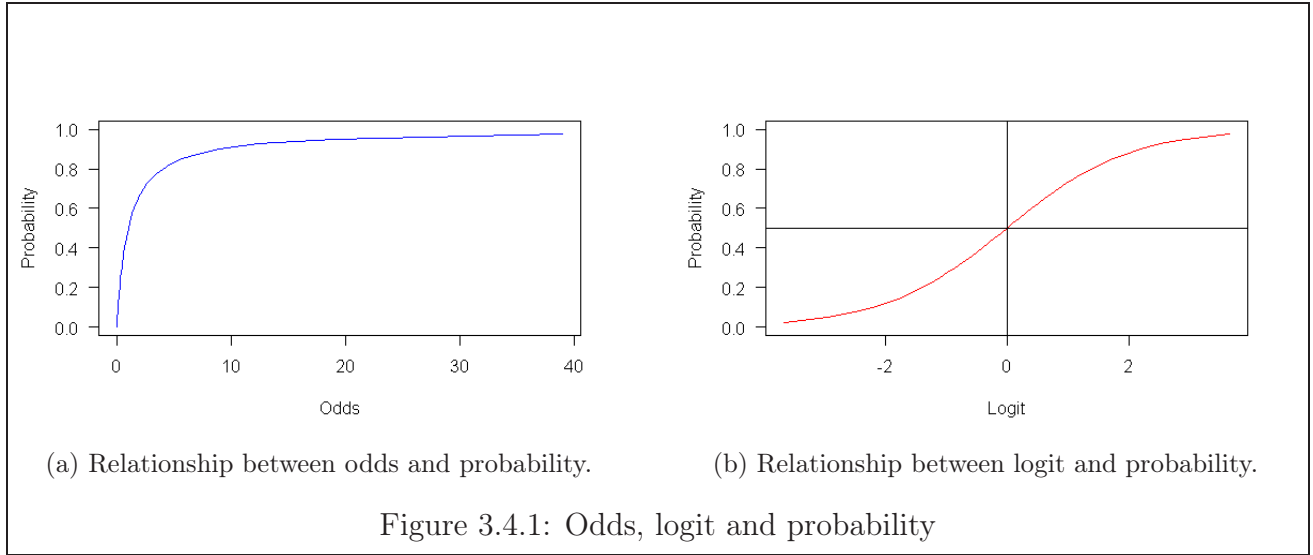
Therefore

$$\begin{aligned}
 \pi_0 &= p(Y = 0 \mid g(\mathbf{x})) = 1 - p(Y = 1 \mid g(\mathbf{x})) \\
 &= 1 - \frac{\exp(g(\mathbf{x}))}{1 + \exp(g(\mathbf{x}))} = \frac{1 + \exp(g(\mathbf{x})) - \exp(g(\mathbf{x}))}{1 + \exp(g(\mathbf{x}))} \\
 &= \frac{1}{1 + \exp(g(\mathbf{x}))}
 \end{aligned}$$

From equation 3.4.1 denote  $\pi = p(Y = 1 \mid \mathbf{X} = \mathbf{x})$  as the probability of observing  $y$  in class 1 given the data  $\mathbf{x}$ . The logistic regression function of  $\pi$  can be expressed as

$$\pi = \frac{\exp(g(\mathbf{x}))}{1 + \exp(g(\mathbf{x}))} \tag{3.4.2}$$

The odds ratio is defined as the ratio of the probability of success and the probability of failure. Given  $\pi$  as the probability of success, the odds ratio is defined as  $\text{odds} = \frac{\pi}{1-\pi}$ . The odds ratio is a non-negative function, that is the  $\text{odds} \geq 0$ , as shown in figure 3.4.1a.



By rearranging the logistic regression function of equation 3.4.2 it can be seen that the odds ratio is an exponential function of  $g(\mathbf{x})$  since

$$\begin{aligned}
 \pi &= \frac{\exp(g(\mathbf{x}))}{1 + \exp(g(\mathbf{x}))} \\
 \pi(1 + \exp(g(\mathbf{x}))) &= \exp(g(\mathbf{x})) \\
 \pi + \pi \exp(g(\mathbf{x})) &= \exp(g(\mathbf{x})) \\
 \pi &= \exp(g(\mathbf{x})) - \pi \exp(g(\mathbf{x})) \\
 \pi &= \exp(g(\mathbf{x}))(1 - \pi) \\
 \frac{\pi}{1 - \pi} &= \exp(g(\mathbf{x})) = \exp(\beta' \mathbf{x}) \tag{3.4.3}
 \end{aligned}$$

The logit of  $\pi$  is observed by taking the logarithm of both sides of the odds ratio of equation 3.4.3. The logistic regression model predicts the logit of  $\pi$ .  $\pi$  is the predicted probability that  $Y = 1$ , from  $\mathbf{x}$ . The logit of the probability of success,  $\pi$ , is a linear function of the predictor variables  $\mathbf{x}$  and unknown parameters  $\beta$ . As shown in figure 3.4.1b, if the probability of success  $\pi = \frac{1}{2}$  then the logit of  $\pi$  is equal to zero. If the probability of success is greater than  $\frac{1}{2}$  then the logit of  $\pi$  is positive. A probability of success less than  $\frac{1}{2}$  corresponds to the negative result of logit of  $\pi$ .

In summary the logistic regression model can be formulated as:

$$\begin{aligned}
 \text{logit}(\pi) &= \ln\left(\frac{\pi}{1-\pi}\right) \\
 &= g(\mathbf{x}) \\
 &= \beta' \mathbf{x} \\
 &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p
 \end{aligned}$$

### 3.4.2 Estimation of the Logistic Regression Model Parameters

The response variable  $y$  is binary such that each  $Y$  belongs to one of the two classes 0 or 1 with fixed probability of being in each class  $\pi_1$  or  $\pi_0 = (1 - \pi_1)$ . The probability distribution function of  $p(Y = l | \mathbf{X} = \mathbf{x})$  is the probability of observing  $Y$  belongs to class  $l \in \{0, 1\}$  given  $\mathbf{X}$ , has a Bernoulli distribution with probability of success is  $\pi$

$$p(Y = l | \mathbf{X} = \mathbf{x}) = \begin{cases} \pi_i^{y_i} (1 - \pi_i)^{1-y_i} & \text{for } y_i = 0, 1 \\ 0 & \text{otherwise} \end{cases}$$

where  $E(Y_i | \mathbf{x}) = \mu_i = \pi_i$  and  $Var(Y_i | \mathbf{x}) = \sigma_i^2 = \pi(1 - \pi_i)$ .

Logistic regression models the number of observation in each class based on the sample observations. Consider taking a random sample of size  $n_i$ . Let  $m_i$  denote the number of successes, thus  $m_i$  is the sum of  $y_i$ ,  $m_i > 0$ . There are  $n_i$  independent Bernoulli random variables with fixed probability of success  $\pi$  and failure  $(1 - \pi)$ .  $m_i$  has a Binomial distribution with parameter  $\pi$ , sample size  $n_i$ ,  $m_i \sim \text{binomial}(n_i, \pi)$ :

$$p(M_i = m_i | \mathbf{X} = \mathbf{x}_j) = \binom{n_i}{m_i} \pi^{m_i} (1 - \pi)^{n_i - m_i}$$

Note that  $E(M_i | \mathbf{x}) = \mu_i = n_i \pi$  and  $Var(M_i | \mathbf{x}) = \sigma_i^2 = n_i \pi (1 - \pi)$ . The likelihood is given by

$$L(\pi, \boldsymbol{\beta}) = \prod_{i=1}^n \binom{n_i}{m_i} \pi^{m_i} (1 - \pi)^{n_i - m_i}$$

Since log is an increasing function, the maximum log likelihood is the same as the maximum likelihood. The log converts the product into a sum, that is

$$\ln \{L(\pi, \boldsymbol{\beta})\} = \sum_i^n (m_i \ln \pi_i + (n_i - m_i) \ln(1 - \pi) + \ln \binom{n_i}{m_i}) \quad (3.4.4)$$

where  $\ln \binom{n_i}{m_i}$  is a constant, independent of the unknown parameters and hence it can be ignored when the derivatives are taken with respect to the unknown parameters.

Substituting equation 3.4.2 into equation 3.4.4 yields

$$\begin{aligned}
 \ln \{L(\pi, \beta)\} &= \ln \left\{ \prod_i^n \binom{n_i}{m_i} \pi^{m_i} (1 - \pi)^{n_i - m_i} \right\} \\
 l(\pi, \beta) &= \sum_{i=1}^n \left\{ \ln(\pi^{m_i}) + \ln(1 - \pi)^{n_i - m_i} + \ln \binom{n_i}{m_i} \right\} \\
 &= \sum_i \left\{ m_i \ln \pi + (n_i - m_i) \ln(1 - \pi) + \ln \binom{n_i}{m_i} \right\} \\
 &= \sum_i \left\{ m_i \ln \pi - m_i \ln(1 - \pi) + n \ln(1 - \pi) + \ln \binom{n_i}{m_i} \right\} \\
 &= \sum_i \left\{ m_i (\ln \pi - \ln(1 - \pi)) + n \ln(1 - \pi) + \ln \binom{n_i}{m_i} \right\} \\
 &= \sum_{i=1}^n \left\{ m_i \ln \left( \frac{\pi}{1 - \pi} \right) + n \ln(1 - \pi) + \ln \binom{n_i}{m_i} \right\}
 \end{aligned}$$

where we assume that

$$\ln \left( \frac{\pi}{1 - \pi} \right) = g(\mathbf{x}_i) = \beta' \mathbf{x}_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \dots + \beta_p x_{pi}$$

and hence

$$\begin{aligned}
 l(\pi, \beta) &= \sum_{i=1}^n \left( m_i \beta' \mathbf{x}_i + n (1 + \exp(\beta' \mathbf{x}_i))^{-1} + \ln \binom{n_i}{m_i} \right) \\
 \therefore l(\beta) &= \sum_{i=1}^n \left( m_i \beta' \mathbf{x}_i - n (1 + \exp(\beta' \mathbf{x}_i)) + \ln \binom{n_i}{m_i} \right)
 \end{aligned}$$

To find the maximum likelihood,  $L(\beta)$ , differentiate the log likelihood with respect to the unknown parameters  $\beta$  and set the derivatives equal to zero. The resulting equations are known as the score equations, that is for  $j = 0, \dots, p$

$$\begin{aligned}
 l'(\beta_j) = \frac{\partial \ln \{L(\beta)\}}{\partial \beta_j} &= \sum_{i=1}^n \left( m_i x_{ij} - \frac{n_i x_{ij} \exp(\beta' \mathbf{x}_i)}{1 + \exp(\beta' \mathbf{x}_i)} \right) \\
 &= \sum_i \left( x_{ij} \left( m_i - \frac{n_i \exp(\beta' \mathbf{x}_i)}{1 + \exp(\beta' \mathbf{x}_i)} \right) \right) \\
 &= \sum_i x_{ij} (m_i - n_i p(y = 1 | \mathbf{X} = \mathbf{x}_i)) = 0
 \end{aligned}$$

where  $\mu_i = n_i p(Y = 1 | \mathbf{X} = \mathbf{x}_i) = n_i \pi$ ,  $\mathbf{x}_i$  is the vector of  $(p + 1)$  of the predictor variables.

Thus the  $j$  equations can be expressed as

$$l'(\beta_j) = \sum_i^n x_i(m_i - n_i\pi) \text{ for } j = 0, 1, \dots, p$$

It is convenient to express these in matrix form, that is as

$$l'(\beta) = \frac{\partial l(\beta)}{\partial \beta} = \mathbf{X}(\mathbf{m} - \boldsymbol{\mu}) \quad (3.4.5)$$

where

$$\begin{aligned} \mathbf{X}(\mathbf{m} - \boldsymbol{\mu}) &= \begin{bmatrix} x_{10} & \cdots & x_{1p} \\ x_{20} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n0} & \cdots & x_{np} \end{bmatrix}' \left( \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_n \end{bmatrix} - \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} \right) \\ &= \begin{bmatrix} \sum_{i=1}^n x_{i0} (m_i - \mu_i) \\ \sum_{i=1}^n x_{i1} (m_i - \mu_i) \\ \vdots \\ \sum_{i=1}^n x_{ip} (m_i - \mu_i) \end{bmatrix} \end{aligned}$$

When setting these  $p + 1$  nonlinear equations of the  $p + 1$  parameters equal to zero there is no closed form solution. The nonlinear equation can be solved using the Newton Raphson algorithm, which requires the second derivatives, called the Hessian matrix

### 3.4.2.1 The Newton Raphson Method

The Newton Raphson method can be used to find solution for the nonlinear score equations [Hastie *et al.*, 2009, page 99; Izenman, 2009, page 243] as follows

$$l'(\beta_j) = \sum_i^n x_i(m_i - n_i\pi_i) = 0 \text{ for } j = 0, 1, \dots, p$$

There are  $p + 1$  equations since  $j = 0, 1, \dots, p$ , that is

$$\begin{aligned} l'(\beta_0) &= \sum_i^n x_{i0}(m_i - n_i\pi_i) = 0 \\ l'(\beta_1) &= \sum_i^n x_{i1}(m_i - n_i\pi_i) = 0 \\ &\vdots = \vdots \\ l'(\beta_p) &= \sum_i^n x_{ip}(m_i - n_i\pi_i) = 0 \end{aligned}$$

These score equations are typically denoted as a vector

$$\begin{aligned} l'(\boldsymbol{\beta}) &= \begin{bmatrix} \frac{\partial \ln\{L(\boldsymbol{\beta})\}}{\partial \beta_0} \\ \frac{\partial \ln\{L(\boldsymbol{\beta})\}}{\partial \beta_1} \\ \vdots \\ \frac{\partial \ln\{L(\boldsymbol{\beta})\}}{\partial \beta_p} \end{bmatrix} = \begin{bmatrix} \sum_i^n x_{i0}(m_i - n_i\pi_i) \\ \sum_i^n x_{i1}(m_i - n_i\pi_i) \\ \vdots \\ \sum_i^n x_{ip}(m_i - n_i\pi_i) \end{bmatrix} \\ &= \mathbf{X}(\mathbf{m} - \boldsymbol{\mu}) \end{aligned}$$

where  $\mathbf{X}$ ,  $\mathbf{m}$  and  $\boldsymbol{\mu}$  are defined above.

The partial derivatives of the components of  $l'(\boldsymbol{\beta})$  are

$$\begin{aligned} l''(\beta_j) &= \frac{\partial^2 \ln \{L(\boldsymbol{\beta})\}}{\partial \beta_i \partial \beta_j} = - \sum_i^n n_i \frac{x_{ij}(x_{ij} \exp(\boldsymbol{\beta}'\mathbf{x})(1 + \exp(\boldsymbol{\beta}'\mathbf{x})) - x_{ij} \exp(\boldsymbol{\beta}'\mathbf{x})\exp(\boldsymbol{\beta}'\mathbf{x}))}{(1 + \exp(\boldsymbol{\beta}'\mathbf{x}))^2} \\ &= - \sum_i^n n_i x'_{ij} \left( \frac{(\exp(\boldsymbol{\beta}'\mathbf{x}) + \exp(2\boldsymbol{\beta}'\mathbf{x})) - \exp(2\boldsymbol{\beta}'\mathbf{x})}{(1 + \exp(\boldsymbol{\beta}'\mathbf{x}))^2} \right) x_{ij} \\ &= - \sum_i^n n_i x'_{ij} \left( \frac{\exp(\boldsymbol{\beta}'\mathbf{x})}{(1 + \exp(\boldsymbol{\beta}'\mathbf{x}))} \right) x_{ij} \\ &= - \sum_i^n n_i x'_{ij} \left( \frac{\exp(\boldsymbol{\beta}'\mathbf{x})}{(1 + \exp(\boldsymbol{\beta}'\mathbf{x}))(1 + \exp(\boldsymbol{\beta}'\mathbf{x}))} \right) x_{ij} \\ &= - \sum_i^n n_i x'_{ij} \pi_i (1 - \pi_i) x_{ij} \\ l''(\boldsymbol{\beta}_j) &= - \sum_i^n x'_{ij} \Sigma_i x_{ij} \end{aligned}$$

The components of the covariance matrix are  $\Sigma_i = \delta^2 = n_i \pi_i (1 - \pi_i)$  and hence the Hessian



matrix is given by

$$\begin{aligned}
 l''(\boldsymbol{\beta}) &= \left[ \frac{\partial'' \ln \{L(\boldsymbol{\beta})\}}{\partial \beta_i \beta_j} \right]_{ij} \text{ for } i, j = 0, 1, 2, \dots, p \\
 &= \begin{bmatrix} \frac{\partial'' \ln \{L(\boldsymbol{\beta})\}}{\partial \beta_0''} & \frac{\partial'' \ln \{L(\boldsymbol{\beta})\}}{\partial \beta_0' \beta_1'} & \dots & \frac{\partial'' \ln \{L(\boldsymbol{\beta})\}}{\partial \beta_0' \beta_p'} \\ \frac{\partial'' \ln \{L(\boldsymbol{\beta})\}}{\partial \beta_1' \beta_0'} & \frac{\partial'' \ln \{L(\boldsymbol{\beta})\}}{\partial \beta_1''} & \dots & \frac{\partial'' \ln \{L(\boldsymbol{\beta})\}}{\partial \beta_1' \beta_p'} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial'' \ln \{L(\boldsymbol{\beta})\}}{\partial \beta_p' \beta_0'} & \frac{\partial'' \ln \{L(\boldsymbol{\beta})\}}{\partial \beta_p' \beta_1'} & \dots & \frac{\partial'' \ln \{L(\boldsymbol{\beta})\}}{\partial \beta_p''} \end{bmatrix} \\
 &= \begin{bmatrix} -\sum_i^n x'_{i0} \Sigma x_{i0} & -\sum_i^n x'_{i0} \Sigma x_{i1} & \dots & -\sum_i^n x'_{i0} \Sigma x_{ip} \\ -\sum_i^n x'_{i0} \Sigma x_{i1} & -\sum_i^n x'_{i1} \Sigma x_{i1} & \dots & -\sum_i^n x'_{i1} \Sigma x_{ip} \\ \vdots & \vdots & \ddots & \vdots \\ -\sum_i^n x'_{i0} \Sigma x_{ip} & -\sum_i^n x'_{i1} \Sigma x_{ip} & \dots & -\sum_i^n x'_{ip} \Sigma x_{ip} \end{bmatrix} \\
 &= - \begin{bmatrix} \sum_i^n x'_{i0} \Sigma x_{i0} & \sum_i^n x'_{i0} \Sigma x_{i1} & \dots & \sum_i^n x'_{i0} \Sigma x_{ip} \\ \sum_i^n x'_{i0} \Sigma x_{i1} & \sum_i^n x'_{i1} \Sigma x_{i1} & \dots & \sum_i^n x'_{i1} \Sigma x_{ip} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_i^n x'_{i0} \Sigma x_{ip} & \sum_i^n x'_{i1} \Sigma x_{ip} & \dots & \sum_i^n x'_{ip} \Sigma x_{ip} \end{bmatrix} \\
 &= -\mathbf{X}'\Sigma\mathbf{X}
 \end{aligned} \tag{3.4.6}$$

The Newton's Raphson iterative approach to finding maximum likelihood  $ML$  estimates is a building block for the iteratively re-weighted least squares ( $IRLS$ ) algorithm.

By beginning with  $\boldsymbol{\beta}^{(0)} = 0$  the  $(j)^{st}$  step gets substituted by the  $(j+1)^{th}$  step until the learning rate,  $\frac{L'(\boldsymbol{\beta})}{L''(\boldsymbol{\beta})}$  is equal or close to zero.

$$\begin{aligned}
 &= \hat{\boldsymbol{\beta}}^{(j)} + \frac{L'(\boldsymbol{\beta})}{[L''(\boldsymbol{\beta})]} \\
 &= \hat{\boldsymbol{\beta}}^{(j)} + [L''(\boldsymbol{\beta})]^{-1} L'(\boldsymbol{\beta})
 \end{aligned} \tag{3.4.7}$$

Keep updating equation 3.4.7 until there is a small change between the components from one iteration to the next.

By referring to equations ?? and 3.4.6 the estimated parameter of  $ML$  by Newton's method in

matrix notation can be written as

$$\begin{aligned}
 \hat{\beta}^{(j+1)} &= \hat{\beta}^{(j)} + [\mathbf{x}'\Sigma\mathbf{x}]^{-1} \mathbf{X}(\mathbf{m} - \boldsymbol{\mu}) \\
 &= [\mathbf{x}'\Sigma\mathbf{x}]^{-1} \mathbf{x}\Sigma\mathbf{x}\hat{\beta}^{(j)} + [\mathbf{x}'\Sigma\mathbf{x}]^{-1} \mathbf{x}\Sigma\Sigma^{-1}(\mathbf{m} - \boldsymbol{\mu}) \\
 &= [\mathbf{x}'\Sigma\mathbf{x}]^{-1} \mathbf{x}\Sigma \left\{ \mathbf{x}\hat{\beta}^{(j)} + \Sigma^{-1}(\mathbf{m} - \boldsymbol{\mu}) \right\} \\
 &= [\mathbf{x}'\Sigma\mathbf{x}]^{-1} \mathbf{x}\Sigma\mathbf{z}
 \end{aligned} \tag{3.4.8}$$

where

$$\mathbf{z} = \left\{ \mathbf{x}\hat{\beta}^{(j)} + \Sigma^{-1}(\mathbf{m} - \boldsymbol{\mu}) \right\}$$

the  $i^{th}$  component of  $\mathbf{z}$  is

$$z_i = x_i' \hat{\beta}^{(j)} + \frac{y_i - \mu_i}{\pi_i (1 - \pi_i)}$$

The update equation, equation 3.4.8, is written in the generalized least squares estimator form with

- $\Sigma$  as the diagonal matrix consisting of weights;
- $\mathbf{z}$  a target vector;
- $\mathbf{X}$  matrix data;
- $\hat{\beta}$  is the update parameter;
- $\mu_i = n_i \pi_i$  mean of the predictor;
- $\pi$  is a probability of success.

$\hat{\beta}, \mathbf{z}$  and  $\Sigma$  are updated at every step since they are dependent on  $\beta^{(j)}$ . Equation 3.4.8 depends on the assumption that  $\mathbf{x}'\Sigma\mathbf{x}$  is invertible. This is acceptable when  $n > p + 1$ . The IRLS algorithm converges in almost all pragmatic situations (Hastie *et al.*, 2009)

### 3.4.3 Using Logistic Regression as a Classifier

There are two different ways to assign observation  $\mathbf{x}$  to a class. Both methods depend on the maximum likelihood estimates. The maximum likelihood estimates the parameters  $\beta$  to give estimates of the logistic function as

$$\begin{aligned}
 \hat{g}(\mathbf{x}) &= \hat{\beta}'\mathbf{x} \\
 &= \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots + \hat{\beta}_p x_p
 \end{aligned} \tag{3.4.9}$$

If  $\hat{g}(\mathbf{x}) \geq 0$  assign  $x_i$  to class 1,  $l = 1$ , otherwise  $x_i$  belongs to class 0,  $l = 0$ . This is typically referred to as logistic analysis. An equivalent classification procedure is to use the likelihood estimated of  $\hat{g}(x)$  in the equation 3.4.9 to estimate the posterior probability  $P(Y = l \mid \mathbf{X} = \mathbf{x})$  as follows

$$\begin{aligned}\hat{p}(y = 1 \mid \mathbf{X} = \mathbf{x}) &= \frac{\exp(\hat{g}(\mathbf{x}))}{1 + \exp(\hat{g}(\mathbf{x}))} \\ &= \frac{1}{1 + \exp(-\hat{g}(\mathbf{x}))} \\ &= \frac{1}{1 + \exp(-\hat{\beta}'\mathbf{x})}\end{aligned}$$

$\mathbf{x}$  is assigned to class one,  $l = 1$  if the posterior probability  $\hat{P}(Y = 1 \mid \mathbf{X} = \mathbf{x}) \geq c$ . Where  $c$  is some cutoff value. alternatively  $\mathbf{x}$  is assigned to the class zero,  $l = 0$  (Izenman, 2009).



# Chapter 4

## Classification Trees

### 4.1 Introduction

This chapter gives an introduction to classification trees. Section 4.2 gives a basic theory of the binary classification tree. Section 4.3 discusses the impurity function. It also states an important theorem of node impurity and tree impurity. Section 4.4 discusses the splitting rule for decision trees using the Gini index and Entropy index. It also discusses how to prune the maximum tree using cost complexity pruning and cross validation methods.

### 4.2 Binary Classification Trees

A classification tree is also known as a decision tree (Ross. J, 1986). A classification tree is built through a process of either splitting or not splitting each parent node in the classification tree into two internal nodes. At each node the tree algorithm searches through all the variables one by one to obtain the best split. The starting point for splitting is denoted by a root node  $t$  at the top of the tree, see figure 4.2.1 below . The split is decided by a condition  $c$  for a single variable  $x$ ,  $x \leq c$  (Breiman *et al.*, 1984) as shown in table 4.1.

Split	Class of $Y$		Row Total
	0	1	
$x_j \geq c$	$n_{11}$	$n_{12}$	$n_{1+}$
$x_j \leq c$	$n_{21}$	$n_{22}$	$n_{2+}$
Column Total	$n_{+1}$	$n_{+2}$	$n_{++}$

Table 4.1: Classification table: The binary split of  $Y$  for variable  $x_j$ .

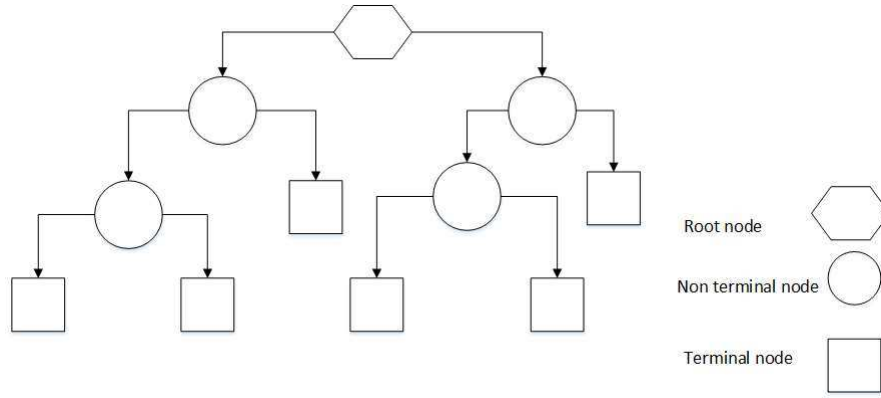


Figure 4.2.1: A figure depicting the root node, non-terminal and terminal nodes of a classification tree.

As shown in figure 4.2.1 nodes are either root, terminal or non-terminal nodes. A non-terminal node is also called a parent or internal node and these nodes have at least one child node. A binary split is decided by a condition value  $c$ , a single variable  $x$  as follows: if  $x \leq c$  then split to  $t_L$  assigned to the right son node, otherwise  $t_R$  assigned to the left son node. The process continues until all the variables in data set  $\mathbf{X}$  are split. A node that does not split, that is a node that has no child, is called terminal node or a leaf node or external node. All observations in a terminal node are assigned to one class and hence a terminal node is assigned a class labels. Each tree has more than one terminal node (Breiman *et al.*, 1984).

## 4.3 Impurity Functions

An impurity function is a function  $\Phi$  defined on the set of all  $L$ -tuples of numbers  $(p_1, \dots, p_L)$ , where  $p_l$  is a non-negative function satisfying  $p_l > 0$ , where  $\sum_{l=0}^L p_l = 1$  with the following properties (Breiman *et al.*, 1984, page 24):

1.  $\Phi$  is maximum when all the classes have equal probability,  $\Phi(\frac{1}{L}, \frac{1}{L}, \dots, \frac{1}{L})$ ;
2.  $\Phi$  is minimum when the probability of one class is one and the remaining classes are zero,  $\Phi(1, 0, 0, \dots, 0), \Phi(0, 1, 0, \dots, 0), \dots, \Phi(0, 0, 0, \dots, 1)$  the nodes contain only one class;
3.  $\Phi$  is symmetric function of  $p_0, p_1, \dots, p_l$  gives small tree and under fitted and loose gives over fit decision tree; if the permutation of  $\Phi$  remain the same then  $\Phi(p_0, p_1, p_2, \dots, p_l) = \Phi(\bar{p}_0, \bar{p}_1, \bar{p}_2, \dots, \bar{p}_l)$ .

### 4.3.1 Node Impurity

Given an impurity function  $\Phi$ , define the impurity of node  $t$  as

$$i(t) = \Phi(p(0|t), p(1|t), p(2|t), \dots, p(l|t))$$

The node impurity function is a maximum when the probability of split  $s$  is equal to  $p_l = \frac{1}{L}$ , a minimum when the probability of split  $s$  is equal to  $p_l = 0$  or 1.  $p_l = 0$  the node contains only one class [Raileanu and Stoffel, 2004; Breiman *et al.*, 1984, page 25]. The node impurity function is calculated at each step of splitting the observations in the node into two subset nodes  $L, R$  with associated probability

$$p_L = \frac{n_{1+}}{n_{++}} \quad (4.3.1)$$

and

$$p_R = \frac{n_{2+}}{n_{++}} \quad (4.3.2)$$

The goodness of split  $s$  at a node  $t$ , denoted by  $\Phi(s, t)$ , is defined as

$$\Delta i(s, t) = \Phi(s, t) = i(t) - p_L i(L) - p_R i(R)$$

The best split  $s$  of the node  $t$  is the split with largest  $\Delta i(s, t)$  or equivalently with the smallest value of  $p_L i(L) + p_R i(R)$  (Ripley, 1996; Breiman *et al.*, 1984, page 32)

$$\Delta i(s^*, t) = \max_{s \in Q_t} \Delta i(s, t)$$

The best split at each node maximizes the decrease in impurity between the nodes  $\Delta i(s^*, t)$ , which partitions the data into the left and right child or son nodes.

### 4.3.2 Tree Impurity

The impurity of the tree  $T$ , denoted by  $I(T)$ , is calculated across the terminal nodes in the tree. The tree impurity is calculated as

$$\begin{aligned} I(T) &= \sum_{t \in T'} I(t) \\ &= \sum_{t \in T'} p(t) i(t) \end{aligned}$$

where  $T'$  denotes the set of terminal nodes of tree  $T_0$ .

The tree impurity is a maximum only if the goodness of the split  $\Delta i(s, t)$  is maximum (Breiman *et al.*, 1984). At any node  $t$  of the sub-tree  $\tilde{T}$  using the split  $s$ , splits the node into  $t_R$  and  $t_L$ . The new sub-tree  $T'$  has impurity function

$$I(\tilde{T}) = \sum_{t \in \tilde{T}} I(t) + I(t_R) + I(t_L)$$

where  $\tilde{T}'$  denotes the set of terminal nodes of sub-tree  $\tilde{T}$ .

The decrease in tree impurity is calculated by subtracting the impurity of sub-tree  $\tilde{T}$  from the impurity of tree  $T$

$$I(T) - I(T') = I(t) + I(t_R) + I(t_L)$$

The goodness of split based on the tree impurity is  $\Delta I(s, t) = I(t) + I(t_R) + I(t_L)$  where

$$\begin{aligned} \Delta I(s, t) &= I(t) + I(t_R) + I(t_L) \\ &= i(t)p(t) - i(t_R)p(t) - i(t_L)p(t) \\ &= (i(t) - i(t_R) - i(t_L))p(t) \\ &= (i(t) - p_R i(t_R) - p_L i(t_L))p(t) \\ &= \Delta i(s, t)p(t) \end{aligned}$$

The goodness of split tree impurity  $\Delta I(s, t)$  is equal to the node impurity  $\Delta i(s, t)$  multiplied by the estimate of the probability of node  $t$ ,  $p(t)$ . The same maximum split point,  $\hat{s}^*$ , will be chosen in either impurity function (Breiman *et al.*, 1984, page 33). A tree is grown by contractually minimizing the split impurity, and hence the tree impurity, until the stopping criterion is satisfied.

## 4.4 Splitting Rules

There are many criteria which can be defined for selecting the best split at each node. In the context of classification the goal is to minimize the impurity. As result we select the split that will reduce the node impurity (Breiman *et al.*, 1984, 37). Two common node impurity functions are the Gini index and the Entropy index functions.

### 4.4.1 Gini Index

The Gini coefficient is a measure of statistical dispersion (Sheen and Anitha, 2012). The Gini index at node  $t$  is defined as

$$\begin{aligned} G(t) = i(t) &= \Phi(p(0|t), p(1|t), \dots, p(L|t)) \\ &= \sum_{l \neq j}^L p(l|t)p(j|t) \end{aligned}$$



Note that for binary splits where  $L = 0, 1$

$$\begin{aligned}
 G(t) &= \sum_{l \neq j} p(l | t) p(j | t) \\
 &= \sum_{l=0}^1 p(l | t) (1 - p(l | t)) \\
 &= \sum_{l=0}^1 p(l | t) - \sum_l (p(l | t))^2 \\
 &= 1 - \sum_{l=0}^1 (p(l | t))^2
 \end{aligned}$$

According to [Breiman *et al.*, 1984; Therneau *et al.*, 1997, page 103] there are two interpretations of the Gini index. In binary classification there are two classes:  $L = 0, 1$ . The Gini index is measure or sum of the variance across the classes. The Gini index uses the probability  $0 \leq p_l \leq 1$ . If the probability is close to zero or one the Gini index gives smaller value. Small values mean that the observations in the node are from one class. The Gini index for binary variables is

$$\begin{aligned}
 G(t) = i(t) &= 1 - p(0 | t)^2 - p(1 | t)^2 \\
 &= 1 - [(1 - p(0 | t))^2 + p(0 | t)^2] \\
 &= 1 - [1 - 2p(0 | t) + p(0 | t)^2 + p(0 | t)^2] \\
 &= 1 - [1 - 2p(0 | t) + 2p(0 | t)^2] \\
 &= 1 - 1 + 2p(0 | t) - 2p(0 | t)^2 \\
 &= 2p(0 | t) - 2p(0 | t)^2 \\
 &= 2p(0 | t) [1 - p(0 | t)] \\
 &= 2p(0 | t) p(1 | t)
 \end{aligned}$$

Using the values in table 4.1, the Gini index of node  $t$  is calculated as

$$G(t) = 2 \left( \frac{n_{+1}}{n_{++}} \right) \left( 1 - \frac{n_{+1}}{n_{++}} \right)$$

The Gini index for splitting the node  $t$  into two child nodes is given by

$$G_{\text{Split}}(t) = p_L G(t_L) + p_R G(t_R)$$

where  $p_L$  is the proportion of cases in the node  $t$  sent to the left child node and  $p_R$  is the proportion sent to the right child node. The best split is the split that has the smallest value of  $G_{\text{Split}}(t)$ . The variable that gives the best split is applied (Muchai and Odongo, 2014). The

goodness of split for node  $t$  and split  $s$  is defined as

$$\begin{aligned}\Delta i(s, t) = \Phi(s, t) &= G(t) - p_L G(t_L) - p_R G(t_R) \\ &= G(t) - G_{\text{Split}}(t)\end{aligned}$$

The split with maximum  $\Delta i(s, t)$  is equivalent to the split with minimum  $G_{\text{Split}}(t)$  (Breiman *et al.*, 1984, page 103). The idea of the CART method (Breiman *et al.*, 1984) is to determine the goodness of split  $\Delta i(s, t)$  at each split  $s, s \in Q_t$ , where  $Q_t$  is the set of all possible splits at node  $t$  and then to select the best split  $s^*$  given as

$$\begin{aligned}\Delta i(s^*, t) &= \max_{s \in Q_t} (G(t) - p_L G(t_L) - p_R G(t_R)) \\ &= \max_{s \in Q_t} \Delta i(s, t)\end{aligned}$$

Example 3.1: Consider the binary split of the node  $t_1$  as shown in figure 4.4.1.

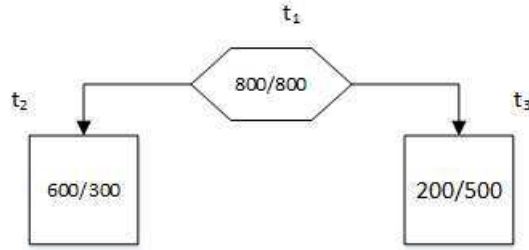


Figure 4.4.1: Binary split the node  $t_1$ .

The Gini index  $G(t_1)$  used for splitting the root node  $t_1$  and the goodness of split  $\Delta i(s, t)$  are calculated below.

The probability of  $p(0 | t_1) = \frac{800}{1600}$  and the probability of  $p(1 | t_1) = \frac{800}{1600}$  and hence

$$G(t_1) = 2 \left( \frac{800}{1600} \right) \left( \frac{800}{1600} \right) = \frac{1}{2}$$

the Gini index for the nodes  $t_2$  and  $t_3$  are calculated as

$$\begin{aligned}G(t_2) &= 2 \left( \frac{600}{900} \right) \left( \frac{300}{900} \right) = 0.44444 \\ G(t_3) &= 2 \left( \frac{200}{700} \right) \left( \frac{500}{700} \right) = 0.40816\end{aligned}$$

The probability of splitting node  $t_1$  into  $t_2$  and  $t_3$  is thus

$$\begin{aligned} p_{t_2} &= \frac{900}{1\ 600} = 0.5625 \\ p_{t_3} &= \frac{700}{1\ 600} = 0.4375 \end{aligned}$$

The Gini split of node  $t_1$  is

$$\begin{aligned} G_{\text{Split}}(t_1) &= p_{t_2}G(t_2) + p_{t_3}G(t_3) \\ &= 0.5625 \times 0.44444 + 0.4375 \times 0.40816 \\ &= 0.42857 \end{aligned}$$

The goodness of split for Gini index at the node  $t_1$  is this

$$\begin{aligned} \Delta_{Gini}i(s, t) &= G(t_1) - G_{\text{Split}}(t_1) \\ &= 0.5 - 0.42857 \\ &= 0.07143 \end{aligned}$$

Example 3.2: Consider the example of binary tree shown in figure 4.4.2.

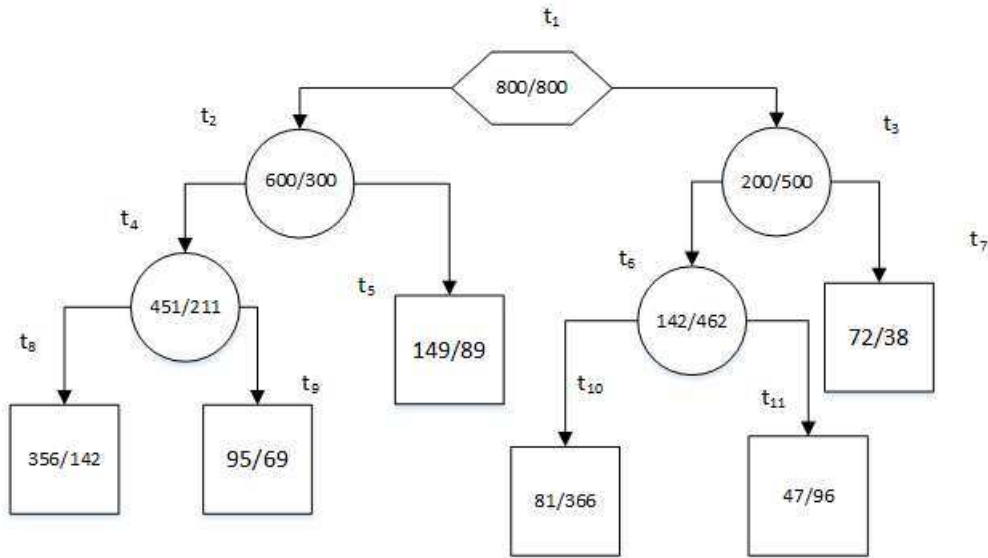


Figure 4.4.2: An example of a binary tree.

For each non-terminal node in the tree the Gini index and split  $\Delta i(s, t)$  calculates are obtained shown in table 4.2

Table 4.2: The Gini split and goodness of the split in the non-terminal nodes in the tree depicted in figure 4.4.2.

Node $t_i$	Class 0	Class 1	Gini index	Gini split $G_{\text{Split}}(t_i)$	Goodness of split $\Delta i(s, t_i)$
1	800	800	0.5	0.42857	0.07143
2	600	300	0.44444	0.44326	0.00119
3	200	500	0.40816	0.35744	0.05073
4	451	211	0.43428	0.42743	0.00685
6	128	462	0.33976	0.33178	0.00799

### 4.4.2 Entropy Index

The Entropy function is defined as (Breiman *et al.*, 1984)

$$\begin{aligned}
 i(t) &= \Phi(p(0 | t), p(1 | t), \dots, p(L | t)) \\
 &= - \sum_{l=0}^L p(l | \tau) \log p(l | \tau)
 \end{aligned}$$

Note that for binary splits where  $L = 0, 1$

$$i(t) = -p(0 | t) \log p(0 | t) - p(1 | t) \log p(1 | t)$$

Using the values in table 4.1. The entropy index of a node is calculated as

$$Entropy(t) = i(t) = -\left(\frac{n_{1+}}{n_{++}}\right) \log\left(\frac{n_{1+}}{n_{++}}\right) - \left(\frac{n_{2+}}{n_{++}}\right) \log\left(\frac{n_{2+}}{n_{++}}\right)$$

The goodness of a split based on the entropy measure is given by

$$\Delta i(s, t) = \Phi(s, t) = Entropy(t) - p_L Entropy(t_L) - p_R Entropy(t_R)$$

Both the entropy and Gini indices have been used widely in the CART methodology, see for example Breiman *et al.* (1984); Strobl *et al.* (2007).

Example 3.3: Refer to figure 4.4.1. The Entropy index,  $Entropy(t)$  used for splitting the root node  $t_1$  is calculated with the goodness of split  $\Delta_{Entropy} i(s, t)$ . The probability of  $p(0 | t_1) = \frac{800}{1600}$  and the probability of  $p(1 | t_1) = \frac{800}{1600}$

$$\begin{aligned}
 Entropy(t_1) &= -\left(\frac{800}{1600}\right) \log\left(\frac{800}{1600}\right) - \left(\frac{800}{1600}\right) \log\left(\frac{800}{1600}\right) \\
 &= 0.30103
 \end{aligned}$$

The Entropy index for the nodes  $t_2$  and  $t_3$  are thus

$$\begin{aligned}
 Entropy(t_2) &= -\left(\frac{600}{900}\right) \log\left(\frac{600}{900}\right) - \left(\frac{300}{900}\right) \log\left(\frac{300}{900}\right) \\
 &= 0.27643 \\
 Entropy(t_3) &= -\left(\frac{500}{700}\right) \log\left(\frac{500}{700}\right) - \left(\frac{200}{700}\right) \log\left(\frac{200}{700}\right) \\
 &= 0.25983
 \end{aligned}$$

The probability of splitting a node  $t_1$  into  $t_2$  and  $t_3$  is thus

$$\begin{aligned}
 p_{t_2} &= \frac{900}{1\ 600} = 0.5625 \\
 p_{t_3} &= \frac{700}{1\ 600} = 0.4375
 \end{aligned}$$

The Entropy split of the node  $t_1$  is thus

$$\begin{aligned}
 Entropy_{split}(t_1) &= 0.5625 \times 0.27643 + 0.4375 \times 0.25983 \\
 &= 0.26917
 \end{aligned}$$

The goodness of this split, based on the entropy measure, at node  $t$ , is

$$\begin{aligned}
 \Delta_{Entropy}i(s, t) &= 0.301029996 - 0.269167974 \\
 &= 0.03186
 \end{aligned}$$

Table 4.3: The Entropy split and goodness of the split in the non-terminal nodes in the tree depicted in figure 4.4.2.

Node $t_i$	Class 0	Class 1	Entropy index	Entropy split $G_{Split}(t_i)$	Goodness of split $\Delta i(s, t_i)$
1	800	800	0.5	0.26917	0.03186
2	600	300	0.44444	0.27286	0.00057
3	200	500	0.40816	0.23544	0.02438
4	451	211	0.43428	0.26850	0.00333
6	128	462	0.33976	0.22236	0.00478

## 4.5 Splitting Procedure

The following procedure is used to select the splitting variable and the splitting value at the node  $t$  (Breiman *et al.*, 1984):

1. Determine the Gini split at node  $t$  among the child branches over all possible decision points for each variable  $X_j$  at each node;

2. Select the variable and the critical value,  $c$ , of that variable with the smallest Gini split at node  $t$ , denoted by  $x_{jc}$  and use it for splitting;
3. Repeat this process at each node until each node has at least one observation or meets the minimum requirement of the observations at each node.

The procedure is the same for other choices of impurity function, for example the entropy impurity function.

### 4.5.1 Maximum Tree

The maximum tree is denoted by  $T_{max} = T_0$ . This tree is the result of splitting the root node into several sub-nodes depending on the data set  $D$  and the impurity function. Splitting of nodes (internal nodes) carries on until all terminal nodes that are generated have at least one observation or all the observation belong to the same class or we declare the node to be a terminal node if satisfies the following condition

$$\max \Delta R(s, t) \leq cR(t)$$

where  $c$  is a constant between  $0 \leq c \leq 1$  and  $R(t)$  is the misclassification rate of node  $t$ . The result of  $T_{max}$  is decreasing sequence of sub-trees where  $T_0$  is the full tree and  $T_\infty$  is the tree without any splitting or terminal nodes:

$$T_0 \geq T_1 \geq T_2 \geq \dots, \geq T_\infty$$

To select the right size of tree from the sequence  $T_0 \geq T_1 \geq T_2 \geq \dots, \geq T_\infty$ , we need to estimate the misclassification error rate  $R(T)$  (Breiman *et al.*, 1984, page 233). The maximum tree can have two problems (Breiman *et al.*, 1984, page 61)

1. High accuracy with low misclassification error rate. This tree provides poor outcomes when applied to new sample data.
2. Understanding and interpreting the maximum tree with large numbers of terminal nodes is complicated.

The maximum tree is a complex tree. The complexity of a trees is determined by the number of terminal nodes (Yohannes and Webb, 1999). Pruning the maximum tree yield the right size tree with the best accuracy and misclassification rate.

## 4.6 Pruning a Tree

The goal of pruning is to remove parts of a classification model that describe random variation in the data set rather than true structure features of the underlying domain Mahmood *et al.* (2010). There are two types of pruning namely, pre and post pruning.

- For the pre-pruning method we stop splitting the tree according to some stopping criteria. Tightly criteria generate small tree are typically underfit (Mahmood *et al.*, 2010);
- The standard algorithm for post-pruning decision trees does not take statistical significance into account. The method is known to be one of the fastest pruning algorithms that produces trees that are both accurate and small (Esposito *et al.*, 1997). Post pruning a tree has two main steps: fitting and simplification. Fitting or growing the maximum sized tree may result in the possibility of over fitting after a certain point. Reducing the size of the tree is accomplished by pruning the tree back, starting from the terminal nodes utilizing a score function (Mahmood *et al.*, 2010).

### 4.6.1 Cost Complexity Pruning

The minimal cost complexity pruning algorithm is defined by (Breiman *et al.*, 1984). The CART technique growing and pruning the tree is based on balancing or optimizing the complexity and misclassification error rates of the tree. Continuous splitting increases the size of the tree the node misclassification error rates decreases as the tree size increases until the misclassification error becomes zero. The main objective is to find the best proportion between the tree complexity and misclassification error  $R(T)$  [Timofeev, 2004; Breiman *et al.* 1984, page 66]. The best proportion is found through use of the cost-complexity function, denoted as

$$R_\alpha(T) = R(T) + \alpha |\tilde{T}| \quad (4.6.1)$$

where  $|\tilde{T}|$  is the number of terminal nodes in the tree  $T$ .  $\alpha |\tilde{T}|$  is a measure of the complexity of the tree.  $\alpha$  is a parameter found through the sequence of training data used to build the tree. The other part of the data is taken as testing data used to evaluate the tree.

$$\alpha = \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1} \quad (4.6.2)$$

We denote  $R(t)$  as the error associated with the node  $t$ ,  $R(T_t)$  is equal to the sum of errors associated with all terminal nodes that are off spring of  $t$ , and  $|\tilde{T}_t|$  is number of the terminal nodes associated with node  $t$ . The sequence of  $\alpha$  can be determined from equation 4.6.2. The minimal error rate cost complexity pruning is the same as the minimal cost complexity pruning in classification is defined as

$$R_{\alpha}(T(\alpha)) = \min_{T \leq T_{max}} R_{\alpha}(T)$$

Example 3.4: The maximum tree was fitted using simulated data that contains 1 600 observations where 800 observations belong to class one and the remaining observations to class zero. the maximum tree has four non-terminal nodes and six terminal nodes as shown in figure 4.6.1.

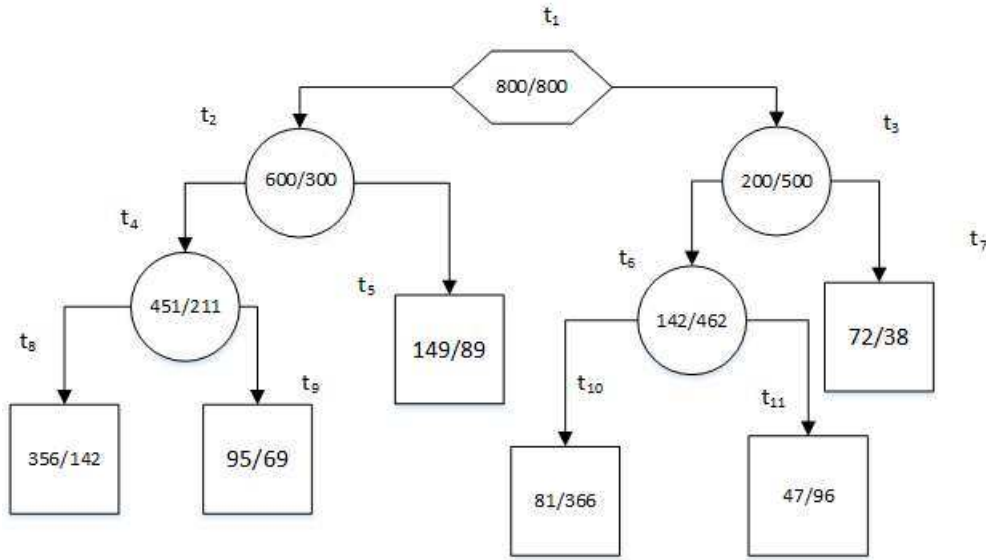


Figure 4.6.1: The maximum tree,  $T_0$ .

Pruning the maximum tree using the alpha method

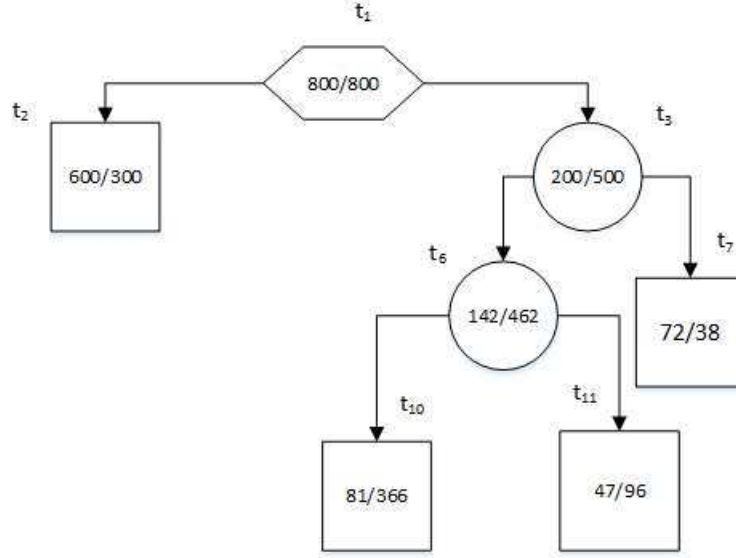
$$\alpha = \frac{R(t_t) - R(T_t)}{(|\tilde{T}_t| - 1)}$$

yields

$$\begin{aligned} \alpha(T_0(t_1)) &= \frac{0.5 - 0.9541}{6 - 1} = -0.0908 \\ \alpha(T_0(t_2)) &= \frac{0.25 - 0.5512}{3 - 1} = -0.1506 \\ \alpha(T_0(t_3)) &= \frac{0.1786 - 0.4028}{3 - 1} = -0.1121 \\ \alpha(T_0(t_4)) &= \frac{0.3194 - 0.4274}{2 - 1} = -0.1080 \\ \alpha(T_0(t_6)) &= \frac{0.2864 - 0.3318}{2 - 1} = -0.0454 \end{aligned}$$

Node  $t_2$  has the lowest alpha of  $-0.1506$ . As a result we prune the tree under node  $t_2$ , that is change the non terminal node  $t_2$  to a terminal node. The first pruning results in sub-tree  $T_1$  which is smaller than  $T_0$  the maximum tree as shown in figure 4.6.2.

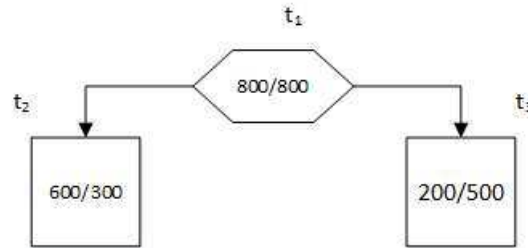


Figure 4.6.2: Sub-tree  $T_1$ .

Sub-tree  $T_1$  contains two non-terminal node and four terminal nodes pruning this tree yields

$$\begin{aligned}\alpha(T_1(t_1)) &= \frac{0.5 - 0.6528}{4 - 1} = -0.0509 \\ \alpha(T_1(t_3)) &= \frac{0.1786 - 0.4028}{3 - 1} = -0.1121 \\ \alpha(T_1(t_6)) &= \frac{0.2864 - 0.3318}{2 - 1} = -0.0454\end{aligned}$$

Node  $t_3$  has the lowest alpha of  $-0.1121$ . Prune the tree under node  $t_3$  change the non-terminal node  $t_3$  to a terminal node. The result is sub tree  $T_3$  less than  $T_2$  and  $T_1$  as shown in figure 4.6.3.

Figure 4.6.3: Sub-tree  $T_3$ .

Sub tree  $T_3$  contains only two terminal nodes and the root node. It is the best sub tree and hence we can stop pruning.

Example 3.5: Consider the pruning tree example 3.1 above . Calculating the minimum cost complexity

$$R_\alpha(T_{t_1}) = R(T_t) + \alpha |T_t|$$

yields

$$R_{\alpha}(T_0(t_1)) = 1.5343 - 6 \times 0.9541 = 0.9895$$

$$R_{\alpha}(T_0(t_2)) = 0.5694 - 3 \times 0.1506 = 0.1176$$

$$R_{\alpha}(T_0(t_3)) = 0.4649 - 3 \times 0.1121 = 0.1331$$

$$R_{\alpha}(T_0(t_4)) = 0.3194 - 2 \times 0.1080 = 0.1034$$

$$R_{\alpha}(T_0(t_6)) = 0.2864 - 2 \times 0.0454 = 0.1956$$

Node  $t_4$  has the lowest cost complexity of 0.1035. Pruning the tree under node  $t_4$ : Change the non-terminal node  $t_4$  to a terminal node. The first pruning result's in sub-tree  $T_1$  which is smaller than  $T_0$ , the maximum tree as shown in figure 4.6.4.

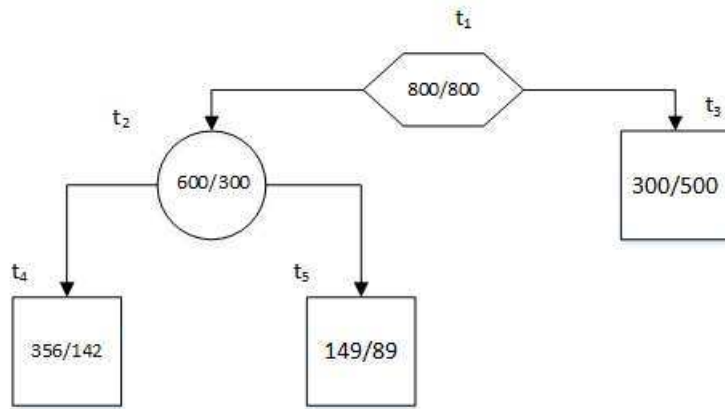


Figure 4.6.4: Sub-tree  $T_1$ .

The cost complexity of sub-tree  $T_1$  is

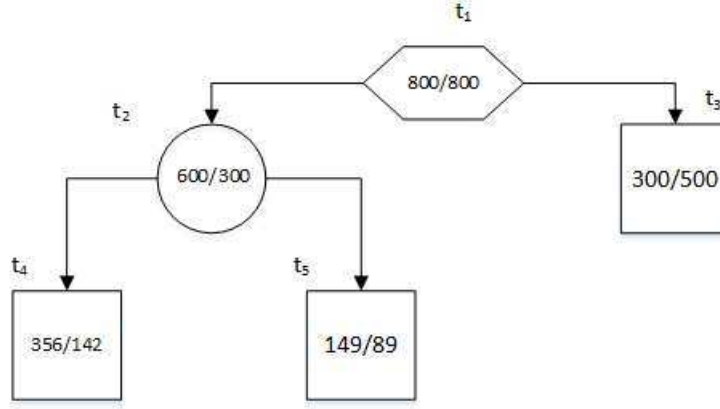
$$R_{\alpha}(T_1(t_1)) = 1.5344 - 5 \times 0.0067 = 1.5009$$

$$R_{\alpha}(T_1(t_2)) = 0.5694 - 2 \times 0.1933 = 0.1828$$

$$R_{\alpha}(T_1(t_3)) = 0.4649 - 3 \times 0.1121 = 0.1286$$

$$R_{\alpha}(T_1(t_6)) = 0.2864 - 2 \times 0.0454 = 0.1956$$

Sub tree  $T_2$  has three non-terminal nodes and five terminal nodes. The minimum cost complexity is found at node  $t_3$ , 0.1285. Prune the tree under node  $t_3$  results in sub tree  $T_2$  as shown in figure 4.6.5.

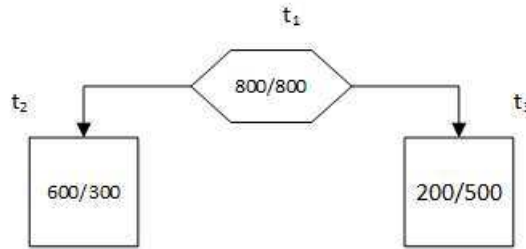
Figure 4.6.5: Sub-tree  $T_2$ .

Recalculating the cost complexity

$$R_\alpha(T_2(t_1)) = 0.75 - 3 \times 0.0609 = 0.5673$$

$$R_\alpha(T_2(t_2)) = 0.25 - 2 \times 0.1933 = -0.1366$$

Sub tree  $T_2$  has one non-terminal node and three terminal nodes. The minimum cost complexity is found at node  $t_2$ ,  $-0.1365$ . Prune the node under  $t_2$  results in sub-tree  $T_3$  as shown in figure 4.6.6.

Figure 4.6.6: Sub-tree  $T_4$ .

The minimum cost complexity method and Alpha method yield the same result where the best sub tree has only the root node and two terminal nodes.

## 4.7 Cross Validation

Cross validation can be used to choose the optimal complexity  $\alpha$  (Therneau *et al.*, 2014):

First fit the maximum tree,  $T_0$ , using the entire data set  $D$ . Prune  $T_0$  and estimate the misclassification error rate of the sub-tree and the tree, complexity parameter  $\alpha$  and cost complexity .

Second divide the data set  $D$  into  $k$  approximately equal sets. Use the  $k - 1$  sets as the training data set  $D_t$ . The remaining set serves as a testing data set. The  $k - 1$  subsets serving as a training data set  $D_t$  are used to build a larger tree  $T_{0_k}$ . Pruning the larger tree creates a sequence of sub-trees  $T_k$ . Use the remaining set, namely the test data set, to estimate the misclassification error as

$$R_{cv}(T_k) = \frac{1}{n_k} \sum_{D_k \in D} R(T_k).$$

Using the misclassification  $R_{cv}(T_k)$  and complexity parameter  $\alpha_k$  we estimate the cost complexity  $R_{\alpha_k}(T_k)$ . Repeating the procedure  $k$  times until each sets has a chance to be selected as testing data set. Similar sub trees are then created and the corresponding values of  $\alpha_k$  for each sub-tree are calculated. For a given value of  $\alpha_k$ , the misclassification error rate are determined for each sub tree of the maximum tree, these misclassification rates are averaged to get a single misclassification score as follows

$$R_{cv}(T) = \frac{1}{K} R_{cv}(T_k)$$

for a single value of  $\alpha$  this procedure is similar to estimating the misclassification for the tree that is created using all the data set, and pruned using the value of  $\alpha$ . during cross validation we determine the misclassification error rate at each node of the maximum tree. Since the maximum tree has different sub-trees, the sub-tree that has smaller misclassification error is selected to be in the final classification tree model.

# Chapter 5

## Prediction of the Protein Secondary Structure

### 5.1 Introduction

This chapter provides an introduction to proteins and amino acids (section 5.2), their structure and function (section 5.3), an introduction to the prediction of protein secondary structure (section 5.4) and a description of the data set used in this study and how this data set is transformed to an appropriate data structure (section 5.5). The accuracy of the classification tree (section 5.6), Naive Bayes (section 5.7) and logistic regression (section 5.8) classifiers, using hold-out and 5-fold cross validation, for the prediction of protein secondary structure for the three classes, namely helix, sheet and coil, are discussed. Section 5.9 compares the predictive accuracy of the logistic regression, Naive Bayes and logistic regression classifiers.

### 5.2 Proteins and Amino Acids

Proteins are made of simple building blocks called amino acids (Pratt *et al.*, 2005; Tsilo, 2009). Proteins are polymer chains of repeating polypeptide units with side chains attached to each polypeptide unit. The side chains, also known as residues, are amino acids with different characteristics (Yüksektepe *et al.*, 2008). Amino acids are built through the connections of central carbon atoms,  $C$ , Hydrogen atoms,  $H$ , amino groups ( $NH_2$ ), carboxyl groups ( $COOH$ ) and other element found in the side chains of certain amino acids in a particular order. The sequence of amino acids in a protein chain is given by the primary structure (Yüksektepe *et al.*, 2008). An example of a amino acid protein structure is shown in figure 5.2.1 below.

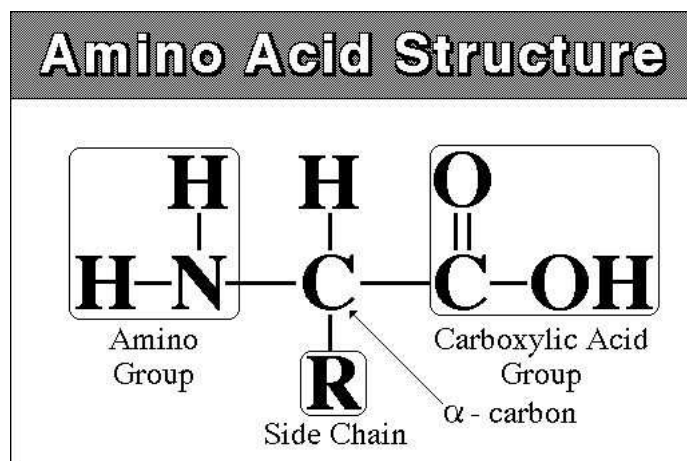


Figure 5.2.1: An example of a protein amino acid structure.

(from [http://www.hcc.mnscu.edu/chem/V.27/page\\_id\\_17100.html](http://www.hcc.mnscu.edu/chem/V.27/page_id_17100.html), accessed 01/12/2014.)

Scientists have discovered more than 300 amino acids, but only 20 amino acids are coded by DNA. Each amino acid has specific characteristics defined by the side chains. The side chains play a unique role in protein structure (Cravedi, 2010). Two methods of abbreviating or coding the primary amino acid structure are shown in table 5.1 below.

Table 5.1: Amino acid abbreviations.

Amino Acid	Three Letter Abbreviation	One Letter Abbreviation
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid ( <i>Aspartate</i> )	Asp	D
Cysteine	Cys	C
Glutamine	Gln	Q
Glutamic Acid (Glutamate)	Glu	E
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrodine	Tyr	Y
Valine	Val	V

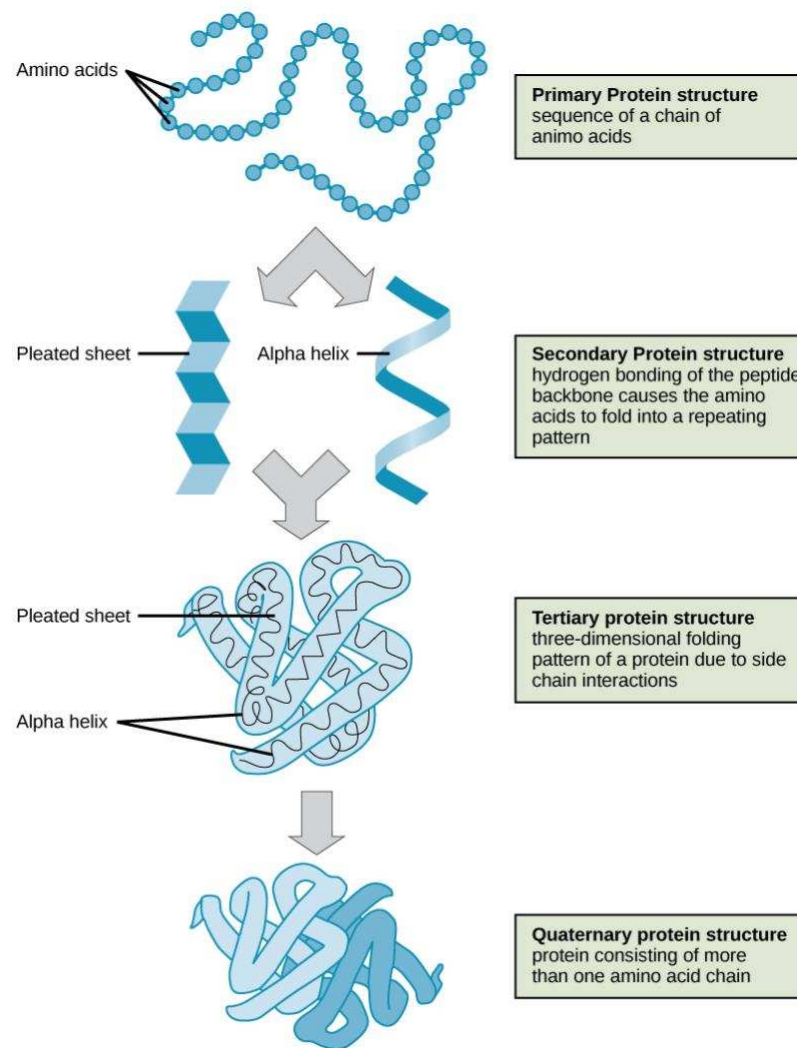


Figure 5.2.2: The four levels of protein structure (From Boundless, 2014).

## 5.3 Protein Structure and Function

Proteins have four different levels of structure: Primary, Secondary, Tertiary and Quaternary structure (Whitford, 2005). These levels are characterized from each other by the degree of complexity in the polypeptide chains. A single protein molecule may hold one or more of these protein structure levels. Primary structure refers to the linear sequence of amino acids. Protein primary structure is the foundation of all the other levels of structure (Tsilo, 2009). Secondary structure is the general three-dimensional form of amino acids. The two most important secondary structures of proteins, the alpha helix and the beta sheet, were predicted in 1951 by Linus Pauling and Robert Corey (Kabsch and Sander, 1983). Tertiary structure is the three dimensional folding of the secondary structures. The quaternary structure is formed from interactions between two or more proteins in their native secondary or tertiary structures (Whitford, 2005). Functionality of most proteins is achieved mainly at the tertiary or quaternary structure

level, though there are examples of functional secondary structure proteins (Whitford, 2005). The four structures of proteins are shown in figure 5.2.2 .

## 5.4 Secondary Structure Prediction

Protein secondary structure prediction is the prediction of the secondary structure of a protein based on the primary structure that is from the linear sequence of amino acid (Tsilo, 2009, page 12). The prediction of the secondary structure depends on the amino acid sequences (Zhang and Rajapakse, 2009). Secondary structure has two properties, hydrogen bond patterns and backbone geometry. Hydrogen bonded features include turns, bridges,  $\alpha$ -helices, ladders and  $\beta$ -sheets while bends, chirality, SS bonds and solvent exposure are features which are determined geometrically (Kabsch and Sander, 1983; Tsilo, 2009). An aim of theoretical chemistry and bioinformatics is to predict the sequence of the protein structure from the primary structure (Zhang and Rajapakse, 2009).

Some of the computationally based methods that can be used to predict the secondary predictions include Naive Bayes, logistic regression classifier, classification trees, neural networks, support vector machines and nearest neighbor methods (Singh *et al.*, 2008).

## 5.5 The RS126 Data Set

The RS126 data set is used in this study. This data set was designed for the prediction of protein secondary structure. The data set consists of 126 proteins which do not share sequence identity with more than 25% over a length of at least 80 residues (Rost and Sander, 1993). The data set consists of the 126 protein primary and secondary sequences, encoded using alphabetic letters. As an example consider the actinoxanthin protein (seq\_1acx) and secondary structure sequences shown in table 5.2 below. The secondary structure sequences are automatically determined or assigned from the experimentally determined tertiary structure by automated software, namely DSSP (Kabsch and Sander, 1983), STRIDE (Frishman and Argos, 1995) or DEFINE (Richards and Kundrot, 1988). DSSP, STRIDE and DEFINE utilize eight secondary structure classes, namely  $\alpha$ -helix (H),  $3_{10}$ -helix (G),  $\pi$ -helix (I),  $\beta$ -strand (E), isolated  $\beta$ -bridge (B), turn (T), bend (S) and rest (-). There are many published 8-to-3 states reduction methods which are known to alter the prediction accuracy of the classifier (Cuff and Barton, 1999). The secondary structure classes were reduced from 8 classes to 3 as follows in this study:

- $\alpha$ -helix:  $\alpha$ -helix (H),  $3_{10}$ -helix (G),  $\pi$ -helix (I)
- $\beta$ -sheet :  $\beta$ -strand (E), isolated  $\beta$ -bridge (B),



- $c$ -coils: turn (T), bend (S), rest (-)

The DSSP sequences were used as the secondary structure sequences in this study and the following three binary classifiers are considered:

- Helix against not helix, denoted as H/ $\sim$ H;
- Sheet against not sheet, denoted as S/ $\sim$ S;
- Coil against not coil, denoted as C/ $\sim$ C.

Table 5.2: The actinoxanthin (1acx.concise) entry in the RS126 data set.

seq_1acx:	APAFSVSPASGASDGQSVSVSVAAAGETYYIAQCAPVGGQD. . .
seq_MACM_STRMA:	APGVTVPATGLSNGQTVTVSATTPGTVYHVGQCAVVEGVI. . .
seq_KEDA_ACTSL:	SAAVSVSPATGLADGATVTVSASATSTSATALQCAILAGRG. . .
seq_NCZS_STRCZ:	APTATVTPSSGLSDGTVVKVAGAQAQGTAYDVGGQCAWVDGVL. . .
OrigSeq:	APAFSVSPASGASDGQSVSVSVAAAGETYYIAQCAPVGGQD. . .
cons:	---EEEE-----EEEEEEE---EEEEEEEEEEEE--E. . .
dssp:	--EEEE-----EEEEEEE---EEEEEEE--EE--EE. . .
define:	EEEE-----EEEEEEE-----EEEE--EE. . .
stride:	--EEEE-----EEEEEEE---EEEEEEEEEEEE--EE. . .

A binary encoding scheme was used to assign numerical values to the primary and secondary structure, represented by letters in the input RS126 data files. For each amino acid in the primary structure sequence there are 21 positions: 20 positions for the letters of amino acids and 21st for a null input denoted by \*. The null is required to extend the start and end of sequences when constructing windows as described below. Each amino acid in the primary structure sequence is encoded as a 20 + 1 dimensional vector as per the mapping shown in table 5.3 below.

Once each input amino acid is encoded as a 21 dimensional vector the encoded vectors are collected in, or concatenated into, a moving window, of a specified size, of encoded amino acid sequences with the aim of predicting the central secondary structure residue, see figure 5.5.1. Thus the secondary structure of the  $j^{th}$  amino acid,  $R_j$ , is predicted from a window of amino acids,  $R_{j-n}, R_{j-n+1}, \dots, R_j, \dots, R_{j+n-1}, R_{j+n}$  where  $w = 2n + 1$  is the window (Holley and Karplus, 1989). These vectors are then appropriately stacked to form the encoded data set for further analysis. This encoding is known as the orthogonal encoding and has the advantage of not introducing any artificial correlations between the amino acids. However this encoding is highly redundant and typically results in classifiers with numerous parameters and hence may take considerable time to estimate/fit (Riis and Krogh, 1996).

Table 5.3: The amino acid encoding matrix.

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
0	1	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	1	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	1	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	1	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	1	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	1	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	1	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	1	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	1	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	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

The secondary structure residues are reduced from 8 to 3 states and encoded as 3 dimensional vectors as shown in table 5.4.

Table 5.4: The secondary structure encoding.

Helix: H	Coil: C	Sheet: S
$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$

Consider as an example encoding the first four observations of the actinoxanthin sequence, table 5.2, as represented in table 5.5 for a window of size 3. The primary sequence of amino acids, used as the input for a classifier, is  $A, P, A, F$ . The associated secondary structure, to be used as the target class for a classifier, is  $-, -, E, E$ .

Step one involves adding null indicators to the start and end of the sequence. Since the window is of size three we need only one null indicator at the start and end of the primary structure/amino acid sequence. In step 2 the amino acid sequence is encoded (figure 5.5.1) using the orthogonal encoding scheme as shown in table 5.6.

Table 5.5: The first four observations of the actinoxanthin (1acx.concise) sequence as considered in this study.

seq_1acx	Primary or amino acid sequence: Input	A	P	A	F	...
DSSP: 8 states	Secondary structure sequence	-	-	E	E	...
DSSP: 3 states	Target class:	C	C	S	S	...

Step three requires that the associated secondary structure is reduced from 8, -, -,  $E$ ,  $E$ , to 3 states  $C$ ,  $C$ ,  $E$ ,  $E$  where  $C$  indicates coil,  $E$  sheet and  $H$  helix, as shown on the bottom row of table 5.5. The secondary structure is then encoded as a 3 dimensional vector, as per table 5.4.

Table 5.6: Encoding of the actinoxanthin (1acx.concise) entry for a window of size 3.

Observation	1	2	3	4	5	6	7	...
Amino Acid	NA	A	P	A	F	S	V	...
A	0	1	0	1	0	0	0	...
R	0	0	0	0	0	0	0	...
N	0	0	0	0	0	0	0	...
D	0	0	0	0	0	0	0	...
C	0	0	0	0	0	0	0	...
Q	0	0	0	0	0	0	0	...
E	0	0	0	0	0	0	0	...
G	0	0	0	0	0	0	0	...
H	0	0	0	0	0	0	0	...
I	0	0	0	0	0	0	0	...
L	0	0	0	0	0	0	0	...
K	0	0	0	0	0	0	0	...
M	0	0	0	0	0	0	0	...
F	0	0	0	0	1	0	0	...
P	0	0	1	0	0	0	0	...
S	0	0	0	0	0	1	0	...
T	0	0	0	0	0	0	0	...
W	0	0	0	0	0	0	0	...
Y	0	0	0	0	0	0	0	...
V	0	0	0	0	0	0	1	...
*	1	0	0	0	0	0	0	...
Sec Structure	NA	-	-	E	E	E	E	...
Target	NA	C	C	S	S	S	S	...
Encoded Target		$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$	...

In step four the input windows are constructed by concatenating the relevant transposed encoded amino acids. For a window of size 3 the input pattern contains 3 transposed, encoded input vectors. In this way each observation, or window, consists of  $21 \times 3$  values, of which 3 are 1's

and the rest are 0's. Prediction is made for the central amino acid residue. The transposed encoded target vector can be concatenated to the end of the input window to create a convenient data structure in step 5. Thus a window and target is constructed for each secondary structure target element of the observed sequence, for all protein sequences contained in the data set. The final step is to stack each window and associated target vector in an appropriate data structure, for example matrix or a data frame in R, suitable for analysis using a classifier.

For a window size of size 13 the input consists of 13 input amino acids that are each encoded as 21 dimensional vectors, that is each amino acid is assigned a one (1) in the appropriate position in the encoding vector while the other positions are filled with zero's (assigned 0's). In this way the input vectors will be of dimension  $(21 \times 13) \times 1$  and consist of 13 1's and 260 0's. The secondary structure of the central amino acid in the window is encoded, and subsequently transposed, to be used as the target vector for the window. These  $276 \times 1$  vectors are stacked to form a matrix for further analysis.

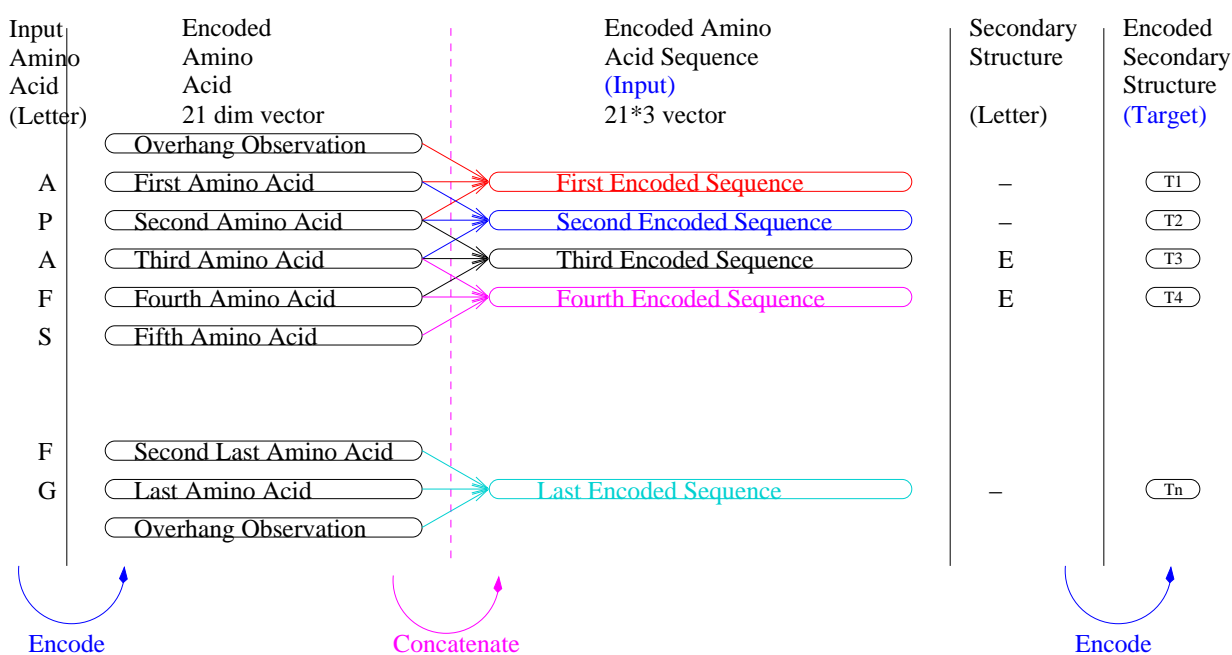


Figure 5.5.1: Creating the data set for a window of size 3 (From Baxter and Jäger, 2011).

## 5.6 Prediction of Protein Secondary Structure using Classification Trees

### 5.6.1 Hold-Out Classification

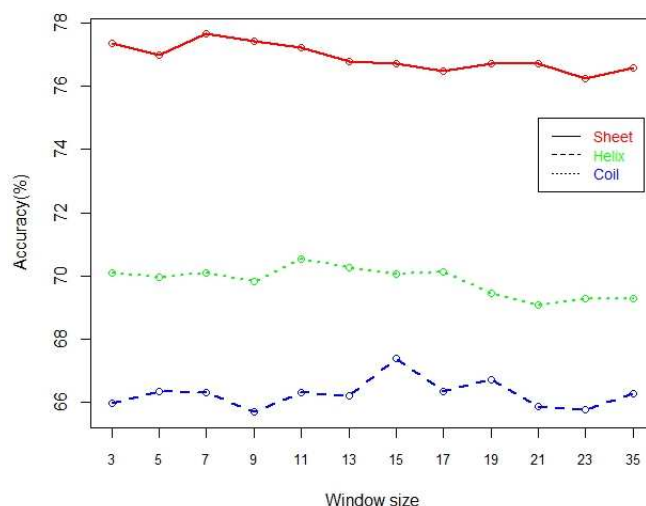
#### 5.6.1.1 Data Set Partition and Application

For each window size of the orthogonally encoded RS126 data set, the data set is randomly partitioned into two sets, 50% as the training data set to build the model and 50% as the test data set to assess the model accuracy as discussed in section 2.5.2. Then for each of the classifications schemes, namely helix against not helix (H/ $\sim$ H), sheet against not sheet (S/ $\sim$ S) and coil against not coil (C/ $\sim$ C) a classification tree is fitted on the training data set, using the *rpart* package (Therneau *et al.*, 2014). This is done in the following manner: the *rpart()* function is used to fit the maximum tree and the *prune()* function is used to prune the maximum tree. Each model's accuracy is assessed using the test data and the *predict()* function to predict the class probability. The accuracy of each classification tree is determined using the *confusionMatrix()* function in the *caret* package (Kuhn, 2014). The R code can be found in appendix A.

#### 5.6.1.2 Classification Tree Accuracy Using Hold-Out Cross Validation

The accuracy of the classification trees are shown in figure 5.6.1, where the x-axis represents the windows size and the y-axis represents the accuracy expressed as a percentage. A summary of the classification tree accuracy (%) across the various window sizes and for each class, is shown in table 5.7 below.

Figure 5.6.1: Test accuracy (%) of the classification trees.



The accuracy of the test data for coil against not coil is shown in figure 5.6.1, represented by the blue dashed line. The minimum accuracy of coil for all windows sizes is 65.70% obtained in windows size 9. The accuracy value rises and falls until it reaches a maximum of 67.40% at windows size 15. The mean accuracy for all the windows sizes is 66.28%. Past window size 15, the accuracy values fluctuate but never exceed the maximum of 67.40%.

The accuracy of the test data for helix against not helix is shown in figure 5.6.1, represented by a green dashed line. The maximum accuracy for helix is 70.53% obtained at windows size 11. The minimum values of 69.09% is attained at windows size 21. The mean accuracy for all the windows sizes is 69.84%. Increasing the windows size above 11 did not increase the accuracy.

Table 5.7: Test accuracy (%) of the classification trees.

	Helix (%)	Sheet (%)	Coil (%)
Minimum	69.09	76.25	65.70
Mean	69.84	76.90	66.28
Maximum	70.53	77.64	67.40

The accuracy of the test data for sheet against not sheet is shown in figure 5.6.1, represented by a red line. The maximum accuracy for sheet is 77.64% obtained at windows size 7. After this the accuracy typically falls until reaching the minimum of 76.90% at windows size 23. The mean accuracy for all the windows sizes is 77.64%. Increasing the windows size above 7 did not increase the accuracy.

## 5.6.2 5-fold Cross Validation

### 5.6.2.1 Data Set Partition and Application

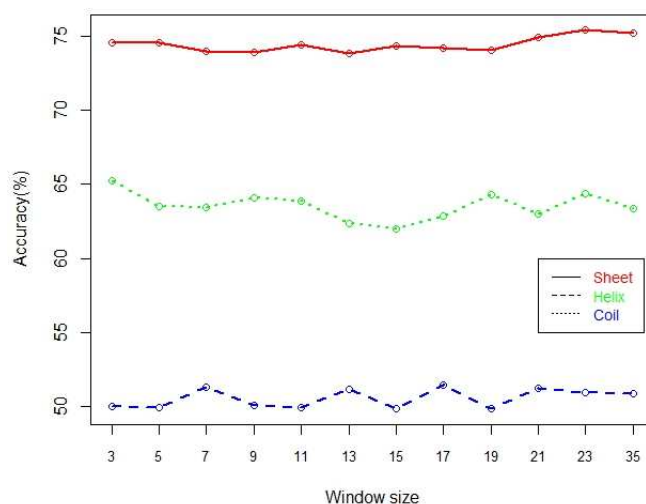
5-fold cross validation of the RS126 data set is applied using the classification tree classifiers. For each of the 12 different window sizes, randomly partition the data set into five folds. 4 folds, that contain 80% of the data set, are used as the training data and the remaining one fold, that contains 20% of the data, is used as the test data as discussed in section 2.5.1. The classification trees are fitted to each training data set using the *rpart* package (Therneau *et al.*, 2014) in the following manner: the *rpart()* function is used to fit the maximum tree using four folds of data and the *prune* function is used to prune the maximum tree. The classification tree accuracy is determined with the remaining fold using the *predict()* function to predict the class. The accuracy of the classification tree for the relevant fold of test data set is determined using the *confusionMatrix()* function in the *caret* package (Kuhn, 2014). The procedure is repeated 5 times until each fold has had a chance of being used as the test data set and the accuracy for each test data set is determined. The overall accuracy is the average of the five accuracy values determined from the procedure above (as per equation 2.5.1 on page 10). The process is

repeated, using the same folds of data, for each of the three classification tasks, namely helix against not helix (H/ $\sim$ H), sheet against not sheet (S/ $\sim$ S) and coil against not coil (C/ $\sim$ C). The R code can be found in appendix B.

### 5.6.2.2 Classification Tree Accuracy Using 5-fold Cross Validation

The accuracy of the classification trees, using 5-fold cross validation is shown in figure 5.6.2, where the x-axis represents the windows sizes and the y-axis represents the accuracy expressed as percentage. A summary of the classification tree accuracy, across the various widow sizes and for each class, is shown in table 5.8 below.

Figure 5.6.2: Test accuracy (%) of the classification trees using 5-fold cross validation.



The accuracy of coil against not coil is shown in figure 5.6.2, represented by a blue dashed line. The minimum accuracy of coil is 49.79% obtained in window size 15. The accuracy values rise and fall until it reaches the maximum of 51.45% at window size 17. The mean accuracy for all the windows sizes is 50.53%. Past window size 21 the accuracy values are decreasing.

Table 5.8: Test accuracy (%) of the classification trees using 5-fold cross validation.

	Helix (%)	Sheet (%)	Coil (%)
Minimum	61.96	73.85	49.79
Mean	63.53	74.48	50.53
Maximum	65.24	75.43	51.45

The accuracy of helix against not helix is shown in figure 5.6.2, represented by a green dashed line. The maximum accuracy for helix is 65.24% obtained at window size 3, the accuracy values

fluctuate until it reaches the minimum value of 61.96% at window size 15. The mean accuracy for all the windows sizes is 63.53%. Increasing the windows size did not increase accuracy.

The accuracy of sheet against not sheet is shown in figure 5.6.2, represented by a red line. The accuracy is 74.62% at windows size 3, after that when we increase the windows size the values for the accuracy rises and falls until it reaches a minimum of 73.85% at window size 13 and continues to fluctuate until reaches the maximum accuracy of 75.43% obtained at windows size 23. The mean accuracy for all the windows sizes is 74.48%.

## 5.7 Prediction of Protein Secondary Structure using Naive Bayes Classifiers

### 5.7.1 Hold out classification

#### 5.7.1.1 Data Set Partition and Application

For each window size of the orthogonally encoded RS126 data set, the data set is randomly partitioned into two sets, 50% as the training data set to build the model and 50% as the test data to assess the model accuracy as discussed in section 2.5.2. Then for each of the classification schemes, namely helix against not helix (H/ $\sim$ H), sheet against not sheet (S/ $\sim$ S) and coil against not coil (C/ $\sim$ C) a Naive Bayes classifier is fitted on the training data set using the *e1071* package (Dimitriadou *et al.*, 2009) in the following manner: the *naiveBayes()* function is used to fit the Naive Bayes model on the training data set. Each model's accuracy is determined using the test data set and the *predict()* function to predict the class probability. The accuracy of each of the Naive Bayes classifiers is determined using the *confusionMatrix()* function in the *caret* package (Kuhn, 2014). The R code can be found in appendix C.

#### 5.7.1.2 Naive Bayes Classifier Accuracy Using Hold-Out Cross Validation

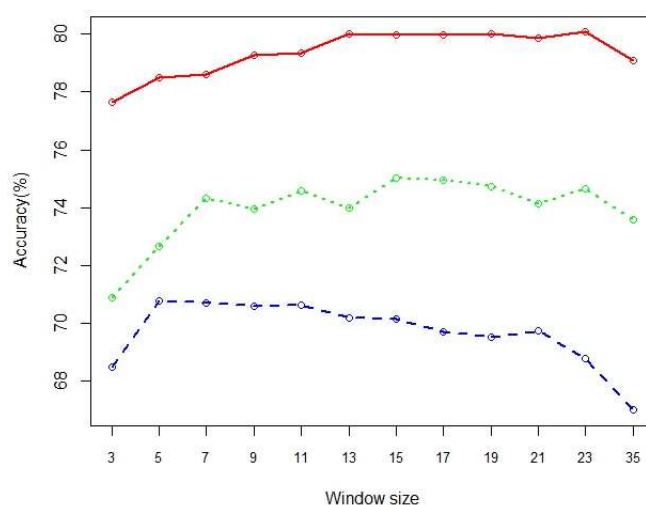
The accuracy of the Naive Bayes classifiers are shown in figure 5.7.1, where the x-axis represents the windows sizes and the y-axis represents the accuracy (%). A summary of the Naive Bayes classifiers accuracy, across the various window sizes and each class, is shown in table 5.9.

The accuracy of the test data for coil against not coil is shown in figure 5.7.1, represented by a blue dashed line. The maximum accuracy of 70.78% is attained at window size 5. After this the accuracies are less, but reasonably similar, until window size 21 where the accuracy drops until it reaches the minimum of 66.98% at window size 35. The mean accuracy for all the windows sizes is 69.68%.



## 59 5.7. Prediction of Protein Secondary Structure using Naive Bayes Classifiers

Figure 5.7.1: Test accuracy (%) of the Naive Bayes classifier.



The accuracy of the test data for helix against not helix is shown in figure 5.7.1, represented by a green dashed line. The minimum accuracy of helix is 70.88% obtained at windows size 3. The accuracy values incerease until the maximum of 75.02% is attained at windows size 15. The mean accuracy for all the windows sizes is 73.96%. Windows sizes above 7 have reasonably similar accuracy.

Table 5.9: Test accuracy (%) of the Naive Bayes classifier.

	Helix(%)	Sheet(%)	Coil(%)
Minimum	70.88	77.63	66.98
Mean	73.96	79.37	69.68
Maximum	75.02	80.07	70.78

The accuracy of the test data for sheet against not sheet is shown in figure 5.7.1, represented by a red line. The maximum accuracy for sheet is 80.07% obtained at windows size 23. The minimum of 77.63% attained at windows size 3 however this never exceed the maximum at window size 23. The mean accuracy for all the windows sizes is 79.37%.

Increasing the windows size did not increase the accuracy of the Naive Bayes approach for coil after window size 5. Incresing the window sizes for each of helix and sheet will increase the accuracy of the Naive Bayes classifier approach.

## 5.7.2 5-fold Cross Validation

### 5.7.2.1 Data Set Partition and Application

5-fold cross validation of the RS126 data set is applied using the Naive Bayes classifiers. For each of the 12 different window sizes, randomly partition the data set into five folds. 4 folds, that contain 80% of the data set, are used as the training data and the remaining one fold, that contain 20% of the data, is used as the test data set as discussed in section 2.5.1. The Naive Bayes classifiers are fitted to the training data set using the *e1071* package (Dimitriadou *et al.*, 2009) in the following manner: The *naiveBayes()* function is used to fit the classifier using the training data. The model accuracy is determined using the *predict()* function to predict the class probability. The accuracy of the Naive Bayes classifier for the relevant test data is determined using the *confusionMatrix()* function in the *caret* package (Kuhn, 2014). The process is repeated 5 times until each fold has had a chance of being used as the test data set and the accuracy for each test data set is determined. The overall accuracy is the average of the five accuracy values determined from the procedure above (as per equation 2.5.1 on page 10). The process is repeated, using the same folds of data, for each of the three classification tasks, namely helix against not helix (H/ $\sim$ H), sheet against not sheet (S/ $\sim$ S) and coil against not coil (C/ $\sim$ C). The R code can be found in appendix D.

### 5.7.2.2 Naive Bayes Classifier Accuracy using 5-fold Cross Validation

The accuracy of the Naive Bayes classifiers, using 5-fold cross validation, is shown in figure 5.7.2, where the x-axis represents the windows sizes and the y-axis represents the accuracy (%). A summary of the Naive Bayes classifiers, across the various window sizes and for each class, is shown in table 5.10.

Table 5.10: Test accuracy (%) of the Naive Bayes classifiers using 5-fold cross validation.

	Helix (%)	Sheet (%)	Coil (%)
Minimum	56.76	67.43	49.79
Mean	59.09	70.14	50.99
Maximum	63.10	73.35	52.09

The accuracy of the test data for coil against not coil is shown in figure 5.7.2, represented by a blue dashed line. The accuracy is 50.70% at window size 3. As the windows size increases the accuracy fluctuates until it reaches the maximum accuracy of 52.09% at window size 7. The accuracy subsequently drops down until reaches a minimum of 49.79% at window size 11. Past window size 7, the accuracy values fluctuate but never exceed the maximum of 52.09%. The mean accuracy for all the window sizes is 50.99%.

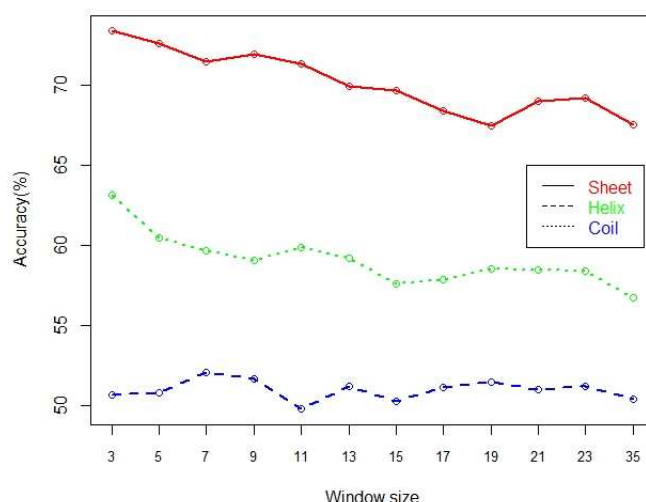
## 5.8. Prediction of Protein Secondary Structure using Logistic Regression

61

Classifiers

The accuracy of the test data for helix against not helix is shown in figure 5.7.2, represented by a green dashed line. The maximum accuracy of helix is 63.10% obtained at window size 3. The accuracy values rise and fall but never exceed the maximum until reaching the minimum of 56.76% at windows size 35. The mean accuracy for all the windows sizes is 59.09%. Increasing the window size reduces the accuracy.

Figure 5.7.2: Test accuracy (%) of the Naive Bayes classifier using 5-fold cross validation.



The accuracy of the test data for sheet against not sheet is shown in figure 5.7.2, represented by a red line. The maximum accuracy for sheet is 73.35% obtained at windows size 3. The accuracy fluctuates, but has a decreasing trend, until reaching the minimum of 67.43% at windows size 19. The accuracy never exceeds the maximum obtained at window size 3. The mean accuracy for all the window sizes is 70.14%. Increasing the windows size did not increase the accuracy.

## 5.8 Prediction of Protein Secondary Structure using Logistic Regression Classifiers

### 5.8.1 Hold-Out Cross Validation

#### 5.8.1.1 Data Set Partition and Application

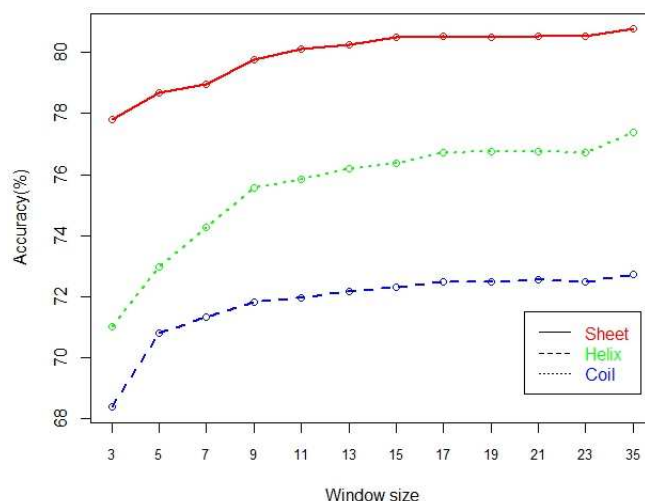
For each window size of the orthogonally encoded RS126 data set, the data set is randomly partitioned into two sets, 50% as the training data set to build the model and 50% as the test data to assess the model accuracy as discussed in section 2.5.2. Then for each of the classification schemes, namely helix against not helix (H/ $\sim$ H), sheet against not sheet (S/ $\sim$ S)

and coil against not coil (C/ $\sim$ C) a binary logistic regression model is fitted to the training data set using SPSS (IBM Corp, 2013). This is done because when we attempted to fit the relevant models in R (R Core Team, 2014) using the *GLM()* function the algorithm didn't converge, probably owing to the large number of independent binary variables with a binary dependent variable. The binary logistic regression function in SPSS was used to predict the class probability of the test data set and the result of the class probability prediction was read from SPSS into R. The accuracy of the logistic regression model was determined using the *confusionMatrix()* function in the caret package (Kuhn, 2014) in R. The R code can be found in appendix E.

### 5.8.1.2 Logistic Regression Classifier Accuracy

The accuracy of the logistic regression classifier is shown in figure 5.8.1, where the x-axis represents the windows sizes and the y-axis represents accuracy(%). A summary of the logistic regression classifier accuracy, across the various window sizes and for each class, is shown in table 5.11 below.

Figure 5.8.1: Test accuracy (%) of the logistic regression classifier.



The accuracy of the test data for coil against not coil is shown in figure 5.8.1, represented by a blue dashed line. The minimum accuracy is 68.38% at window size 3. The accuracy values tend to fluctuate in this range increase with increasing window size and reaches a maximum accuracy of 72.73% at window size 35. The mean accuracy for all the windows sizes is 71.79%.

The accuracy of the test data for helix against not helix is shown in figure 5.8.1, represented by a green dashed line. The minimum accuracy of helix is 71.02% obtained at windows size 3,

## 63 5.8. Prediction of Protein Secondary Structure using Logistic Regression Classifiers

the accuracy values increase with increasing window size and reaches a maximum accuracy of 77.39% at windows size 35. The mean accuracy for all the windows sizes is 75.55%.

The accuracy of the test data for sheet against not sheet is shown in figure 5.8.1, represented by a red line. The minimum accuracy for sheet is 77.79% obtained at windows size 3, the accuracy values increase with increasing window size and reaches a maximum accuracy of 80.76% at windows size 35. The mean accuracy for all the windows sizes is 79.91%.

Table 5.11: Test accuracy (%) of the logistic regression classifier.

	Helix (%)	Sheet (%)	Coil (%)
Minimum	71.02%	77.79%	68.38%
Mean	75.55%	79.91%	71.79%
Maximum	77.39%	80.76%	72.73%

### 5.8.2 5-fold Cross Validation

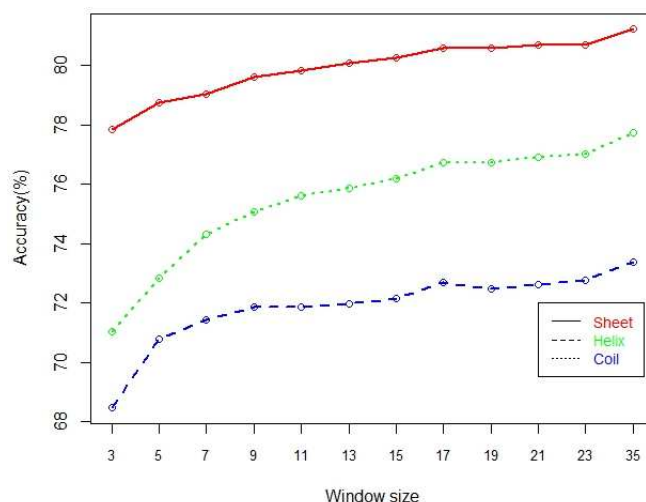
#### 5.8.2.1 Data Set Partition and Application

5-fold cross validation of the RS126 data set is applied using the logistic regression classifiers. For each of the 12 different window sizes, randomly partition the data set into five folds. 4 folds, that contain 80% of the data set, are used as the training data and the remaining one fold, that contains 20% of the data, is used as the test data as discussed in section 2.5.1. The accuracy of the 5-fold cross validated logistic regression classifier is determined in the following manner: fit the logistic regression using the binary logistic regression function in SPSS (IBM Corp, 2013) to the training data. Predict the class probability of the test data set. The result of the class probability prediction was read from SPSS into R. The accuracy of the logistic regression classifier was determined using the *confusionMatrix()* function in the caret package (Kuhn, 2014) in R. Repeat the procedure 5 times until each fold has had a chance of being used as the test data set. The overall accuracy is the average of the five accuracy values determined from the procedure above (as per equation 2.5.1 on page 10). The process is repeated, using the same folds of data, for each of the three classification tasks, namely helix against not helix (H/ $\sim$ H), sheet against not sheet (S/ $\sim$ S) and coil against not coil (C/ $\sim$ C). The R code can be found in appendix F.

#### 5.8.2.2 Logistic Regression Classifier Accuracy Using 5-fold Cross Validation

The accuracy of the logistic regression using 5-fold cross validation is shown in figure 5.8.2, where the x-axis represents the windows sizes and the y-axis represents the accuracy(%). A summary of the logistic regression classifier accuracy using 5-fold cross validation, across the various windows sizes and for each class, is shown in table 5.12 below.

Figure 5.8.2: Test accuracy (%) of the logistic regression classifier using 5-fold cross validation.



The accuracy of the test data for coil against not coil is shown in figure 5.8.2, represented by a blue dashed line. The minimum accuracy is 68.46% at window size 3. The accuracy values increase with window size and reach the maximum accuracy of 73.39% at window size 35. The mean accuracy for all the windows sizes is 71.87%.

The accuracy of the test data for helix against not helix is shown in figure 5.8.1, represented by a green dashed line. The minimum accuracy of helix is 71.03% obtained at windows size 3. The accuracy values increase with the window size and reach the maximum accuracy of 77.74% at window size 35. The mean accuracy for all the windows sizes is 75.50%.

The accuracy of the test data for sheet against not sheet is shown in figure 5.8.1, represented by a red line. The minimum accuracy for sheet is 77.85% obtained at windows size 3. The accuracy values increase with the window size and reach the maximum accuracy of 81.22% at windows size 35. The mean accuracy for all the windows sizes is 79.93%.

Table 5.12: Test accuracy (%) of the logistic regression classifier using 5-fold cross validation.

	Helix (%)	Sheet (%)	Coil (%)
Minimum	71.03	77.85	68.46
Mean	75.50	79.93	71.87
Maximum	77.74	81.22	73.39

## 5.9. Comparison of the Naive Bayes, Classification Tree and Logistic Regression Classifiers

### 5.9.1 Helix against not Helix

The objective of the study is to compare the performance of the Naive Bayes, classification tree and logistic regression classifiers using hold-out and 5-fold cross validation when predicting protein secondary structure. This study has considered the three main or primary secondary structure classes, namely helix, sheet and coil.

In this section, the best classifier for each classification task, namely helix against not helix (H/ $\sim$ H), sheet against not sheet (S/ $\sim$ S) and coil against not coil (C/ $\sim$ C), are compared in terms of their performance. This is done by looking at the highest prediction accuracies for each method at each window size. 95% confidence intervals for each classifier are constructed.

The protein secondary structure prediction accuracies (%) for predicting helix against not helix using the 12 different window sizes for the appropriately encoded RS126 test data for the Naive Bayes, classification trees and logistic regression classifiers using hold-out and 5-fold cross validation are shown in table 5.13. These data are graphed in figure 5.9.1, where the x-axis represents the window size and the y-axis represents the accuracy (%). The performance of the six classifiers when predicting helix against not helix secondary protein structure, as shown in table 5.13, are summarised in table 5.14.

Table 5.13: Test accuracy (%), for all window sizes, when predicting helix against not helix for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation.

Window Size	Hold-out validation			5-fold cross validation		
	Naive Bayes	Logistic regression	Classification tree	Naive Bayes	Logistic regression	Classification tree
3	70.88402	71.01679	70.11307	63.09805	71.03055	65.23725
5	72.67432	72.98698	69.95032	60.48195	72.82603	63.51824
7	74.33613	74.27189	70.11307	59.64003	74.31218	63.42588
9	73.93353	75.55251	69.81326	59.06529	75.05042	64.08596
11	74.56741	75.83519	70.53281	59.87823	75.59200	63.90423
13	73.98492	76.20353	70.27583	59.23375	75.87493	62.37284
15	75.02142	76.38342	70.06168	57.58713	76.19554	61.96240
17	74.94432	76.72606	70.12164	57.86377	76.72499	62.86486
19	74.7216	76.73891	69.46205	58.50574	76.73651	64.29855
21	74.14768	76.76889	69.08515	58.47828	76.90562	62.98154
23	74.66164	76.72606	69.29930	58.43479	77.00964	64.35569
35	73.59089	77.38564	69.27360	56.75881	77.73634	63.38914



The mean accuracy for the three classifiers are in the range of 59.09% to 75.55%. From the graph and summary tables it is apparent that the Naive Bayes classifier, using 5-fold cross validation, achieved the lowest accuracies. The classification tree classifiers, using 5-fold cross validation, performed second worst. The logistic regression classifier, using 5-fold cross validation, achieves the highest accuracy when compared to all other classification approaches considered in this study. The difference in the accuracy between the two logistic regression approaches is very small at all 12 different window sizes.

Figure 5.9.1: Comparison of the test accuracies (%) for predicting helix against not helix for each of the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation.

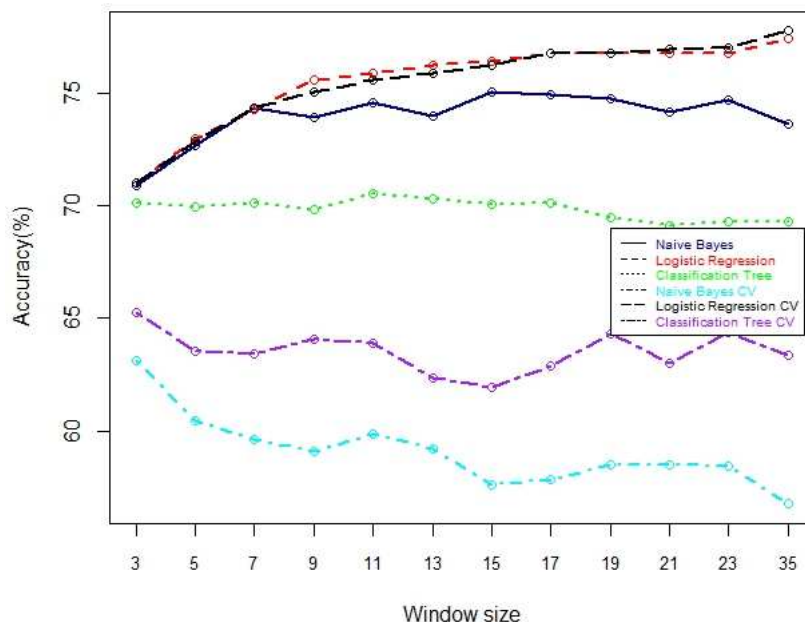


Table 5.14: Summary of the test accuracy (%), amongst all window sizes, when predicting helix against not helix for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation.

Classification approach	Hold-out validation			5-fold cross validation		
	Naive Bayes	Logistic regression	Classification tree	Naive Bayes	logistic regression	Classification tree
Minimum	70.88	71.02	69.09	56.76	71.03	61.96
Mean	73.96	75.55	69.84	59.09	75.50	63.53
Maximum	75.02	77.39	70.53	63.10	77.74	65.24

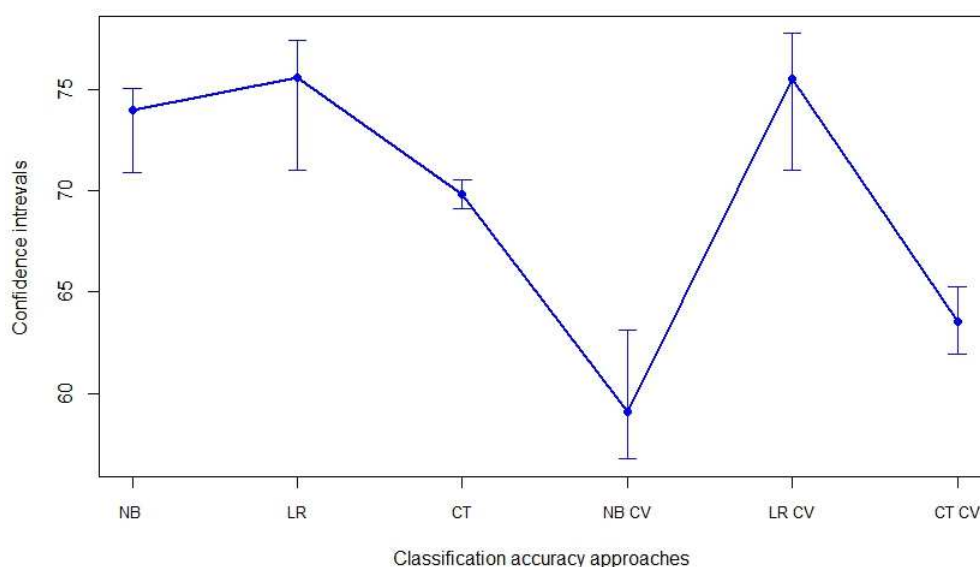
To determine where the prediction accuracies are different from each other, heuristic confidence intervals based on the accuracy estimate at each window size for each binary classifier



## 5.9. Comparison of the Naive Bayes, Classification Tree and Logistic Regression Classifiers

were constructed. The results were obtained using the R function in appendix G. The plots of the confidence intervals for the three classifiers, using different validation estimators, are shown in figure 5.9.2 below. Some of the confidence intervals overlap. The Naive Bayes 5-fold cross validation achieves accuracy between 58.05% and 60.12% accuracy and is the lowest 95% confidence interval. The logistic regression classifier, using 5-fold cross validation, achieves the highest 95% confidence interval. There is no different in the accuracy for either validation approaches using the logistic regression classifier. We can conclude that logistic regression classifier, with a large window size, performs best for predicting helix against not helix secondary structure.

Figure 5.9.2: Heuristic confidence intervals: Comparison of the test accuracies (%) for predicting helix against not helix for each of the Naive Bayes, classification tree and logistic regression classifiers using 5-fold cross validation.



### 5.9.2 Coil against not Coil

The protein secondary structure prediction accuracies (%) for predicting coil against not coil using the 12 different window sizes for the appropriately encoded RS126 test data for the Naive Bayes, classification trees and logistic regression classifiers using hold-out and 5-fold cross validation are shown in table 5.17. These data are graphed in figure 5.16, where the x-axis represents the window size and the y-axis represents the accuracy (%). The performance of the six classifiers when predicting helix against not helix secondary protein structure, as shown in table 5.13, are summarised in table 5.14.

Table 5.15: Test accuracy (%), for all window sizes, when predicting coil against not coil for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation.

Window Size	Hold-out validation			5-fold cross validation		
	Naive Bayes	Logistic regression	Classification tree	Naive Bayes	Logistic regression	Classification tree
3	68.49139	68.37845	65.98132	50.70346	68.45487	49.99315
5	70.77872	70.80264	66.36683	50.81007	70.77875	49.90343
7	70.71019	71.32517	66.30686	52.09042	71.43977	51.30394
9	70.59882	71.81771	65.69862	51.65751	71.85148	50.0258
11	70.62452	71.96762	66.33256	49.7881	71.86007	49.89632
13	70.16191	72.16892	66.20406	51.22451	71.96419	51.12405
15	70.12764	72.3017	67.40341	50.2668	72.15164	49.79900
17	69.69931	72.48158	66.35826	51.15903	72.69632	51.45122
19	69.50227	72.49872	66.72663	51.50126	72.46507	49.83768
21	69.75071	72.57153	65.89566	50.99672	72.63283	51.22780
23	68.78266	72.48158	65.77572	51.23207	72.77674	50.93262
35	66.98364	72.73000	66.28116	50.40041	73.38771	50.87119

The mean accuracy for the three classifiers are in the range of 50.09% to 71.87%. From the graph and summary tables it is apparent that the Naive Bayes classifier, using 5-fold cross validation, achieved the lowest accuracies. The logistic regression classifier, using 5-fold cross validation, achieves the highest accuracy when compared to all other classification approaches considered in this study. The difference in the accuracy between the two logistic regression approaches is very small at all 12 different window sizes.

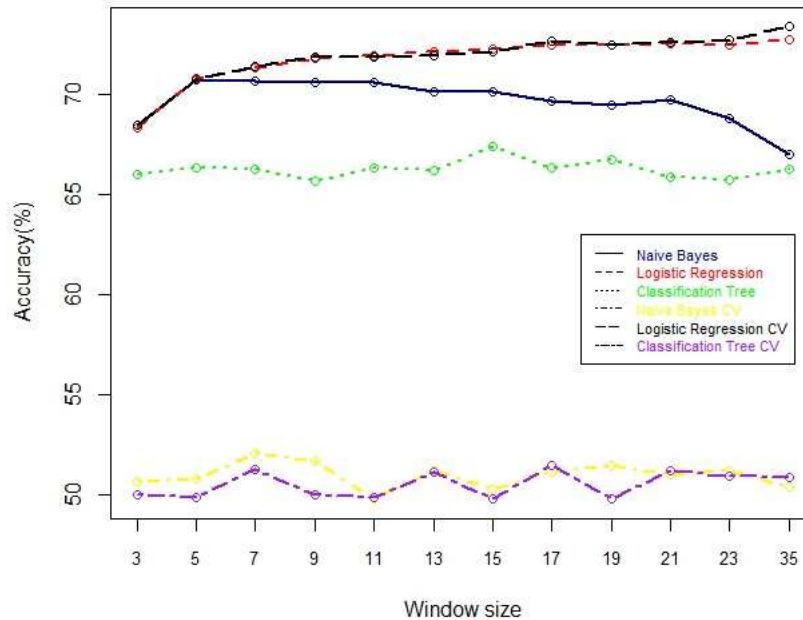
Table 5.16: Summary of the test accuracy (%), amongst all window sizes, when predicting coil against not coil for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation.

Classification approach	Hold-out validation			5-fold cross validation		
	Naive Bayes	Logistic regression	Classification tree	Naive Bayes	logistic regression	Classification tree
Minimum	66.98	68.38	65.70	49.79	68.46	49.80
Mean	69.68	71.79	66.28	50.99	71.87	50.53
Maximum	70.78	72.73	67.40	52.09	73.39	51.45

## 5.9. Comparison of the Naive Bayes, Classification Tree and Logistic Regression

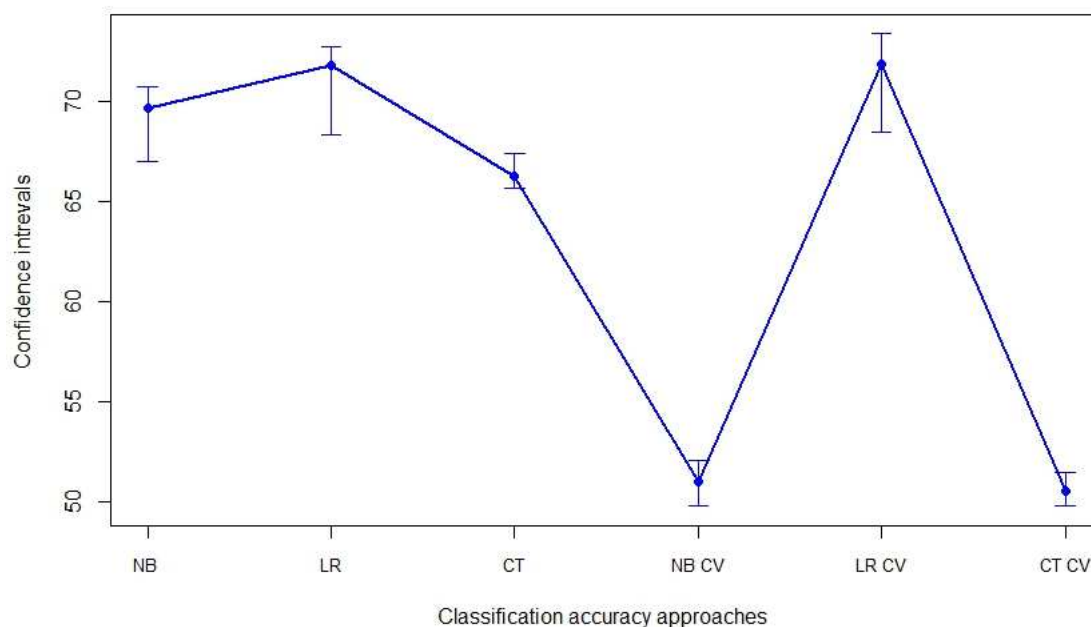
69 **Classifiers**

Figure 5.9.3: Comparison of the test accuracies (%) for predicting coil against not coil for each of the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation.



Heuristic confidence intervals based on the accuracy estimate at each window size for each binary classifier were constructed. The results were obtained using the R function in appendix G. The plots of confidence intervals for the three classifiers, using different validation estimators, are shown in figure 5.9.4 below. Some of the confidence intervals overlap. The Naive Bayes 5-fold cross validation achieves accuracy between 50.58% and 51.39% accuracy and is the lowest 95% confidence interval. The logistic regression classifier, using 5-fold cross validation, achieves the highest 95% confidence interval. There is no different in the accuracy for either validation approaches using the logistic regression classifier. We can conclude that logistic regression classifier, with a large window size, performs best for predicting coil against not coil secondary structure.

Figure 5.9.4: Heuristic confidence intervals: Comparison of the test accuracies (%) for predicting coil against not coil for each of the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation.



### 5.9.3 Sheet against not Sheet

The protein secondary structure prediction accuracies (%) for predicting sheet against not sheet using the 12 different window sizes for the appropriately encoded RS126 test data for the Naive Bayes, classification trees and logistic regression classifiers using hold-out and 5-fold cross validation are shown in table 5.17. These data are graphed in figure 5.9.5, where the x-axis represents the window size and the y-axis represents the accuracy (%). The performance of the six classifiers when predicting helix against not helix secondary protein structure, as shown in table 5.17, are summarised in table 5.18.

## 5.9. Comparison of the Naive Bayes, Classification Tree and Logistic Regression

71  
Table 5.17: Test accuracy (%), for all window sizes, when predicting sheet against not sheet for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation.

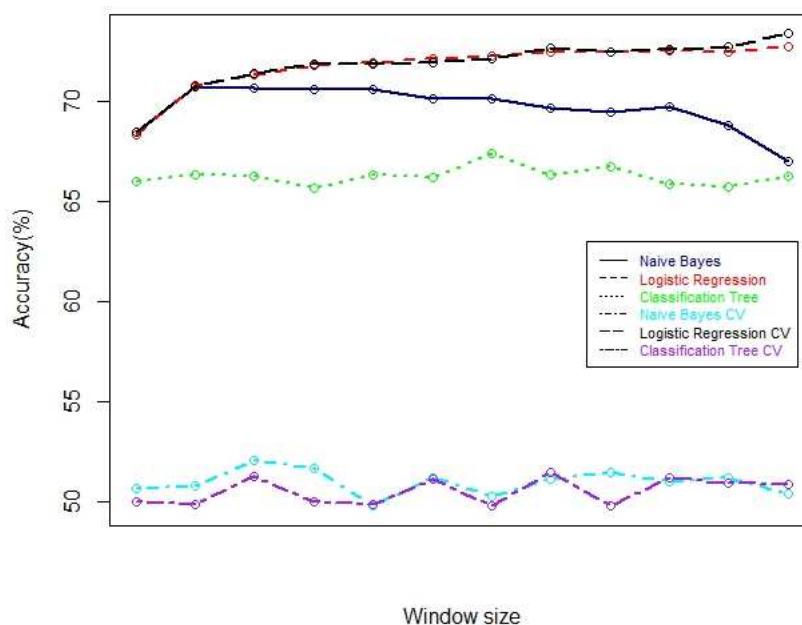
Window Size	Hold-out validation			5-fold cross validation		
	Naive Bayes	Logistic regression	Classification tree	Naive Bayes	Logistic regression	Classification tree
3	70.88402	77.79253	77.34087	73.35286	77.85146	74.62300
5	72.67432	78.67912	76.97250	72.56641	78.75358	74.55771
7	74.33613	78.97036	77.64071	71.43852	79.03474	74.02791
9	73.93353	79.76700	77.41797	71.91735	79.61413	73.91903
11	74.56741	80.0968	77.21237	71.282	79.81754	74.47340
13	73.98492	80.25527	76.77546	69.93202	80.06732	73.84522
15	75.02142	80.49512	76.72406	69.6489	80.26074	74.40113
17	74.94432	80.52082	76.47563	68.38525	80.60060	74.21308
19	74.7216	80.48655	76.71550	67.42764	80.58639	74.09829
21	74.14768	80.53366	76.72406	68.98405	80.68086	74.94365
23	74.66164	80.52082	76.25289	69.16735	80.69902	75.43447
35	73.59089	80.76066	76.56986	67.52183	81.2179	75.25989

The mean accuracy for the three classifiers are in the range of 70.14% to 79.93%. From the graph and associated tables it is apparent that the Naive Bayes classifier, using 5-fold cross validation, achieved the lowest accuracies. The logistic regression classifier, using 5-fold cross validation, achieves the highest accuracy when compared to all other classification approaches considered in this study. The difference in the accuracy between the two logistic regression approaches is very small at all 12 different window sizes.

Table 5.18: Summary of the test accuracy (%), amongst all window sizes, when predicting sheet against not sheet for the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation.

Classification approach	Hold-out validation			5-fold cross validation		
	Naive Bayes	Logistic regression	Classification tree	Naive Bayes	logistic regression	Classification tree
Minimum	70.88	77.79	76.25	67.43	77.85	73.85
Mean	73.96	79.91	76.90	70.14	79.93	74.48
Maximum	75.02	80.76	77.64	73.35	81.22	75.44

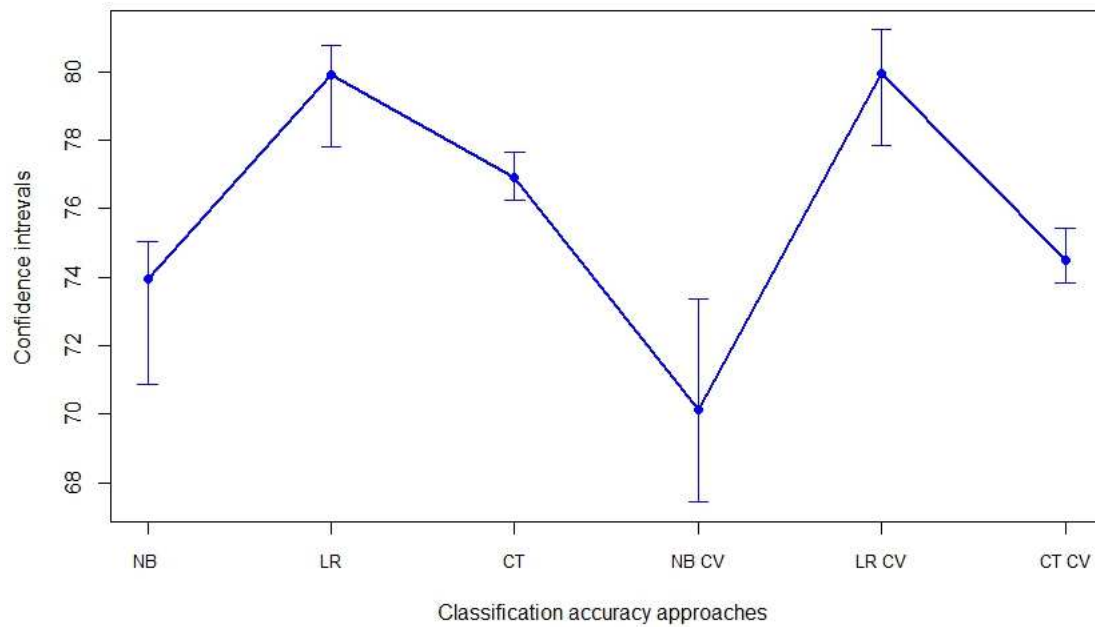
Figure 5.9.5: Comparison of the test accuracies (%) for predicting sheet against not sheet for each of the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation.



Heuristic confidence intervals based on the accuracy estimate at each window size for each binary classifier were constructed. The results were obtained using the R function in appendix G. The plots of confidence intervals for the three classifiers, using different validation estimators, are shown in figure 5.9.5 below. Some of the confidence intervals overlap. The Naive Bayes 5-fold cross validation achieves accuracy between 58.05% and 60.12% accuracy and is the lowest 95% confidence interval. The logistic regression classifier using 5-fold cross validation classifier achieves the highest 95% confidence interval. There is no difference in the accuracy for either validation approaches using the logistic regression classifier. We can conclude that logistic regression classifier, with a large window size, performs best for predicting sheet against not sheet secondary structure.

## 5.9. Comparison of the Naive Bayes, Classification Tree and Logistic Regression

**Classifiers**  
Figure 5.9.6: Heuristic confidence intervals: Comparison of the test accuracies (%) for predicting sheet against not sheet for each of the Naive Bayes, classification tree and logistic regression classifiers using hold-out or 5-fold cross validation.







# Chapter 6

## Conclusion

### 6.1 Introduction

This chapter provides conclusion and discussion of the study investigated in section 6.2. Future research is identified in section 6.3.

### 6.2 Conclusions and Discussion

Predicting the protein secondary structure is an important step in the prediction of the tertiary structure of proteins. This is because knowledge of the tertiary structure of proteins help in determining their functions.

The main aim of this thesis was to compare the performance of classification tree, Naive Bayes and logistic regression using hold-out and 5-fold cross validation in predicting the secondary structure of proteins from their amino acid sequences using the three classification tasks, namely helix against not helix (H/ $\sim$ H), sheet against not sheet (S/ $\sim$ S) and coil against not coil (C/ $\sim$ C). The following conclusions were derived:

- Naive Bayes using 5-fold cross validation achieved the lowest accuracy for predicting helix against not helix and classification trees using 5-fold cross validation achieved the lowest accuracies for both coil against not coil and sheet against not sheet classifications.
- The logistic regression classifier approach achieved the highest accuracy when compared to classification tree and Naive Bayes classifiers for each classification task, namely predicting helix against not helix, sheet against not sheet and coil against not coil.
- The logistic regression classifier accuracy is dependent on the window size; there is a positive relationship between the accuracy (%) and the window size.

- The Naive Bayes and classification tree have optimal window size.
- The classification tree, Naive Bayes and logistic regression classifiers using the hold-out cross validation approach achieved higher accuracy than the 5-fold cross validation for predicting the protein secondary structure for the three classes, namely helix, sheet and coil as given in section 5.9.

The results of the classification tree, Naive Bayes and logistic regression classifiers using hold-out and 5-fold cross validation were compared with the result of a support vector machine classifier based on the work of Hua and Sun (2001) and Tsilo (2009). Both these studies used the RS126 data set, utilised the same orthogonal primary structure encoding scheme and similar 8 to 3 secondary structure class reductions (Cuff and Barton, 1999; Tsilo, 2009). For all approaches three binary classifiers were considered, namely helix against not helix (H/ $\sim$ H), sheet against not sheet (S/ $\sim$ S) and coil against not coil (C/ $\sim$ C). The accuracy results obtained using the logistic regression classifier using hold-out and 5-fold cross validation compared are with these support vector machine classifiers constructed using 7-fold cross validation that were fitted to 7 different window sizes are shown in table 6.1.

Table 6.1: Comparison of the effect of change in window size on logistic regression and support vector machine Hua and Sun (2001).

Binary classifier	Logistic regression						Support Vector Machines		
	Hold-out			5-fold			7-fold		
	cross validation			cross validation			cross validation		
Window	Helix	Sheet	Coil	Helix	Sheet	Coil	Helix	Sheet	Coil
5	72.99	78.68	70.80	72.83	78.75	70.78	77.55	80.89	71.19
7	74.27	78.97	71.33	74.31	79.03	71.44	79.36	81.22	71.20
9	75.55	79.77	71.82	75.05	79.61	71.85	80.28	81.25	71.12
11	75.84	80.10	71.97	75.59	79.82	71.86	80.36	80.82	69.77
13	76.20	80.26	72.17	75.88	80.07	71.96	79.74	80.14	68.82
15	76.38	80.50	72.30	76.20	80.26	72.15	79.63	79.63	68.29
17	76.73	80.52	72.48	76.73	80.60	72.70	79.36	79.36	67.10
	17	17	17	17	17	17	11	9	7

Table 6.1 shows how a change in the window size effects the protein secondary structure prediction accuracy using either a support vector machine classifier Hua and Sun (2001) or a logistic regression classifier. For the logistic regression classifier, increasing the window size leads to an increase in accuracy prediction, while for the support vector machine there is an optimal window size. Using support vector machines to predict helix against not helix the optimum window size is 11, for sheet against not sheet 9 and for coil against not coil 7.

Table 6.2: Comparison of the logistic regression result with support vector machine using result of Hua and Sun (2001) and Tsilo (2009).

Binary classifier	Helix against not helix (H/ $\sim$ H)	Sheet against not sheet (S/ $\sim$ S)	Coil against not coil (C/ $\sim$ C)
Support vector machines (Hua and Sun, 2001)	79.74	80.40	69.77
Support vector machines (Tsilo, 2009)	73.74	80.32	68.31
Logistic regression cross validation [This study]	77.74	81.22	73.39

There is a clear difference when classifying coil against not coil (C/ $\sim$ C), a smaller difference when classifying helix against not helix (H/ $\sim$ H) and a small difference is observed for the overall accuracy when classifying sheet against not sheet (S/ $\sim$ S). Among all the methods the highest accuracy is observed for sheet against not sheet (S/ $\sim$ S) with the highest average accuracy being approximately 81.22%. Coil against not coil (C/ $\sim$ C) has the lowest average accuracies. Overall, the logistic regression classifier approach gives better result as compared to support vector machine classifiers based on the result in table 6.2.

## 6.3 Future Study

A problem was encountered when attempting to fit the logistic regression model in R (R Core Team, 2014). This is probably as a result of the iterative method used to estimate the various parameters being affected by the large number of independent binary variables. The binary logistic regression function in SPSS (IBM Corp, 2013) was used to predict the class probability of the test data set and the result of the class probability prediction was read from SPSS into R. Ideally it would be easier to compare these classifiers if the classification process could be facilitated in R. Alternatively it would be easier to assess these logistic regression classifiers if SPSS had a function to determine the accuracy of the logistic regression classifier.



# Bibliography

- Altman, D. G. and Bland, J. M. (1994). Statistics Notes: Diagnostic tests 1: sensitivity and specificity. *British Medical Journal*, 308(6943), 1552.
- Baxter, J. and Jäger, G. (2011). Protein secondary structure prediction: an application of Bayesian adaptive regression trees. In *53rd Annual Conference of the South African Statistical Association*, Pretoria, South Africa. CSIR Convention Centre.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer, New York.
- Boundless (2014). Protein Structures. In *Boundless Anatomy and Physiology*. Boundless. <https://www.boundless.com/physiology/textbooks/boundless-anatomy-and-physiology-textbook/general-chemistry-2/organic-compounds-39/protein-structures-317-11437/> Accessed: 03/01/2015.
- Breiman, L., Friedman, J. H., Stone, C. J., and Olshen, R. A. (1984). *Classification and Regression Trees*. Chapman & Hall/CRC, Boca Raton, London, New York, Washington.
- Burman, P., Chow, E., and Nolan, D. (1994). A cross-validatory method for dependent data. *Biometrika*, 81, 377–403.
- Cravedi, K. (2010). Expanding the Genetic Code with Unnatural Amino Acids. Department of Chemistry, The Catholic University of America.
- Cuff, J. A. and Barton, G. J. (1999). Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. *Proteins: Structure, Function, and Bioinformatics*, 34(4), 508–519.
- Dimitriadou, E., Hornik, K., Leisch, F., Meyer, D., Weingessel, A., and Leisch, M. F. (2009). Package e1071. *R Software package, available at <http://cran.rproject.org/web/packages/e1071/index.html>*.
- Esposito, F., Malerba, D., Semeraro, G., and Kay, J. (1997). A comparative analysis of methods for pruning decision trees. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(5), 476–491.

- Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27, 861–874.
- Foody, G. M. (2002). Status of land cover classification accuracy assessment. *Remote Sensing of Environment*, 80(1), 185–201.
- Frishman, D. and Argos, P. (1995). Knowledge-based protein secondary structure assignment. *Proteins: Structure, Function, and Bioinformatics*, 23(4), 566–579.
- Gareth, J., Daniela, W., Hastie, T., and Tibshirani, R. (2013). *Introduction to Statistical Learning with Applications in R*. Springer.
- Gelman, A. and Wang, W. (2013). A problem with the use of crossvalidation for selecting among multilevel models. In *International Society for Bayesian Analysis*, OBayes13. 2015.20.01.
- Han, J., Kamber, M., and Pei, J. (2012). *Data Mining: Concepts and Techniques* (Third ed.). The Morgan Kaufmann Series in Data Management Systems. Amsterdam: Elsevier/Morgan Kaufmann.
- Han, J. and Micheline, K. (2006). *Data Mining: Concepts and Techniques* (Second ed.). Morgan Kaufmann.
- Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*, Volume 2. Springer.
- Holley, L. H. and Karplus, M. (1989). Protein secondary structure prediction with a neural network. *Proceedings of the National Academy of Sciences*, 86(1), 152–156.
- Hua, S. and Sun, Z. (2001). A novel method of protein secondary structure prediction with high segment overlap measure: support vector machine approach. *Journal of Molecular Biology*, 308(2), 397–407.
- IBM Corp (Released 2013). *IBM SPSS Statistics for Windows, Version 22.0*. IBM Corp: Armonk, NY.
- Izenman, A. J. (2009). *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. Springer.
- Kabsch, W. and Sander, C. (1983). Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12), 2577–2637.
- Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5), 1–26.
- Kuhn, M. (2014). *caret: Classification and Regression Training*. R package version 6.0-30.

- Labatut, V. and Cherifi, H. (2012). Accuracy measures for the comparison of classifiers. *arXiv preprint arXiv:1207.3790*.
- Lewis, D. D. (1998). Naive (Bayes) at forty: The independence assumption in information retrieval. In *Machine learning: ECML-98*, 4–15. Springer.
- Mahmood, A. M., Mrithumjaya, P. G. V. G. K., and Kuppa, R. (2010). A new pruning approach for better and compact decision trees. *International Journal on Computer Science & Engineering*.
- McCallum, A. and Nigam, K. (1998). A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, Volume 752, 41–48.
- Muchai, E. and Odongo, L. (2014). Comparison of crisp and fuzzy classification trees using Gini index impurity measure on simulated data. *European Scientific Journal*, 10(18).
- Peng, C.-Y. J. and So, T.-S. H. (2002). Logistic regression analysis and reporting: A primer. *Understanding Statistics: Statistical Issues in Psychology, Education, and the Social Sciences*, 1, 31–70.
- Pratt, C. W., Voet, D., and Voet, J. G. (2005). *Fundamentals of Biochemistry: Life at the Molecular level*. John Wiley & Sons, New York.
- R Core Team (2014). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing.
- Raileanu, L. E. and Stoffel, K. (2004). Theoretical comparison between the Gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1), 77–93.
- Refaeilzadeh, P., Tang, L., and Liu, H. (2009). Cross-Validation. In L. Liu and M. T. Özsu (Eds.), *Encyclopedia of Database Systems*, 532–538. Springer US.
- Richards, F. M. and Kundrot, C. E. (1988). Identification of structural motifs from protein coordinate data: Secondary structure and first-level supersecondary structure. *Proteins: Structure, Function, and Bioinformatics*, 3(2), 71–84.
- Riis, S. K. and Krogh, A. (1996). Improving prediction of protein secondary structure using structured neural networks and multiple sequence alignments. *Journal of Computational Biology*, 3, 163–183.
- Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press.
- Ross, J. Q. (1986). Induction of decision trees. *Machine Learning*, 1(1), 81–106.
- Rost, B. and Sander, C. (1993). Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*, 232(2), 584–599.

- Sammut, C. and Webb, G. I. (2011). *Encyclopedia of Machine Learning*. Springer.
- Sheen, S. and Anitha, R. (2012). A Novel Node Splitting Criteria for Decision Trees Based on Theil Index. In T. Huang, Z. Zeng, C. Li, and C. Leung (Eds.), *Neural Information Processing*, Volume 7664 of *Lecture Notes in Computer Science*, 435–443. Springer Berlin Heidelberg.
- Singh, M., Sandhu, P. S., and Kaur, R. K. (2008). Protein secondary structure prediction. *World Academy of Science, Engineering and Technology*, 42, 458–461.
- Strobl, C., Boulesteix, A.-L., and Augustin, T. (2007). Unbiased split selection for classification trees based on the Gini index. *Computational Statistics & Data Analysis*, 52(1), 483–501.
- Therneau, T. M., Atkinson, E. J., and Foundation, M. (2014). *An introduction to recursive partitioning using the RPART routines*. R package version 4.1-8.
- Therneau, T. M., Atkinson, E. J., and Mayo, F. (1997). *Technical Report 61: An introduction to recursive partitioning using the RPART routines*. Section of Biostatistics, Mayo Clinic, Rochester.
- Timofeev, R. (2004). Classification and Regression Trees (CART): Theory and Applications. Master’s thesis, Humboldt University, Berlin.
- Tsilo, L. C. (2009). Protein Secondary Structure Prediction using Neural Networks and Support Vector Machines. Master’s thesis, Rhodes University.
- Whitford, D. (2005). *Proteins: Structure and Function*. John Wiley & Sons.
- Yohannes, Y. and Webb, P. (1999). *Classification and Regression Trees, (CART): a User Manual for Identifying Indicators of Vulnerability to Famine and Chronic Food Insecurity*, Volume 3. International Food Policy Research Institute.
- Yüksektepe, F. Ü., Yılmaz, Ö., and Türkay, M. (2008). Prediction of secondary structures of proteins using a two-stage method. *Computers & Chemical Engineering*, 32, 78–88.
- Zaki, M. J. and Meira Jr, W. (2014). *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press.
- Zhang, Y. and Rajapakse, J. C. (2009). *Machine Learning in Bioinformatics*, Volume 4. John Wiley & Sons.
- Zhu, X. (2010). CS769 Spring 2010 Advanced Natural Language Processing: Naive Bayes Classifier. <http://pages.cs.wisc.edu/~jerryzhu/cs769.html>. Accessed: 2015-07-02.



# Appendix A

## Classification Trees: Hold-Out Cross Validation

```
1 library(rpart)
2 rs126_data_window_3 <- read.delim("C:/temp/Data/Binary Excel Data/rs126_data_
   window_3.txt")
3 gdata_3 <- rs126_data_window_3
4 gdata_3<- data.frame(gdata_3[,1:66])
5 xdata_3<- data.frame(gdata_3[,4:66])
6 names(gdata_3)[3] <- "ydata_3"
7 ydata_3 <- as.factor(gdata_3[,3])
8 cartdata_3<-data.frame(ydata_3,xdata_3)
9 ### randomize the data
10 set.seed(999)
11 cartdata_3 <- data.frame(ydata_3,xdata_3)
12 trainIndex_3 <- createDataPartition(cartdata_3$ydata_3, p=0.50, list=FALSE)
13 cartdata.training_3 <- cartdata_3[ trainIndex_3,]
14 cartdata.testing_3 <- cartdata_3[-trainIndex_3,]
15 ### fitting the maximum tree on training data set for windows size 3 using
16 ### the rpart function from rpart package
17 training.orthogonal.rpart_3 <- rpart(ydata_3 ~ ., cartdata.training_3, method="
   class", control=rpart.control(cp = 0.001))
18 summary(training.orthogonal.rpart_3 )
19 ### plotting the maximum tree
20 plot(training.orthogonal.rpart_3 ,uniform = TRUE, margin=.1,main = "Using five
   fold cross first fold validation to growing maximum Classification tree for
   protein secondary struction data size 126 windows 3")
21 text(training.orthogonal.rpart_3 , splits = TRUE, use.n = TRUE, all = FALSE,
   fancy = FALSE, minlength = 4L, cex =0.6)
22 ### prediction class
23 pred.class_3<- predict(training.orthogonal.rpart_3, newdata = cartdata.testing_
   3, type = "class")
24 ### Residual tree
```

```

25 resid_3<- residuals(training.orthogonal.rpart_3)
26 #### Cost complexity omplexity
27 plotcp(training.orthogonal.rpart_3, minline = TRUE, lty = 3, col = 1, upper = "
    splits", main = "Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
28 plotcp(training.orthogonal.rpart_3, minline = TRUE, lty = 3, col = 1, upper = "
    size", main = "Cost complexity for Classification tree for protein secondary
    struction data size 126 windows 3")
29 cp.training_set.3_rpart <- printcp(training.orthogonal.rpart_3, digits =
    getOption("digits") - 2)
30 #### location of minimum cost complexity cross-validation error
31 cp.choice.set_3 <- training.orthogonal.rpart_3$cptable[which.min(training.
    orthogonal.rpart_3$cptable[, "xerror"]), "CP"]
32 prune.training.rpart_3 <- prune(training.orthogonal.rpart_3, cp = cp.choice.set_
    3)
33 summary(training.orthogonal.rpart_3)
34 plot(training.orthogonal.rpart_3 , uniform = TRUE, main = " Pruning maximum
    Classification tree for protein secondary struction data size 126 windows 3")
35 text(training.orthogonal.rpart_3 , splits = TRUE, use.n = TRUE, all = FALSE,
    fancy = FALSE, minlength = 4L, cex =0.8)
36 thepredictedclass.testdata_3 <- predict(training.orthogonal.rpart_3, newdata =
    cartdata.testing_3, type="class")
37 library(caret)
38 ydata.testing_3 <- cartdata.testing_3[,1]
39 testdataconfusion_w3_fold_3 <- confusionMatrix(data = thepredictedclass.testdata
    _3, reference= ydata.testing_3)
40 #### testing
41 #### fitting the maximum tree on testing data set for windows size 3 using rpart
42 #### function from rpart package
43 testing.orthogonal.rpart_3 <- rpart(ydata_3 ~ ., cartdata.testing_3 , method="
    class", control=rpart.control(cp = 0.001))
44 summary(testing.orthogonal.rpart_3 )
45 #### plotting the maximum tree
46 plot(testing.orthogonal.rpart_3 , uniform = TRUE, margin=.1, main = "Using hold
    out cross validation to growing maximum Classification tree for protein
    secondary struction data size 126 windows 3")
47 text(testing.orthogonal.rpart_3 , splits = TRUE, use.n = TRUE, all = FALSE, fancy
    = FALSE, minlength = 4L, cex =0.6)
48 #### prediction class
49 pred.classt_3 <- predict(testing.orthogonal.rpart_3, newdata = cartdata.training
    _3, type = "class")
50 #### Residual tree
51 residt_3 <- residuals(testing.orthogonal.rpart_3)
52 #### Cost complexity omplexity
53 plotcp(testing.orthogonal.rpart_3, minline = TRUE, lty = 3, col = 1, upper = "
    splits", main = " Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")

```

```

54 plotcp(testing.orthogonal.rpart_3, minline = TRUE, lty = 3, col = 1, upper = "
    size", main = " Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
55 cp.testing_set.3_rpart <- printcp(testing.orthogonal.rpart_3, digits = getOption
    ("digits") - 2)
56 ### location of minimum cost complexity cross-validation error
57 cp.choice.sett_3 <- testing.orthogonal.rpart_3$cptable[which.min(testing.
    orthogonal.rpart_3$cptable[, "xerror"]), "CP"]
58 prune.testing.rpart_3 <- prune(testing.orthogonal.rpart_3, cp = cp.choice.sett_
    3)
59 summary(testing.orthogonal.rpart_3)
60 plot(testing.orthogonal.rpart_3, uniform = TRUE, main = " Pruning maximum
    Classification tree for protein secondary struction data size 126 windows 3")
61 text(testing.orthogonal.rpart_3, splits = TRUE, use.n = TRUE, all = FALSE, fancy
    = FALSE, minlength = 4L, cex = 0.8)
62 thepredictedclass.traindata_3 <- predict(testing.orthogonal.rpart_3, newdata =
    cartdata.training_3, type="class")
63 library(caret)
64 ydata.training_3 <- cartdata.training_3[,1]
65 traindataconfusion_w3_fold_3 <- confusionMatrix(data = thepredictedclass.
    traindata_3, reference= ydata.training_3)
66 accuracy.w3.hold.out <- c(testdataconfusion_w3_fold_3$overall[1],
    traindataconfusion_w3_fold_3$overall[1])
67 accuracy.w3 <- mean(accuracy.w3.hold.out)

```

Listing A.1: Classification Trees: Hold-Out Cross Validation



# Appendix B

## Classification Trees: 5-fold Cross Validation

```
1 library(rpart)
2 gdata <- read.delim("C:/temp/Data/Binary Excel Data/rs126_data_window_3.txt")
3 gdata <- data.frame(gdata[,1:66])
4 xdata <- data.frame(gdata[,4:66])
5 ydata <- as.factor(gdata[,3])
6 names(gdata)[3] <- "ydata"
7 cartdata<-data.frame(ydata,xdata)
8 ## randomize the data
9 set.seed(9999)
10 gdata <- sample(gdata)
11 gdata_sample <- sample(1:23348)
12 set.1 <- gdata_sample[1:4669]
13 set.2 <- gdata_sample[4670:9338]
14 set.3 <- gdata_sample[9339:14007]
15 set.4 <- gdata_sample[14008:18676]
16 set.5 <- gdata_sample[18677:23348]
17 #### Training and testing data set for first fold of windows size 3
18 cartdata.training.31 <- cartdata[-set.1,]
19 cartdata.testing.31 <- cartdata[set.1,]
20 #### fitting the maximum tree on training data set for windows size 3 using
    rpart function from rpart package
21 training.orthogonal.rpart.31 <- rpart(ydata ~ .,cartdata.training.31 , method="
    class", control=rpart.control(cp = 0.001))
22 #### plotting the maximum tree
23 plot(training.orthogonal.rpart.31, uniform = TRUE, margin=.1, main = "Using five
    fold cross first fold validation to growing maximum Classification tree for
    protein secondary struction data size 126 windows 3")
24 text(training.orthogonal.rpart.31, splits = TRUE, use.n = TRUE, all = FALSE,
    fancy = FALSE, minlength = 4L, cex =0.6)
25 #### prediction of the class
```

```

26 pred.class.31 <- predict(training.orthogonal.rpart.31, newdata = cartdata.
    testing.31, type = "class")
27 ##### Residual tree
28 resid.31 <- residuals(training.orthogonal.rpart.31)
29 #### Cost complexity omplexity
30 plotcp(training.orthogonal.rpart.31, minline = TRUE, lty = 3, col = 1, upper =
    "splits", main = "Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
31 plotcp(training.orthogonal.rpart.31, minline = TRUE, lty = 3, col = 1, upper =
    "size", main = "Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
32 cp.training_set.11_rpart <- printcp(training.orthogonal.rpart.31, digits =
    getOption("digits") - 2)
33 ##### location of minimum cost complexity cross-validation error
34 cp.choice.set.31<-training.orthogonal.rpart.31$cptable[which.min(training.
    orthogonal.rpart.31$cptable[, "xerror"]), "CP"]
35 ##### pruning the maximum tree
36 prune.training.rpart.31 <- prune(training.orthogonal.rpart.31, cp = cp.choice.
    set.31)
37 plot(prune.training.rpart.31, uniform = TRUE, main = " Pruning maximum
    Classification tree for protein secondary struction data size 126 windows 3")
38 text(prune.training.rpart.31, splits = TRUE, use.n = TRUE, all = FALSE, fancy =
    FALSE, minlength = 4L, cex =0.8)
39 thepredictedclass.testdata.31 <- predict(prune.training.rpart.31, newdata =
    cartdata.testing.31, type="class")
40 #### Determine the accuracy
41 library(caret)
42 ydata.testing.31 <- cartdata.testing.31[set.1,1]
43 testdataconfusion_w3_fold.31 <- confusionMatrix(data = thepredictedclass.
    testdata.31, reference= ydata.testing.31)
44
45 #### 2 fold
46 #### Training data set for windows size 3
47 cartdata.training.32 <- cartdata[-set.2,]
48 cartdata.testing.32 <- cartdata[set.2,]
49 #### fitting the maximum tree on training data set for windows size 3 using
    rpart function from rpart package
50 training.orthogonal.rpart.32 <- rpart(ydata ~ ., cartdata.training.32 , method=
    "class", control=rpart.control(cp = 0.001))
51 #### plotting the maximum tree
52 plot(training.orthogonal.rpart.32 ,uniform = TRUE, margin=.1,main = "Using five
    fold cross first fold validation to growing maximum Classification tree for
    protein secondary struction data size 126 windows 3")
53 text(training.orthogonal.rpart.32, splits = TRUE, use.n = TRUE, all = FALSE,
    fancy = FALSE, minlength = 4L, cex =0.6)
54 #### prediction class

```

```

55 pred.class.32 <- predict(training.orthogonal.rpart.32, newdata = cartdata.
    testing.32, type = "class")
56 ##### Residual tree
57 resid.32 <- residuals(training.orthogonal.rpart.32)
58 #### Cost complexity omplexity
59 plotcp(training.orthogonal.rpart.32, minline = TRUE, lty = 3, col = 1, upper =
    "splits", main = "Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
60 plotcp(training.orthogonal.rpart.32, minline = TRUE, lty = 3, col = 1, upper =
    "size", main = " Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
61 cp.training_set.32_rpart <- printcp(training.orthogonal.rpart.32, digits =
    getOption("digits") - 2)
62 ##### location of minimum cost complexity cross-validation error
63 cp.choice.set.32 <- training.orthogonal.rpart.32$cptable[which.min(training.
    orthogonal.rpart.32$cptable[, "xerror"]), "CP"]
64 ##### Pruning the tree
65 prune.training.rpart.32 <- prune(training.orthogonal.rpart.32, cp = cp.choice.
    set.31)
66 plot(prune.training.rpart.32, uniform = TRUE, main = " Pruning maximum
    Classification tree for protein secondary struction data size 126 windows 3")
67 text(prune.training.rpart.32, splits = TRUE, use.n = TRUE, all = FALSE, fancy =
    FALSE, minlength = 4L, cex = 0.8)
68 thepredictedclass.testdata.32 <- predict(prune.training.rpart.32, newdata =
    cartdata.testing.32, type="class")
69 ##### Determine the accuracy
70 library(caret)
71 ydata.testing.32 <- cartdata.testing.32[set.2,1]
72 testdataconfusion_w3_fold.32 <- confusionMatrix(data = thepredictedclass.
    testdata.32, reference= ydata.testing.32)
73
74 #### 3 folds
75 #### Training data set for windows size 3
76 cartdata.training.33 <- cartdata[-set.3,]
77 cartdata.testing.33 <- cartdata[ set.3,]
78 #### fitting the maximum tree on training data set for windows size 3 using
    rpart function from rpart package
79 training.orthogonal.rpart.33 <- rpart(ydata ~ ., cartdata.training.33 , method="
    class", control=rpart.control(cp = 0.001))
80 #### plotting the maximum tree
81 plot(training.orthogonal.rpart.33, uniform = TRUE, margin=.1, main = "Using five
    fold cross first fold validation to growing maximum Classification tree for
    protein secondary struction data size 126 windows 3")
82 text(training.orthogonal.rpart.33, splits = TRUE, use.n = TRUE, all = FALSE,
    fancy = FALSE, minlength = 4L, cex = 0.6)
83 #### prediction

```

```

84 pred.class.33 <- predict(training.orthogonal.rpart.33, newdata = cartdata.
    testing.33 , type = "class")
85 ##### Residual tree
86 resid.33 <- residuals(training.orthogonal.rpart.33)
87 #### Cost complexity omplexity
88 plotcp(training.orthogonal.rpart.33, minline = TRUE, lty = 3, col = 1, upper =
    "splits", main = "Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
89 plotcp(training.orthogonal.rpart.33, minline = TRUE, lty = 3, col = 1, upper =
    "size", main = " Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
90 cp.training_set.33_rpart <- printcp(training.orthogonal.rpart.33, digits =
    getOption("digits") - 2)
91 ##### location of minimum cost complexity cross-validation error
92 cp.choice.set.33 <- training.orthogonal.rpart.33$cptable[which.min(training.
    orthogonal.rpart.33$cptable[, "xerror"]), "CP"]
93 ##### Puning tree
94 prune.training.rpart.33 <- prune(training.orthogonal.rpart.33, cp = cp.choice.
    set.31)
95 plot(prune.training.rpart.33 , uniform = TRUE, main = " Pruning maximum
    Classification tree for protein secondary struction data size 126 windows 3")
96 text(prune.training.rpart.33, splits = TRUE, use.n = TRUE, all = FALSE, fancy =
    FALSE, minlength = 4L, cex = 0.8)
97 thepredictedclass.testdata.33 <- predict(prune.training.rpart.33, newdata =
    cartdata.testing.33, type="class")
98 ##### Determine the accuracy
99 library(caret)
100 ydata.testing.33 <- cartdata.testing.33[set.3,1]
101 testdataconfusion_w3_fold.33 <- confusionMatrix(data = thepredictedclass.
    testdata.33, reference= ydata.testing.33)
102
103 #### 4 folds
104 #### Training data set for windows size 3
105 cartdata.training.34 <- cartdata[-set.4,]
106 cartdata.testing.34 <- cartdata[ set.4,]
107 #### fitting the maximum tree on training data set for windows size 3 using
    rpart function from rpart package
108 training.orthogonal.rpart.34 <- rpart(ydata ~ ., cartdata.training.34 , method="
    class", control=rpart.control(cp = 0.001))
109 #### plotting the maximum tree
110 plot(training.orthogonal.rpart.34, uniform = TRUE, margin=.1, main = "Using
    five fold cross first fold validation to growing maximum Classification tree
    for protein secondary struction data size 126 windows 3")
111 text(training.orthogonal.rpart.34, splits = TRUE, use.n = TRUE, all = FALSE,
    fancy = FALSE, minlength = 4L, cex = 0.6)
112 #### prediction class

```



```

113 pred.class.34 <- predict(training.orthogonal.rpart.34, newdata = cartdata.
      testing.34, type = "class")
114 ##### Residual tree
115 resid.34 <- residuals(training.orthogonal.rpart.34)
116 #### Cost complexity omplexity
117 plotcp(training.orthogonal.rpart.34, minline = TRUE, lty = 3, col = 1, upper =
      "splits", main = " Cost complexity for Classification tree for protein
      secondary struction data size 126 windows 3")
118 plotcp(training.orthogonal.rpart.34, minline = TRUE, lty = 3, col = 1, upper =
      "size", main = " Cost complexity for Classification tree for protein
      secondary struction data size 126 windows 3")
119 cp.training_set.34_rpart <- printcp(training.orthogonal.rpart.34, digits =
      getOption("digits") - 2)
120 ##### location of minimum cost complexity cross-validation error
121 cp.choice.set.34 <- training.orthogonal.rpart.34$cptable[which.min(training.
      orthogonal.rpart.34$cptable[, "xerror"]), "CP"]
122 #####pruning the tree
123 prune.training.rpart.34 <- prune(training.orthogonal.rpart.34, cp = cp.choice.
      set.31)
124 plot(prune.training.rpart.34, uniform = TRUE, main = "Pruning maximum
      Classification tree for protein secondary struction data size 126 windows 3")
125 text(prune.training.rpart.34, splits = TRUE, use.n = TRUE, all = FALSE, fancy =
      FALSE, minlength = 4L, cex = 0.8)
126 thepredictedclass.testdata.34 <- predict(prune.training.rpart.34, newdata =
      cartdata.testing.34, type="class")
127 ##### Determine the accuracy
128 library(caret)
129 ydata.testing.34 <- cartdata.testing.34[set.4,1]
130 testdataconfusion_w3_fold.34 <- confusionMatrix(data = thepredictedclass.
      testdata.34, reference= ydata.testing.34)
131
132 #####
133 ### 5 folds
134 ### Training data set for windows size 3
135 cartdata.training.35 <- cartdata[-set.5,]
136 cartdata.testing.35 <- cartdata[set.5,]
137 training.orthogonal.rpart.35 <- rpart(ydata ~ ., cartdata.training.35, method="
      class", control=rpart.control(cp = 0.001))
138 ### plotting the maximum tree
139 plot(training.orthogonal.rpart.35, uniform = TRUE, margin=.1, main = "Using five
      fold cross first fold validation to growing maximum Classification tree for
      protein secondary struction data size 126 windows 3")
140 text(training.orthogonal.rpart.35, splits = TRUE, use.n = TRUE, all = FALSE,
      fancy = FALSE, minlength = 4L, cex = 0.6)
141 ### prediction
142 pred.class.35 <- predict(training.orthogonal.rpart.35, newdata = cartdata.
      testing.35, type = "class")

```

```

143 ##### Residual tree
144 resid.35 <- residuals(training.orthogonal.rpart.35)
145 ##### Cost complexity complexity
146 plotcp(training.orthogonal.rpart.35, minline = TRUE, lty = 3, col = 1, upper = "
    splits", main = " Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
147 plotcp(training.orthogonal.rpart.35, minline = TRUE, lty = 3, col = 1, upper =
    "size", main = " Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
148 cp.training_set.35_rpart <- printcp(training.orthogonal.rpart.35, digits =
    getOption("digits") - 2)
149 ##### location of minimum cost complexity cross-validation error
150 cp.choice.set.35 <- training.orthogonal.rpart.35$cptable[which.min(training.
    orthogonal.rpart.35$cptable[, "xerror"]), "CP"]
151 ##### pruning the tree
152 prune.training.rpart.35 <- prune(training.orthogonal.rpart.35, cp = cp.choice.
    set.31)
153 summary(prune.training.rpart.35)
154 plot(prune.training.rpart.35, uniform = TRUE, main = " Pruning maximum
    Classification tree for protein secondary struction data size 126 windows 3")
155 text(prune.training.rpart.35, splits = TRUE, use.n = TRUE, all = FALSE, fancy =
    FALSE, minlength = 4L, cex = 0.8)
156 thepredictedclass.testdata.35 <- predict(prune.training.rpart.35, newdata =
    cartdata.testing.35, type="class")
157 library(caret)
158 ydata.testing.35 <- cartdata.testing.35[set.5,1]
159 testdataconfusion_w3_fold.35 <- confusionMatrix(data = thepredictedclass.
    testdata.35, reference= ydata.testing.35)
160 testdataconfusion_w3 <- data.frame( testdataconfusion_w3_fold.31$overall,
    testdataconfusion_w3_fold.32$overall, testdataconfusion_w3_fold.33$overall,
    testdataconfusion_w3_fold.34$overall, testdataconfusion_w3_fold.35$overall)
161 bClasstestdataconfusion_w3 <- data.frame(testdataconfusion_w3_fold.31$byClass,
    testdataconfusion_w3_fold.32$byClass, testdataconfusion_w3_fold.33$byClass,
    testdataconfusion_w3_fold.34$byClass, testdataconfusion_w3_fold.35$byClass)
162 accuracy_windows_3 <- c(testdataconfusion_w3_fold.31$overall[1],
    testdataconfusion_w3_fold.32$overall[1], testdataconfusion_w3_fold.33$overall
    [1], testdataconfusion_w3_fold.34$overall[1], testdataconfusion_w3_fold.35$
    overall[1])
163 Accuracy_3 <- mean(accuracy_windows_3 )
164 Accuracy_3

```

Listing B.1: Classification Trees: 5-fold Cross Validation

# Appendix C

## Naive Bayes: Hold-Out Cross Validation

```
1 library(rpart)
2 rs126_data_window_3 <- read.delim("C:/temp/Data/Binary Excel Data/rs126_data_
   window_3.txt")
3 gdata_3 <- rs126_data_window_3
4 gdata_3<- data.frame(gdata_3[,1:66])
5 xdata_3<- data.frame(gdata_3[,4:66])
6 names(gdata_3)[3] <- "ydata_3"
7 ydata_3 <- as.factor(gdata_3[,3])
8 cartdata_3<-data.frame(ydata_3,xdata_3)
9 #### randomize the data
10 set.seed(999)
11 cartdata_3 <- data.frame(ydata_3,xdata_3)
12 trainIndex_3 <- createDataPartition(cartdata_3$ydata_3, p=0.50, list=FALSE)
13 cartdata.training_3 <- cartdata_3[ trainIndex_3,]
14 cartdata.testing_3 <- cartdata_3[-trainIndex_3,]
15 #### fitting the maximum tree on training data set for windows size 3 using
16 #### the rpart function from rpart package
17 training.orthogonal.rpart_3 <- rpart(ydata_3 ~ ., cartdata.training_3, method="
   class", control=rpart.control(cp = 0.001))
18 summary(training.orthogonal.rpart_3 )
19 #### plotting the maximum tree
20 plot(training.orthogonal.rpart_3 ,uniform = TRUE, margin=.1,main = "Using five
   fold cross first fold validation to growing maximum Classification tree for
   protein secondary struction data size 126 windows 3")
21 text(training.orthogonal.rpart_3 , splits = TRUE, use.n = TRUE, all = FALSE,
   fancy = FALSE, minlength = 4L, cex =0.6)
22 #### prediction class
23 pred.class_3<- predict(training.orthogonal.rpart_3, newdata = cartdata.testing_
   3, type = "class")
24 #### Residual tree
25 resid_3<- residuals(training.orthogonal.rpart_3)
26 #### Cost complexity omplexity
```

```

27 plotcp(training.orthogonal.rpart_3, minline = TRUE, lty = 3, col = 1, upper = "
    splits", main = "Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
28 plotcp(training.orthogonal.rpart_3, minline = TRUE, lty = 3, col = 1, upper = "
    size", main = "Cost complexity for Classification tree for protein secondary
    struction data size 126 windows 3")
29 cp.training_set.3_rpart <- printcp(training.orthogonal.rpart_3, digits =
    getOption("digits") - 2)
30 #### location of minimum cost complexity cross-validation error
31 cp.choice.set_3 <- training.orthogonal.rpart_3$cptable[which.min(training.
    orthogonal.rpart_3$cptable[, "xerror"]), "CP"]
32 prune.training.rpart_3 <- prune(training.orthogonal.rpart_3, cp = cp.choice.set_
    3)
33 summary(training.orthogonal.rpart_3)
34 plot(training.orthogonal.rpart_3, uniform = TRUE, main = " Pruning maximum
    Classification tree for protein secondary struction data size 126 windows 3")
35 text(training.orthogonal.rpart_3, splits = TRUE, use.n = TRUE, all = FALSE,
    fancy = FALSE, minlength = 4L, cex = 0.8)
36 thepredictedclass.testdata_3 <- predict(training.orthogonal.rpart_3, newdata =
    cartdata.testing_3, type="class")
37 library(caret)
38 ydata.testing_3 <- cartdata.testing_3[,1]
39 testdataconfusion_w3_fold_3 <- confusionMatrix(data = thepredictedclass.testdata
    _3, reference= ydata.testing_3)
40 #### testing
41 #### fitting the maximum tree on testing data set for windows size 3 using rpart
42 #### function from rpart package
43 testing.orthogonal.rpart_3 <- rpart(ydata_3 ~ ., cartdata.testing_3, method="
    class", control=rpart.control(cp = 0.001))
44 summary(testing.orthogonal.rpart_3)
45 #### plotting the maximum tree
46 plot(testing.orthogonal.rpart_3, uniform = TRUE, margin=.1, main = "Using hold
    out cross validation to growing maximum Classification tree for protein
    secondary struction data size 126 windows 3")
47 text(testing.orthogonal.rpart_3, splits = TRUE, use.n = TRUE, all = FALSE, fancy
    = FALSE, minlength = 4L, cex = 0.6)
48 #### prediction class
49 pred.classt_3 <- predict(testing.orthogonal.rpart_3, newdata = cartdata.training
    _3, type = "class")
50 #### Residual tree
51 residt_3 <- residuals(testing.orthogonal.rpart_3)
52 #### Cost complexity omplexity
53 plotcp(testing.orthogonal.rpart_3, minline = TRUE, lty = 3, col = 1, upper = "
    splits", main = " Cost complexity for Classification tree for protein
    secondary struction data size 126 windows 3")
54 plotcp(testing.orthogonal.rpart_3, minline = TRUE, lty = 3, col = 1, upper = "
    size", main = " Cost complexity for Classification tree for protein

```

```

    secondary struction data size 126 windows 3")
55 cp.testing_set.3_rpart <- printcp(testing.orthogonal.rpart_3, digits = getOption
    ("digits") - 2)
56 #### location of minimum cost complexity cross-validation error
57 cp.choice.sett_3 <- testing.orthogonal.rpart_3$cptable[which.min(testing.
    orthogonal.rpart_3$cptable[, "xerror"]), "CP"]
58 prune.testing.rpart_3 <- prune(testing.orthogonal.rpart_3, cp = cp.choice.sett_
    3)
59 summary(testing.orthogonal.rpart_3)
60 plot(testing.orthogonal.rpart_3, uniform = TRUE, main = " Pruning maximum
    Classification tree for protein secondary struction data size 126 windows 3")
61 text(testing.orthogonal.rpart_3, splits = TRUE, use.n = TRUE, all = FALSE, fancy
    = FALSE, minlength = 4L, cex = 0.8)
62 thepredictedclass.traindata_3 <- predict(testing.orthogonal.rpart_3, newdata =
    cartdata.training_3, type="class")
63 library(caret)
64 ydata.training_3 <- cartdata.training_3[,1]
65 traindataconfusion_w3_fold_3 <- confusionMatrix(data = thepredictedclass.
    traindata_3, reference= ydata.training_3)
66 accuracy.w3.hold.out <- c(testdataconfusion_w3_fold_3$overall[1],
    traindataconfusion_w3_fold_3$overall[1])
67 accuracy.w3 <- mean(accuracy.w3.hold.out)

```

Listing C.1: Naive Bayes: Hold-Out Cross Validation



# Appendix D

## Naive Bayes: 5-fold Cross Validation

```
1 library(class)
2 library(e1071)
3 library(caret)
4 library(rpart)
5 rs126_data_window_3 <- read.delim("C:/temp/Data/Binary Excel Data/rs126_data_
  window_3.txt")
6 gdata_3 <- rs126_data_window_3
7 gdata_3 <- data.frame(gdata_3[,1:66])
8 xdata_3 <- data.frame(gdata_3[,4:66])
9 names(gdata_3)[3] <- "ydata_3"
10 ydata_3 <- as.factor(gdata_3[,3])
11 cartdata_3 <- data.frame(ydata_3,xdata_3)
12 ### Training and testing data set for windows size_31
13 set.seed(22)
14 gdata_3 <- sample(gdata_3)
15 gdata_sample <- sample (1:23348)
16 set.1 <- gdata_sample[1:4669]
17 set.2 <- gdata_sample[4670:9338]
18 set.3 <- gdata_sample[9339:14007]
19 set.4 <- gdata_sample[14008:18676]
20 set.5 <- gdata_sample[18677:23348]
21 cartdata.training_31 <- cartdata_3[-set.1,]
22 cartdata.testing_31 <- cartdata_3[set.1,]
23 classifier.naive._31 <- naiveBayes(xdata_3, ydata_3, data = cartdata.training_
  31)
24 predict.classifier.naive._31<- predict(classifier.naive._31, newdata=cartdata.
  testing_31)
25 ydata.testing_31 <- cartdata.testing_31[set.1,1]
26 library(caret)
27 predict.naive_31 <- confusionMatrix(data = predict.classifier.naive._31,
  reference= ydata.testing_31)
28 ##### Accuracy
29 accuracy_31 <- predict.naive_31$overall[1]
```

```

30 Kappa_31 <- predict.naive_31$overall[2]
31 AccuracyLower_31 <- predict.naive_31$overall[3]
32 AccuracyUpper_31 <- predict.naive_31$overall[4]
33 AccuracyNull_31 <- predict.naive_31$overall[5]
34 AccuracyPValue_31 <- predict.naive_31$overall[6]
35 McNemarPValue_31 <- predict.naive_31$overall[7]
36 sensitivity_31 <-predict.naive_31$byClass[1]
37 Specificity_31 <- predict.naive_31$byClass[2]
38 Pos_Pred_Value_31 <- predict.naive_31$byClass[3]
39
40 #### Second fold
41 cartdata.training_32 <- cartdata_3[-set.2,]
42 cartdata.testing_32 <- cartdata_3[set.2,]
43 classifier.naive._32 <- naiveBayes(xdata_3, ydata_3, data = cartdata.training_
    32)
44 predict.classifier.naive._32 <- predict(classifier.naive._32, newdata=cartdata.
    testing_32)
45 ydata.testing_32 <- cartdata.testing_32[set.2,1]
46 library(caret)
47 predict.naive_32<- confusionMatrix(data = predict.classifier.naive._32,
    reference= ydata.testing_32)
48 ##### Accuracy
49 accuracy_32 <- predict.naive_32$overall[1]
50 Kappa_32 <- predict.naive_32$overall[2]
51 AccuracyLower_32 <- predict.naive_32$overall[3]
52 AccuracyUpper_32 <- predict.naive_32$overall[4]
53 AccuracyNull_32 <- predict.naive_32$overall[5]
54 AccuracyPValue_32 <- predict.naive_32$overall[6]
55 McNemarPValue_32 <- predict.naive_32$overall[7]
56 sensitivity_32 <-predict.naive_32$byClass[1]
57 Specificity_32 <- predict.naive_32$byClass[2]
58 Pos_Pred_Value_32 <- predict.naive_32$byClass[3]
59
60 #### Third Fold
61 cartdata.training_33 <- cartdata_3[-set.3,]
62 cartdata.testing_33 <- cartdata_3[set.3,]
63 classifier.naive._33 <- naiveBayes(xdata_3, ydata_3, data = cartdata.training_
    33)
64 predict.classifier.naive._33 <- predict(classifier.naive._33, newdata=cartdata.
    testing_33)
65 ydata.testing_33 <- cartdata.testing_3[set.3,1]
66 library(caret)
67 predict.naive_33 <- confusionMatrix(data = predict.classifier.naive._33,
    reference= ydata.testing_33)
68 ##### Accuracy
69 accuracy_33 <- predict.naive_33$overall[1]
70 Kappa_33 <- predict.naive_33$overall[2]

```



```

71 AccuracyLower_33 <- predict.naive_33$overall[3]
72 AccuracyUpper_33 <- predict.naive_33$overall[4]
73 AccuracyNull_33 <- predict.naive_33$overall[5]
74 AccuracyPValue_33 <- predict.naive_33$overall[6]
75 McNemarPValue_33 <- predict.naive_33$overall[7]
76 sensitivity_33 <- predict.naive_33$byClass[1]
77 Specificity_33 <- predict.naive_33$byClass[2]
78 Pos_Pred_Value_33 <- predict.naive_33$byClass[3]
79
80 #### fourth fold
81 cartdata.training_34 <- cartdata_3[-set.4,]
82 cartdata.testing_34 <- cartdata_3[set.4,]
83 classifier.naive._34 <- naiveBayes(xdata_3, ydata_3, data = cartdata.training_
      34)
84 predict.classifier.naive._34 <- predict(classifier.naive._34, newdata=cartdata.
      testing_34)
85 ydata.testing_34 <- cartdata.testing_34[set.1,1]
86 library(caret)
87 predict.naive_34 <- confusionMatrix(data = predict.classifier.naive._34,
      reference= ydata.testing_34)
88 ##### Accuracy
89 accuracy_34 <- predict.naive_34$overall[1]
90 Kappa_34 <- predict.naive_34$overall[2]
91 AccuracyLower_34 <- predict.naive_34$overall[3]
92 AccuracyUpper_34 <- predict.naive_34$overall[4]
93 AccuracyNull_34 <- predict.naive_34$overall[5]
94 AccuracyPValue_34 <- predict.naive_34$overall[6]
95 McNemarPValue_34 <- predict.naive_34$overall[7]
96 sensitivity_34 <- predict.naive_34$byClass[1]
97 Specificity_34 <- predict.naive_34$byClass[2]
98 Pos_Pred_Value_34 <- predict.naive_34$byClass[3]
99
100 #### fifth fold
101 cartdata.training_35 <- cartdata_3[-set.5,]
102 cartdata.testing_35 <- cartdata_3[set.5,]
103 classifier.naive._35 <- naiveBayes(xdata_3, ydata_3, data = cartdata.training_
      35)
104 predict.classifier.naive._35 <- predict(classifier.naive._35, newdata=cartdata.
      testing_35)
105 ydata.testing_35 <- cartdata.testing_35[set.5,1]
106 library(caret)
107 predict.naive_35 <- confusionMatrix(data = predict.classifier.naive._35,
      reference= ydata.testing_35)
108 ##### Accuracy
109 accuracy_35 <- predict.naive_35$overall[1]
110 Kappa_35 <- predict.naive_35$overall[2]
111 AccuracyLower_35 <- predict.naive_35$overall[3]

```

```
112 AccuracyUpper_35 <- predict.naive_35$overall[4]
113 AccuracyNull_35 <- predict.naive_35$overall[5]
114 AccuracyPValue_35 <- predict.naive_35$overall[6]
115 McnemarPValue_35 <- predict.naive_35$overall[7]
116 sensitivity_35 <- predict.naive_35$byClass[1]
117 Specificity_35 <- predict.naive_35$byClass[2]
118 Pos_Pred_Value_35 <- predict.naive_35$byClass[3]
119 Accuracy.naive_3_cv <- c(accuracy_31, accuracy_32, accuracy_33, accuracy_34,
    accuracy_35)
120 Accuracy.naive_3 <- mean(Accuracy.naive_3_cv)
```

Listing D.1: Naive Bayes: 5-fold Cross Validation

# Appendix E

## Logistic Regression: Hold-Out Cross Validation

```
1 library(caret)
2 windows.3 <- read.delim("U:/cross validation excel result/Logistic regression
   result section code and data/Logistic regression result/windows 3.txt")
3
4 #logistic regression result win.3
5 # Helix variable and predicted logistic regression class from SPSS
6 logistic.predict.test.H.w3 <- windows.3[,1]
7 Respond.test.H.w3 <- windows.3[,2]
8 testDataconfusion.test_H_w3 <- confusionMatrix(data = logistic.predict.test.H.w3
   , reference=Respond.test.H.w3 )
9 logistic.predict.train.H.w3 <- windows.3[,1]
10 Respond.train.H.w3 <- windows.3[,2]
11 testDataconfusion.train_H_w3 <- confusionMatrix(data = logistic.predict.train.H.
   w3, reference=Respond.train.H.w3 )
12 accuracy.hold.out.H_w3 <- mean(c(testDataconfusion.test_H_w3$overall[1],
   testDataconfusion.train_H_w3$overall[1]))
13
14 # Sheet variable and predicted logistic regression class from SPSS
15 logistic.predict.test.S.w3 <- windows.3[,3]
16 Respond.test.S.w3 <- windows.3[,4]
17 testDataconfusion.test_S_w3 <- confusionMatrix(data = logistic.predict.test.S.w3
   , reference=Respond.test.S.w3 )
18 logistic.predict.train.S.w3 <- windows.3[,3]
19 Respond.train.S.w3 <- windows.3[,4]
20 testDataconfusion.test_S_w3 <- confusionMatrix(data = logistic.predict.train.S.w3
   , reference=Respond.train.S.w3 )
21 testDataconfusion.train_H_w3 <- confusionMatrix(data = logistic.predict.train.S.
   w3, reference=Respond.train.S.w3 )
22 accuracy.hold.out.S_w3 <- mean(c(testDataconfusion.test_S_w3$overall[1],
   testDataconfusion.train_S_w3$overall[1]))
```

```
23 |
24 | # Coil variable and predicted logistic regression class from SPSS
25 | logistic.predict.test.C.w3 <- windows.3[,5]
26 | Respond.test.C.w3 <- windows.3[,6]
27 | testdataconfusion.test_C_w3 <- confusionMatrix(data = logistic.predict.test.C.w3
28 |   , reference=Respond.test.C.w3 )
29 | logistic.predict.train.C.w3 <- windows.3[,5]
30 | Respond.train.C.w3 <- windows.3[,6]
31 | testdataconfusion.train_C_w3 <- confusionMatrix(data = logistic.predict.train.C.
32 |   w3, reference=Respond.train.C.w3 )
33 |
34 | accuracy.hold.out.C_w3 <- mean(c(testdataconfusion.test_C_w3$overall[1],
35 |   testdataconfusion.train_C_w3$overall[1]))
36 |
37 | accuracy.hold.out.H_w3
38 | accuracy.hold.out.S_w3
39 | accuracy.hold.out.C_w3
```

Listing E.1: Logistic Regression: Hold-Out Cross Validation

# Appendix F

## Logistic Regression: 5-fold Cross Validation

```
1 library(caret)
2 windows_31 <- read.delim("U:/cross validation excel result/New folder/windows_
   31.txt")
3 windows_32 <- read.delim("U:/cross validation excel result/New folder/windows_
   32.txt")
4 windows_33 <- read.delim("U:/cross validation excel result/New folder/windows_
   33.txt")
5 windows_34 <- read.delim("U:/cross validation excel result/New folder/windows_
   34.txt")
6 windows_35 <- read.delim("U:/cross validation excel result/New folder/windows_
   35.txt")
7 #logistic regression result win.3
8 # Helix variable and predicted logistic regression class from SPSS
9 logistic.predict.H.w31 <- windows_31[,1]
10 Respond.H.w31 <- windows_31[,4]
11 testDataconfusion_H_w31 <- confusionMatrix(data = logistic.predict.H.w31,
   reference=Respond.H.w31)
12
13 logistic.predict.H.w32 <- windows_32[,1]
14 Respond.H.w32 <- windows_32[,4]
15 testDataconfusion_H_w32 <- confusionMatrix(data = logistic.predict.H.w32,
   reference=Respond.H.w32)
16
17 logistic.predict.H.w33 <- windows_33[,1]
18 Respond.H.w33 <- windows_33[,4]
19 testDataconfusion_H_w33 <- confusionMatrix(data = logistic.predict.H.w33,
   reference=Respond.H.w33)
20
21 logistic.predict.H.w34 <- windows_34[,1]
22 Respond.H.w34 <- windows_34[,4]
```

```
23 testdataconfusion_H_w34 <- confusionMatrix(data = logistic.predict.H.w34,  
      reference=Respond.H.w34)  
24  
25 logistic.predict.H.w35 <- windows_35[,1]  
26 Respond.H.w35 <- windows_35[,4]  
27 testdataconfusion_H_w35 <- confusionMatrix(data = logistic.predict.H.w35,  
      reference=Respond.H.w35)  
28  
29 thedata.H.accuracy_w3 <- c(testdataconfusion_H_w31$overall[1], testdataconfusion  
      _H_w32$overall[1], testdataconfusion_H_w33$overall[1], testdataconfusion_H_  
      w34$overall[1], testdataconfusion_H_w35$overall[1])  
30 accuracy.H.cv_w3 <- mean(thedata.H.accuracy_w3)  
31 accuracy.H.cv_w3
```

Listing F.1: Logistic Regression: 5-fold Cross Validation

# Appendix G

## Confidence Interval: R Code

```
1 all.result <- read.csv("C:/temp/Data/Thesis code/new result hold out/only sheet.
  csv")
2
3 the.mean <- apply(all.result, 2, mean)
4 the.sd <- apply(all.result, 2, sd)
5 the.min <- apply(all.result, 2, min)
6 the.max <- apply(all.result, 2, max)
7 themin <- min(all.result)
8 themax <- max(all.result)
9 plot(the.mean, type="p", pch=16, xaxt="n", , xlab="Classification accuracy
  approaches", ylab="Confidence intervals", ylim=c(themin, themax), col="blue")
10 lines(the.mean, lwd=2, lty="solid", col="blue")
11 y <- c(1:6)
12 arrows(the.mean+1.96*the.sd, y, the.mean-1.96*the.sd, y, code = 3, angle = 90,
  length = 0.1, col="blue")
13 arrows(y, the.min, y, the.max, code = 3, angle = 90, length = 0.1, col="blue")
14 axis(1, at = y, labels=c("NB ", "LR", "CT", "NB CV", "LR CV", "CT CV"), cex.axis
  =0.8)
```

Listing G.1: Confidence Intervals: R Code