

A COMPUTATIONAL ENVIRONMENT FOR DATA PREPROCESSING IN
SUPERVISED CLASSIFICATION

By

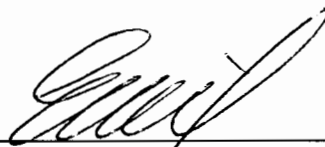
Caroline K. Rodríguez

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

Master of Science
in
Scientific Computation

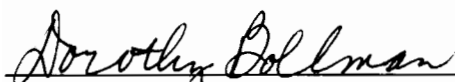
UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
July, 2004

Approved by:



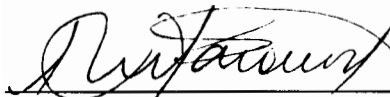
Edgar Acuña, Ph.D.
President, Graduate Committee

7/9/04
Date




Dorothy Bollman, Ph.D.
Member, Graduate Committee

7/9/04
Date



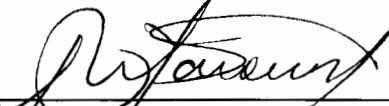
Pedro Vázquez, D.Sc.
Member, Graduate Committee

7/13/04
Date



Agustín Rullán, Ph. D.
Representative of Graduate Studies

7/9/04
Date



Pedro Vázquez, D.Sc.
Department Head

7/13/04
Date

Abstract

In this thesis, a data preprocessing environment has been created, for use in a supervised classification context, with the Windows platform of the *R* programming language and environment for statistical computing and graphics.. The functions that compose the environment have been selected based on the results of empirical studies on the effects of the data preprocessing techniques investigated on the misclassification error of well-known classifiers used on real datasets. Visualization techniques were also included in the environment to support data exploration, as well as data preprocessing decisions. The techniques considered in this thesis were applied to twelve real datasets found at the *Machine Learning Database Repository at the University of California, Irvine*. The datasets varied in the number of dimensions from 4 to 60, in the number of observations from 150 to 4435, and in the number of classes from 3 to 7. Other existing studies on data preprocessing study the effects of applying these techniques to the whole dataset, but not by class.

The functions that form the data preprocessing environment were placed in a package that can be downloaded to the *R* directory `R_HOME/library` and then, loaded to the user's workspace to create a data preprocessing environment for supervised classification applications. Future investigations may explore the use of these functions for data mining projects that involve very-high dimensional and very large datasets.

Resumen

En esta tesis, se ha creado un ambiente de pre-procesamiento de datos para usarse en aplicaciones de clasificación supervisada para la plataforma de *Windows* del lenguaje de programación y ambiente estadístico y gráfico llamado **R**. Las funciones que componen el ambiente han sido seleccionadas en base a los resultados de estudios empíricos sobre el efecto del pre-procesamiento de datos en el error de la mala clasificación de tres clasificadores muy conocidos. Las doce bases de datos usadas, cuyas dimensionalidades varían de 4 a 60, número de observaciones de 150 a 4435 y número de clases de 3 a 7, fueron tomadas del *Machine Learning Database Repository at the University of California, Irvine*. Otros estudios existen en el área de pre-procesamiento de datos, pero aplican las técnicas mencionadas a datos completos y no a los datos agrupados por clase.

Las funciones codificadas han sido empaquetadas y el paquete puede ser bajado al directorio de **R** “R_HOME/library”. Una vez ahí, el usuario puede montar el paquete en su “workspace”, creando así un ambiente propicio para el pre-procesamiento de datos para aplicaciones de clasificación supervisada. Investigaciones futuras podrán explorar el uso de estas funciones para proyectos de minería de datos.

©Copyright by Caroline Kay Rodriguez on June 2004

Dedication

"If I have seen further it is by standing on the shoulders of giants."

- Isaac Newton

This work is dedicated to the two *giants* who, over this past year, have lent me a shoulder to stand on. It was only with one foot on the shoulder of knowledge and the other on the shoulder of love that this work was made possible.

... to my thesis advisor

... to my husband and children

Acknowledgements

I would like to thank the following organizations and institutions for their financial support during the last year and a half. Without the aid provided, the completion of this work would have been improbable.

- The Office of Naval Research (ONR) – (grant number: N00014-03-1-0359)
- PRECISE group of the Engineering School at the University of Puerto Rico at Mayagüez funded by NSF grant EIA 99-77071
- The Mathematics Department of the University of Puerto Rico at Mayagüez

I would also like to thank the emotional support and comradeship provided by the Math Department administrative staff, faculty members and the members of the CASTLE group. Your faith in my ability to complete this endeavor has made this a rewarding research experience.

Table of Contents

<i>Chapter 1</i>	<i>Introduction</i>	1
<i>Chapter 2</i>	<i>A Brief Review of Data preprocessing</i>	3
2.1	Introduction	3
2.2	Motivation for applying data preprocessing	4
2.2.1	<i>Data Characterization</i>	5
2.2.2	<i>Data Visualization</i>	6
2.3	Techniques for Data Preprocessing	6
2.3.1	<i>Data cleaning</i>	7
2.3.2	<i>Data Integration</i>	7
2.3.3	<i>Data Transformation</i>	8
2.3.4	<i>Data Reduction</i>	8
2.4	Aspects of data preprocessing studied in this thesis	9
<i>Chapter 3</i>	<i>Visualization</i>	10
3.1.	Introduction	10
3.2.	Overview of some high-dimensional data visualizations	12
3.2.1.	<i>Scatter Plots</i>	12
3.2.2.	<i>Survey Plots</i>	15
3.2.3.	<i>Parallel Coordinate Plot</i>	16
3.2.4.	<i>Radial Coordinate Visualization (RADVIZ)</i>	17
3.2.5.	<i>Grand Tours</i>	18
3.3.	Current visualization techniques available for R	18
3.4.	New visualization functions for R	19
3.4.1.	<i>An implementation in R of the survey plot – surveyplot()</i>	20
3.4.2.	<i>An implementation in R of the parallel coordinate plot – parallelplot()</i>	24
3.5.	Limitations and future work	26
<i>Chapter 4</i>	<i>Outlier Detection</i>	30
4.1.	Introduction	30
4.2.	Univariate Outliers	31
4.3.	Multivariate Outliers	32
4.3.1.	Statistical based outlier detection	34
4.3.2.	<i>Detection of outliers using clustering</i>	42
4.3.3.	<i>Distance based outlier detection</i>	44
4.3.4.	<i>Density-based local outliers</i>	47
<i>Chapter 5</i>	<i>Feature Selection</i>	54
5.1.	Introduction	54
5.2.	Filter methods	55
5.2.1.	<i>The RELIEF Algorithm</i>	55
5.2.2.	<i>The Las Vegas Filter (LVF)</i>	59
5.2.3.	<i>The FINCO method</i>	63
5.3.	Wrapper methods	64
5.3.1.	<i>Sequential Forward selection</i>	65
5.3.2.	<i>Sequential Floating Forward Selection</i>	67
<i>Chapter 6</i>	<i>Missing Values</i>	72
6.1.	Introduction	72

6.2.	Mechanisms that lead to missing data	73
6.3.	Methods of handling missing data	77
6.3.1.	Case Deletion	78
6.3.2.	Mean Imputation	78
6.3.3.	Median Imputation (MDI)	79
6.3.4.	KNN Imputation (KNNI)	80
6.4.	Other imputations methods	82
6.5.	Effect of imputation on the misclassification error rate	83
Chapter 7	<i>Experimental Results</i>	87
7.1.	Programming in R	87
7.2.	The Datasets	89
7.3.	Classifiers used in this thesis	89
7.5.	Applications of Visualization Techniques	96
7.6.	The effect of outliers on the misclassification error and their treatment in a supervised classification context	105
7.7.	The effect on the misclassification error rate for both filters and wrappers	109
7.8.	Effect of imputation on the misclassification error of the contaminated real datasets	112
Chapter 8	<i>Conclusions and Future Projections</i>	117
8.1	Conclusions	117
8.2.	Future Projections	118
Chapter 9	<i>Appendix</i>	120
Chapter 10	<i>References</i>	156

List of Figures

FIGURE 3.1: 2D SCATTER PLOT FOR THE <i>IRIS</i> DATASET.....	13
FIGURE 3.2: 3D SCATTER PLOT OF <i>IRIS</i> DATA	14
FIGURE 3.3 MATRIX OF SCATTER PLOTS FOR <i>IRIS</i> DATA	14
FIGURE 3.4 SURVEY PLOT OF THE <i>CAR</i> DATASET SORTED BY <i>CYLINDERS</i> AND <i>MPG</i> (FAYYAD ET AL, 2002)....	15
FIGURE 3.5: PARALLEL COORDINATE PLOT OF <i>IRIS</i> DATASET TAKEN FROM GRINSTEIN ET AL, (2002)	16
FIGURE 3.6: RADVIZ PLOT OF <i>IRIS</i> DATASET.....	18
FIGURE 3.7: SURVEY PLOT OF <i>IRIS</i> DATASET PRODUCED IN R BY A CALL TO SURVEYPLOT().....	22
FIGURE 3.8: SURVEY PLOT FOR <i>CARS</i> DATASET PRODUCED IN R BY A CALL TO SURVEYPLOT()	23
FIGURE 3.9: SURVEY PLOT OF <i>IRIS</i> DATASET FOR WHICH SEVERAL OBSERVATIONS HAVE BEEN EMPHASIZED.	23
FIGURE 3.10: PARALLEL COORDINATE PLOT FOR <i>IRIS</i> DATASET.....	25
FIGURE 3.11: DISTINCT PERMUTATIONS FOR THE <i>IRIS</i> DATASET AS PRODUCED BY THE PARALLEL PLOT() FUNCTION	26
FIGURE 3.12: PLOT THAT REFLECTS THE “CLUTTER PROBLEM” FOR LARGE NUMBER OF ATTRIBUTES	27
FIGURE 3.13: DISPLAY DEPICTING THE “CROSSOVER PROBLEM”	28
FIGURE 4.1: OUTLIERS OF THE FEATURES IN CLASS 1 OF THE <i>IRIS</i> DATASET	33
FIGURE 4.2: OUTLIERS OF THE FEATURES IN CLASS 2 OF THE <i>IRIS</i> DATASET	33
FIGURE 4.3: OUTLIERS OF THE FEATURES IN CLASS 3 OF THE <i>IRIS</i> DATASET.	34
FIGURE 4.4: EXAMPLE OF A BI-DIMENSIONAL OUTLIER THAT IS NOT AN OUTLIER IN EITHER OF ITS PROJECTIONS.	35
FIGURE 4.5: DETECTING MULTIVARIATE OUTLIERS BY BOXPLOTS IN THE <i>IRIS</i> DATASET	36
FIGURE 4.6: THE MASKING EFFECT OF MULTIVARIATE OUTLIERS IN THE HAWKINS DATASET.....	37
FIGURE 4.7: PLOT OF THE INSTANCES OF THE <i>IRIS</i> DATASET RANKED BY THEIR MAHALANOBIS DISTANCE USING MVE ESTIMATOR.....	41
FIGURE 4.8: PLOT OF THE INSTANCES OF <i>IRIS</i> CLASS 3, RANKED BY THEIR MAHALANOBIS DISTANCE USING MCD ESTIMATOR.....	41
FIGURE 4.9: INSTANCES OF THE CLASS 3 IN <i>IRIS</i> DATASET RANKED BY THE BAY’S ALGORITHM OUTLYINGNESS MEASURE	47
FIGURE 4.10: BAY’S ALGORITHM FOR FINDING DISTANCE-BASED OUTLIERS	46
FIGURE 4.11: EXAMPLE TO SHOW THE WEAKNESS OF THE DISTANCE-BASED METHOD TO DETECT OUTLIERS.....	48
FIGURE 4.12: THE LOF ALGORITHM.....	50
FIGURE 4.13: RUNTIME FOR THE COMPUTATION OF LOFs FOR DIFFERENT DATASETS.....	52
FIGURE 4.14: INSTANCES OF THE CLASS 3 IN <i>IRIS</i> DATASET RANKED BY THE LOF’S ALGORITHM OUTLYINGNESS MEASURE	53
FIGURE 5.1: THE RELIEF ALGORITHM.....	56
FIGURE 5.2: PLOT OF RELEVANCE WEIGHTS FOR THE FEATURES OF THE DATASET <i>IONOSPHERA</i>	59
FIGURE 5.3: THE LVF ALGORITHM	62
FIGURE 5.4: THE FINCO ALGORITHM.....	64
FIGURE 5.5: THE SFS ALGORITHM.....	66
FIGURE 5.6: THE SEQUENTIAL FLOATING FORWARD SELECTION ALGORITHM (SFFS).	68
FIGURE 5.7: PLOT OF THE LOGARITHMS OF TIME COMPUTATION OF THE WRAPPERS VERSUS THE LOGARITHM OF INSTANCES IN EACH OF THE TWELVE DATASETS.....	69
FIGURE 5.8: PLOT OF THE LOGARITHM OF THE COMPUTATION TIME OF THE WRAPPERS VERSUS THE LOGARITHM OF THE NUMBER OF FEATURES IN EACH OF THE TWELVE DATASETS.	69
FIGURE 5.9: PLOT IN LOGARITHM SCALE THAT SHOWS A LINEAR RELATIONSHIP BETWEEN THE NUMBER OF INSTANCES AND THE COMPUTER RUNNING TIME OF THE FILTER ALGORITHMS.	70

FIGURE 5.10: PLOT IN LOGARITHM SCALE THAT SHOWS THE QUADRATIC RELATIONSHIP BETWEEN THE NUMBER OF FEATURES AND THE COMPUTER RUNNING TIME OF THE FILTER ALGORITHMS.	71
FIGURE 6.1: AN EXAMPLE OF MISSING VALUES OBTAINED BY A MAR MECHANISM.	75
FIGURE 6.2: AN EXAMPLE OF MISSING VALUES OBTAINED BY THE NMAR MECHANISM.	76
FIGURE 6.3: THE KNN IMPUTATION ALGORITHM.	81
FIGURE 7.1: SCATTER PLOT OF CREDIT HISTORY DATA WITH LINEAR DISCRIMINANT LINE.	91
FIGURE 7.2: DECISION TREE FOR <i>IRIS</i> DATASET.	92
FIGURE 7.3: SCATTERPLOTS OF THE FIRST ATTRIBUTE OF <i>IRIS</i> BEFORE AND AFTER APPLYING DIFFERENT NORMALIZATION METHODS.	95
FIGURE 7.4: PARALLEL COORDINATE PLOT FOR CLASS 2 OF <i>IRIS</i>	97
FIGURE 7.5: PARALLEL COORDINATE PLOT FOR CLASS 3 OF <i>IRIS</i>	97
FIGURE 7.6: PARALLEL COORDINATE PLOT FOR CLASS 1 OF <i>BUPA</i>	98
FIGURE 7.7: PARALLEL COORDINATE PLOT FOR CLASS 2 OF <i>BUPA</i>	98
FIGURE 7.8: PARALLEL COORDINATE PLOT FOR CLASS 1 OF <i>HEART</i>	99
FIGURE 7.9: PARALLEL COORDINATE PLOT FOR CLASS 2 OF <i>HEART</i>	99
FIGURE 7.10: PARALLEL COORDINATE PLOT FOR CLASS 1 OF <i>CRX</i>	100
FIGURE 7.11: PARALLEL COORDINATE PLOT FOR CLASS 2 OF <i>CRX</i>	100
FIGURE 7.12: PARALLEL COORDINATE PLOT FOR CLASS 1 OF <i>DIABETES</i>	101
FIGURE 7.13: PARALLEL COORDINATE PLOT FOR CLASS 2 OF <i>DIABETES</i>	101
FIGURE 7.14: PARALLEL COORDINATE PLOT FOR CLASS 4 OF <i>VEHICLE</i>	102
FIGURE 7.15: SURVEY PLOT OF <i>CRX</i> DATASET BEFORE FEATURE SELECTION SORTED BY THE SECOND ATTRIBUTE.	103
FIGURE 7.16: <i>CRX</i> SORTED BY V9.	104
FIGURE 7.17: <i>CRX</i> SORTED BY V10.	104
FIGURE 7.18: PERCENTAGE OF FEATURES SELECTED BY THE WRAPPER METHODS.	110
FIGURE 7.19: PERCENTAGE OF FEATURES SELECTED BY THE FILTER METHODS.	110

List of Tables

TABLE 3-1: DESCRIPTION OF VISUALIZATION FUNCTIONS CURRENTLY AVAILABLE IN R	20
TABLE 4-1: TOP OUTLIERS PER CLASS IN THE IRIS DATASET BY FREQUENCY AND THE OUTLYINGNESS MEASURE USING THE MVE ESTIMATOR	40
TABLE 4-2: TOP OUTLIERS IN CLASS 1 BY FREQUENCY AND THE OUTLYINGNESS MEASURE USING THE MCD ESTIMATOR	40
TABLE 4-3: OUTLIERS IN THE IRIS DATASET ACCORDING TO THE PAM ALGORITHM.....	44
TABLE 4.4: EXPERIMENTAL RUNNING TIME FOR COMPUTING THE LOF FOR ALL OBSERVATIONS.....	51
TABLE 6-1: DESCRIPTION OF INCOMPLETE DATASETS USED IN THIS STUDY	85
TABLE 6-2 CROSS-VALIDATION ERRORS AND RATIOS FOR THE THREE CLASSIFIERS AND THE FOUR METHODS USED TO DEAL WITH MISSING DATA	86
TABLE 7-1 : INFORMATION ABOUT THE DATASETS USED IN THESIS.	89
TABLE 7-2: CREDIT DATA FOR EXAMPLE 1.....	91
TABLE 7-3: FEATURES SELECTED BY DIFFERENT METHODS FOR <i>CRX</i>	103
TABLE 7-4: THE MISCLASSIFICATION ERROR RATE FOR THE LDA, KNN AND RPART CLASSIFIERS USING THREE DIFFERENT TYPES OF SAMPLES	106
TABLE 7-5: FEATURES SELECTED USING SFS AND RELIEF FOR THE THREE TYPE OF SAMPLES	106
TABLE 7-6: MISCLASSIFICATION ERROR RATE AFTER SFS FOR THE THREE TYPE OF SAMPLES	107
TABLE 7-7 : THE MISCLASSIFICATION ERROR RATE FOR THE LDA, KNN AND RPART CLASSIFIERS USING THREE DIFFERENT TYPES OF SAMPLES.....	108
TABLE 7-8: FEATURES SELECTED USING SFS AND RELIEF FOR THE THREE TYPE OF SAMPLES	108
TABLE 7-9 MISCLASSIFICATION ERROR RATE AFTER FEATURE SELECTION FOR THE THREE TYPE OF SAMPLES	109
TABLE 7-10: <i>MISCLASSIFICATION ERROR RATIOS OF AFTER FEATURE SELECTION TO BEFORE FEATURE SELECTION (LDA CLASSIFIER)</i>	111
TABLE 7-11: <i>MISCLASSIFICATION ERROR RATIOS OF AFTER FEATURE SELECTION TO BEFORE FEATURE SELECTION (KNN CLASSIFIER)</i>	111
TABLE 7-12: <i>MISCLASSIFICATION ERROR RATIOS OF AFTER FEATURE SELECTION TO BEFORE FEATURE SELECTION (RPART CLASSIFIER)</i>	112
TABLE 7-13: RATIOS OF CROSS VALIDATION ERRORS USING KNN.....	114
TABLE 7-14: RATIOS OF CROSS VALIDATION ERRORS USING LDA	115
TABLE 7-15: RATIOS OF CROSS VALIDATION ERRORS USING RPART	116

Chapter 1 Introduction

Data mining, sometimes called knowledge discovery, is the process of analyzing data using statistical techniques and knowledge-based methods to extract meaningful patterns from large datasets and turn these into useful information. During a data mining process, different techniques are applied to find the patterns, associations or relationships among the variables or attributes of the dataset, which can be converted into knowledge about historical patterns and future trends. Generally any of four types of relationships are sought: classes, clusters, associations and sequential patterns. Data mining techniques that search for class and cluster relationships fall under the category of Classification.

Classification refers to the data mining problem of attempting to predict the category to which each observation of the dataset belongs by building a model based on some predictor variables or features. Classification methods can be of two types: supervised or unsupervised. In supervised classification, one feature of the dataset contains values that represent a predetermined grouping of the data. These groups are generally called *classes*. For unsupervised classification the goal is to divide the observations of the dataset into groups or clusters based on some logical relationship that exists among the values of the features but that must yet be discovered. Classification may be considered the most well studied data mining problem.

The datasets used for current data mining projects are highly susceptible to anomalies and impurities, sometimes referred to as noise. If this dirty data is used for data analysis, the conclusions drawn from the process may be worthless. Since the use of noisy data for data analysis is not recommended, a wide range of methods have

been developed with the intention of eliminating the noise. This practice, known as data preprocessing, is now considered a necessary step in the data mining process.

This work uses the results of empirical studies to create a data preprocessing environment to be used within the **R** statistical analysis software. The main contribution of this work will be a library of data preprocessing functions that may be downloaded and installed to the user's **R** workspace to create an environment in which data may be cleaned and visualized, missing values imputed, outliers detected and dimensionality reduced before the data is analyzed in a supervised learning context. The real datasets used have been taken from the *Machine Learning Database Repository at the University of California, Irvine*. The electronic address of the site is <http://www.ics.uci.edu/~mlearn/MLRepository.html>. The functions to create the data preprocessing environment can be downloaded from <http://academic.uprm.edu/~eacuna/softw.htm>.

To create the environment, decompress the package in the **R** directory "R\src\library". Then, open **R**, and choose the *load* option from the *package* menu. Select the library *drep* from the list of available libraries. The functions have been created and tested for version 1.8.1 of **R**.

Chapter 2 A Brief Review of Data preprocessing

2.1 Introduction

Data mining is a class of database applications that looks for hidden patterns that can be used to predict future behavior in a group of data. Azzopardi (2002) breaks the data mining process into five stages:

- *Selecting the domain* - Data mining should be assessed to determine whether there is a viable solution to the problem at hand and a set of objectives should be defined to characterize these problems.
- *Selecting the target data* - This entails the selection of data that is to be used in the specified domain; for example, selection of subsets of features or data samples from larger databases.
- *Preprocessing the data* – This phase is primarily aimed at preparing the data in a suitable and useable format, so that a knowledge extraction process can be applied.
- *Extracting the knowledge/information* – During this stage the types of data mining operations (classification, regression, segmentation or clustering, etc), the data mining techniques, and data mining algorithms are chosen and the data is then mined.
- *Interpretation and evaluation* – This stage of the data mining process is the interpretation and evaluation of the discoveries made. It includes filtering information that is to be presented, visualizing graphically or locating the

useful patterns and translating the patterns discovered into an understandable form.

In the process of data mining many patterns are found in the data. Patterns that are interesting for the miner are those that are easily understood, valid, potentially useful, and novel. (Fayyad, 1996) These patterns should validate a hypothesis that the user seeks to confirm. The quality of patterns obtained depends on the quality of the analyzed data. It is a common practice to prepare data before applying traditional data mining techniques such as: statistical analysis, clustering, and supervised classification.

Section two of this chapter provides a more precise justification for the use of data preprocessing techniques. This is followed by a description in section three of some of the data preprocessing techniques currently in use as well as those used in this work.

2.2 Motivation for applying data preprocessing

Pyle (1999) suggests that about 60% of the total time required to complete a data mining project should be spent on data preparation since it is one of the most important contributors to the success of the project. Transforming the data at hand into a format appropriate for knowledge extraction has a significant influence on the final models generated, as well as on the amount and quality of the knowledge discovered during the process (Engels, 1998). At the same time, the effect caused by changes made to a dataset during data preprocessing can either facilitate or complicate even further the knowledge discovery process, thus changes made must be selected with care.

Today's real-world datasets are highly susceptible to noise, missing and inconsistent data due to human errors, mechanical failures and to their typically large size. Common error existence rates are estimated to be at 5% (Muller, 2003). Data affected in this manner is known as "dirty". During the past decades, a number of

techniques have been developed to preprocess data gathered from real world applications before the data is further processed for other purposes.

Cases where data mining techniques are applied directly to raw data without any kind of data preprocessing are still frequent; yet, data preprocessing has been recommended as an obligatory step. Data preprocessing techniques should never be applied blindly to a dataset, however. Prior to any data preprocessing effort, the dataset should be explored and characterized. Two methods for exploring the data prior to preprocessing are *data characterization* and *data visualization*.

2.2.1 Data Characterization

Data characterization describes data in ways that are useful to the miner and begins the process of understanding what is in the data. Engels (1998) describes the following characteristics as standard for a given dataset: the number of classes, the number of observations, the number of attributes, the number of features with numeric data type and the number of features with symbolic data type. These characteristics can provide a first indication of the complexity of the problem being studied.

In addition to the above mentioned characteristics, parameters of location and dispersion can be calculated as single dimensional measurements that describe the dataset. Location parameters are measurements such as minimum, maximum, arithmetic mean, median, and empirical quartiles. On the other hand, dispersion parameters such as range, standard deviation, and quartile deviation, provide measurements that indicate the dispersion of values of the feature.

Location and dispersion parameters can be divided in two classes: those that can deal with extreme values and those that are sensitive to them. A parameter that can deal well with extreme values is called robust. Some statistical software packages provide for the computation of robust parameters in addition to the

traditional non-robust parameters. Comparing robust and non-robust parameter values can provide insight to the existence of outliers during the data characterization phase.

2.2.2 Data Visualization

Visualization techniques can also be of assistance during this exploration and characterization phase. Visualizing the data before preprocessing it can improve the understanding of the data, thereby, increasing the likelihood that new and useful information will be gained from the data. Visualization techniques can be used to identify the existence of missing values, and outliers, as well as to identify relationships among attributes. These techniques can, in effect, assist in ranking the “impurity” of the data and in selecting the most appropriate data preprocessing techniques to apply.

2.3 Techniques for Data Preprocessing

Applying the correct data preprocessing techniques can improve the quality of the data, thereby helping to improve the accuracy and efficiency of the subsequent mining process (Han, 2000). Pyle (1999) and Azzopardi (2002) present descriptions of common techniques for preparing data for analysis. The techniques described by both authors can be summarized as follows:

- *Data cleaning* – filling in missing values, smoothing noisy data, removing outliers and resolving inconsistencies.
- *Data reduction* – reducing the volume of data (but preserving the patterns) by removing repeated observations and applying feature selection techniques.

- *Data transformation* – converting text and graphical data to a format which can be processed, normalizing or scaling the data, aggregation, generalization.
- *Data integration* – correcting differences in coding schemes due to the combining of several sources of data.

2.3.1 *Data cleaning*

Data cleaning provides methods to deal with dirty data. Since dirty datasets can cause problems for data exploration and analysis, data cleaning techniques have been developed to clean data by filling in missing values (value imputation), smoothing noisy data, identifying and/or removing outliers, and resolving inconsistencies.

Noise is a random error or variance in a measured feature (Han, 2000). Given a numeric attribute data, several methods can be applied to remove the noise. Binning methods smooth a sorted data value by consulting the neighborhood or values around it. Data can also be smoothed by using regression to find a mathematical equation to fit the data. Smoothing methods that involve *discretization* are also methods of data reduction since they reduce the number of distinct values per attribute. Clustering methods can also be used to remove noise by detecting outliers.

2.3.2 *Data Integration*

Some studies require the integration of multiple databases, or files. This process is known as *data integration*. Since attributes representing a given concept may have different names in different databases, care must be taken to avoid causing inconsistencies and redundancies in the data. Inconsistencies are observations that have the same values for each of the attributes but that are assigned to different

classes. Redundant observations are observations that contain the same information. Attributes that have been derived or inferred from others may create redundancy problems. Again, having a large amount of redundant and inconsistent data may slow down the knowledge discovery process for a given dataset.

2.3.3 *Data Transformation*

Many data mining algorithms provide better results if the data has been normalized or scaled to a specific range before these algorithms are applied. The use of normalization techniques is justified by the fact that if attributes are left unnormalized and distance-based algorithms are applied, the distance measurements taken on by attributes that assume many values will generally outweigh distance measurements taken by attributes that assume fewer values (Han, 2000). Other methods of *data transformation* include data aggregation and generalization techniques. These methods create new attributes from existing information by applying summary operations to data or by replacing raw data by higher level concepts. For example, monthly sales data may be aggregated to compute annual sales.

2.3.4 *Data Reduction*

The increased size of current real-world datasets has led to the development of techniques that can reduce the size of the dataset without jeopardizing the data mining results. The process known as *data reduction* obtains a reduced representation of the dataset that is much smaller in volume, yet maintains the integrity of the original data (Han, 2000). This means that data mining on the reduced dataset should be more efficient yet produce similar analytical results. Strategies for data reduction include the following.

- *Dimension reduction*, where algorithms are applied to remove irrelevant, weakly relevant or redundant attributes.

- *Data compression*, where encoding mechanisms are used to obtain a reduced or compressed representation of the original data. Two common types of data compression are wavelet transforms and principal component analysis.
- *Numerosity reduction*, where the data are replaced or estimated by alternative, smaller data representations such as parametric models (which store only the model parameters instead of the actual data) or nonparametric methods such as clustering, and the use of histograms.
- *Discretization and concept hierarchy generation*, where raw data values for attributes are replaced by ranges or higher conceptual levels. For example, concept hierarchies can be used to replace a low level concept such as age, with a higher level concept such as young, middle-aged or senior. Some detail may be lost by such data generalizations.

2.4 *Aspects of data preprocessing studied in this thesis*

The main objective of this thesis is to create a data preprocessing environment to be used in a supervised classification context with the **R** statistical analysis program. Functions have been created to implement techniques for data cleaning, data reduction and data transformation. Specific techniques that have been investigated include: visualization techniques, methods for outlier detection, feature selection and missing value imputation. The functions have been programmed in the **R** programming language for the Windows platform.

Chapter 3 Visualization

3.1. Introduction

Visualization is the process of transforming information into a visual form enabling the user to observe the information. Using successful visualizations for data mining and knowledge discovery projects can reduce the time it takes to understand the underlying data, find relationships, and discover information. According to Keim (2001), three different goals of data visualization are:

- *explorative analysis*
- *confirmative analysis*
- *presentation*

In *explorative analysis*, the starting point is a set of data for which no hypothesis has yet been constructed. The process involves a search for structures and the result is a visualization of the data which provides a hypothesis about the data. In *confirmative analysis*, the starting point is a set of hypotheses about the data. The data is examined with the intention of confirming or rejecting the hypotheses. In *presentation*, an appropriate presentation technique is chosen to create a high-quality visualization of a set of fixed points. In this work, we focus on the use of visualization techniques with real datasets to create spatial representations that are conducive to explorative analysis.

Explorative Analysis

Visualization techniques can be applied during the data preprocessing phase of data mining to obtain insight into the data before classification techniques are applied. The ability to observe partitions of relevant data and navigate among “data slices” of varied detail before applying these traditional data analysis techniques is

essential for obtaining more efficient classification results. This use of visualization may improve the understanding that users have of their data, thereby, increasing the likelihood that new and useful information will be gained from the data.

The following is a possible classification of visualization techniques used for exploratory data analysis as given by Poulet (1999):

- basic techniques
- geometric techniques
- symbolic techniques
- hierarchical techniques
- 3D techniques

Basic techniques are those used in various spreadsheets: pie charts, histograms, etc. Their main advantage is comprehensibility but they can represent only simple relations between data. Techniques that perform geometric transformations and projections are called *Geometric techniques*. Examples of these techniques are 2D and 3D Scatterplot Matrices, Parallel Coordinates and Permutation Matrices, and Survey Plots. In *symbolic techniques*, the basic idea is the visualization of the data values as features of icons. Examples of these techniques are: Chernoff-faces, glyphs, and stick-figures. In the last class of visualizations techniques *hierarchical* and *3D-techniques*, we find specific visualization algorithms such as decision tree visualizations.

Data analysts need tools for creating hypotheses about large and/or high dimensional datasets since these datasets are becoming commonplace in an increasing number of applications. It is no longer unusual to have datasets with hundreds or even thousands of dimensions and hundreds of thousands of instances. Visualization tools can provide the ability to explore and understand data, allowing analysts to examine “what if” scenarios while interacting with multivariate visual displays. Constructing a visual display of the data that is useful for the researcher who works with large and/or high dimensional datasets is currently one of the most difficult tasks in visualization. For this reason, Sahling (2002) points out that the challenge for any visualization

method is to reduce the dimensionality of a dataset and create a visualization from which relevant information can be extracted without changing the characteristics of the original data.

In this thesis, some geometric techniques have been implemented for use in the Windows platform of *R* to strengthen the set of visualization tools already available for supervised classification in this statistical environment. The sections that follow describe these techniques. Section 2 presents an overview of some of the most frequently used visualization tools for high-dimensional data. Section 3 discusses the visualization tools that are currently available in *R* whereas, in Section 4 we describe the tools that we have created for the *R* environment. Finally, Section 5 describes some limitations of our plots and future directions for continued work.

3.2. Overview of some high-dimensional data visualizations

Many data visualization techniques stand out as high-dimensional visualizations because they are better able to create informative displays for data that contain a high number of attributes than the standard multidimensional displays. Yet, it is not always precisely clear what characterizes a “high-dimensional” dataset. Ferreira and Levkowitz (2003), state that the conceptual boundary between low and high-dimensional data is around three to four data attributes. They suggest a general guideline for characterizing dimensionality as: low – up to four attributes, medium – five to nine attributes and high – 10 or more. We give a brief description of some techniques that are mentioned most frequently for their ability to provide information even for high-dimensional datasets.

3.2.1. Scatter Plots

A scatter plot is a point projection of the data into a 2D or 3D dimensional space represented on the screen in classic (X, Y) or (X, Y, Z) format. This is the most commonly utilized data visualization method because it is a useful exploratory method for providing a first look at relationships between pairs of attributes, clusters of points, trends, and outliers. Conventional scatter plots lose their effectiveness;

however, as the number of attributes of the dataset becomes large. The insight into the higher dimensions is rarely as good as with the standard 2D plot.

A scatter plot matrix is a tool for displaying multivariate data, in which each plot of the matrix shows the data points based on two attributes. It is the standard means for extending the scatter plot to higher dimensions since it is useful for looking at all possible two-way interactions or correlations between dimensions. For n -dimensional data this yields $\frac{p(p-1)}{2}$ scatter plots with shared scales. The power of this plot resides in the possibility of making a visual link between the features of one scatter plot with features on another.

Figures 3.1, 3.2 and 3.3 show 2D and 3D scatter plots and a scatter plot matrix of the *Iris* dataset created by using functions available in existing *R* libraries. The three graphs show an obvious formation of clusters indicating the discriminating power of *Petal length* and *Petal width*.

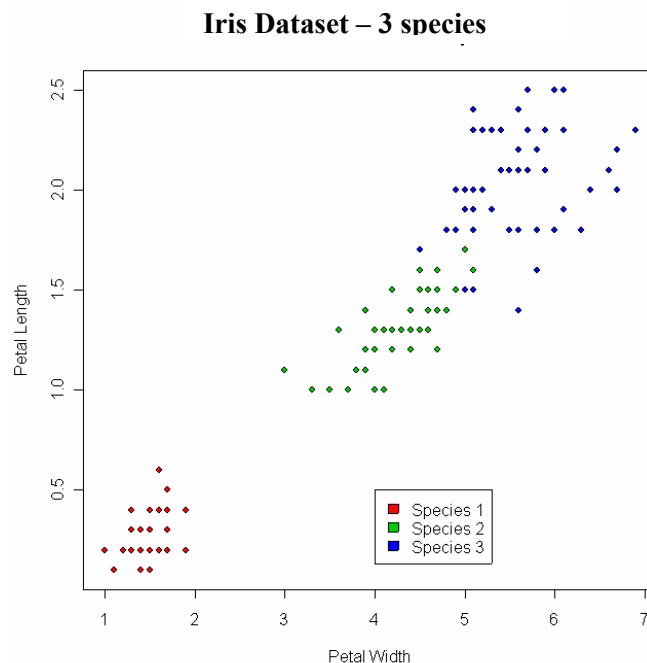


Figure 3.1: 2D scatter plot for the *Iris* dataset.

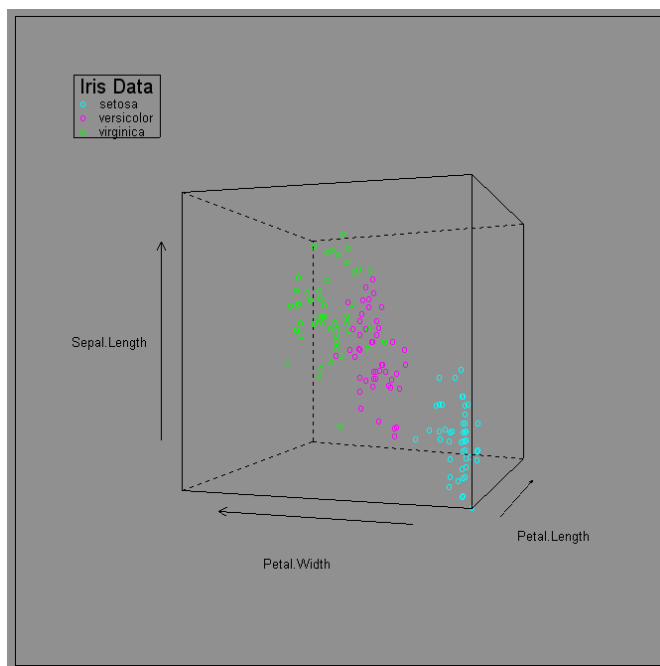


Figure 3.2: 3D scatter plot of *Iris* data

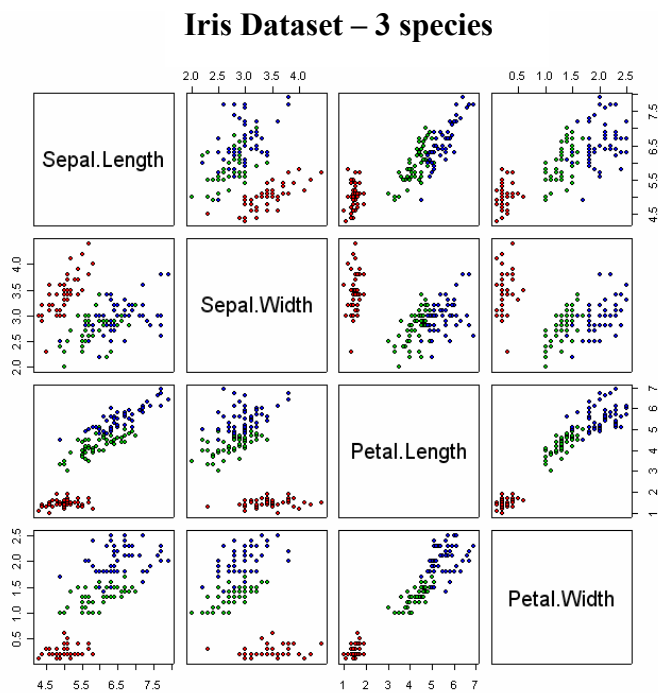


Figure 3.3 Matrix of scatter plots for *Iris* data

3.2.2. Survey Plots

A survey plot is closely related to bar graphs and the permutation matrices that were invented by a French cartographer Jaques Bertin in 1967. Survey plots usually consist of n rectangular areas or lines (depending on the number of the observations and the size of the plotting screen) – one for each dimension – that are vertically arranged in rows. Each data value of an attribute is mapped to a point on the vertical line and the point is extended to a line with length proportional to the corresponding value. The strength of this visualization lays in its ability to show the relations and dependencies between any two attributes, especially when the data is sorted on a particular dimension (Fayyad et al, 2002). Tendencies and outliers can also be extracted easily. In addition, survey plots can aid in identifying rules for classification. (Grinstein et al, 2002). Figure 3.4 presents a survey plot created for the *Car* dataset as presented in Fayyad et al. (2002).

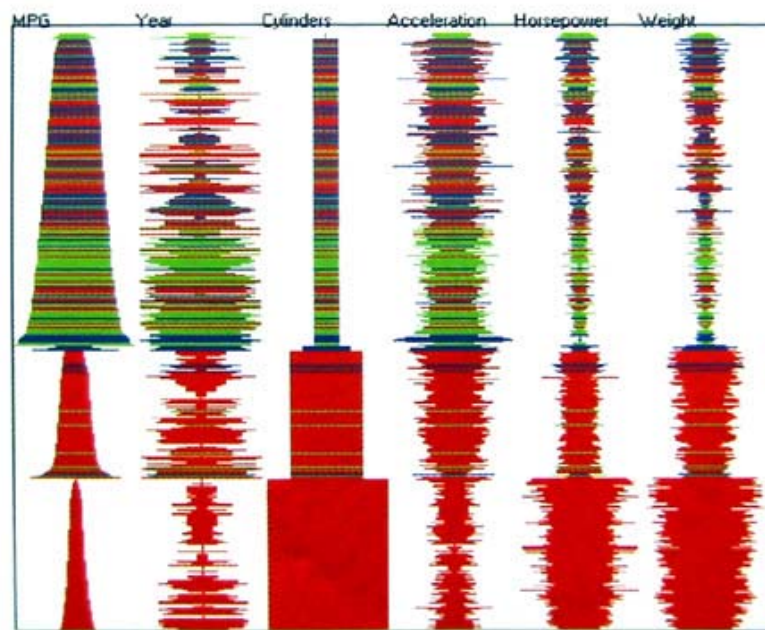


Figure 3.4 Survey plot of the *Car* dataset sorted by *cylinders* and *mpg* (Fayyad et al, 2002)

3.2.3. *Parallel Coordinate Plot*

The parallel coordinate plot, which was first described by Al Inselberg in 1985, represents multidimensional data using lines. Whereas in traditional Cartesian coordinates all axes are mutually perpendicular, in parallel coordinate plots, all axes are parallel to one another and equally spaced. In this approach, a point in m -dimensional space is represented as a series of $m-1$ line segments (Inselberg and Dimsdale, 1990) in 2-dimensional space. Thus, if the original data observation is written as (x_1, x_2, \dots, x_m) , then its parallel coordinate representation is the $m-1$ line segments connecting points $(1, x_1), (2, x_2), \dots, (m, x_m)$. Typically, continuous features will be standardized before a parallel coordinate plot is drawn.

Wegman (1990) provide a description of the highly structured mathematical nature of the transformation from Cartesian coordinates to parallel coordinates, as well as some basic facts about parallel coordinate geometry. In summary, if two attributes are highly positively correlated, lines passing from one feature to another tend not to intersect between the parallel coordinate axes. For highly negatively correlated attributes, the line segments tend to cross near a single point between the two parallel coordinate axes. Figure 3.5 presents a parallel coordinate plot of the Iris dataset as presented in Grinstein et al, (2002).

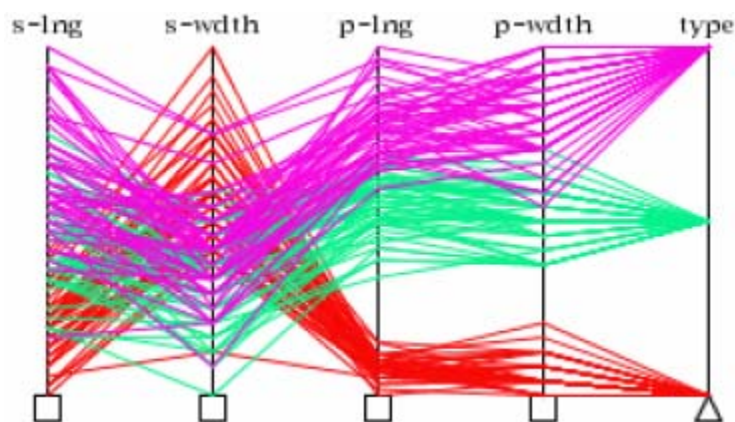


Figure 3.5: Parallel Coordinate plot of Iris dataset taken from Grinstein et al, (2002)

The following visualization techniques are described here because they are frequently used with high dimensional datasets. However, they have not been implemented in this thesis.

3.2.4. *Radial Coordinate Visualization (RADVIZ)*

Like the previous visualization technique, the RADVIZ method (Ankerst et al, 1996) maps a set of n -dimensional points onto a two dimensional space. However, in this case the mapping is not linear. This technique applies the idea of using spring constants to represent relational values between points. In the RADVIZ implementation, n -dimensions are laid out as points equally spaced around the perimeter of a circle. One end of n springs is attached to the n perimeter points. The other ends of the springs are attached to a data point. The spring constant, k , equals the value of the particular dimension of the fixed point. For each point the position is placed where the sum of the spring forces equals 0. The data values are usually scaled to have values between 0 and 1. Many points can map to the same position.

Some features of this visualization are:

- Points where all the values of an attribute have approximately the same value lie closer to the center.
- If one or two dimensional values are greater, points will lie closer to those dimensional points.
- Where the point will lie depends on the layout of the attributes around the circle.
- Certain symmetries of the data will be preserved.

Figure 3.6 shows a RADVIZ plot for the *Iris* dataset as taken from Grinstein et al, (2002).

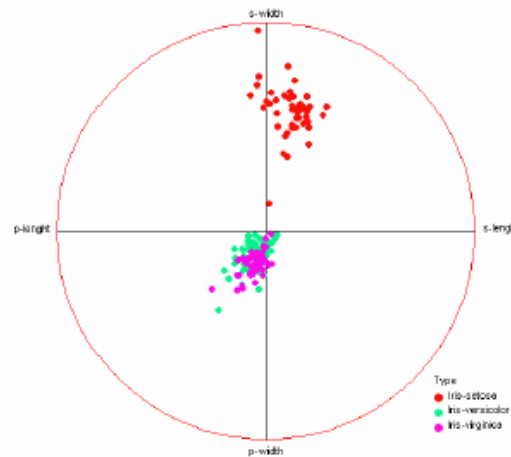


Figure 3.6: RADVIZ plot of *Iris* dataset

3.2.5. *Grand Tours*

The 2-dimensional grand tour was introduced by Asimov. A 2-dimensional grand tour is a method for viewing multidimensional data by using linear projections onto a sequence of two dimensional subspaces and then moving continuously from one projection to the next. Wegman (1993) worked on generalizing the plot for d -dimensions and Yang (2000) extended the work specifically to three dimensions. In all cases, the aim is to automatically aid the user in finding interesting, informative projections which are hard to find in the original data when the number of attributes is high. Yang (1999) sustains that in the data preprocessing stage of a data mining project, grand tours are efficient ways to examine the distribution of values of each feature, the correlations among features, and to decide which features should be included in further analysis. Wegman and Carr (1993) suggest coupling grand tours with parallel coordinate displays, to allow for an in-depth study of high dimensional data.

3.3. *Current visualization techniques available for R*

Several packages currently available in R include visualization functions. Table 3.1 lists several current R packages and the visualization functions they contain. The

list is not meant to be exhaustive but to provide a general idea of the options that are currently available to R users. At the same time, the differences between the visualizations provided by these functions and the functions designed as part of this work are emphasized to highlight the need to gather a useful set of visualization functions into one package.

Other visualization tools that can be embedded into the R environment are currently available. A well known example is GGobi (Temple et al., 2001). GGobi, a direct descendant of XGobi (Swayne et al., 1991), is a data visualization system with interactive dynamic methods for the manipulation of views of data. It offers 2D displays of projections of points and lines in high-dimensional spaces, such as scatterplots, parallel coordinate plots, scatterplot matrices and time series plots. GGobi runs on Linux systems as well as under the Microsoft Windows and Macintosh OS operating system. GGobi is a stand-alone application that has been constructed as a programming library to provide visualization functionality that can be embedded within other applications either through language bindings or plugins.

3.4. New visualization functions for R

Two visualization functions that work well with datasets containing up to 20 features (high-dimensional) are the parallel coordinate and survey plots. Functions to implement these techniques have been included in the data preprocessing environment proposed in this thesis. These visualizations have been chosen to be included in the environment we propose because they provide support for formulating hypotheses, identifying clusters, outlier detection, determining feature relevance, as well as for rule or pattern detection. The plots that are produced by these functions are low-resolution graphics that provide not only an automated means for visualizing a dataset but, also, provide several ways to modify the displays and increase the information that can be obtained.

Table 3-1: Description of visualization functions currently available in R

R package	Function name	Visualization technique	Description
MASS	parcoord()	parallel coordinates	Creates a parallel coordinate plot for an input dataset with the column of classes removed on a white background. Order of features of the dataset must be arranged externally before calling the plotting function. Individual observations cannot be highlighted.
lattice	cloud()	3D scatter plot	Creates a 3D scatter plot based on a specified formula for the indicated dataset on a gray background.
lattice	splom()	scatter plot matrix	Creates a scatter plot matrix based on a specified formula for the indicated dataset on a gray background.
lattice	parallel()	parallel coordinates	Creates a parallel coordinate plot for input data on a gray background. The class column is used to divide the plot into panels, one for each class. The order of features of the dataset must be arranged externally before calling the plotting function. Individual observations cannot be highlighted.
base	plot()	scatter plot	Plot points for two columns of a dataset, one column provides the x-coordinates and the other the y-coordinates
base	pairs()	scatter plot matrix	Creates a scatter plot matrix for the specified columns of a dataset on a white background.

3.4.1. *An implementation in R of the survey plot – surveyplot()*

The survey plot is most useful at determining correlations between attributes and their relevance to classification if the data has been sorted by at least one of the attributes. This sorting option has been provided in the version of the survey plot created for this work, however, the plot is not an exact replicate of the original survey plot as presented by program Inspect (Lohninger, 1994). This original version extends a line around a center point, where the length of the line corresponds to the value of

the attribute. The survey plot shown in Figure 3.4 was created following Lohninger's (1994) design. It can be observed that each column is symmetric about its center (the left side has been flipped horizontally and repeated on the right), in this way displaying redundant information. The proposed version of the survey plot eliminates this redundancy, simplifying the computations needed to create the plot and saving space on the display screen. In this sense, this newer version of the plot is closer to a permutation matrix display (introduced by a French cartographer, Jacques Bertin) in that it allows for the observation of statistical information such as attribute means and the shape of the distribution of the whole dataset. A more complete discussion of the permutation matrix plot can be found in Schmid and Hinterberger (1994).

The R function that has been programmed for this work takes as minimum input the dataset. First the column containing the classes is removed, and then the data is standardized. Next, the screen space is distributed evenly among the number of observations and the number of attributes. Finally the horizontal lines corresponding to each normalized data value are plotted. Optional parameters to the function are: the name of the dataset, the column number of an attribute by which to order the dataset, the number of the class for which to limit the plot, and a vector containing the observation number of those observations that are to be highlighted. Survey plots of the *Iris* and *Cars* datasets that were created using this function are shown in figures 3.7 and 3.8.

In figure 3.7, the survey plot of the *Iris* dataset, the three clusters of flower types are easily observed. The plot has been sorted by *petal length* and a correlation between *petal length* and *petal width* can be observed since *petal width* tends to increase with *petal length*. It can also be seen that *petal length* and *petal width* appear to be good discriminators for this dataset, since observations in each class assume similar values (observations are grouped by colors). Several observations with extreme values can also be observed.

In Figure 3.8, the *Cars* dataset has been sorted by the third column, the number of cylinders. From this survey plot, the clustering of American cars (blue

segments) with respect to increased horsepower, weight, and cylinders can be observed. The Japanese cars (green segments) have high mpg, low weight, and smaller number of cylinders. The European cars (red segments) have more intermediate values. Again, extreme values can be observed in different classes. Some discrimination power can be observed, yet it appears that this power might be shared by more than one feature.

Figure 3.9 shows the survey plot for the *Iris* data that has been ordered by column one and in which several observations are being emphasized. If compared to Figure 3.7, a difference in the discrimination power of the first and third features can be observed.

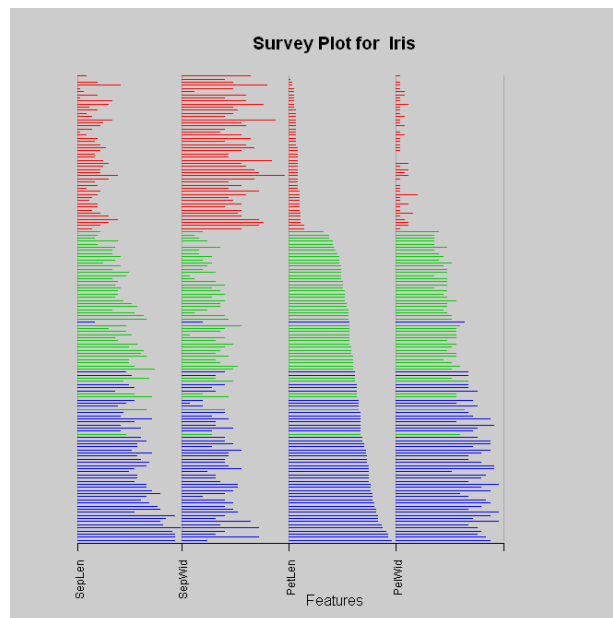


Figure 3.7: Survey Plot of *Iris* Dataset produced in R by a call to `surveyplot()`

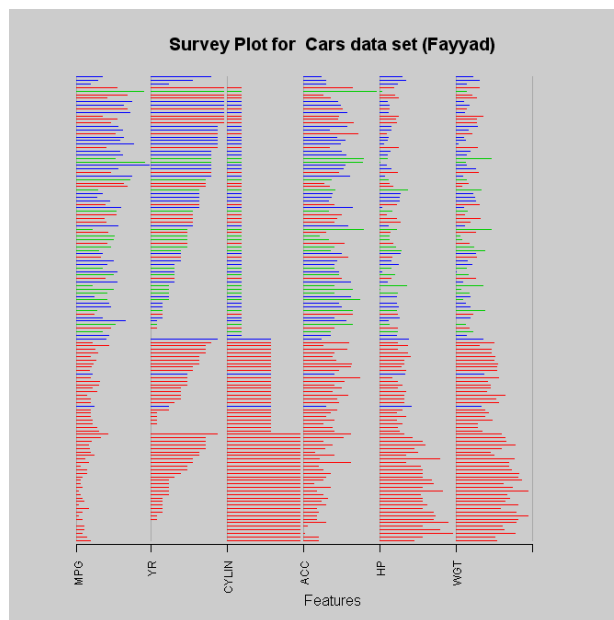


Figure 3.8: Survey plot for *Cars* dataset produced in R by a call to `surveyplot()`

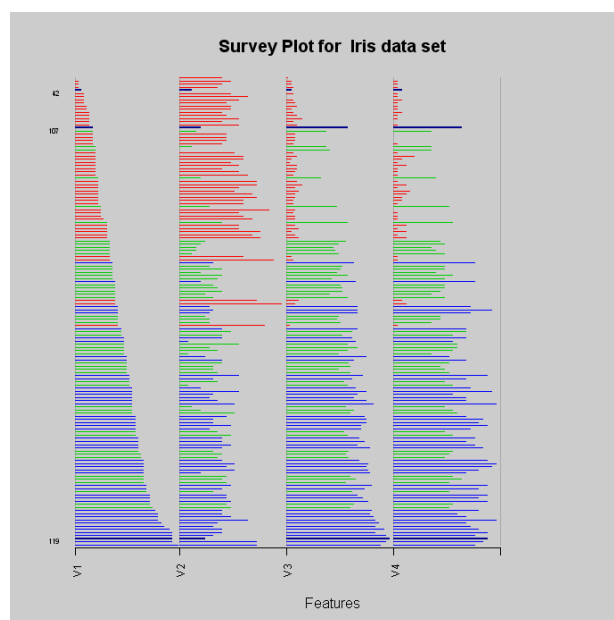


Figure 3.9: Survey plot of *Iris* dataset for which several observations have been emphasized.

3.4.2. *An implementation in R of the parallel coordinate plot – parallelplot()*

The R function designed for this work that creates a parallel coordinate plot takes as minimum input the dataset. Once again, the column containing the classes is removed, and then the data is standardized. The number of distinct permutations of the attributes is determined and a temporary matrix that stores the order of the attributes for each combination is created. If a particular combination is specified at input, then p axes are placed, where p is the number of attributes, by evenly dividing the horizontal plotting space in $p+1$ rectangular spaces and then plotting the standardized observation values on the axes for each observation. The color of the poly-line will depend on the class to which the observation belongs. If no particular combination of feature ordering is specified at input, all distinct combinations are plotted (in reduced format) on a series of panels, four plots per display. This provides the user with the opportunity to observe all combinations and select the one which is believed to provide the most information.

Optional parameters to the function are: the name of the dataset, the number of the class for which to limit the plot, and a vector containing the observation number of those observations that are to be highlighted. The high-resolution counterparts to these options would be the ability to zoom in on a particular class or observation and the ability to obtain a different view.

Figure 3.10 shows a parallel coordinate plot for the *Iris* dataset created by the function we designed for the R environment. Each different segment color represents a different class. It can be observed, once again, that the attributes *sepal width* (V3) and *sepal length* (V4) appear to be better discriminators, than the attributes *petal width* (V1) and *petal length* (V2).

Other information that is conveyed by this plot is:

- Clustering is propagated through all attributes.
- The relationship between *petal width* and *petal length* shows relatively little crossover, suggesting positive correlation. (Thin flowers tend to have small petals.)

- Several segments can be observed in groups of segments of the same color that do not follow the general pattern of the group. This suggests the presence of outliers.

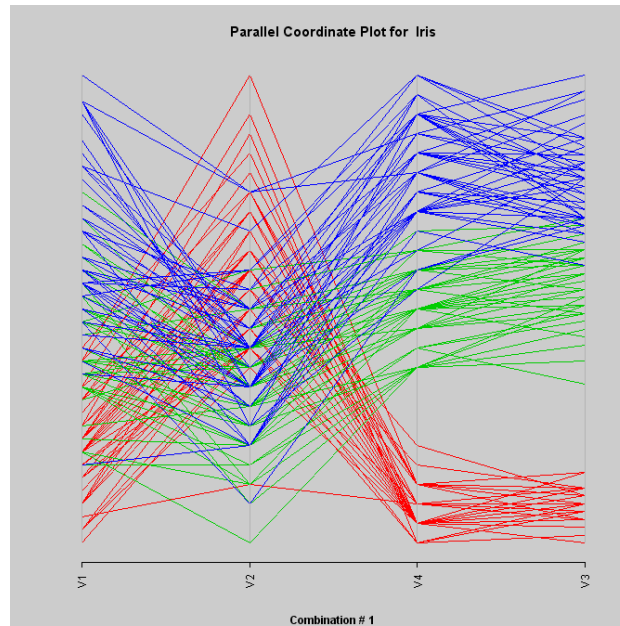


Figure 3.10: Parallel coordinate plot for *Iris* dataset.

In any parallel coordinate plot, pairwise comparison is limited to those axes that are adjacent. Therefore, theoretically one could create $n!$ permutations of the n attributes in the dataset so that in some permutation every axis is adjacent to every other axis. However, many of these are duplicate adjacencies. Wegman (1990) provides the details for determining that there are actually $\left\lceil \frac{p+1}{2} \right\rceil$ permutations required for an p -dimensional dataset, where $\lceil * \rceil$ is the greatest integer function.

Figure 3.11 shows parallel coordinate plots for the two distinct combinations of the *Iris* dataset using Wegman's results (1990) as provided by the parallel coordinate function that has been created. In the second plot, the pairwise comparisons that were not available in the plot of the first combination are V1 with V4, V2 with V3, and V1 with V3.

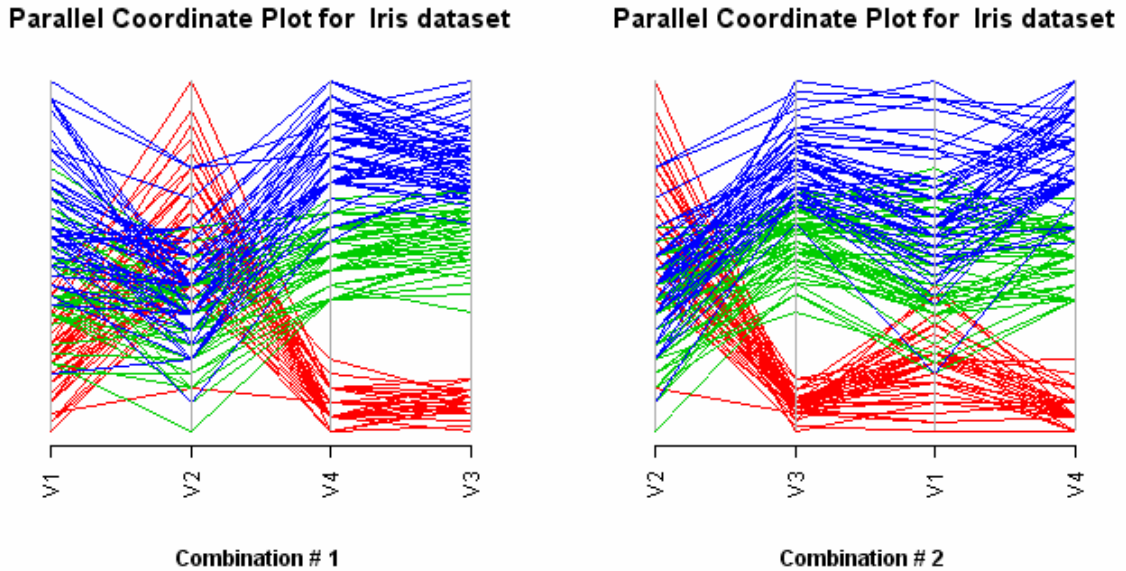


Figure 3.11: Distinct Permutations for the *Iris* dataset as produced by the `parallelplot()` function

3.5. Limitations and future work

Though these two visualizations have been found useful for exploring the data sets used in this thesis, they suffer from limitations as the number of attributes and the number of observations increases. Both graphs suffer from the “clutter problem” which occurs when individual data items can no longer be seen clearly from the display due to the large number of attributes and/or observations. When the number of attributes is over fifteen, the large number of axes needed to create these displays tends to crowd the figure, limiting the value of the plot for detecting patterns or other useful information. Even with a low number of observations, a high dimensionality presents a serious challenge for these techniques. Figure 3.12 illustrates the “clutter problem” for high number of attributes using the *Ionosphere* dataset which contains 2 classes, 32 attributes and 351 observations.

In addition to the “clutter problem”, a parallel plot may suffer from the “crossover problem”. When many of the poly-lines of the parallel coordinate plot crossover each other, following the lines that share common points on axes becomes

very difficult, if not impossible. Figure 3.13 shows the *Segmentation* dataset, which contains 7 classes, 17 attributes and 2310 observations. Individual observations are difficult to identify although it is still possible to observe patterns related to discriminating features.

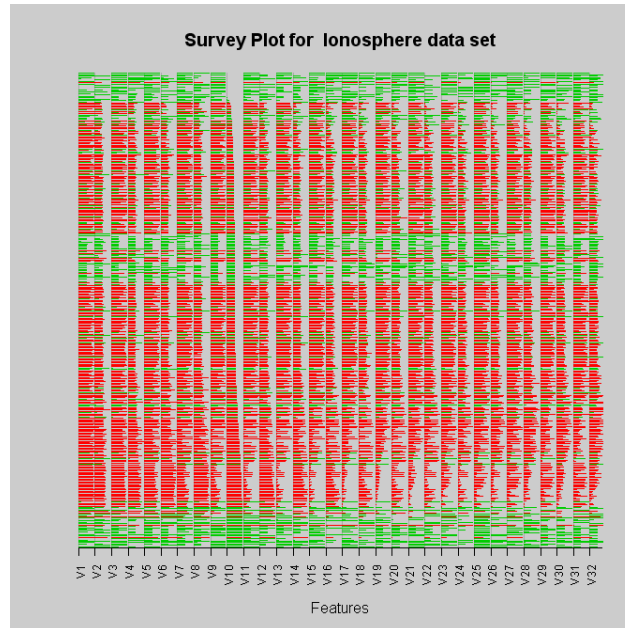


Figure 3.12: Plot that reflects the “clutter problem” for large number of attributes

One other limitation of these displays is the loss of the information that is encoded into the lines between the axes for discrete, heterogeneous data attributes. If A is a continuous attribute and B is discrete, line segments reflecting a positive slope from attribute A to attribute B , may no longer imply that the value in attribute B is higher than the value of attribute A , due to the normalization that was applied before the graph was created. For example, $x = \{1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0\}$ maps to $x = \{0, 0.125, 0.250, 0.375, 0.500, 0.625, 0.750, 0.875, 1.000\}$, whereas $z = \{0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1\}$ maps to $z = \{0, 0.5, 1.0, 0, 0.5, 1.0, 0, 0.5, 1.0, 0, 0.5\}$ under the normalization method we used before constructing the plots. It can be noticed that line segments crossing from x to z will not carry any correlation information.

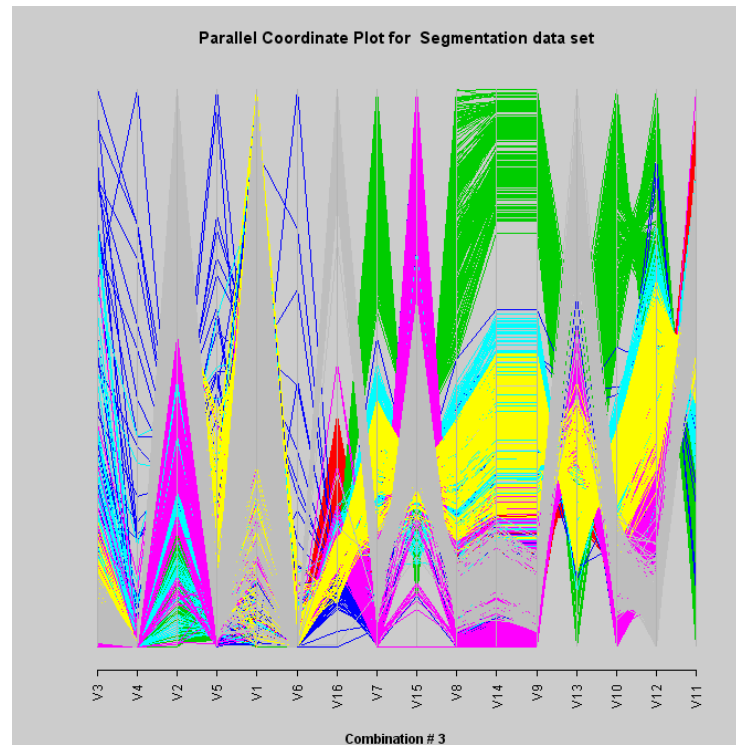


Figure 3.13: Display depicting the “crossover problem”

Much effort is being dedicated by many researchers to overcome these limitations. The authors of this work believe that the “cluttering problem” could be addressed by modifying the current functions to include the ability to “slice” the attribute set and view the complete display as a series of panel displays. A different approach could be to apply dimensionality reduction techniques before plotting the data as proposed by Yang et al (2003). Their approach combines automation and user interaction to generate a meaningful attribute subspace that can be displayed using traditional multidimensional techniques. Yang has developed a prototype of a framework that forms clusters of attributes, automatically selects a representative attribute for each cluster and then maps the high-dimensional dataset into the subspace composed of these representatives and displays the projected subspace using multidimensional visualization techniques.

Graham et al. (2003) proposes a number of refinements to the parallel coordinate graphing technique to solve the above mentioned problems. Graham suggests replacing the traditional set of poly-lines with a collection of smooth curves across the attribute axes. The curves allow the user to discern individual paths more easily. Though this increases the utility of the plot by allowing for paths to be followed more easily after crossovers occur, the plots may still become cluttered after a large number of observations.

Chapter 4 Outlier Detection

4.1. Introduction

According to Hawkins (1980), “An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism”. Almost all the studies that consider outlier identification as their primary objective are in the field of statistics. A comprehensive treatment of outliers appears in Barnett and Lewis (1994). They provide a list of about 100 discordancy tests for detecting outliers in data that follow well-known distributions. The choice of an appropriate discordancy test depends on:

- a) the distribution,
- b) the knowledge of the distribution parameters,
- c) the number of expected outliers, and
- d) the type of expected outliers.

These methods have two main drawbacks. First, almost all of them are for univariate data, making them unsuitable for multidimensional datasets. Second, all of them are distribution-based, and most of the time real-world data is multivariate with an unknown distribution.

Detecting outliers is an important data mining task. The data mining community became interested in outliers after Knorr and Ng (1998) proposed a non-parametric approach to outlier detection based on the distance of an instance to its nearest neighbors. Outlier detection has many applications such as: fraud detection, network intrusion, and data cleaning. Frequently, outliers are removed to improve the accuracy of estimators. However, this practice is not always recommendable because sometimes outliers can have very useful information. The presence of outliers can

indicate individuals or groups that exhibit a behavior that is very different from the norm. A growing practice in the data mining community is to rank the instances using an outlyingness measure rather than classifying the instances as outliers or non-outliers.

Section 2 of this chapter includes a brief discussion of the detection of outliers for univariate data. Section 3 focuses on methods for the detection of multivariate outliers. Four methods of outlier detection are considered: a method based on robust estimation of the Mahalanobis distance, a method based on the PAM algorithm for clustering, a distance-based method and a density-based method. The effect and treatment of outliers in supervised classification will be discussed in Chapter 7.

4.2. *Univariate Outliers*

Given a dataset of n observations of a feature x , let \bar{x} be the mean and let s be the standard deviation of the data distribution. It is well known that an observation of the dataset is declared as an outlier if it lies outside of the interval

$$(\bar{x} - ks, \bar{x} + ks), \quad (4.1)$$

where the value of k is usually taken as 2 or 3. The justification of these values relies on the fact that when assuming normal distribution one expects to have 95 percent of the data in the interval centered about the mean, with a radius equal to two standard deviations. Also, one expects to have all of the data inside an interval that is centered at the mean and that has a radius of three standard deviations.

From Equation 4.1, the observation x is considered an outlier if

$$\frac{|x - \bar{x}|}{s} > k. \quad (4.2)$$

The problem with the above criteria is that it assumes normal distribution of the data, something that frequently does not occur. Furthermore, the mean and standard deviation are highly sensitive to outliers themselves.

John Tukey (1977) introduced several methods for explorative data analysis, one of which was the *Boxplot*. The Boxplot is a graphical display in which the outliers appear tagged. Two types of outliers are distinguished: *mild outliers* and *extreme outliers*. An observation x is declared a *mild outlier* if it lies outside of the interval $[Q_1 - 1.5(IQR), Q_3 + 1.5(IQR)]$. The interval has a center at $(Q_1 + Q_3)/2$ and a radius of $2(IQR)$. An observation x is declared as an *extreme outlier* if it lies outside of the interval $[Q_1 - 3(IQR), Q_3 + 3(IQR)]$. Notice that the center of the interval is $(Q_1 + Q_3)/2$ with a radius of $3.5(IQR)$, where $IQR = Q_3 - Q_1$. IQR , called the *Interquartile Range*, is a robust estimator of variability which can replace s in Equation 4.1. The numbers 1.5 and 3 are chosen by comparison with a normal distribution. On the other hand $(Q_1 + Q_3)/2$ is a robust estimator of the center that can be used instead of \bar{x} in Equation 4.1.

All the major statistical softwares include boxplots among their graphical displays. Figures 4.1-4.3 show the outliers of the features in the three classes of the dataset *Iris* detected through their boxplots.

4.3. *Multivariate Outliers*

Given a dataset D with p features and n instances (in a supervised classification context) we must also know the class to which each of the instances belongs. It is very common to include the classes as the last column of the data matrix. The objective of outlier detection in supervised classification is to identify all the complete instances that seem to be unusual in each class, these will be the multivariate outliers. One might think that multivariate outliers can be detected based on the univariate outliers for each feature but, as it is shown in the Figure 4.4, this is not always true.

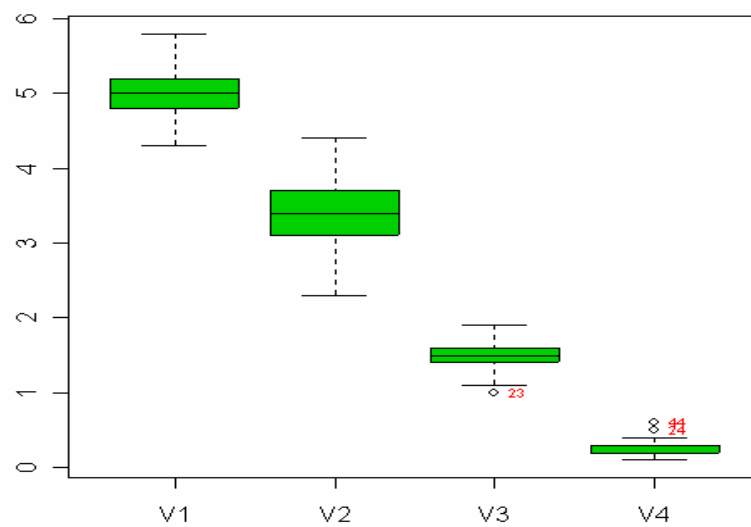


Figure 4.1: Outliers of the features in class 1 of the *Iris* dataset

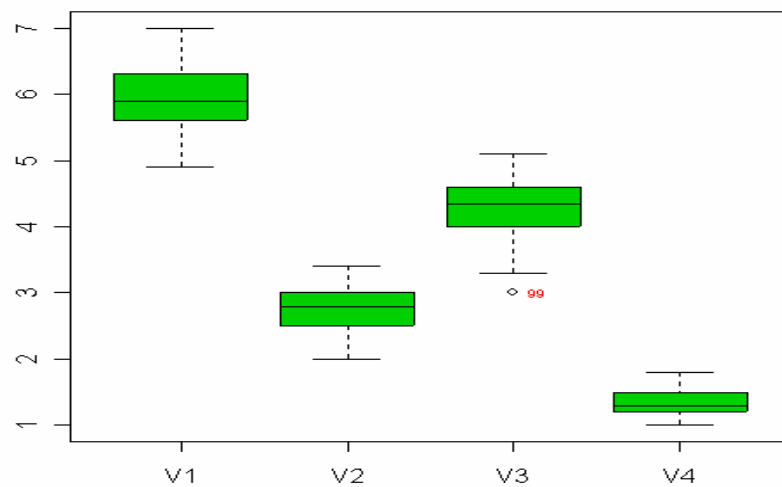


Figure 4.2: Outliers of the features in class 2 of the *Iris* dataset

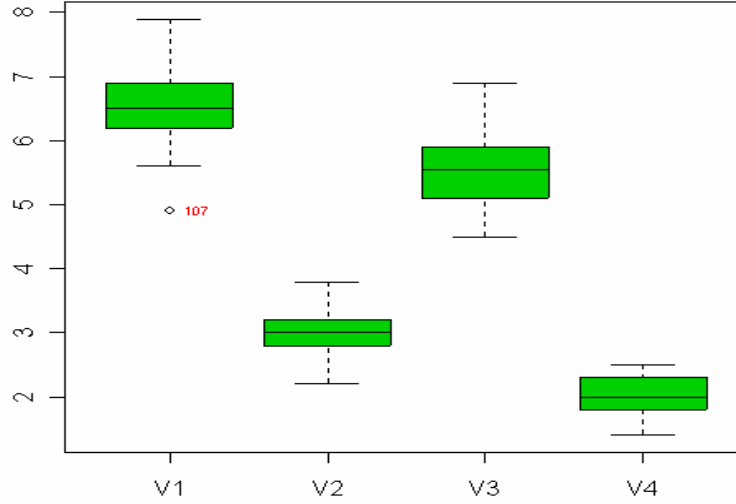


Figure 4.3: Outliers of the features in class 3 of the *Iris* dataset.

The instance appearing in the upper right corner is a multivariate outlier but it is not an outlier in either feature. On the other hand, an instance can have values that are outliers in several features and yet not be a multivariate outlier as a whole.

There are several methods for detecting multivariate outliers. The methods discussed in this thesis are: statistical-based outlier detection, outlier detection by clustering, distance-based outlier detection and density-based local outlier detection. The before mentioned methods are discussed in the next sections.

4.3.1. Statistical based outlier detection.

Let \mathbf{x} be an observation of a multivariate dataset consisting of n observations and p features. Let $\bar{\mathbf{x}}$ be the centroid of the dataset, which is a p -dimensional vector with the mean of each feature as components. Let \mathbf{X} be the matrix of the original dataset with columns centered by their means. Then the $p \times p$ matrix $\mathbf{S} = \left(\frac{1}{n-1} \right) \mathbf{X}'\mathbf{X}$ represents the covariance matrix of the p features.

The multivariate version of Equation 4.2 is

$$D^2(\mathbf{x}, \bar{\mathbf{x}}) = (\mathbf{x} - \bar{\mathbf{x}})\mathbf{S}^{-1}(\mathbf{x} - \bar{\mathbf{x}}) > k, \quad (4.3)$$

where D^2 is called the Mahalanobis square distance from \mathbf{x} to the centroid of the dataset. An observation with a large Mahalanobis distance can be considered as an outlier.

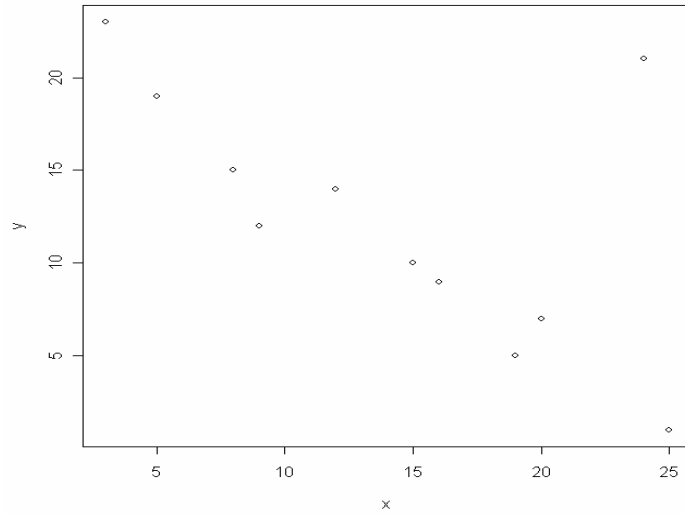


Figure 4.4: Example of a bi-dimensional outlier that is not an outlier in either of its projections.

Assuming that the data follows a multivariate normal distribution, it has been shown that the distribution of the Mahalanobis distance behaves as a Chi-Square distribution for a large number of instances. The proposed cutoff point for Equation 4.3 is given by $k = \chi^2_{(p, 1-\alpha)}$, where χ^2 stands for the Chi-Square distribution and α is a signification level, usually taken as 0.05. (Rousseeuw and Leroy, 1987)

A basic method for detecting multivariate outliers is observing the outliers that appear in the boxplot of the distribution of the Mahalanobis distance of all the instances. Rocke and Woodruff (1996) stated that the Mahalanobis distance works well identifying scattered outliers however, it may fail to detect clustered outliers.

Example 1. Find the multivariate outliers in each of the classes of the *Iris* dataset by building boxplots of the Mahalanobis distance of all of the instances.

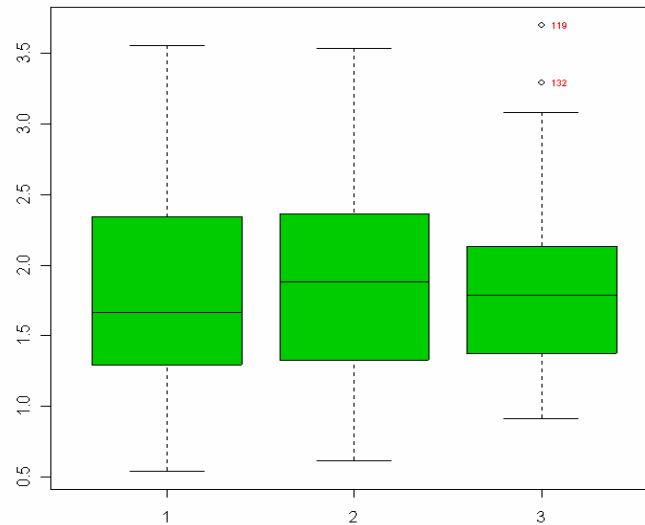


Figure 4.5: Detecting multivariate outliers by boxplots in the *Iris* dataset

Figure 4.5 shows the boxplots of the Mahalanobis distances for each feature in each class of the *Iris* dataset. Notice that only two outliers (119 and 132) are detected in class 3.

Datasets with multiple outliers or clusters of outliers are subject to the *masking* and *swamping* effects.

- **Masking effect.** It is said that an outlier masks a second one that is close by if the latter can be considered an outlier by itself, but not if it is considered along with the first one. Equivalently, after the deletion of one outlier, the other instance may emerge as an outlier. Masking occurs when a group of outlying points skews the mean and covariance estimates towards it, and the resulting distance of the outlying point from the mean is small.

- **Swamping effect.** It is said that an outlier swamps other instances if the latter can be considered as outliers only under the presence of the first one. In other words after the deletion of one outlier, the other outlier may become a “well-behaved” instance. Swamping occurs when a group of outlying instances skew the mean and covariance estimates towards it and away from other “good” instances, and the resulting distance from these “good” points to the mean is large making them look like outliers.

Example 2: Consider the dataset due to Hawkins, Bradu, and Kass (Rousseeuw and Leroy, 1987) consisting of 75 instances and 3 features, in which the first fourteen instances have been contaminated to become outliers. Using the Mahalanobis distance, only observation 14 is detected as an outlier as is shown in Figure 4.6. The remaining 13 outliers appear to be masked.

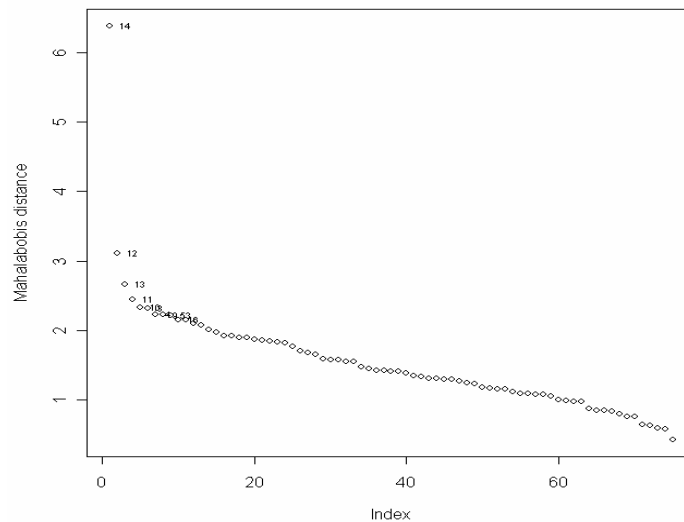


Figure 4.6: The Masking effect of multivariate outliers in the Hawkins dataset

Masking and swamping can be solved by using robust estimates of the centroid (location) and the covariance matrix (dispersion), which by definition are

less affected by outliers. Outlying points are less likely to enter into the calculation of the robust statistics, so they will not be able to influence the parameters used in the Mahalanobis distance. Two robust estimators of the centroid and the covariance matrix include the minimum covariance determinant (MCD) and the minimum volume ellipsoid (MVE), both introduced by Rousseeuw (1985).

The *Minimum Volume Ellipsoid (MVE) estimator* is the center and the covariance of a subsample of size h ($h \leq n$) that minimizes the volume of the covariance matrix associated to the subsample. Formally,

$$\mathbf{MVE}=(\bar{\mathbf{x}}_J^*, S_J^*), (4.4)$$

where $J=\{\text{set of } h \text{ instances: } Vol(S_J^*) \leq Vol(S_K^*) \text{ for all } K \text{ s.t. } \#(K)=h\}$.

The value of h can be thought of as the minimum number of instances which must not be outlying and is usually equal to $\left\lceil \frac{n+p+1}{2} \right\rceil$, where $\lceil . \rceil$ is the greatest integer function, n is the number of observations and p is the number of features. The volume of the ellipsoid is calculated using the formula:

$$Vol(S_k) = \left\{ \left| S_k \right| \text{median}_{i=1,2,\dots,h} d_i^2 \right\}^{\frac{1}{2}}.$$

The *Minimum Covariance Determinant (MCD) estimator* is the center and the covariance of a subsample of size h ($h \leq n$) that minimizes the determinant of the covariance matrix associated with the subsample. Formally,

$$\mathbf{MCD}=(\bar{\mathbf{x}}_J^*, S_J^*), (4.5)$$

where $J=\{\text{set of } h \text{ instances: } |S_J^*| \leq |S_K^*| \text{ for all } K \text{ s.t. } \#(K)=h\}$. As before, it is common to take $h = \lceil (n+p+1)/2 \rceil$, where $\lceil . \rceil$ is the greatest integer function.

The MCD estimator underestimates the scale of the covariance matrix, so the robust distances are slightly too large, and too many instances tend to be nominated as outliers. A scale correction has been implemented, and it seems to work well. The

algorithms used to compute the MVE and MCD estimators are based on combinatorial arguments (for more details see Rousseeuw and Leroy, 1987).

In this thesis, both estimators, MVE and MCD, have been computed using the function ***cov.rob*** available in the package ***lqs*** of **R**. This function uses the best algorithms available so far to compute both estimators (Rousseeuw, 1989). Taking into account their statistical efficiency and computational accuracy, the MCD is preferred over the MVE.

Replacing the classical estimators of the center and the covariance in the usual Mahalanobis distance, Equation 4.3, by either the MVE or MCD estimator, outlying instances will not skew the estimates and can be identified as outliers by large values of the Mahalanobis distance. The most common cutoff point k is again the one based on a Chi-Square distribution, although Hardim and Rocke (2004) propose a cutoff point based on the F distribution that they claim to be better.

In this thesis, two strategies to detect outliers using robust estimators of the Mahalanobis distances have been used. The first method involves choosing a given number of instances appearing at the top of a ranking based on their robust Mahalanobis measure. The second method chooses as multivariate outliers the instances that are tagged as outliers in the boxplot of the distribution of these robust Mahalanobis distance.

Example 3: Find the multivariate outliers in each of the classes of the *Iris* dataset by building boxplots for the distribution of the robust version of the Mahalanobis distance.

Using the ***robout*** function we have written in **R** (see appendix) and considering 10 repetitions the results appearing in the tables 4.1 and 4.2 have been obtained from the boxplots for the distribution of the robust version of the Mahalanobis distances. Notice that both methods detect two outliers in the first class, but the MVE method detects the instance 42 as a second outlier whereas the MCD

method detects the instance 24. All the remaining outliers detected by both methods are the same. Three more outliers are detected in comparison with the use of the Mahalanobis distance.

Table 4.1: Top outliers per class in the *Iris* dataset by frequency and the outlyingness measure using the MVE estimator

Instance	class	Frequency	Outlyingness
44	1	8	5.771107
42	1	8	5.703519
69	2	9	5.789996
119	3	8	5.246318
132	3	6	4.646023

Table 4.2: Top outliers per class in the *Iris* dataset by frequency and the outlyingness measure using the MCD estimator

Instance	Class	Frequency	Outlyingness
44	1	10	6.557470
24	1	10	5.960466
69	2	10	6.224652
119	3	10	5.390844
132	3	7	4.393585

Figure 4.7 shows a plot of the ranking of the instances in class 3 of the *Iris* dataset by their robust Mahalanobis distance using the MVE estimator. Figure 4.8 shows a plot of the ranking of the instances in class 3 of *Iris* by their robust Mahalanobis distance using the MCD estimator. According to Rocke (2002) robust methods work well detecting scattered outliers but fail to detect clustered outliers. For this type of outlier it is better to use a clustering algorithm as will be discussed in the next section.

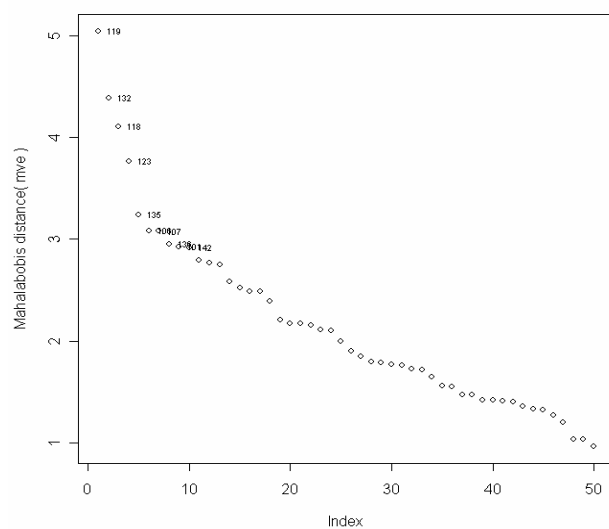


Figure 4.7: Plot of the instances of the *Iris* dataset ranked by their Mahalanobis distance using MVE estimator

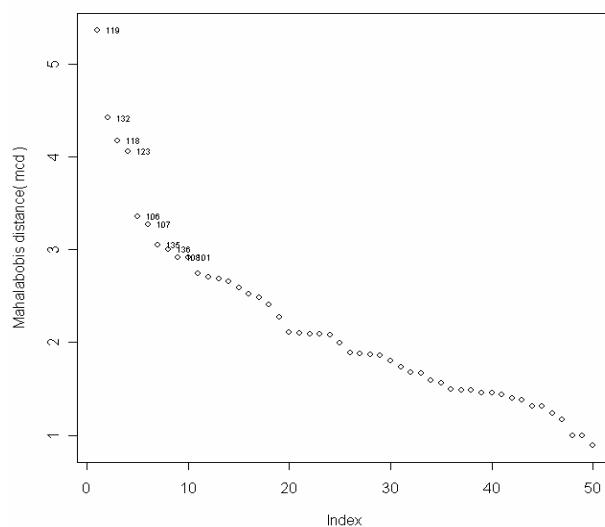


Figure 4.8: Plot of the instances of *Iris* class 3, ranked by their Mahalanobis distance using MCD estimator

4.3.2. *Detection of outliers using clustering*

A clustering technique can be used to detect outliers. Scattered outliers will form a cluster of size 1 and clusters of small size can be considered as clustered outliers. There are a large number of algorithms for finding clusters. In this thesis, only the Partitioning around Medoids (PAM) method will be considered. PAM was introduced by Kaufman and Rousseeuw (1990) to improve the well-known k-means clustering method. It works efficiently on small datasets, but is extremely costly for larger ones. This led to the development of CLARA (Clustering Large Applications) (Kauffman and Rousseeuw, 1990), where multiple samples of the dataset are generated, and then PAM is applied to each sample. CLARA chooses the best clustering as the output, basing quality on the similarity and dissimilarity of objects in the entire set, not just the samples. A modification of CLARA that is applied to very large datasets is CLARANS (Ng and Han, 1994).

Given k , the number of partitions to construct, PAM creates an initial partitioning. It then uses an iterative relocation technique that attempts to improve the partitioning by moving instances from one group to another. The general criterion of “good” partitioning is that instances in the same cluster are “close” or related to each other, whereas instances of different clusters are “far apart” or very different.

In order to find k clusters, PAM’s approach is to determine a representative instance for each cluster. This representative instance called *medoid*, is meant to be the most centrally located instance within the cluster. More specifically, a medoid can be defined as that instance of a cluster, whose average dissimilarity to all the objects in the cluster is minimal. After finding the set of *medoids*, each object of the dataset is assigned to the nearest *medoid*.

If O_j is a non-selected instance and O_i is a selected medoid, we say that O_j belongs to cluster represented by O_i if $d(O_i, O_j) = \min_{O_e} d(O_j, O_e)$ where the minimum is taken over all medoids O_e , and $d(O_a, O_b)$ denotes the dissimilarity or distance between instances O_a and O_b .

The PAM algorithm consists of two steps:

1. *The BUILD-step*: This step sequentially selects k centrally located instances, to be used as initial medoids.
2. *The SWAP-step*: If the objective function $J = \sum d(i, mv_i)$

$d(i, mv_i)$, which is the sum of the dissimilarities of all instances to their nearest medoid mv , can be reduced by interchanging (swapping) a selected object with an unselected object, then the swap is carried out. This is continued until the objective function J can no longer be decreased.

There are $k(n-k)$ possible pairs of (O_i, O_h) . For each pair, computing J requires the examination of $(n-k)$ non-selected instances. Thus, the combined complexity is: $O(k(n-k)^2)$. Hence, PAM becomes very costly for large values of n and k . However, PAM is very robust to the presence of outliers and does not depend on the order in which instances are examined.

After the allocation of the instances to the k clusters, one must determine the *separation* between them. The *separation* of the cluster C is defined as the smallest dissimilarity between two objects; one which belongs to Cluster C and the other that does not. That is, $\text{separation}_c = \min(d_{lh}), l \in C, h \notin C$.

If the separation of a cluster is large enough, then all of the instances that belong to the cluster are considered outliers. In order to detect the clustered outliers one must vary the number of clusters, k , until clusters of small size are obtained that have a large separation from others clusters.

The algorithm PAM can be evaluated using the function **pam** available in the library **cluster** in R.

Example 4: Find the outliers of the *Iris* dataset using the PAM algorithm.

Looking at the separation measures of ten clusters generated for each class, the detected outliers are shown in the table 4.3.

Table 4.3: Outliers in the Iris dataset according to the PAM algorithm

Instance	Class	Separation
42	1	0.6244998
58	2	0.6480741
61	2	0.6480741
94	2	0.6480741
99	2	0.6480741
107	3	0.9110434
118	3	0.8185353
132	3	0.8185353

Notice that in the class 3, PAM detects the instance number 107 as an outlier but it does not detect the instance 119.

4.3.3. *Distance based outlier detection*

Given a distance measure on a feature space, two different definitions of distance-based outliers are the following.

1. An instance \mathbf{x} in a dataset D is an outlier with parameters p and λ if at least a fraction b of the objects are a distance greater than λ from \mathbf{x} . (Knorr and Ng, 1997, 1998, Knorr et al. 2000). This definition has certain difficulties such as the determination of λ and the lack of a ranking for the outliers. Thus an instance with very few neighbors within a distance λ can be regarded as strong an outlier as an instance with more neighbors within a distance λ . Furthermore, the time complexity of the algorithm is $O(pn^2)$, where p is the number of features and n is the number of instances. Hence it is not an adequate definition to use with datasets having a large number of instances.
2. Given the integers k and n ($k < n$), outliers are the top n instances with the largest distance to their k th nearest neighbor (Ramaswamy et al., 2000).

One shortcoming of this definition is that it only considers the distance to the k th neighbor and ignores information about closer points. An alternative is to the use of the average distance to the k nearest neighbors instead of the greatest distance. The drawback of this alternative is that it takes longer to calculate.

In this thesis a variant of a simple algorithm for distance-based outlier detection, that is based on nested loops (Bay and Schwabacher, 2003), has been used.

Bay's Algorithm.

Bay and Schwabacher (2003) proposed a simple nested loop algorithm that tries to reconcile definitions 1 and 2. The main idea in the algorithm is that for each instance in D one keeps track of the closest k neighbors found so far. When an instance's k closest neighbors achieve a score that is lower than a cutoff, then the instance is removed from the list of possible candidates for outliers because it can no longer be an outlier.

In this thesis the score function used has been the median distance to the k neighbors. Bay used the average distance to the k neighbors, but the median is more robust than the mean. As more instances are processed, the algorithm finds more extreme outliers and the cutoff increases along with pruning efficiency. The performance of the algorithm in the worst case is of quadratic order. The algorithm is shown in Figure 4.9.

Bay and Schwabacher (2003) determined experimentally that the algorithm obtains linear performance with respect to the number of neighbors and almost linear with respect to the number of instances, when the data is in random order and a simple pruning rule is used. Using 6 datasets they found a complexity of order $O(n^\alpha)$ where α varied from 1.13 to 1.32. In this thesis work, an α value near 1.5 has been obtained for three datasets: Ionosphere, Vehicle and Diabetes, sustaining that the pruning rule has the effect of lowering the theoretical time complexity. A function

called *baysout* has been written in the *R* language to perform Bay's algorithm. The algorithm is shown in Figure 4.9.

```

Input: k: number of nearest neighbors
         n: number of outliers to return
         D: dataset randomly ordered
         BS: size of blocks in which D is divided.

Let distance(x,y) return the Euclidean distance between x and y.
Let maxdist(x,Y) return the maximum distance between the instance x and the set of instances Y.
Let Closest(x,Y,k) return the k closest instances in Y to x.
Let score(x) return median distance to the k neighbors

begin
c ← 0 Set the cutoff for pruning to 0.
O ← ∅ Initialize the set of outliers as the empty set.
NB ← ceiling(# instances in  $\frac{D}{BS}$ )
  while nb < NB {
    Neighbors(b) ← ∅ for all b in Bnb
    for each d in D {
      for each b in Bnb, b ≠ d {
        if |Neighbors(b)| < k or distance(b,d) < maxdist(b, Neighbors(b)) {
          Neighbors(b) ← Closest(b, Neighbors(b) ∪ d, k)
        }
        if (score(Neighbors(b),b) < c {
          remove b from Bnb
        }
      }
    }
    O ← Top(Bnb ∪ O, n) //Keep only the top n outliers
    c ← min(score(o)) for all in O //The cutoff is the score of the weakest outlier
  }
end

Output: O, a set of outliers

```

Figure 4.9: Bay's Algorithm for finding distance-based outliers

Example 5: Find the outliers of the class 3 in the *Iris* dataset using the Bay's algorithm.

Using the *baysout* function the top 20 outliers are shown in Figure 4.10. Clearly the instance 107 is detected as an outlier. There is a second group that includes 119, 120, 132, 123 and 118.

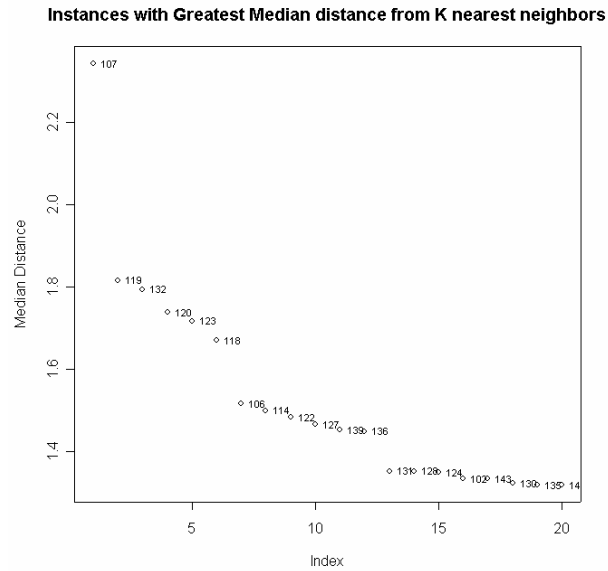


Figure 4.10: Instances of the class 3 in Iris dataset ranked by the Bay's algorithm outlyingness measure

4.3.4. *Density-based local outliers*

The term *density-based local outlier* was introduced by Breunig et al (2000). For density-based local outliers the density of an instance and the density of its neighbors play a key role in classifying an instance as an outlier. Furthermore, an instance is not explicitly classified as either an outlier or a non-outlier. Instead, for each instance, a local outlier factor (LOF) is computed which will give an indication of how strong of an outlier an instance has been found to be.

Figure 4.11 that follows, taken from Breunig et al (2000), shows the weakness of the distance-based outlier detection method which would identify the instance o_1 as an outlier, but would not consider o_2 as an outlier. Several definitions are needed in order to formalize the algorithm.

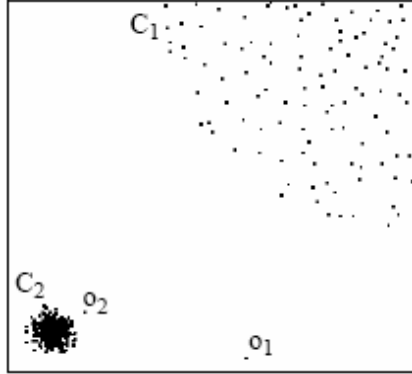


Figure 4.11: Example to show the weakness of the distance-based method to detect outliers

Definition 1. *k-distance of an instance x*

For any positive integer k , the k -distance of an instance x , denoted by $k\text{-distance}(x)$, is defined as the distance $d(x,y)$ between x and an instance $y \in D$ such that:

- (i) for at least k instances $y' \in D - \{x\}$, $d(x,y') \leq d(x,y)$
- (ii) for at most $k-1$ instances $y' \in D - \{x\}$, $d(x,y') < d(x,y)$.

Definition 2. *k-distance neighborhood of an instance x*

Given an instance x of a dataset D its k -distance neighborhood contains every instance whose distance from x is not greater than the k -distance. That is, the set of k -nearest neighbors of x is given by

$$N_{k\text{-distance}(x)} = \{y \in D - \{x\} \text{ s.t. } d(x,y) \leq k\text{-distance}(x)\}. \quad (4.6)$$

Definition 3. *Reachability distance of an instance x w.r.t. instance y*

Let k be a positive integer number. The reachability distance of the instance x with respect to the instance y is defined as

$$\text{reach-dist}_k(x,y) = \max\{k\text{-distance}(y), d(x,y)\}. \quad (4.7)$$

The density-based local algorithm to detect outliers requires only one parameter, *MinPts*, which is the number of nearest neighbors used in defining the local neighborhood of the instance.

Definition 4. *Local reachability density of an instance x*

Given an instance x of a dataset D its local reachability density is defined by

$$lrd_{MinPts}(x) = \left\{ \frac{\sum_{y \in N_{MinPts}(x)} reach-dist_{MinPts}(x, y)}{|N_{MinPts}(x)|} \right\}^{-1} . \quad (4.8)$$

This is the inverse of the average reachability distance based on the *MinPts*-nearest neighbor of x . Finally the definition of the outlyingness measure is given below.

Definition 5. *Local outlier factor (LOF) of an instance x*

The LOF measures the degree to which an instance x can be considered an outlier and is defined by

$$LOF_{MinPts}(x) = \left\{ \frac{\sum_{y \in N_{MinPts}(x)} \frac{lrd_{MinPts}(y)}{lrd_{MinPts}(x)}}{|N_{MinPts}(x)|} \right\}^{-1} . \quad (4.9)$$

Breunig et al showed that instances deep inside a cluster have LOF's that are close to 1 and should not be labeled local outliers.

Since LOF is not monotonic, Breuning et al recommends finding the LOF for each instance of a dataset using *MinPts*-nearest neighbors, where *MinPts* assumes a range of values from *MinPtsLB* to *MinPtsUB*. They reported that for the datasets they experimented with, *MinPtsLB*=10 and *MinPtsUB*=20 seemed to work well. In this thesis, LOF were found in a range of *MinPtsLB*=10 and *MinPtsUB*=20 or *MinPtsLB*=20 and *MinPtsUB*=30, depending on the range which produced a more monotonic plot.

After deciding upon the values to use for MinPtsLB and MinPtsUB, the LOF of each instance is computed over this range. Finally all the instances are ranked with respect to the maximum LOF value within the specified range. That is, the ranking of an instance x is based on:

$$\text{Max}\{\text{LOF}_{\text{MinPts}(x)} \text{ s.t. } \text{MinPtsLB} \leq \text{MinPts} \leq \text{MinPtsUB}\}. \quad (4.10)$$

A **maxlof** (see appendix) function has been written in the **R** language to perform the LOF algorithm as part of this work. The algorithm is shown in figure 4.12.

Input: Dataset D, MinptsLB, MinptsUB
Let maxlofvect= ϕ
for each i in the interval [MinPtsLB, MinPtsUB]
 {
 1. Find the i nearest neighbors and their distance from each observation in D
 2. Calculate the local reachability density for each observation in D
 3. Compute the local outlier factor of each observation in D
 4. maxlofvect=max(maxlofvect, lof)
 }
end
Output: maxlofvect, the local outlier factor for each observation in D

Figure 4.12: The maxLOF Algorithm

Breunig et al.(2000) states that the time complexity of the maxLOF algorithm can be analyzed by studying independently the time complexity of the two main steps required to produce the LOF factor for each instance of the dataset. The first step, finding the k -distance neighborhood, has a runtime complexity of $O(n \cdot \text{time for a } k\text{-nn query})$. Therefore, the actual time complexity of this step is determined by the method used to perform the k -nn queries. For low dimensionality (no more than 5 features), if a grid based approach is used the query can be performed in constant time leading to a complexity of $O(n)$ to complete the entire step. For medium

dimensionality (between 5 and 10 features), an index can be used that would provide an average complexity of $O(\log n)$ for the k-nn queries, leading to a total complexity of $O(n \log n)$. Finally, for high dimensional data, a sequential scan may be used with a complexity of $O(n)$ that would lead to a total complexity of $O(n^2)$. Finding the maximum outlier factors of all observations in the dataset can be done in linear time.

Table 4.4 shows the experimental running time for the local outlier factors for all observations of the datasets used in this work. The times have been computed using two different values for k , the number of neighbors. Note that the number of neighbors does not affect the running time. Using $k=15$, the regression line of $\log(\text{time})$ versus $\log(n)$ and $\log(p)$ is

$$\log(\text{time}) = -4.80 + 1.62 \log(n) + 1.09 \log(p)$$

with a $R^2 = 90.1\%$. Therefore a good estimate of the complexity would be $O(n^{1.62} p^{1.09})$. Looking at the relationship of time versus the number of instances n , one gets an estimated regression line given by

$$\log(\text{time}) = -4.21 + 1.87 \log(n)$$

with $R^2 = 75.1\%$, and a good estimate for the complexity would be $O(n^{1.87})$.

Table 4.4: Experimental running times for computing the LOF for all observations

	k=15	k=25	n	p	log(time)	log(n)	log(p)
Iris	0.54	0.55	150	4	0.26761	2.17609	0.60206
Sonar	11.45	11.3	208	60	1.05881	2.31806	1.77815
Heartc	1.32	1.33	297	13	0.12057	2.47276	1.11394
Bupa	1.21	1.33	345	6	0.08279	2.53782	0.77815
Ionosfera	16.19	15.92	351	32	1.20925	2.54531	1.50515
Crx	8.51	8.56	653	15	0.92993	2.81491	1.17609
Breastw	4.8	4.72	683	9	0.68124	2.83442	0.95424
Diabetes	8.89	8.99	768	8	0.9489	2.88536	0.90309
Vehicle	11.18	11.43	846	18	1.04844	2.92737	1.25527
German	11.16	11.14	1000	20	1.04766	3	1.30103
Segment	323.45	321.3	2310	16	2.50981	3.36361	1.20412
Landsat	746.71	746.38	4435	36	2.87315	3.64689	1.5563

This agrees with the Breuning's claim that for a high-dimensional data the LOF algorithm has time complexity $O(n^2)$.

A log-scale plot that shows time required to compute the LOFs for all the datasets in our study as the number of instances increases is shown in Figure 4.13. This log-scale plot suggests a near quadratic relationship between the computing running time of the LOFs and the number of instances.

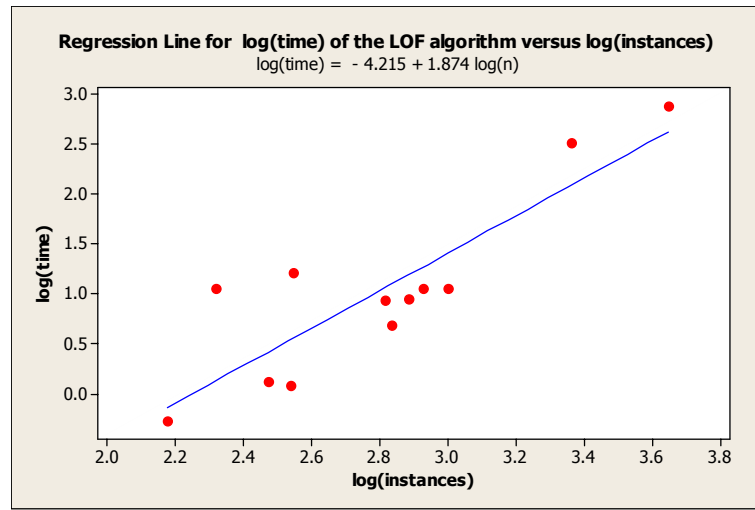


Figure 4.13: Runtime for the computation of LOFs for different datasets

Example 6. Find the outliers of the third class in the *Iris* dataset using the LOF algorithm.

Using the *maxlof* function, the top 10 outliers are shown in Figure 4.14. Clearly the instance 107 is detected as an outlier. There is a second group that includes 119, 118, 132 and 123. Finally, instance 106 appears as a third group.

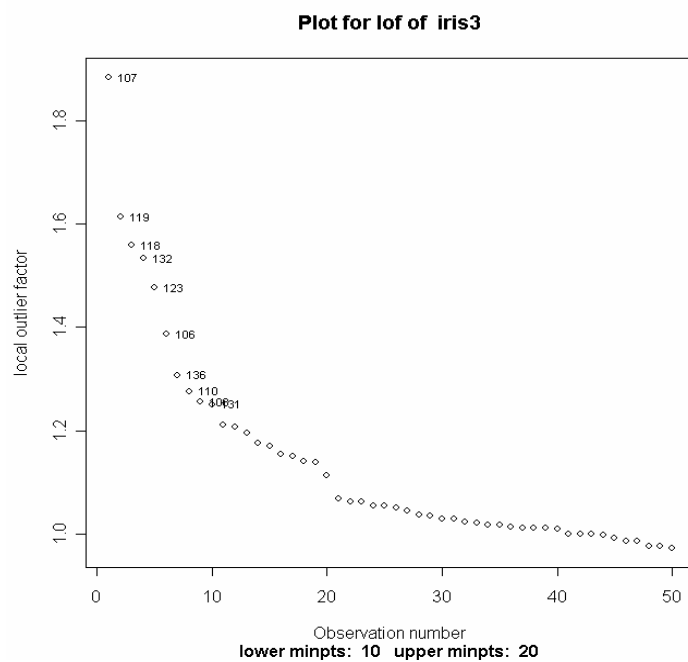


Figure 4.14: Instances of the class 3 in Iris dataset ranked by the LOF's algorithm outlyingness measure

Chapter 5 Feature Selection

5.1. Introduction

The dimensionality problem in a knowledge discovery process still remains as a very important problem to be investigated in spite of the fact that computers are becoming more powerful everyday. The classification task in knowledge discovery is more conveniently done with few features for two reasons: a saving of computing time and an easy interpretation of the model. The goal of feature selection is to choose a small subset of features such that the recognition rate of the classifier does not decrease significantly. Feature selection methods are classified by the way they generate subsets and by the evaluation function used to measure the quality of the subset produced. The subset generation procedure can be one of three types: complete, heuristic and random. The evaluation function can be a distance measure, an information measure, a dependence measure, a consistency measure, or the misclassification error rate.

Dash and Liu (1997) established 15 categories of feature selection procedures, based on the generation procedure of the subsets and the evaluation function used to compare them. In this thesis, the performance of five feature selection procedures are evaluated: The RELIEF, Las Vegas Filter, FINCO, Sequential forward selection (SFS) and, Sequential floating forward selection (SFFS). The first three are considered filter methods because they do not use a classifier to select features, whereas the last two require a classifier and are known as wrapper methods.

The classifiers used in this thesis are: the discriminant linear analysis (LDA), the k-nearest neighbor classifier, and the recursive partitioning classifier (rpart). Filters and wrappers are compared according to the percentages of features selected

and the effect on the misclassification error rate of the feature subsets produced. The comparison is carried out in twelve datasets available in the Machine Learning Repository at the University of California, Irvine. The number of features of these datasets varies from 4 to 60.

Section 2 of this chapter deals with the filter methods. Wrappers methods are discussed in section 3.

5.2. *Filter methods*

These methods do not require the use of a classifier to select the best subset of features. They use general characteristics of the data to evaluate features. In this paper we considered three filter methods: the RELIEF, Las Vegas Filter (LVF) and a new procedure introduced by Acuña et al (2003) called FINCO. We will describe each of them briefly.

5.2.1. *The RELIEF Algorithm*

This method was introduced for a two class problem by Kira and Rendell in 1992. The general idea of this method is to choose the features that can be most distinguished between classes. These are known as the relevant features. In a two class problem, initially each of the p features of the dataset, D , have a relevance weight w_j ($j=1, \dots, p$) equal to zero. Then, at each step of an iterative process, an instance \mathbf{x} is chosen randomly from D and the weights w_j are updated according to the distance of \mathbf{x} to its *Nearmiss* and *NearHit*. The *Nearmiss* is the instance in the dataset, D , that is closest to \mathbf{x} but that belongs to the other class. The *NearHit* is the instance in D that is closest to \mathbf{x} and belongs to its same class. The updating formula of w_j is given by

$$w_j = w_j - \text{diff}(x_j, \text{Nearhit}_j)^2 + \text{diff}(x_j, \text{Nearmiss}_j)^2, \quad (5.1)$$

where x_j the j -th component of \mathbf{x} , and the function *diff* computes the distance between the values of a feature for two given instances. For nominal and binary

features, *diff* is either 1 (the values are different) or 0 (the values are equal). For ordinal and continuous features, *diff* is the difference of the values of the feature for the two instances normalized by the range (maximum-minimum) of the feature.

The process is repeated M times, where M is a predefined parameter, usually taken as the number of instances in D . The repetition must be done to reduce the variability generated by the randomization. In this work, the algorithm was repeated ten times for datasets with less than 10 features and, twenty times for datasets with 10 or more features. The output of the algorithm is the best subset of features that includes those features with relevance greater than a specified threshold. The RELIEF works well in datasets containing mixed types of features, as well as with datasets containing noise and correlated features. Its time complexity is $O(n \times M \times p)$, that is, linear in the number of features as well as in the number of instances.

The RELIEF algorithm appears in Figure 5.1.

Input: D =Dataset, p = number of features in D , M =number of instances randomly drawn, Threshold= τ .

1. **let** $T = \phi$, T is the subset containing the features being selected
2. Initialize all weights, w_j ($j=1, \dots, p$), to zero
3. **for** $i=1$ to M
 - { Choose at random an instance \mathbf{x} in D .
 - Find its *Nearhit* and *NearMiss*
 - for** $j=1$ to p

$$w_j = w_j - \text{diff}(x_j, \text{Nearhit}_j)^2 + \text{diff}(x_j, \text{Nearmiss}_j)^2$$
 - }
4. **for** $j=1$ to p
 - if** $w_j > \tau$ **then** append *feature_j* to T
5. **Output:** T , the set of most relevant features.

Figure 5.1: The RELIEF algorithm

The RELIEF algorithm presents the following disadvantages:

- The selection of the threshold τ and of the parameter M , are unclear. Yet the choice of the threshold is very critical for the efficiency of the RELIEF algorithm.
- The algorithm eliminates irrelevant features, but redundant features, and those that might be correlated with others, can be selected.

In this thesis, the threshold was chosen iteratively. First, the threshold was set to 0. Then, after looking at the plots of the feature weights, the threshold was increased to equal the weight at which the lowest gap occurs in the plot (if one was present). Otherwise, the value was moved away from zero, trying to obtain frequencies that were near 10 for the chosen features. In some cases when the gap did not appear clearly, then the threshold was refined with additional repetitions of the algorithm. However, this situation did not occur very often.

In this thesis, for small datasets, M was chosen equal to the number of instances of the dataset D . However, for large datasets, M was taken to be equal to a value as low as a 10 percent of the total number of instances (depending on the presence of a gap in the plot of relevance weights).

The RELIEF algorithm was extended to multiclass problems by Kononenko (1994) and Kononenko et al. (1997). The new algorithm was named RELIEF-F. In this case, one *Nearmiss* is found for every class distinct to the class containing \mathbf{x} and the distance from \mathbf{x} to each *Nearmiss* is weighted according to the proportion of instances in each class. The updating formula of the relevance weight w_j is as follows:

$$w_j = w_j - \text{diff}(x_j, \text{Nearhit}_j)^2 + \sum_{C \neq \text{class}(x_j)} \frac{P(C)}{1 - p(\text{class}(x_j))} \text{diff}(x_j, \text{Nearmiss}(C))^2. \quad (5.2)$$

The RELIEF algorithm used in this thesis was the one modified by Kononenko.

An **R** function named *relief*, (see appendix) was written to implement the RELIEF algorithm.

Example 1. Apply the RELIEF algorithm to select the best subset of features for the *Ionosphere* dataset, which has 32 features and 351 instances.

The results obtained after applying the *relief* function are:

Frequencies and average weights of most relevant features in 10 replicates:

	feature	frequency	weight
[1,]	25	10	0.04112852
[2,]	26	10	0.03926254
[3,]	22	10	0.03513727
[4,]	28	10	0.02904335
[5,]	13	10	0.02780220
[6,]	3	10	0.02767617
[7,]	27	7	0.02761284
[8,]	24	8	0.02418991
[9,]	23	7	0.02417275
[10,]	32	10	0.02339732
[11,]	8	6	0.02265841
[12,]	1	6	0.01999221
[13,]	30	6	0.01981515

Selected features

[1] 25 26 22 28 13 3 27 24 23 32 8 1 30

The corresponding plot of the relevance weights is shown in figure 5.2. Notice the gap at $weights = 0.02$. There are other gaps above and below 0.02, but choosing a high value as a threshold will yield less features selected and choosing a lower threshold value will cause the selection of a large number of features.

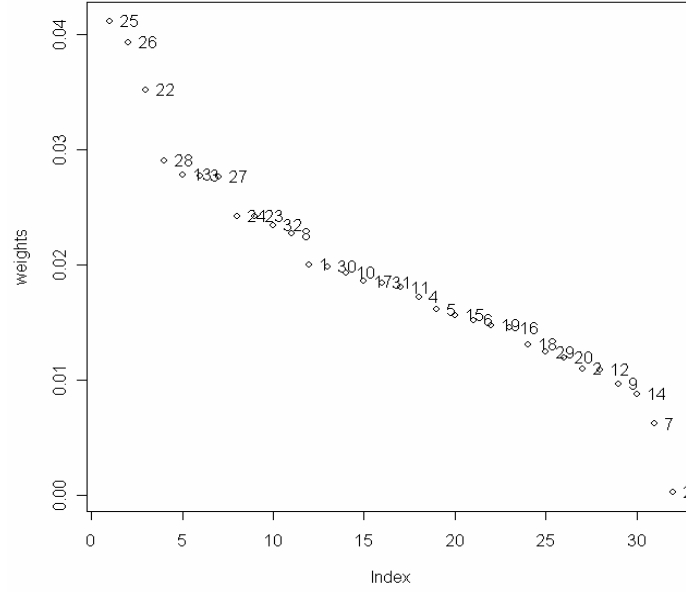


Figure 5.2: Plot of relevance weights for the features of the dataset *Ionosfera*

5.2.2. The Las Vegas Filter (LVF)

The Las Vegas Filter method (LVF) was introduced by Liu and Setiono (1996). LVF uses a random generation of subsets and an inconsistency measure as the evaluation function. Two instances of a dataset D are inconsistent if they have the same feature values but belong to different classes. The inconsistency measure of a given subset of features, T , relative to a dataset D , is defined as

$$Inconsistency(T, D) = \frac{\sum_{i=1}^K |D_i| - |M_i|}{N}, \quad (5.3)$$

where $|D_i|$ is the number of occurrences of the i^{th} feature value combination on T , K is the number of the distinct combinations of feature values on T , $|M_i|$ is the cardinality of the class to which the majority of instances on the i^{th} feature values belong, and N

is the number of instances in the dataset D . An R function *inconsist* was written to compute the inconsistency level of a dataset.

Example 2. The following table shows the inconsistency level of the *Breastw* dataset for several combinations of its features. The dataset has 9 features that take on integer values from 1 to 10.

Subset	Inconsistency
1	0.14055
1,4	0.05710
1,4,6	0.01171
1,4,8	0.02489
1,2,4,6	0.001464
1,2,4,5,6	0.001464
1,2,3,4,5,6,7,8,9	0

Notice that inconsistency is monotonically decreasing on the number of features. Thus, given two subsets of features A and B , such that $A \subseteq B$ then $\text{inconsistency}(A) \leq \text{inconsistency}(B)$.

The inconsistency measure can be applied to datasets with continuous features after applying a discretization process, such as the method of Fayyad and Irani that is based on partitioning by minimization of the class information entropy with Minimal Description Length Principle as a stopping criterion (Dougherty et al., 1995). In this thesis, we have applied a simple equal width interval discretization method based on Scott's formula (see Venables and Ripley (1997), p. 169) to estimate the width interval. In this formula a feature with n values and standard deviation s can be discretized on $k = \frac{R}{h}$ integer values, where $h = 3.5 \times s \times n^{-1/3}$ and, $R = \text{Max} - \text{Min}$, is the

range of the feature. An *R* function named ***discretar*** has been written to perform discretization using Scott's formula.

The LVF algorithm requires several input parameters. An inconsistency threshold that is close to or equal to zero must be set beforehand and sent to the algorithm. To determine this value, the inconsistency measure of the dataset, including all the features, was first computed. Then a value a little larger than the measure obtained was taken as the inconsistency threshold. Any candidate subset having an inconsistency measure greater than the threshold is rejected.

Another parameter is the maximum number of subsets to be generated randomly. This number was varied between 1000 and 20,000 depending on the number of features of the dataset and the variability of the subsets obtained. The LVF method is suitable for datasets having only nominal features. If there are any continuous features in the dataset it must be discretized previously. The LVF algorithm is shown in Figure 5.3.

An **R** function named ***lvf*** (see appendix) has been written to perform the LVF algorithm on the discretized dataset.

Example 3. Apply the LVF algorithm to select the best subset of features for the *Ionosphere* dataset.

The inconsistency level of the complete set of features is zero. After performing 20,000 iterations using a threshold of 0.001, and implementing a voting process consisting of 10 repetitions of the algorithm, the features 1, 5, 12, 14, 20, 29, 30 were selected.


```

Input:  $D$ =Dataset,  $p$ =number of features in  $D$ ,  $S$ =set of all features in  $D$ ,
 $MaxTries$  = maximum number of trials,  $\delta$ =Threshold.

begin {
  let  $S_{best}=S$ 
  let  $C_{best} = \text{card}(S_{best}) = p$ 
  for  $i=1$  to  $MaxTries$  {
     $S_i$ = subset of  $S$  randomly selected.
     $C_i=\text{card}(S_i)$ 
    if ( $C_i < C_{best}$ ) {
      if ( $\text{Inconsistency}(S_i, D) \leq \delta$ )
        then  $\delta = \text{Inconsistency}(S_i, D)$ 
      let  $S_{best} = S_i$ 
      let  $C_{best} = C_i$ 
    }
    if ( $C_i = C_{best}$ ) {
      if  $\text{Inconsistency}(S_i, D) < \delta$  then  $S_{best} = S_i$ 
    } }
Output:  $S_{best}$ , the best subset of features

```

Figure 5.3: The LVF algorithm

One disadvantage of the LVF algorithm is that it presents a high variation of the subsets being chosen. If one repeats the LVF algorithm a second time, several features will appear that did not appear the first time. For instance, for the *Ionosphere* dataset, LVF selects in one repetition the features: 5, 10, 13, 14, 28, 31, 32 and after a second repetition it selects 1, 2, 3, 8, 14, 27. Only the feature number 14 appears in the two “best” subsets. One can reduce this variability either by choosing a large number of iterations or a smaller threshold but this will slow down the computation of the LVF algorithm.

5.2.3. The *FINCO* method

The *FINCO* method was introduced by Acuña (2003). *FINCO* stands for Forward and Inconsistency. It uses a sequential forward generation of subsets along with the inconsistency measure used in the LVF method. The best subset of features, T , is initialized as the empty set and in each step we add to T the feature that gives the lowest inconsistency rate along with the features already included in T . The process continues until the inconsistency rate given by T and each of the features not yet selected is less than a predefined inconsistency threshold, which is chosen as in the LVF algorithm. A feature entering T can not be removed from it. Continuous features need to be discretized before applying FINCO. The FINCO algorithm appears in Figure 5.4.

In order to avoid the nesting problem, a floating forward selection may be applied, but it was not considered in this work. An **R** function called *finco* has been written to perform the FINCO algorithm.

Example 4. Apply the FINCO algorithm to select the best subset of features for the *Ionosphere* dataset.

Choosing a threshold of 0.001 for the inconsistency the features 3, 4, 14, 32 were selected.

FINCO seems to be biased towards small feature subsets, particularly if the number of instances is small relative to the number of features. One example of this is the *Sonar* dataset. FINCO and LVF have similar performance with respect to misclassification error reduction, but FINCO has a lower computation time.

The computation time of the FINCO algorithm depends on the number of instances, the number of features and the threshold for the inconsistency level. If a small inconsistency level is chosen then there are more comparisons to carry out and the computation time slows down.

Input: D =Dataset, p =number of features in D , S =set of all the features in D .
 ∂ =Threshold

Initialization:
 Let $k=0$ and $T=\phi$ (T_k : Subset of features selected until the k -th step).

Inclusion step:
for $k=1$ to p {
 $x^+ = \arg \min_{x \in S - T_k} Incons(T_k + x)$
 /* $S - T_k$ is a subset of features not yet selected, $Incons(T_k + x)$ is the inconsistency level using the features in T_k along with feature x . Thus, x^+ is the most important feature with respect to T_k */
if ($Incons(T_k + x^+) \leq Incons(T_k)$ **and** $Incons(T_k + x^+) > \partial$)
then {
 $T_{k+1} = T_k + x^+$
 $k := k + 1$
}
}

Termination:
if ($Incons(T_{k+1}) > Incons(T_k)$ or $Incons(T_{k+1}) \leq \partial$)
then print T_k

Output: T_k , a subset of the features of D .

Figure 5.4: The FINCO algorithm

The discretization method required for LVF and FINCO may affect their performance. On average, RELIEF reduces the misclassification error rate more than LVF and FINCO. The RELIEF algorithm also has a lower computation time than the other two methods.

5.3. Wrapper methods

The wrapper methods use the misclassification error rate of a given classifier as the evaluation function. A large discussion of wrappers can be found in Kohavi and

John (1997). In this thesis, three classifiers have been used in the wrapper methods: the linear discriminant analysis (LDA), the k-nn classifier, and rpart, a decision tree classifier. The misclassification error rate is estimated by a 10-fold cross-validation technique. In this work, only wrapper methods where the subsets are generated heuristically, have been considered. The three main approaches are: Sequential Forward selection (SFS), Sequential Backward selection (SBS), and the Sequential Floating Forward selection (SFFS).

In SBS the best subset of features, T , is initialized as the set containing all the features and in each step we remove from T the feature x for which T gives the highest correct classification rate (CCR) when x is excluded. Thus, the worst feature with respect to T is removed. The process continues until the CCR decreases when excluding from T each of the remaining features. The SBS method has not been considered in this thesis due to its slow computing time.

5.3.1. *Sequential Forward selection*

In Sequential Forward selection (SFS) the best subset of features T is initialized as the empty set. In each step, the feature that gives the highest correct classification rate (CCR) along with the features already included in T is added to T . The process continues until none of the remaining features not yet included in T produces an increase in the CCR when added to T . The complete algorithm is shown in Figure 5.5. An **R** function called *sfs* has been written to perform the sequential forward feature selection algorithm.

Example 5. Use *SFS* with the classifiers LDA, knn and rpart to perform feature selection on the *Ionosphere* dataset.

The algorithm was repeated 20 times in order to reduce the variability of the subset of selected features. The following features were selected.

	LDA	KNN	Rpart
Selected Features	3,6,19	1,3,4,14	1,3,5,6,32

In practice the size of the best subset was determined by averaging the number of features selected in each repetition, and then a subset was formed using the features with the highest frequencies. This subset was called the “best subset”.

Input: D =Dataset, p =number of features in D , S = set of all features in D .

Initialization:

Let $k = 0$ and $T = \phi$ (T : Subset of features selected until the k -th step).

Inclusion step:

for $k=1$ to p {

$$x^+ = \arg \max_{x \in S - T_k} CCR(T_k + x)$$

// $S - T_k$: Subset of features not yet selected, $CCR(T_k + x)$ is the correct classification rate of the classifier using the features in T_k along with the feature x . Thus, x^+ is the most important feature with respect to T_k //

if ($CCR(T_k + x^+) > CCR(T_k)$)

then {

$$T_{k+1} = T_k + x^+$$

$$k = k + 1$$

}

Termination:

if $CCR(T_{k+1}) \leq CCR(T_k)$

Output: T_k : Subset of selected features.

Figure 5.5: The SFS algorithm.

Both methods, SFS and SBS, suffer from the nesting problem. This means that a feature that is included (removed) in some step of the iterative process can not be excluded (included) in a later step.

5.3.2. Sequential Floating Forward Selection

The Sequential Floating Forward selection (SFFS) method was introduced by Pudil et al. (1994) to deal with the nesting problem. The best subset of features, T is initialized as the empty set and at each step first a new feature is added to T as in the SFS method but after that it searches for features that can be removed from T as in the SBS method until the CCR decreases. The iterations continue until no new feature can be added to T because the CCR does not increase. The SFFS algorithm is shown in Figure 5.6.

An **R** function named *sffs* has been written to perform the sequential floating forward feature selection algorithm.

Example 6. Using sequential floating forward selection with the classifiers LDA, knn and rpart for the *Ionosfera* dataset the features selected are:

The algorithm was repeated 20 times in order to reduce the variability of the subset of selected features. The following features were selected.

	LDA	KNN	Rpart
Selected Features	3,23	1,4,14	3,32,1,6

In practice the size of the best subset was determined by averaging the number of features selected in each repetition, and then a subset was formed using the features with the highest frequencies. This subset was called the “best subset”.

Input: D =Dataset, p =number of features in D , S = set of all features in D .

Initialization:

T = Subset of selected features after applying *SFS* twice. Set $k=2$.

for ($i=k$ to p) {

//Inclusion Step:

$$x^+ = \arg \max_{x \in S - T_k} CCR(T_k + x)$$

//Thus, x^+ is the most important feature with respect to T_k

let $T_{k+1} = T_k + x^+$

let $k = k + 1$

repeat {

//Conditional exclusion

$$x^- = \arg \max_{x \in T_k} CCR(T_k - x)$$

//Thus, x^- is the least important feature in T_k .

if $CCR(T_k - x) > CCR(T_k)$

then {

$$\text{let } T_{k-1} = T_k - x^-$$

let $k = k-1$

 }

until $CCR(T_k - x) < CCR(T_k)$

}

Termination:

if $CCR(T_{k+1}) \leq CCR(T_k)$

Output: T = Subset of selected features.

Figure 5.6: The sequential floating forward selection algorithm (SFFS).

Figure 5.7 shows the computation time of the wrapper methods used in this study. We observed that it is linear on the number of instances. For smaller datasets, SFS and SFFS with the k-nn classifier, is computed a little bit faster but the graph suggests that this difference will not be reflected in datasets with a large number of instances.

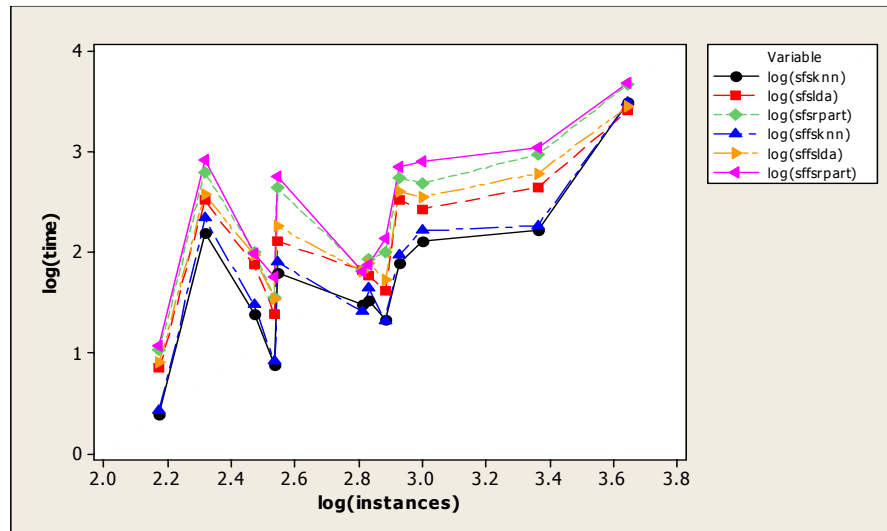


Figure 5.7 : Plot of the logarithms of time computation of the wrappers versus the logarithm of instances in each of the twelve datasets

Figure 5.8 shows the relationship between the logarithms of the computation time of all of the combinations of wrapper-classifier versus the logarithm of the number of features. The graph suggests a linear relationship of slope 2 and therefore a quadratic relationship between the computation time and the number of features in the dataset.

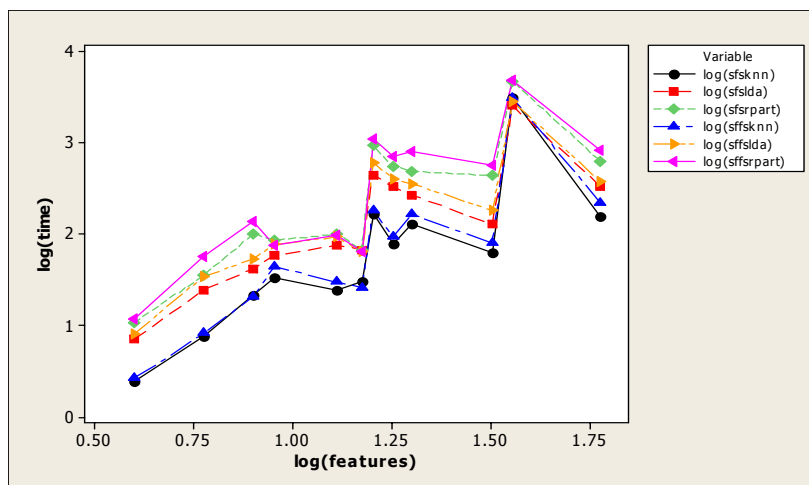


Figure 5.8: Plot of the logarithm of the computation time of the wrappers versus the logarithm of the number of features in each of the twelve datasets.

Both figures and the corresponding estimated regression lines suggest, empirically, that the complexity time of the algorithms for the wrappers is $O(np^2)$. Approximated time complexities for feature selection methods can be found in Kudo and Sklansky (2000).

Below we present two plots in logarithmic scale that show the relationship between the number of instances and features versus computer running time for the filter methods. Figure 5.9 suggests that a linear relationship exists between the number of instances and the computer running time.

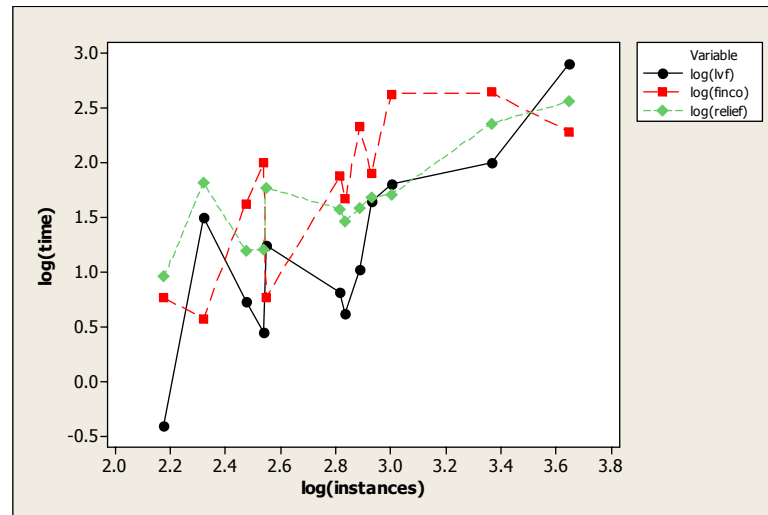


Figure 5.9: Plot in logarithm scale that shows a linear relationship between the number of instances and the computer running time of the filter algorithms.

Figure 5.10 suggests the existence of a quadratic relationship between the number of features and the computer running time for filter algorithms. Two datasets, Sonar and Ionosphere, have been omitted from the plot because these datasets have a high proportion of features as compared to the number of instances. This fact seems to affect the performance of filter algorithms.

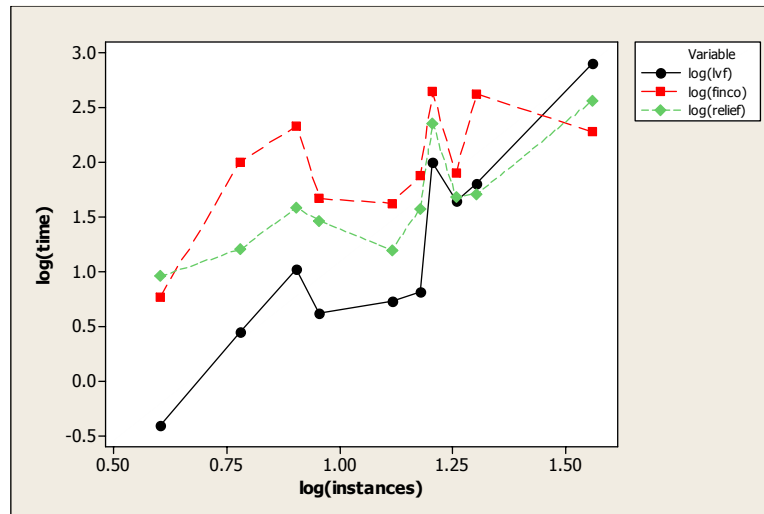


Figure 5.10: Plot in logarithm scale that shows the quadratic relationship between the number of features and the computer running time of the filter algorithms.

A discussion of the effect on the misclassification error rate for both filters and wrappers will be considered in chapter 7.

Chapter 6 Missing Values

6.1. Introduction

Missing data is a common problem in statistical analysis. In particular, missing values in a dataset can affect the performance of a classifier constructed using such a dataset as a training sample. Rates of less than 1% missing data are generally considered trivial, 1-5% manageable. However, 5-15% must be handled by sophisticated methods, and more than 15% may severely impact any kind of interpretation. (Pyle, 1999)

Several methods have been proposed in the literature to treat missing data. Many of these methods were developed for dealing with missing data in sample surveys (Kalton and Kasprzyk (1986), Little and Rubin (2002)) and have some drawbacks when they are applied to classification tasks. Chan and Dunn (1972) considered the treatment of missing values in supervised classification using the LDA classifier but only for two-class problems considering a simulated dataset from a multivariate normal model. Dixon (1979) introduced the KNN imputation technique for dealing with missing values in supervised classification. Tresp et al. (1995) also considered the missing value problem in a supervised learning context for neural networks.

The interest in dealing with missing values has continued with the statistical applications to new areas such as Data Mining (Grzymala-Busse and Hu , 2000) and Microarrays (Hastie et al, 1999, Troyanskaya et al, 2001). These applications include supervised classification as well as unsupervised classification (clustering). When working with data from microarrays, some people even replace missing values by zero. Bello (1995) compared several imputation techniques in regression analysis, a related area to classification.

In this thesis work, four methods for dealing with missing values are considered: the case deletion method, mean imputation, median imputation and KNN imputation procedure. In section 2 an explanation of the mechanisms that lead to missing data is given. The four methods to treat of missing values considered in this thesis are described in section 3. In section 4 other existing imputation methods that are not included in this thesis work are described.

6.2. *Mechanisms that lead to missing data*

According to Little and Rubin (2002) there are three classes of missing data mechanisms.

i) *Missing completely at random (MCAR)*: If the probability of an instance having a missing value for an attribute is the same for all the instances. This means that such probability does not depend on either the known values or the missing data. As an example, suppose weight and age are features of interest for a particular study. If the likelihood that a person will provide his or her weight information is the same for all individuals regardless of their weight or age, then the missing data in the attribute weight is considered to be MCAR. More formally,

Let $x = (x_{ij})$ be the data matrix containing missing values. Let I be a random variable that assumes the value of 1 if x_{ij} is present and 0 if x_{ij} is missing. Then, in the MCAR mechanism, it is assumed that the distribution of I does not depend on the data. Thus,

$$\text{Prob}(I/x_{\text{miss}}, x_{\text{obs}}) = \text{Prob}(I)$$

Data that are missing because a researcher dropped the test tubes or survey participants' accidentally skipped questions are likely to be MCAR. The case deletion

method for dealing with missing values seems to give good results in this situation. Unfortunately, most missing data are not MCAR.

In the context of supervised classification, the missingness in MCAR is unrelated to the values of any attributes or even to the classes, whether missing or observed. This mechanism is more suitable for data to be used in an unsupervised classification task.

ii) *Missing at random (MAR)*: If the probability of an instance having a missing value for an attribute depends on a known value, such as the class to which the instance belongs, but does not depend on the missing data itself, the missingness is classified as MAR. Again, using the example of weight and age, if the likelihood that a person will provide his or her weight varied according to an individual's weight but not his or her age, then the missing data is considered to be MAR. Formally,

$$\text{Prob}(I/x_{\text{miss}}, x_{\text{obs}}) = \text{Prob}(I/x_{\text{obs}})$$

Most missing data methods are designed under this assumption. In MAR, cases with incomplete data differ from cases with complete data, but the pattern of missingness of the data is traceable or predictable from other features in the database. In other words, the actual features where data are missing are not the cause of the incomplete data. Instead, the cause of the missing data is due to some other external influence.

MAR is an assumption that is justifiable more frequently, though not always. The more relevant and related the predictors that are to be included in statistical models are, the more likely it is that the MAR assumption will be met.

In this thesis work, the MAR mechanism has been considered in two situations:

- a) In datasets with a low percentage of missing values.
- b) In datasets that contained simulated missing values, since the assigned missing values have been assigned at random but proportional to the class size.

An **R** function *imagmiss* has been written to display the distribution of the missing values in the dataset.

Example 1. The image of Figure 6.1 shows the location of the missing values assigned randomly to the three most relevant features of the Iris dataset in 19% of the instances. Notice that a certain pattern exists in the distribution of the missing values.

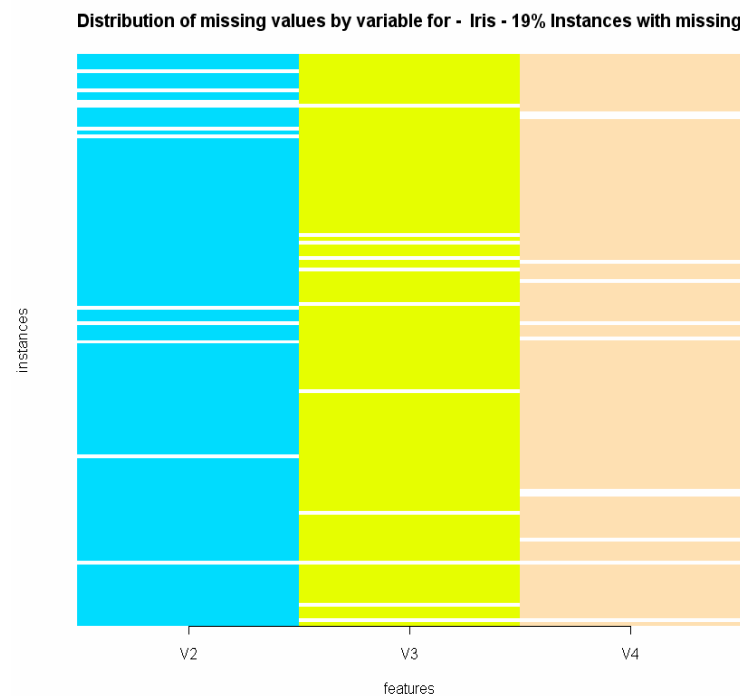


Figure 6.1: An example of missing values obtained by a MAR mechanism.

iii) *Non-ignorable or Not Missing at Random (NMAR)*: If the probability that a value of an attribute is missing depends on the missing data itself, then it is considered to be not missing at random. An example of this would be if the likelihood of an individual providing his or her weight varied according to a person's weight in each age category. Typically this type of missing data is the hardest condition to deal with, but unfortunately, the most likely to occur as well. It commonly occurs when interviewed people do not want to reveal something very personal or unpopular about

themselves. Some methods for dealing with missing data can give highly biased results for NMAR data.

In NMAR the pattern of data missingness is non-random and it is not predictable from other features in the database. If a participant in a weight-loss study does not attend a weigh-in due to concerns about his weight loss, his data are missing due to non-ignorable factors. In contrast to the MAR situation outlined above where data missingness is explainable by other measured features in a study, non-ignorable missing data arises due to the fact that the data missingness pattern is explainable --- and *only* explainable --- by the very feature(s) on which the data are missing.

Example 2: For the *Hepatitis* dataset which has a high percentage of missing values a NMAR mechanism is considered. Figure 2 shows the missing values distribution of *Hepatitis*. Note that *Hepatitis* has a high percentage of instances containing missing values.

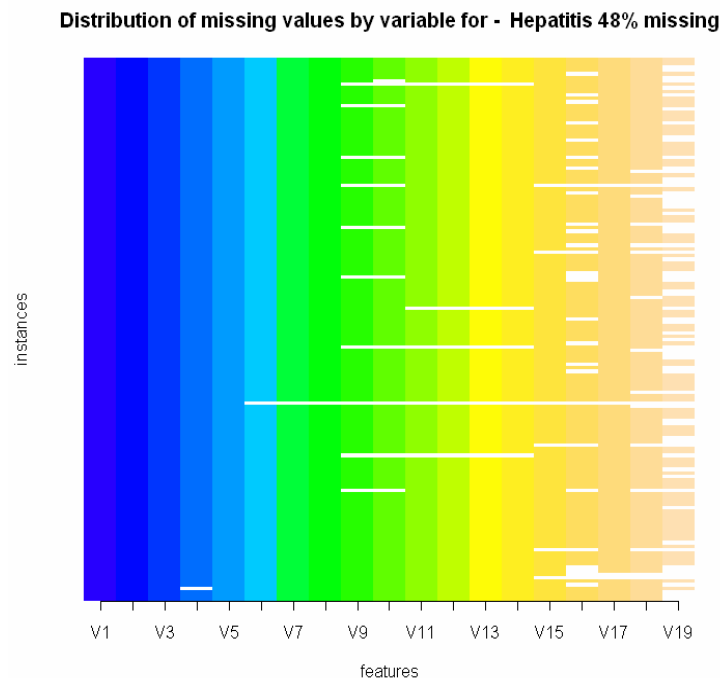


Figure 6.2: An example of missing values obtained by the NMAR mechanism

6.3. *Methods for handling missing data*

In general, methods for treating missing data can be divided into the three categories listed below (Little and Rubin, 2002).

- a) **Case/Pairwise Deletion**, which are the easiest and most commonly applied methods.
- b) **Parameter estimation**, where maximum likelihood procedures that use variants of the Expectation-Maximization algorithm can handle parameter estimation in the presence of missing data. These methods are generally superior to case deletion methods, because they utilize all the observed data, especially when the probability mechanism leading to missingness can be included in the model. However, they suffer from several limitations, including a strict assumption of a model distribution for the features, such as a multivariate normal model, which has a high sensitivity to outliers and a high degree of complexity (slow computation).
- c) **Imputation techniques**, where missing values are replaced with estimated ones based on information available in the dataset. The objective is to employ known relationships that can be identified in the valid values of the dataset to assist in estimating the missing values. There are many options varying from naive methods, like mean imputation, to other robust methods based on relationships among attributes.

In this thesis work four methods to treat missing values in supervised classification problems are considered. The chosen techniques are:

- i) The case deletion technique (CD),
- ii) The mean imputation (MI),
- iii) The median imputation (MDI) and,
- iv) The k-nearest neighbor (KNN) imputation.

In the following sections a description of the above four methods for treating missing values in the supervised classification context is given.

6.3.1. *Case Deletion*

Case Deletion is also known as Complete Case Analysis. It is available in some statistical packages and is the default method in many of those programs that include it. This method consists of discarding all instances (cases) with missing values for at least one feature. A variation of this method consists of determining the extent of missing data on each instance and attribute and deleting the instances and/or attributes with high levels of missing data. However, before deleting any attribute, it is necessary to evaluate its relevance to the analysis. Unfortunately, relevant attributes should be kept even with a high degree of missing values.

CD is less hazardous if it involves minimal loss of sample size (minimal missing data or a sufficiently large sample size) and there is no structure or pattern to the missing data. For other situations where the sample size is insufficient or some structure exists in the missing data, CD has been shown to produce more biased estimates than alternative methods. CD should be applied only in cases in which data are missing completely at random (see Little and Rubin (2002)).

6.3.2. *Mean Imputation*

Mean Imputation is one of the most frequently used methods. It consists of replacing the missing data for a given feature (attribute) by the mean of all known values of that attribute. In a supervised classification context, the value is replaced by the mean of all known values of the attribute that are in the class to which the instance with the missing attribute belongs. Let us consider that the value x_{ij} of the k -th class, C_k , is missing then it will be replaced by

$$\hat{x}_{ij} = \sum_{i: x_{ij} \in C_k} \frac{x_{ij}}{n_{jk}}, \quad (6.1)$$

where n_{jk} represents the number of non-missing values in the j -th feature of the k -th class. In some studies the overall mean is used but we consider that this does not take

into account the sample size of the class to which the instance with the missing values belongs.

According to Little and Rubin (2002) among the drawbacks of mean imputation are:

- (a) the sample size is overestimated,
- (b) the variance of the estimator is underestimated,
- (c) the correlation between two features can be negatively biased, and
- (d) the distribution of the new values might be an incorrect representation of the population values because the shape of the distribution is distorted by adding values equal to the mean.

Replacing all missing records with a single value will deflate the variance and artificially inflate the significance of any statistical tests based on it. Surprisingly though, mean imputation has given good experimental results in datasets used for supervised classification purposes (Chan and Dunn, 1972, Mundfrom and Whitcomb, 1998).

6.3.3. *Median Imputation (MDI)*

Since the mean is affected by the presence of outliers it seems natural to use the median instead just to assure robustness. In this case, the missing data for a given feature is replaced by the median of all known values of that attribute in the class to which the instance with the missing value belongs. This method is also a recommended choice when the distribution of the values of a given feature is skewed. Let us consider that the value x_{ij} of the k -th class, C_k , is missing, then it will be replaced by

$$\hat{x}_{ij} = \text{median}\{x_{ij}\}_{i:x_{ij} \in C_k}. \quad (6.2)$$

In the case of a missing value in a categorical feature we can use mode imputation instead of either mean or median imputation. These imputation methods are applied separately in each feature containing missing values. Notice that the

correlation structure of the data is not being considered in the above methods. The existence of others features with similar information (high correlation), or similar predicting power can make the missing data imputation useless, or even harmful.

6.3.4. KNN Imputation (KNNI)

In this method the missing values of an instance are imputed considering a given number of instances that are most similar to the instance of interest. The similarity of two instances is determined using a distance function. The algorithm is shown in Figure 6.3.

The advantages of KNN imputation are:

- (i) k-nearest neighbor can predict both qualitative attributes (the most frequent value among the k nearest neighbors) and quantitative attributes (the mean among the k nearest neighbors).
- (ii) It does not require the creation of a predictive model for each attribute with missing data. Actually, the k-nearest neighbor algorithm does not create explicit models.
- (iii) It can easily treat instances with multiple missing values.
- (iv) It takes in consideration the correlation structure of the data.

The disadvantages of KNN imputation are:

- (i) There are no set criteria for the choice of the distance function to be used. It could be Euclidean, Manhattan, Mahalanobis, Pearson, etc. In this work we have considered the Euclidean distance.
- (ii) The KNN algorithm searches through all instances belonging to a class looking for the most similar instances. This is a very time consuming process and it can be very critical in data mining where large databases are analyzed.

Input: D =Dataset, p =number of features in D , S =set of all features in D

begin{

1. Divide the dataset D into two parts. Let D_m be the set containing the instances in which at least one of the feature values is missing. The remaining instances with complete feature information form a set called D_c .
2. **for** each vector \mathbf{x} in D_m {
 - a) Divide the instance vector into observed and missing parts as $\mathbf{x}=[\mathbf{x}_o;\mathbf{x}_m]$.
 - b) Calculate the distance between the \mathbf{x}_o and all the instance vectors from the set D_c . Use only those features in the instance vectors from the complete set D_c , which are observed in the vector \mathbf{x} .
 - c) Use the K closest instance vectors (K -nearest neighbors) and perform a majority voting estimate of the missing values for categorical attributes. For continuous attributes replace the missing value using the mean value of the attribute in the k -nearest neighborhood. The median could be used instead of the mean.

Output: D , a complete matrix.

Figure 6.3: The KNN imputation Algorithm.

No fixed criteria exist for the choice of k , the number of neighbors, either. Troyanskaya et al. (2001), set the value of k empirically. Several numbers were tried and it was decided to use $k=10$ based on the accuracy of the classifier after the imputation process. The choice of a small k produces a deterioration in the performance of the classifier after imputation due to overemphasis of a few dominant instances in the estimation process of the missing values. On the other hand, a neighborhood of large size would include instances that are significantly different

from the instance containing missing values hurting their estimation process and therefore the classifier's performance declines. For small datasets, a k value smaller than 10 can be used.

6.4. *Other imputations methods*

Several other imputation methods exist, but few of them are suitable for classification tasks. Next, additional methods that have not been used in this work are described briefly.

i) Hot deck Imputation. In this method, a missing attribute value is filled in with a value from an estimated distribution for the missing value from the current data. In Random Hot deck, a missing value (the recipient) of an attribute is replaced by an observed value (the donor) of the attribute chosen randomly. There are also cold deck imputation methods that are similar to hot deck but in this case the data source of the donor must be different from the current data source. For more details see Kalton and Kasprzyk (1986).

ii) Imputation using a prediction model. These methods consist of creating a predictive model to estimate values that will substitute the missing data. The attribute with missing data is used as the response attribute, and the remaining attributes are used as input for the predictive model. The disadvantages of this approach are:

- a) the model estimated values are usually more well-behaved than the true values would be.
- b) if there are no relationships among complete attributes in the dataset and the attribute with the missing data, then the model will not be precise for estimating missing values.
- c) the computational cost is high since we have to build a large amount of models to predict missing values.

This method is not suitable for either high dimensionality datasets or very large datasets having a large number of missing values because it will be very slow.

iii) Imputation using decision trees algorithms. All the decision tree classifiers handle missing values by using built in approaches. For instance, CART replaces a missing value of a given attribute using the corresponding value of a surrogate attribute which has the highest correlation with the original attribute. C4.5 uses a probabilistic approach to handle missing data in both the training and the test sample. In this thesis some comparisons with these method have been done through the use of the *rpart* library available in R

iv) Multiple imputation. In this method, the missing values in a feature are replaced with values drawn randomly (with replacement) from a fitted distribution for that feature. This is repeated a number of times, say $M=5$ times. After each iteration, a classifier can be applied to the "complete" dataset and the misclassification error computed for each dataset. The misclassification error rates are averaged to obtain a single estimate and also to estimate variances of the error rate. More details can be found in Little and Rubin (2002) and Schaffer (1997). There are libraries available in R to perform multiple imputation.

6.5. *Effect of imputation on the misclassification error rate*

In this thesis, four methods for treating missing values were compared according to the effect on the misclassification rate of three classifiers: Linear Discriminant Analysis (LDA), K-nn (KNN), and Rpart (rpart). Twelve datasets, coming from the Machine Learning Database Repository at the University of California Irvine, were used. Four of the datasets used contained missing values and eight of them did not contain missing values. Tests were run using the incomplete datasets and using the complete sets but applying a randomization allocation of certain percentages of missing values to them. The percentage of simulated missing values was varied from 1% to 20%.

To evaluate more precisely the effect of missing value imputation on the accuracy of the classifier only the relevant features in each dataset were used. In doing so, the imputation process was also sped up. The relevant features were

selected using the RELIEF, a filter method for feature selection in supervised classification discussed in chapter 4 (see Acuña et al. (2003) for more details). Batista and Monard (2002) ran a similar experiment but they chose only the three most important features and computed the misclassification error after the imputation of each missing value rather than after the imputation of all values.

In this work, first, each of the four datasets having missing values was passed through a cleaning process where features with more than 30% of missing as well as instances with more than 50% of missing were eliminated. An *R* function *clean* was written to perform this task. The function allows the user to modify the percentage of missing in the instances and attributes to tolerate. This cleaning process is carried out in order to perform the smallest number of imputations on each dataset.

After this first step, the four methods to treat missing values were applied but using only the relevant features. An *R* function *ce.mimp* was written to treat the missing values with the mean, median or mode and the function *ec.knnmimp* was written to treat missing values using k-nn imputation. Once a "complete" dataset was obtained, the 10-fold cross-validation estimates of the misclassification error were computed for the three classifiers under consideration. A similar sequence was carried out for datasets with simulated missing values. The results of the experimentation for the incomplete datasets are presented here. The remaining results appear in chapter 7.

Table 6.1 below offers a description of the incomplete datasets used in this work. It also describes the cleaning process needed to prepare the datasets for analysis. Hepatitis has the largest percent of missing, both overall and in the number of affected instances. Based on the tolerance we have set for our experiments, only Hepatitis will have a column and a row eliminated. After cleaning, the datasets ordered in decreasing order of number of missing are: Hepatitis, CRX, Breast and Heartc. It can be observed from table 6.1, that since only the missing values in the relevant features will be imputed, Hepatitis will only have 18 imputations performed.

Table 6.2 shows the cross validation errors (using the three classifiers: lda, knn and rpart) for each of the incomplete datasets when the observations containing

missing values are eliminated.. The table also shows that ratios between the errors obtained after imputation and the base error for each dataset. To interpret the data, we have applied the criteria that errors that change by 5% or more have been affected by the imputation method.

From this table, we can conclude that in the datasets with a small amount of instances containing missing values, there is not much difference between case deletion and other imputation methods for all of the classifiers used. This is observed by ratios that are close to one.

Table 6-1: Description of incomplete datasets used in this study

Datasets (number of missing)	Heartc (6)	Breast-W (16)	CRX (67)	Hepatitis (167)
Percent of missing overall	0.15 %	0.25 %	0.64 %	5.67 %
Features with missing - % missing	1. V12 (1.32)	1. V7 - (2.29)	1. V1 (1.73)	1. V4 (0.65)
	2. V13 (0.66)		2. V2 (1.74)	2. V6 (0.65)
			3. V4 (0.87)	3. V7 (0.65)
			4. V5 (0.87)	4. V8 (0.65)
			5. V6 (1.30)	5. V9 (6.45)
			6. V7 (1.30)	6. V10 (7.097)
			7. V14 (1.88)	7. V11(3.23)
				8. V12 (3.23)
				9. V13 (3.23)
				10. V14 (3.23)
				11. V15 (3.87)
				12. V16 (18.71)
				13. V17(2.58)
				14. V18 (10.32)
				15. V19 (43.23)
Number of rows with missing:	6	16	37	75
Percent of rows with missing:	1.98	2.29	5.36	48.38
Number of values to be imputed (relevant features only):	6	16	30	18

Table 6-2 Cross-validation errors and ratios for the three classifiers and the four methods used to deal with missing data

Cross-validation error/Datasets with missing	Heartc	Breast-W	CRX	Hepatitis
ECV10(lda) - NA omit	0.1644	0.0367	0.1363	0.1725
ECV10(knn) - NA omit	0.1943	0.0344	0.2505	0.21825
ECV10(rpart) -NA omit	0.1933	0.0442	0.1273	0.199
Err_MI LDA ratio	0.9782	1.0161	1.0633	0.9577
Err_MI KNN ratio	0.9672	1.1168	1.0061	0.9146
Err_MI Rpart ratio	1.0093	1.2101	1.0238	1.0207
Err_MDI LDA ratio	0.9831	0.9989	1.0633	0.981
Err_MDI KNN ratio	0.9584	1.1285	0.9869	0.8361
Err_MDI Rpart ratio	0.9546	1.1680	1.0227	0.9992
Err_KNNI LDA ratio	0.9728	1.0816	1.0633	1.0069
Err_KNNI KNN ratio	0.9626	1.0511	0.9814	0.9777
Err_KNNI Rpart ratio	0.9546	1.1712	1.0124	1.0377

Chapter 7 Experimental Results

7.1. *Programming in R*

R is a computer language which can be considered an implementation of the S language of Becker, Chambers and Wilks (1988) that has adopted the evaluation model of the Lisp dialect, Scheme. R was initially written by Robert Gentleman and Ross Ihaka (1996). The project began as an effort to bring some of the power of modern statistical systems to the class of low-end computers which were generally available for teaching purposes at the time. R has since been created for several platforms, including, UNIX, Windows and MacIntosh.

R is not object oriented in the same sense as Java or C++, but in the sense that the basic entities of the language are objects, i.e. complex entities with type and defined behavior. These objects are characterized by their names and their content, but also by attributes which specify the kind of data represented by the object. For example, consider a feature that takes on the value 1,2, or 3. Such a feature could be an integer variable (i.e., the number of birds in a nest), or the coding of a categorical variable (i.e., sex in a population of crustaceus: male, female, or hermaphrodite). It is clear that the statistical analysis of this feature will not be the same in both cases. With R, the attributes of the object give the necessary information to determine the type of analysis allowed for this feature. Another difference between R and classical object oriented languages is that R doesn't (generally) have methods that are semantically part of objects. However, it does have class-based function dispatch and generic functions, so that a function can do different things for different kinds of objects, and for combinations of objects.

R has elements of being a functional language like the fact that programs are composed of expressions that are turned into function objects that get evaluated. Yet,

it is not a pure functional language because functions can and do have side effects. Other distinctive features of the language are that basic operations are vectorized, and that "lists" in R are really generic vectors where each element can be of a different type.

R has lazy evaluation (objects are not evaluated until their values are required) and weak dynamic typing (a variable can change type at will: `a = 1` ; `a = "a"` is allowed). Conversion between types is often automatic or can be programmed to be so, hence operations on disparate types can often be carried out. Another characteristic of the language is that parameter passing follows call-by-value and copy-on-modify disciplines. This means that arguments are not copied unless the function seeks to mutate their values.

All actions of R are done on objects stored in the active memory of the computer, that is, no temporary files are used. The results of functions are displayed directly on the screen, stored in an object, or written on the disk. Since results are themselves objects, they can be considered as data and analysed as such. The functions available to the user are stored in a library localized on the disk in a directory called `R_HOME/library` where `R_HOME` is the directory where R is installed. This directory contains packages of functions, which are themselves structured in directories. The package named **base** contains the basic functions of the language for reading and manipulating data, some graphic functions, and a few statistical functions. Each package has a directory called `R` with a file named like the package (for instance, `R_HOME/library/base/R/base`). This last file is in ASCII format and contains all the functions of the package.

R currently has a large population of users and collaborators. Its ability to be both a statistical environment and a programming language provides the duality required for the rapid elaboration of code that implements advanced algorithms for statistical analysis. Since the creation of visualizations that would aid in statistical analysis was a primary goal of this work, the ease with which graphics are created in Windows was considered another advantage of this platform. In addition, the

portability of the R language will allow the authors of this work to cross over to the Unix platform to improve memory usage of the code that was written for future work.

7.2. *The Datasets*

A summary of the characteristics of the complete datasets used in this work appears in Table 7.1. The number in parenthesis in the column “Number of Classes” indicates the number of instances in each class of the dataset. The asterisk (*) indicates that some features of the original dataset have not been considered.

Table 7-1 : Information about the datasets used in thesis.

<i>Datasets</i>	Observations				Number of Features
	Number of Instances	Number of classes (Instances per class)	Number of missing values	Percent of instances with missing values	
<i>Iris</i>	150	3(50, 50, 50)			4
<i>Hepatitis</i>	155	2(32, 123)	75	48.38%	19
<i>Sonar</i>	208	2 (111, 97)			60
<i>Heart-c</i>	303	2(164, 139)	6	1.98%	13
<i>Bupa</i>	345	2(145, 200)			6
<i>Ionosphere*</i>	351	2(225, 126)			34
<i>Crx</i>	690	2(383, 307)	37	5.36%	15
<i>Breast-w</i>	699	2(458, 241)	16	2.29%	9
<i>Diabetes</i>	768	2(500, 268)			8
<i>Vehicle</i>	846	4(218, 212, 217,			18
<i>German</i>	1000	2(700,300)			20
<i>Segment*</i>	2310	7(330, 330, 330, 330, 330, 330,			19
<i>Landsat</i>	4435	6(1072, 479, 961, 415, 470, 1038)			36

7.3. *Classifiers used in this thesis*

Throughout this thesis, three classifiers have been used to obtain classification results for the different experiments. These are: Linear Discriminant Analysis (LDA),

K-nearest neighbor (KNN) and Recursive Partitioning (rpart). A brief description of each classification method follows.

Linear Discriminant Analysis (LDA) is a commonly used classification method of parametric nature that uses the assumption of multivariate and equality of variance matrices. This method maximizes the ratio of between-class variance to the within-class variance in any particular data set, thereby guaranteeing maximal separability. LDA tries to draw a decision region between the classes of the dataset using the rule: assign x to the class j that has the closest mean. In general the discriminant function is written as:

$$D = \beta_o + \beta_1 x_1 + \dots + \beta_p x_p,$$

where β_1, \dots, β_p are the discriminant coefficients and x_1, \dots, x_p are the features of the dataset.

Example 1. Given the data on loans from table 7.2, compute the Linear Discriminant Function and plot the decision region.

In Table 7.2 below, the column labeled status contains the class information for the observations. Those observations labeled 0 represent customers whose credit history has been classified as “Bad risk” and those labeled 1 as “No risk”.

In R, the function *lda* from the library MASS is used to obtain the linear discriminant function. After obtaining the linear discriminant function, a scatter plot of the 11 observations is plotted as displayed in Figure 7.1. In this plot, M1 and M2 represent the mean values of the “No risk” class marked with G’s and the “Bad risk” class marked with B’s respectively.

Table 7-2: Credit data for Example 1.

Name	Yearly income	Age	Status
1	18,000	22	1
2	26,000	26	1
3	14,000	31	0
4	20,000	42	1
5	12,000	31	0
6	21,000	24	0
7	35,000	32	1
8	40,000	46	1
9	36,000	37	1
10	22,000	28	0
11	24,000	32	1

The line crossing the plot is the *linear discriminant* function for the data. It can be observed that two observations would be incorrectly classified if this method is used.

**Figure 7.1: Scatter plot of Credit History data with linear discriminant line**

The K-nearest neighbor classifier finds for each observation of the dataset, the k-nearest observations (using a distance measure), and the classification is decided by majority vote, with ties broken at random. If there are ties for the kth nearest observations, all candidates are included in the vote.

Recursive Partitioning (rpart) is a tree structure that uses a classification tree to classify the observations. A classification tree will determine a set of logical if-then conditions (instead of linear equations) for predicting or classifying cases. Figure 7.2 shows a classification tree developed for the Iris dataset. The interpretation of this tree is: if the petal length is less than 2.45, the respective flower would be classified as Setosa. If the petal length is greater than or equal to 2.45, but less than 4.95 and the petal width is less than 1.75, then the respective flower should be classified as Versicolor; else, if the petal length is greater than or equal to 4.95, and the petal width is less than 1.75, then the respective flower belongs to class Virginica. Finally, if the petal width is greater than or equal to 1.75, then the flower should be placed in the Virginica group. Tree classifiers use no implicit assumption about the underlying relationships between the predictor variables and the dependent variable.

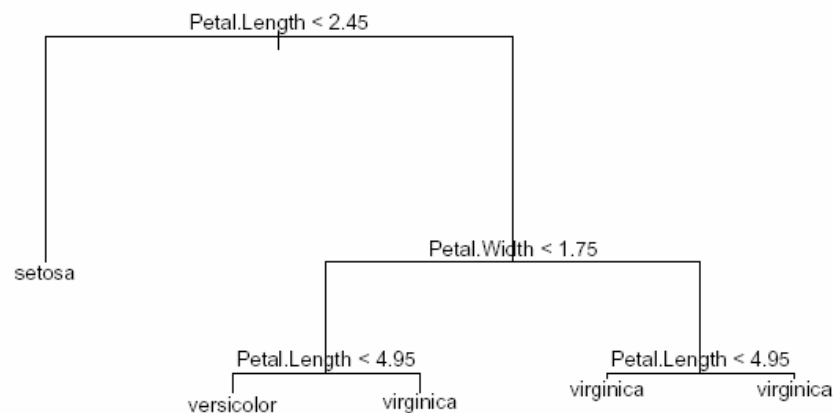


Figure 7.2: Decision Tree for *Iris* dataset

7.4. Normalization

An attribute is normalized by scaling its values so that these fall within a small specified range, for instance from 0 to 1. Normalization is very useful for distance-based algorithms such as k-nearest neighbors, because it helps to prevent attributes with large values from overweighing attributes with small values. In this work, normalization was applied to the datasets to construct parallel coordinate and survey plots and to study the effect of normalization on feature selection for the SFS wrapper method. The filter methods normalize data before determining the relevance as part of their algorithms.

There are many methods for data normalization. The proposed data preprocessing environment will include R functions to implement the methods described below.

Normalization methods included in this study

1) *Min-Max Normalization*:- This type of normalization transforms the data into a desired range, usually [0,1]. The transformation formula for a value $v(i)$ of the attribute A is given by:

$$v'(i) = \frac{v(i) - \min A}{\max A - \min A} * (\text{newmax } A - \text{newmin } A) + \text{newmin } A, \quad (7.1)$$

where $[\min A, \max A]$ is the initial range of the attribute A and $[\text{newmin } A, \text{newmax } A]$ is the new range.

2) *z-Score normalization*: By using this type of normalization, the mean of the transformed set of data points is reduced to zero. For this, the mean and standard deviation of the initial set of data values are required. The transformation formula for a value $v(i)$ of the attribute A is given by:

$$v'(i) = \frac{v(i) - \text{mean}A}{\text{stdev}A}, \quad (7.2)$$

where $\text{mean}A$ and $\text{std_dev}A$ are the mean and standard deviation of attribute A.

3) *Decimal Scaling*: This type of scaling transforms the data into a range between [-1,1]. The transformation formula is given by

$$v'(i) = \frac{v(i)}{10^k}, \quad (7.3)$$

for the smallest k such that $\max(|v'(i)|) \leq 1$.

4) *Sigmoidal normalization*. This method transforms the input data nonlinearly into the range -1 to 1, using a sigmoid function. It calculates the mean and standard deviation of the input data. Data points within a standard deviation of the mean are mapped to the almost linear region of the sigmoid. Outlier points are compressed along the tails of the sigmoidal function. The transformation formula is given by

$$V'(i) = \frac{1 - e^{-a(i)}}{1 + e^{-a(i)}}, \quad (7.4)$$

where $a(i) = \frac{V(i) - \text{mean}A}{\text{stdev}A}$.

Sigmoidal normalization is especially appropriate when you have outlier data points you wish to include in the dataset. It prevents the most commonly occurring values from being compressed into essentially the same values without losing the ability to represent very large outlier values.

5) *Softmax normalization*. It is so called because it reaches "softly" toward its maximum and minimum value, never quite getting there. The transformation is more or less linear in the middle range, and has a smooth nonlinearity at both ends. The whole output range covered is 0 to 1 and the transformation assures that no present value lies outside this range. The transformation formula value $v(i)$ of the attribute A is given by

$$V'(i) = \frac{1}{1 + e^{-a(i)}}, (7.5).$$

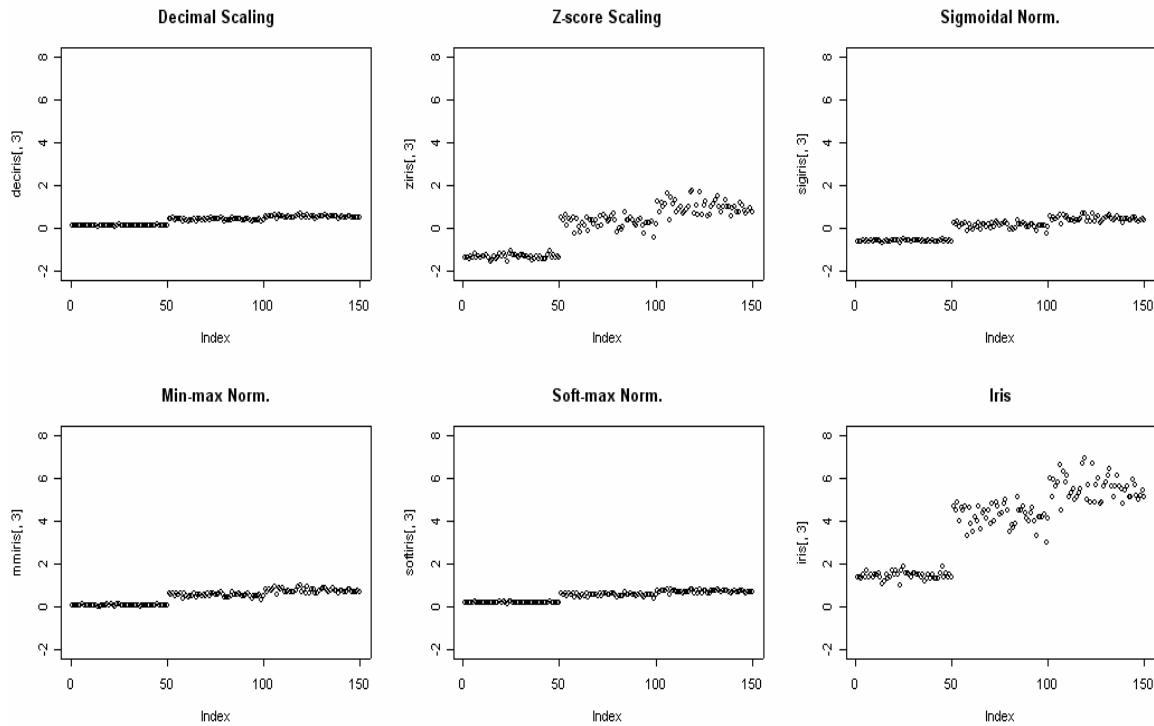


Figure 7.3: Scatterplots of the first attribute of *Iris* before and after applying different normalization methods

Some observed effects of normalization on feature selection and outlier detection

Normalization seems to affect the wrapper feature selection methods, in particular those based on knn and rpart as shown in the Example 2 that follows.

Example 2. Normalize the Diabetes dataset using *z-Score normalization*. Then apply feature reduction methods to reduce the dimensionality of the dataset. Find the misclassification error associated with the new subsets.

Using the functions created in *R* to implement forward selection and floating selection on the normalized dataset, the following results are obtained:

```

> sfs(diabetes,"lda",repet=10)
The best subset of features is:
[1] 7 6 2 8

> sfs(as.data.frame(sdiabetes),"lda",repet=10)
The best subset of features is:
[1] 6 2 7 8

> sfs(diabetes,"knn",repet=10)
The best subset of features is:
[1] 2 6 7

> sfs(as.data.frame(sdiabetes),"knn",repet=10)
The best subset of features is:
[1] 2 8 1 6

> sfs(diabetes,"rpart",repet=10)
The best subset of features is:
[1] 2 3 4

> sfs(as.data.frame(sdiabetes),"rpart",repet=10)
The best subset of features is:
[1] 2 7

```

We can observe how the features selected changed for the KNN and RPART classifiers after the dataset had been normalized.

7.5. Applications of Visualization Techniques

Parallel Coordinate Plots by class

Parallel Coordinate Plots and Survey Plots were created for each dataset used in this work to aid in the identification of relevant features and outliers. Since in a classification context, feature relevance is determined with respect to the entire dataset but outliers are determined with respect to each class, plots were created for each class of the dataset, as well. Below, graphs of the several datasets by class are presented. In each graph outliers that were suggested by the different algorithms are highlighted.

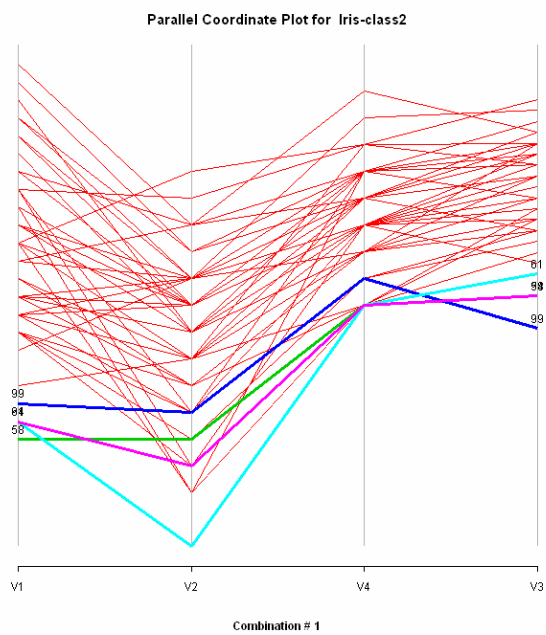


Figure 7.4: Parallel coordinate plot for class 2 of *Iris*

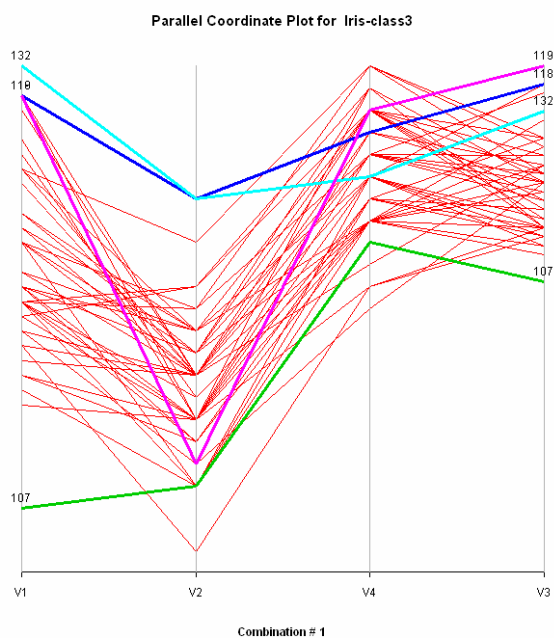


Figure 7.5: Parallel coordinate plot for class 3 of *Iris*

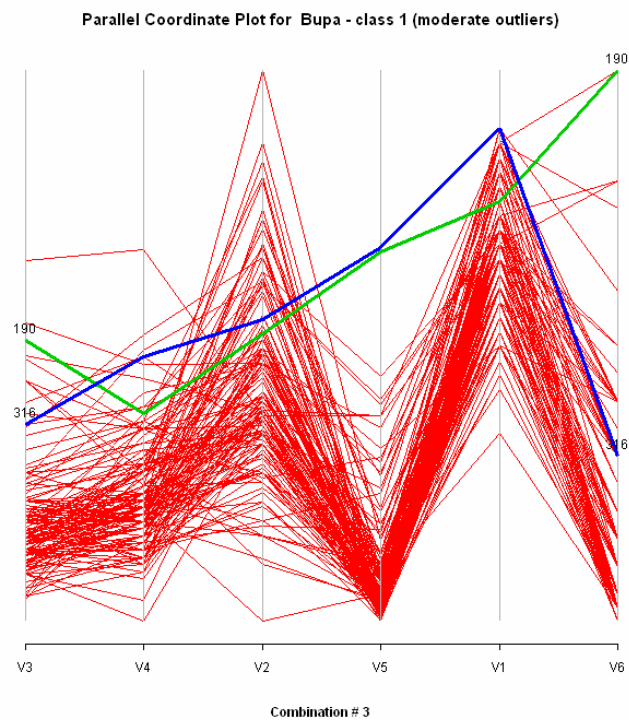


Figure 7.6: Parallel coordinate plot for class 1 of Bupa

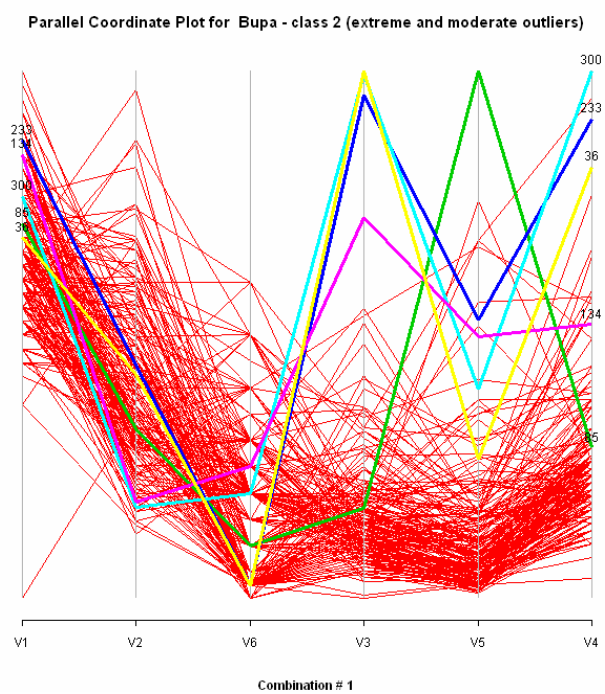


Figure 7.7: Parallel coordinate plot for class 2 of Bupa

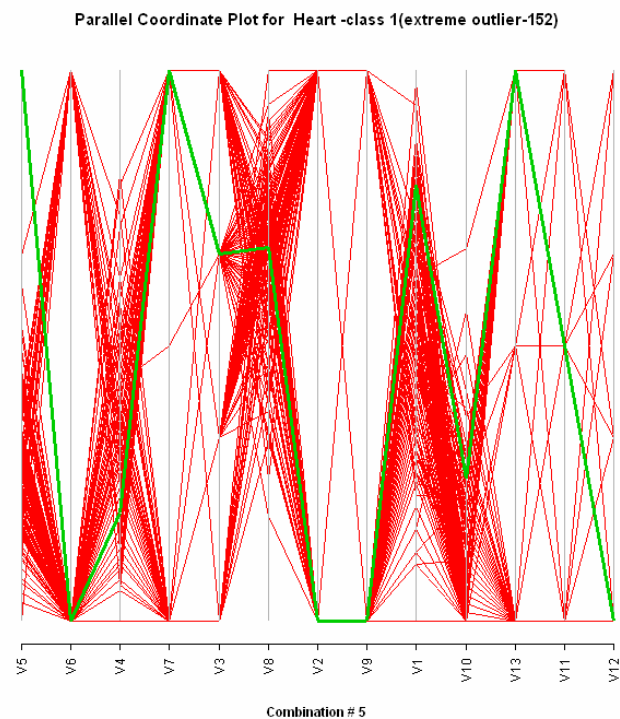


Figure 7.8: Parallel coordinate plot for class 1 of Heart

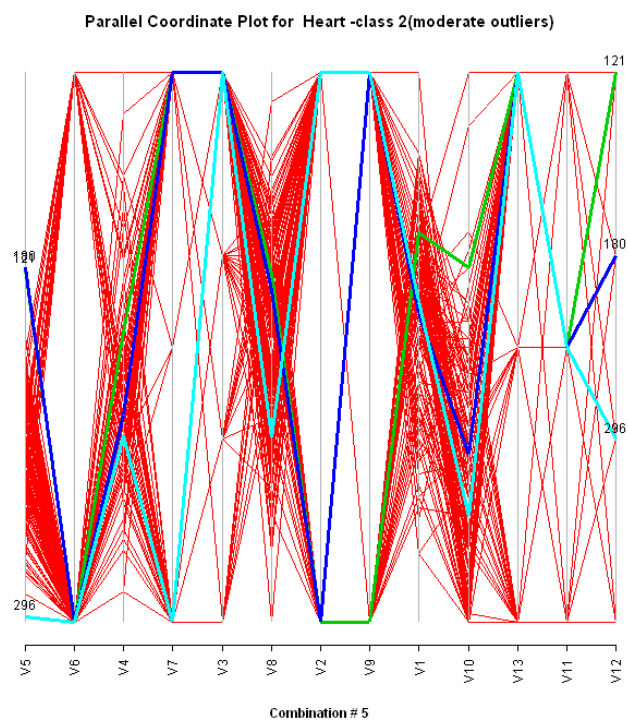


Figure 7.9: Parallel coordinate plot for class 2 of Heart

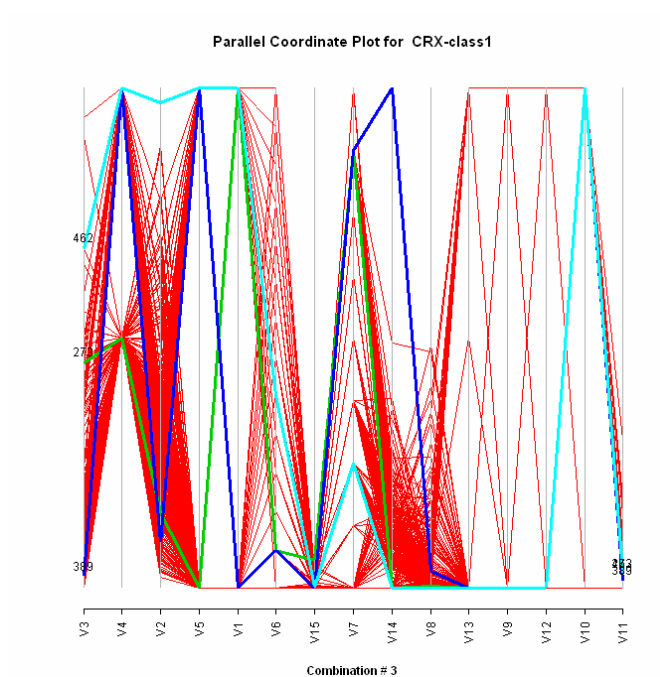


Figure 7.10: Parallel coordinate plot for class 1 of CRX

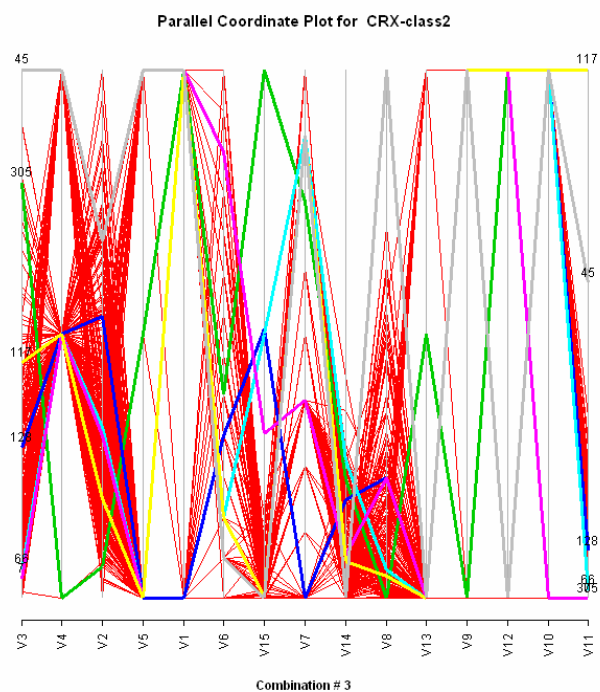


Figure 7.11: Parallel coordinate plot for class 2 of CRX

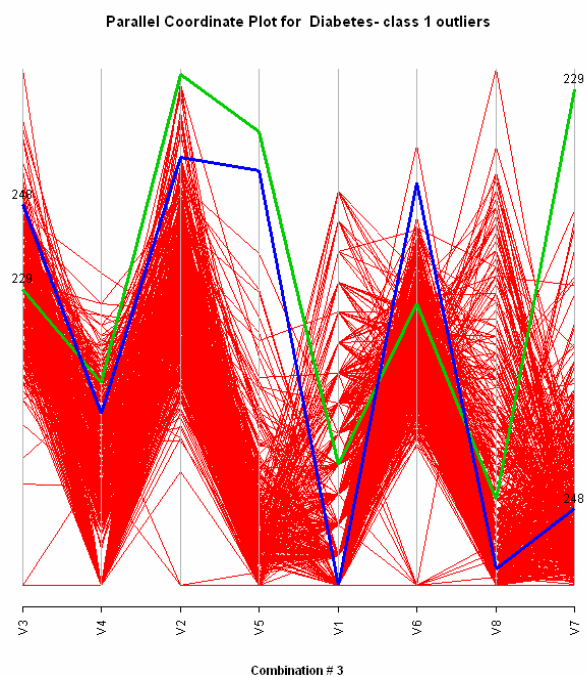


Figure 7.12: Parallel coordinate plot for class 1 of Diabetes

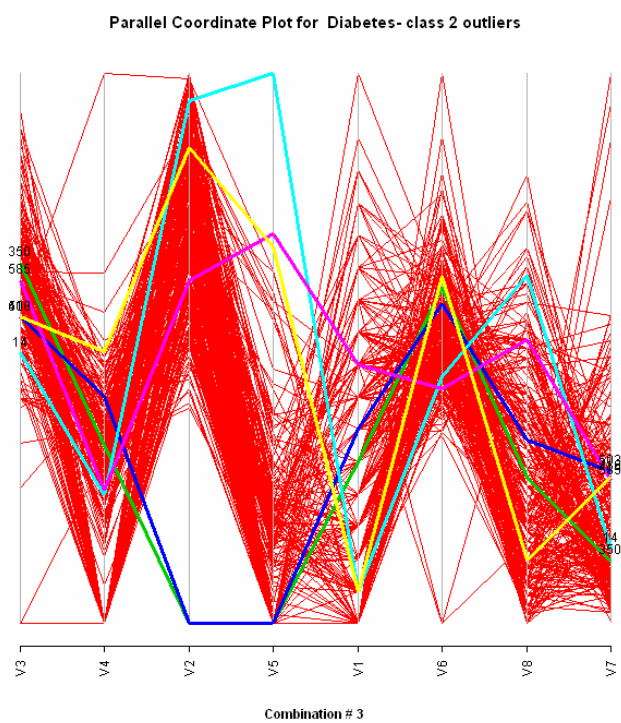


Figure 7.13: Parallel coordinate plot for class 2 of Diabetes

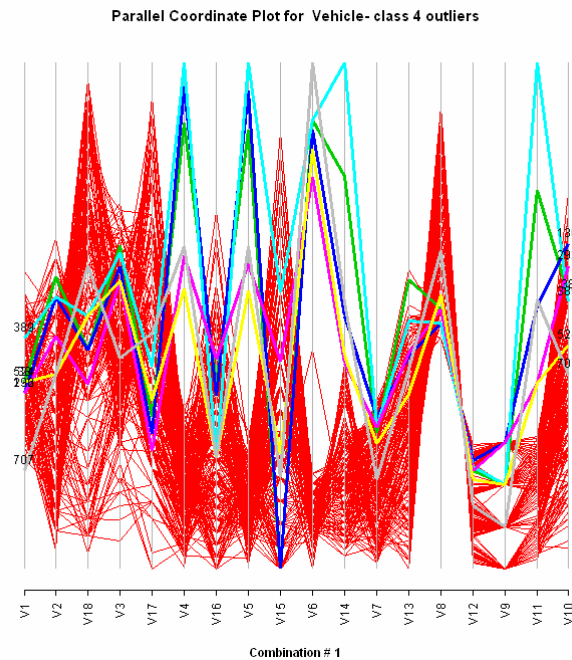


Figure 7.14: Parallel coordinate plot for class 4 of Vehicle

Survey Plots after feature selection

When a data analyst first encounters a dataset, little is known about the relationships among the features of the dataset. The survey plot was mentioned earlier as a possible visualization technique to aid in the identification of relevant features for classification, particularly if allowed the ability to sort on one or more features. Once a feature selection method has been applied, this visualization can further aid in the identification of features that possess stronger discrimination power than others. For example, notice the survey plot of the *Crx* dataset before feature selection presented in Figure 7.15. The attributes selected by the feature selection methods used in this study are shown in Table 7.3.

A survey plot after feature selection allows for the visualization of a display that reflects the discrimination capability of the attributes selected. Figures 7.16 and 7.17 show the surveyplot of the *Crx* dataset when the dataset is ordered by the first

and second attributes (feature 9 and 10). Observe the clustering of colors in the graphs.

Table 7-3: Features selected by different methods for *Crx*.

Feature Selection Method	Relevant Features	% of Features selected
Relief	9, 10, 13	20 %
SFS – LDA	9,15	13.3%
SFS- KNN	9	6.67%
SFS-Rpart	9	6.67%
FFS – KNN	9,15	13.3 %
FFS – LDA	9	6.67%
FFS – Rpart	9	6.67%

With, the help of the survey plot, it can be confirmed that feature 9 has the strongest discrimination power of all other attributes.

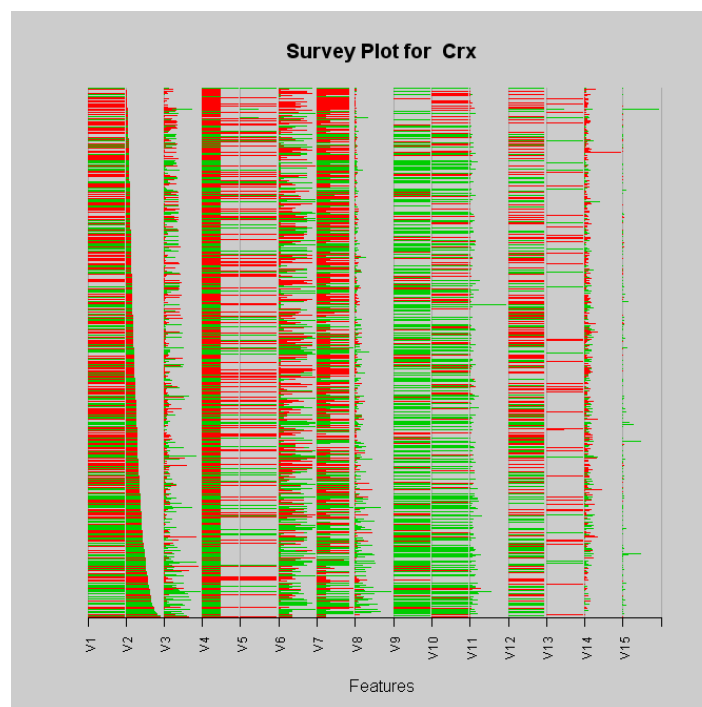


Figure 7.15: Survey plot of *Crx* dataset before feature selection sorted by the second attribute

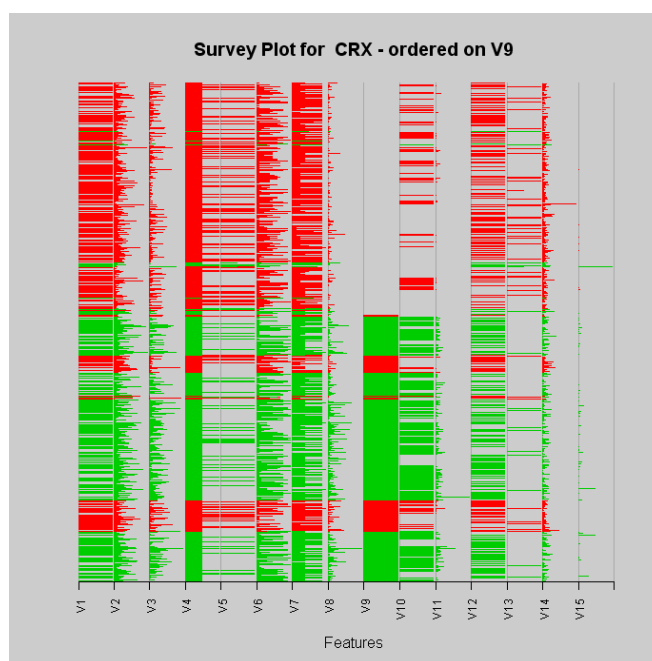


Figure 7.16: *Crx* sorted by V9

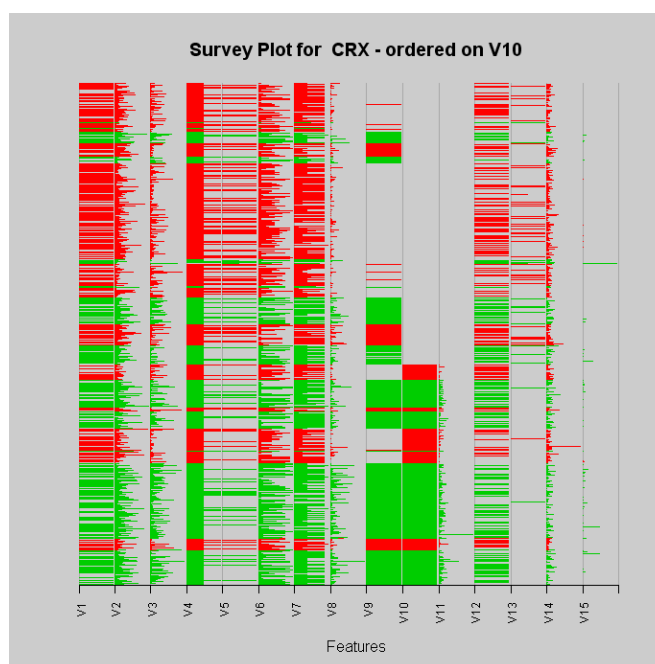


Figure 7.17: *Crx* sorted by V10

7.6. *The effect of outliers on the misclassification error and their in a supervised classification context*

In literature related to outliers, it is frequently mentioned that the presence of outliers affects the performance of classifiers, but there are few studies verifying this claim. This is not the case in a regression context where a large number of studies showing the effect of outliers in regression problems can be found. Two main aspects in supervised classification are feature selection and the misclassification error rate. In this thesis, an evaluation of the effect of outliers in these aspects is considered. The *Iris* and *Bupa* datasets will be used to show the effect of outliers.

Example 3: Use the *Iris* dataset to show the effect of outliers on feature selection and the estimation of the misclassification error rate.

Using all the criteria described in Chapter 4 and the visual help provided by the parallel coordinate plot to decide about the doubtful outliers, the following outliers have been detected in the *Iris* dataset.

Outliers in class 1: (7)

16, 15 ,34, 42, 44, 24, 23

Outliers in class 2 (7)

71, 63 ,58 ,61, 94, 99, 69

Outliers in class 3 (5)

107, 119, 132, 118 ,120

In Table 7.4, the misclassification error of three classifiers: LDA, knn and rpart, has been computed using the original sample, the original sample without outliers and the original sample extracting a random sample of size equal to the number of outliers.

Table 7-4: The misclassification error rate for the LDA, knn and rpart classifiers using three different types of samples

	Original Sample	Original sample without outliers	Original sample excluding a random sub-sample
LDA	2.02	1.54	2.30
Knn(k=1)	4.05	2.35	4.10
Rpart	6.69	2.90	7.32

Notice that all three classifiers are affected when outliers are removed, whereas there is only a minimum change on the misclassification error when a random subsample of instances is removed.

Table 7.5 shows the feature selected using the three types of samples described before. The feature selected methods used here are the sequential forward selection (SFS) with the three classifiers used in Table 7.5 and the Relief method. Some differences can be observed between the subset of features selected by the four methods.

Table 7-5: Features selected using SFS and Relief for the three type of samples

	Original Sample	Original sample without outliers	Original sample excluding a random subsample
SFS(lda)	4,2	4,2	4,3
SFS(knn)	4,3	4,3	4,3
SFS(rpart)	4	4	4
Relief	2,3,4	4,3	4,3

Finally, in Table 7.6, the misclassification error rates of the three classifiers after forward feature selection and for the three types of samples are shown.

Table 7-6: Misclassification error rate after SFS for the three type of samples

	Original Sample	Original sample without outliers	Original sample excluding a random subsample
LDA	3.70	2.33	5.31
knn(k=1)	4.01	1.87	4.80
Rpart	5.29	2.29	5.25

Notice that the lower misclassification errors are obtained for samples where the feature selection is performed after eliminating outliers. Another option for treating outliers is to treat them as if they were missing values and impute the values. Some data analysts prefer the latter because it avoids the loss of sample size but others dislike this method because it can create bias on the estimation.

Example 4: Use the *Bupa* dataset to show the effect of outliers on feature selection and the estimation of the misclassification error rate.

Using all the criteria described in Chapter 7 and the visual help provided by the parallel coordinate plot to decide about the doubtful outliers, the following outliers have been detected in the *Bupa* dataset.

Outliers in class 1: (22)

168, 175, 182, 190, 205, 316, 317, 335, 345, 148, 183, 26, 1 311, 25, 167, 189, 312
326, 343, 313, 20, 22

Outliers in class 2 (26)

36, 77, 85, 115, 134, 179, 233, 300, 323, 331, 342, 111, 139, 252, 294, 307, 123, 186,
286, 2, 133, 157, 187, 224, 278, 337

In Table 7.7, the misclassification error of three classifiers: LDA, knn and rpart, have been computed using the original sample, the original sample without outliers and the original sample extracting a random sample of size equal to the number of outliers.

Table 7-7 : The misclassification error rate for the LDA, knn and rpart classifiers using three different types of samples

	Original Sample	Original sample without outliers	Original sample excluding a random sub-sample
LDA	31.82	26.23	31.17
Knn(k=7)	31.55	27.65	32.26
Rpart	31.86	33.24	35.07

Notice that LDA and knn are the classifiers that are most affected, while Rpart is the least affected. The latter makes sense since it is well known that Rpart is a classifier that is robust to outliers.

Table 7.8 shows the features selected using the three types of samples described before. The feature selection methods used here are the sequential forward selection (SFS) with the three classifiers mentioned earlier along with the Relief method. Differences can be observed between the subset of features selected by the four methods.

Table 7-8: Features selected using SFS and Relief for the three type of samples

	Original Sample	Original sample without outliers	Original sample excluding a random subsample
SFS(lda)	5,4,3,6	5,3,4	5,4,3,6
SFS(knn)	5,3,1	5,3,1,6	5,3,4,1
SFS(rpart)	5,3,6,2	5,3,2	5,3,2
Relief	6,3,4	4,2,5,3	2,4,3

Finally, in Table 7.9, the misclassification error rates of the three classifiers after feature selection and for the three types of samples are shown.

Table 7-9 Misclassification error rate after feature selection for the three type of samples

	Original Sample	Original sample without outliers	Original sample excluding a random subsample
LDA	34.94	26.72	35.62
knn(k=7)	36.53	30.65	40.99
Rpart	37.47	32.48	39.78

Notice that the lower misclassification errors are obtained for samples where the feature selection is performed after eliminating outliers. Another option for treating outliers is to treat them as if they were missing values and impute the values. Some data analysts prefer the latter because it avoids the loss of sample size but others dislike this method because it can create bias on the estimation.

7.7. The effect on the misclassification error rate for both filters and wrappers

Figures 7.18 and 7.19 show the percentages of features selected by the filter and wrapper methods. Among the wrapper methods, the SFFS performs better than SFS, since they both select similar percentages of features (a lower percentage of features than the filter methods) and they both obtain similar classification accuracies, but SFFS has a faster computation time. Among the filters methods, FINCO appears to have the smallest percentage of features selected.

In Tables 7.10 through 7.12, misclassification errors that have deteriorated are those that have increased by at least 5% after feature selection was applied. It can be

observed from these tables that wrapper methods are more effective than filter methods in reducing the misclassification error rate.

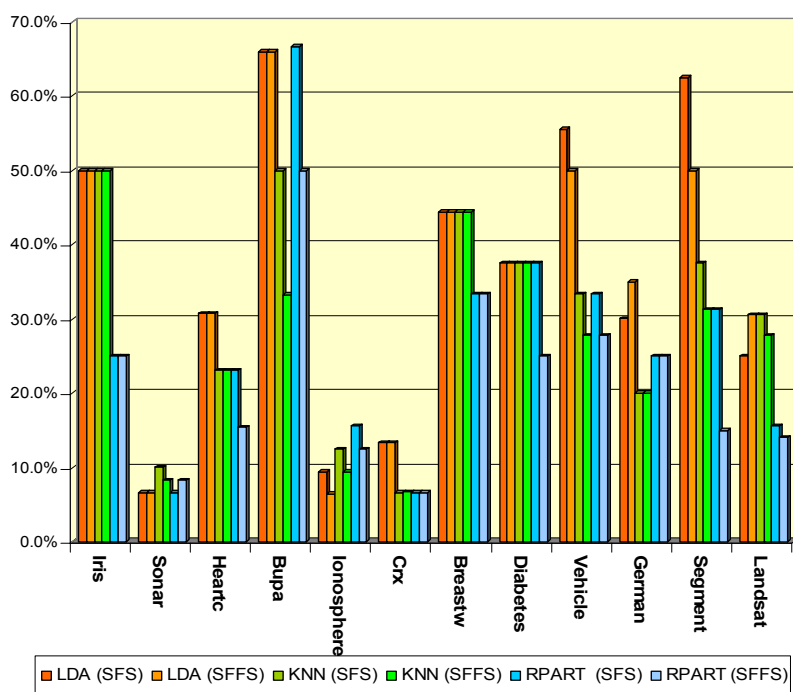


Figure 7.18: Percentage of features selected by the wrapper methods

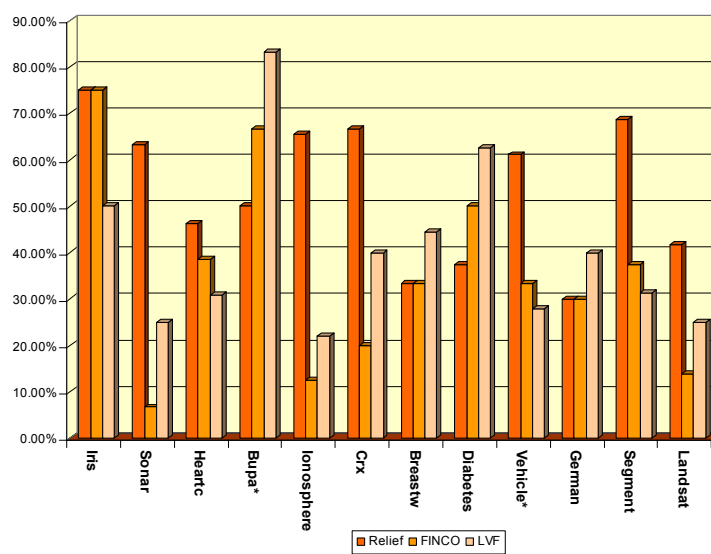


Figure 7.19: Percentage of features selected by the filter methods

Table 7-10: Misclassification error ratios of after-feature-selection to before-feature-selection (LDA classifier)

Dataset	SFS	SFFS	RELIEF	FINCO	LVF
<i>Iris</i>	1.85	1.88	1.58	1.50	2.14
<i>Sonar</i>	1.01	0.82	1.06	1.00	0.94
<i>Heartc</i>	0.94	0.94	0.99	1.18	1.40
<i>Bupa</i>	1.02	1.01	1.33	1.42	1.08
<i>Ionosfera</i>	1.13	1.19	1.05	1.25	1.16
<i>Crx</i>	1.00	1.00	1.01	1.01	1.01
<i>Breastw</i>	0.94	0.93	1.34	0.99	1.15
<i>Diabetes</i>	1.01	1.01	1.46	1.00	1.01
<i>Vehicle</i>	1.11	1.16	1.24	1.93	2.31
<i>German</i>	1.00	1.00	1.05	1.20	1.12
<i>Segment</i>	0.98	1.00	0.55	1.38	1.42
<i>Landsat</i>	1.01	1.02	1.03	1.14	1.14
Mean	1.08	1.08	1.14	1.25	1.32
Median	1.01	1.01	1.06	1.19	1.15

Table 7-11: Misclassification error ratios of after feature selection to before feature selection (KNN classifier)

Dataset	SFS	SFFS	RELIEF	FINCO	LVF
<i>Iris</i>	0.98	0.97	1.16	1.17	1.00
<i>Sonar</i>	1.03	1.27	0.81	1.43	1.02
<i>Heartc</i>	0.48	0.48	0.55	1.07	1.05
<i>Bupa</i>	1.04	1.05	1.19	1.35	1.25
<i>Ionosfera</i>	0.46	0.43	0.79	0.67	0.84
<i>Crx</i>	0.45	0.45	0.47	0.99	1.27
<i>Breastw</i>	1.08	1.08	1.51	1.20	1.20
<i>Diabetes</i>	0.96	1.02	1.28	0.98	0.95
<i>Vehicle</i>	0.82	0.84	1.00	1.32	1.22
<i>German</i>	0.78	0.77	0.84	1.02	1.01
<i>Segment</i>	0.67	0.70	1.97	2.36	2.60
<i>Landsat</i>	1.10	1.10	1.09	1.54	1.36
Mean	0.82	0.85	1.06	1.26	1.23
Median	0.89	0.91	1.05	1.19	1.13

Table 7-12: Misclassification error ratios of after feature selection to before feature selection (RPART classifier)

Dataset	SFS	SFFS	RELIEF	FINCO	LVF
Iris	0.70	0.70	1.00	1.03	1.00
Sonar	0.83	0.75	1.01	0.92	0.93
Heartc	0.77	1.35	0.94	0.85	1.19
Bupa	0.97	1.03	1.01	1.28	1.26
Ionosfera	0.70	0.68	0.93	0.84	0.73
Crx	0.97	0.97	0.97	0.97	1.01
Breastw	0.74	0.85	0.93	0.84	0.87
Diabetes	1.01	0.99	1.38	1.01	1.01
Vehicle	0.97	0.98	1.10	1.14	1.29
German	0.96	0.96	1.02	1.11	1.08
Segment	0.98	0.97	1.01	1.01	1.35
Landsat	1.01	0.99	0.99	0.98	1.16
Mean	0.88	0.94	1.02	1.00	1.07
Median	0.96	0.97	1.01	1.00	1.05

7.8. Effect of imputation on the misclassification error of the contaminated real datasets

Tables 7.14 through 7.16 below, show the ratios of the misclassification errors over the base errors of the datasets used in this work for each of the imputation methods studied and the classifiers used. The ratios labeled in red show misclassification errors that suffered a change that was greater than 5%.

In general, small differences can be observed between the results obtained with mean and median imputation. As mentioned before, most of the datasets used in this work contain features with distributions that have outliers in both directions and their effect cancels out. Otherwise, one could expect a better performance from median imputation. The KNN imputation method yields somewhat better results than the other methods.

It can also be observed, that the percent of instances with missing appears to have a larger influence on the effect of classification accuracy than the percent of missing overall. This is seen through the fact that for datasets with small amounts of missing values, little difference is observed between the errors obtained by the imputation methods employed. However, our results suggest that the case deletion method is more likely to yield a highly affected error when there are a high percentage of instances with missing values.

Some datasets appear to be more resistant to a deteriorating misclassification error, even for large percentages of instance with missing. For example, *Bupa* and *Ionosfera* under the KNN classifier resist close to 50 and 80 percent of instances with missing respectively, and German and Segment resists close to 80% under the LDA classifier. It can also be observed that some datasets can be reduced severely (case deletion due to missing) and yet reflect a misclassification error that has only been slightly affected. Some cases are: Bupa under the KNN and RPART classifiers, Diabetes under the KNN classifier and Segment under the LDA classifier.

Table 7-13: Ratios of Cross Validation Errors using KNN

Dataset	% Missing (overall)	% Missing (instances)	Base error	CD Ratio	MI Ratio	MDI Ratio	KNNI Ratio
Iris	1	2	4.68	1.0299	1.0278	1.0043	1.0385
	7	21.33	4.68	1.2586	1.0171	1.0021	0.9936
	13	33.33	4.68	0.9145	0.4872	0.5662	0.7350
Sonar	1	33.65	14.74	1.1811	0.9851	1.035	0.998
	3	69.71	14.74	1.6323	0.7802	0.8304	0.8813
	5	81.73	14.74	1.8562	0.9064	0.9281	0.8996
Heartc*	5	31.64	18.55	1.0421	0.9342	0.9536	0.9369
	11	56.9	18.55	0.6367	0.9574	0.9655	0.834
	21	79.46	18.55	0.8216	0.7121	0.7596	0.8259
Bupa	3	8.69	36.49	0.9411	0.9792	0.9553	0.9553
	11	30.14	36.49	0.9619	0.9197	0.9164	0.9268
	19	46.95	36.49	1.0271	0.9720	1.0022	0.9942
Ionosphere	1	19.65	13.28	1.1017	0.9428	0.9368	0.9691
	5	65.24	13.28	1.3118	0.9518	0.9729	0.948
	9	86.61	13.28	1.4287	1.0399	0.9654	1.0301
Crx*	3	23.73	25.19	0.9575	0.9619	0.9841	0.9266
	13	70.59	25.19	1.1679	0.9547	0.9984	0.9758
	21	87.28	25.19	1.395	0.7527	0.8384	0.8634
Breastw	3	14.64	3.41	0.9677	0.9736	0.956	0.9765
	11	45.09	3.41	0.9795	0.827	0.827	0.8387
	21	66.18	3.41	0.6217	0.5631	0.5777	0.607
Diabetes	3	14.32	27.47	1.0076	0.9687	0.9709	0.9651
	11	45.31	27.47	0.8580	0.8198	0.7674	0.7951
	21	66.88	27.47	0.9884	0.6858	0.5872	0.7477
Vehicle	5	42.08	34.87	1.1012	1.0419	1.0023	0.9535
	13	74.82	34.87	1.1557	0.9687	0.9415	0.9295
	21	92.55	34.87	1.2234	0.916	0.9036	0.8652
German	5	48.7	29.78	1.091	0.9681	0.9785	0.954
	13	83.7	29.78	1.0121	0.823	0.8251	0.9103
	19	93	29.78	1.1061	0.8103	0.7898	0.8408
Segment	5	43.03	4.64	1.403	1.3448	1.3168	0.9461
	13	77.96	4.64	2.028	1.5172	1.4224	1.1444
	21	92.46	4.64	1.9655	1.653	1.4677	1.0927

Table 7-14: Ratios of Cross Validation Errors using LDA

Dataset	% Missing (overall)	% Missing (cases)	Base error	CD_Ratio	MI_Ratio	MDI_Ratio	KNNI_Ratio
Iris	1	2	3.18	0.9088	1.2013	1.1698	1.1447
	7	20.67	3.18	1.0503	1.0629	1.044	0.8868
	13	32.67	3.18	1.2013	0.934	0.9937	0.956
Sonar	1	31.25	26.6	1.1229	0.9996	0.9993	0.9835
	3	65.86	26.6	1.1891	0.9579	0.9741	0.9827
	5	81.73	26.6	1.741	0.8748	0.8797	0.879
Heartc*	5	26.93	16.51	0.785	0.8989	0.8546	0.9485
	11	50.16	16.51	1.0903	0.8807	0.8328	0.9219
	21	75.75	16.51	0.7117	0.7838	0.6445	0.8262
Bupa	3	8.4	35.14	0.9807	1.0051	1.0062	0.9713
	11	29.56	35.14	1.0148	1.0996	1.1104	0.9878
	19	47.53	35.14	1.0976	1.0683	1.0461	0.9693
Ionosphere	1	19.37	16.59	0.9385	0.9669	0.9740	0.9747
	5	66.67	16.59	1.3207	0.956	0.94274	0.9675
	9	86.04	16.59	1.6799	0.9210	0.912	0.9542
Crx*	3	23.58	13.62	1.105	0.9677	0.9662	0.9802
	13	71.05	13.62	0.8935	0.8767	0.8767	0.9192
	21	88.36	13.62	1.2071	0.7944	0.7863	0.7863
Breastw	3	13.32	3.66	0.9836	1.0055	0.9672	1.0683
	11	43.77	3.66	1.2787	0.9453	0.9809	1.0328
	21	70.13	3.66	1.3798	0.8005	0.847	0.9481
Diabetes	3	14.32	24.64	1.0341	1.0142	1.011	1.011
	11	45.05	24.64	0.9826	0.9688	0.9692	0.9765
	21	69.53	24.64	1.0629	0.9436	0.9224	0.9322
Vehicle	5	38.65	29.15	1.0621	1.0388	1.0415	0.9897
	13	74.11	29.15	1.0604	1.1832	1.1942	0.9883
	21	90.89	29.15	1.1252	1.1979	1.1485	1.1235
German	5	48.4	24.38	1.0685	0.9934	0.9959	1.0008
	13	84.8	24.38	1.0665	0.9795	0.9311	0.9828
	19	94.6	24.38	1.1952	0.9081	0.8831	0.9680
Segment*	5	42.9	9.15	0.9705	1.0186	1.0240	1.0021
	13	78.74	9.15	0.906	1.0219	1.0317	0.9661
	21	92.16	9.15	0.9792	0.8536	0.8404	0.8175

Table 7-15: Ratios of Cross Validation Errors using RPART

Dataset	% Missing	% Missing	Base error	CD Ratio	MI Ratio	MDI Ratio	KNNI Ratio
Iris*	1	2	6.66	1.03904	1.003	1.02102	1.02102
	7	20	6.66	0.7958	0.43994	0.46246	0.43544
	13	33.33	6.66	0.98498	0.99099	1.06456	1.05706
Sonar*	1	30.28	26.97	1.08194	1.00111	1.03411	0.94549
	3	65.38	26.97	1.26993	1.02966	1.01965	1.00742
	5	84.61	26.97	0.5191	0.92659	0.93585	0.91991
Heartc*	5	29.62	17.75	0.97239	1.07211	1.04507	1.01915
	11	58.25	17.75	1.47606	0.92958	0.87268	1.03437
	21	83.16	17.75	1.36338	1.11944	1.08169	0.96789
Bupa**	3	8.98	37.87	0.96884	0.96462	0.94243	0.98125
	11	30.72	37.87	0.90863	0.85556	0.88566	0.85001
	19	47.53	37.87	0.98495	0.70003	0.70267	0.76076
Ionosphere	1	19.65	12.07	0.87655	1.01243	1.02568	0.98592
	5	68.94	12.07	0.83099	0.96852	0.86578	0.99751
	9	86.61	12.07	1.27258	0.93455	0.74731	0.93621
Crx*	3	23.12	12.67	1.12628	1.05209	1.05288	1.08051
	13	74.12	12.67	0.82794	0.90845	0.98185	0.93449
	21	87.9	12.67	1.26046	0.75533	0.69613	0.68903
Breastw*	3	13.61	4.69	0.9979	0.9702	0.9787	0.9936
	11	44.5	4.69	1.2559	0.791	0.7932	0.7847
	21	68.37	4.69	2.0192	0.9574	0.9211	0.9424
Diabetes	3	14.45	27	0.9756	0.98	0.9682	0.9733
	11	45.7	27	0.9822	0.7559	0.7707	0.7896
	21	68.22	27	1.0822	0.6537	0.6537	0.8107
Vehicle	5	39.59	34.58	1.0743	1.0234	1.0463	1.0272
	13	75.76	34.58	1.1212	0.9501	0.9286	0.9569
	21	90.3	34.58	1.4685	0.9069	0.895	0.9633
German	5	46.4	28.06	1.0321	0.9608	0.9797	0.9704
	13	84.3	28.06	1.1525	0.7719	0.788	0.8378
	19	95.3	28.06	1.1586	0.6508	0.729	0.7659
Segment	5	43.46	8.23	1.079	1.0316	1.0024	1.017
	13	79.3	8.23	0.9101	0.8566	0.7995	0.9405
	21	92.2	8.23	2.2819	0.7533	0.7436	0.932

Chapter 8 Conclusions and Future Projections

8.1 Conclusions

The experiments carried out sustain the belief that datasets taken from real world applications are frequently “dirty”, that is, they may contain missing values and/or both univariate and multivariate outliers. Both the existence of missing values and the existence of outliers has an effect on the misclassification error of datasets used in a supervised classification context. Increasing percentages of either can have a detrimental on the results of the classification process.

Through this study, it was repeatedly observed that feature selection could prove to be a valuable tool for improving the time complexity of many other algorithms. If a subset of features can be chosen that maintains the intrinsic characteristics of the original dataset, then all other computations related to the classification of the observations in the dataset will be less time consuming. One limitation is that the most common complexity of the feature selection algorithms themselves is $O(np^2)$, which deteriorates the performance of these algorithms as the dimensionality and the number of observations increases.

During this work, visualization techniques have proven to be extremely useful for data exploration to identify patterns in missing data, as well as to aid in the identification of relevant features and the existence of outliers, early during the data analysis process. The visualization techniques included in the data preprocessing package of functions that was created can also assist in identifying correlations and patterns among attributes. Limitations to these visualizations exist, for example, the “cross-over” and “clutter” problems that occur as the size of the datasets grows.

After studying empirically the effect of the different data preprocessing techniques on the misclassification error of the supervised classification datasets used

in this study, the following functions have been included to form a data preprocessing environment for the Windows *R* platform. The environment will include functions to:

- Clean datasets by eliminating rows and columns with a percent of missing above a given tolerance provided by the user.
- Perform imputation using mean, median or knn methods
- Perform feature selection using filter algorithms such as: RELIEF, FINCO, or Las Vegas and wrapper methods such as: forward or floating sequential selection methods.
- Detect outliers using Bay's or LOF algorithms, clustering algorithms and robust estimators.
- Normalize data using min-max, decimal scaling, z-score, sigmoidal and softmax normalizations
- Visualize data using parallel coordinate or survey plots.

8.2. *Future Projections*

This study has been carried out using datasets that vary in dimensionality from four attributes to sixty and in number of observations from 150 to over 4435. Future investigations would involve extending the use of the functions created to a data mining context using datasets of very-high dimensionality and very-large datasets. Though the algorithms created were promising in terms of their effectiveness and their state-of-the-art character, modifications to all algorithms could be proposed in the attempt to improve the experimental running times. Modifications that could be investigated, that include the use of alternative data structures and search algorithms for the construction of the distance tables used in the outlier detection algorithms.

Other possibilities for investigation in this area include the use of block management suggested by Bay (2003), in which the entire database is not written to memory at the same time, but the database is processed in blocks. This same idea could be applied to the visualization of high dimensional datasets by producing a "slide-show" of panels instead of visualizing the entire dataset. The combination of

feature reduction and visualization techniques could also be investigated for the possibility of obtaining a subset of features that characterized the entire dataset before visualization is carried out. The possibility of parallelizing some of the algorithms could also be explored.

Future projections for this work also include the incorporation of other classifiers and other techniques for data processing and visualization into the environment that has been developed. These techniques could include techniques for detecting and treating data redundancy.

Chapter 9 Appendix

Function Codes

```
imagmiss=function(data,name="")
{
#Function to create a graph of the observations of the dataset
#leaving white lines where data is missing.
#The main idea is to use the original dataset to create
#a temporary dataset containing 1 if a value is found or
#0 is the value is missing. The temporary data set is graphed by column
#changing color for each feature and leaving a blank horizontal line if
#value is missing.
#Uses the R function image from the base library.

#data: the dataset
#name: the name of the dataset as desired in the graph title

ncol=dim(data)[2]
nrow=dim(data)[1]

xaxis=colnames(data)
#xaxis=xaxis[-ncol]

ticks=1:(dim(data)[2]-1)

data=as.matrix(data)
#data=data[,-ncol]
data[which(data!="NA")]=1
data[-which(data!="NA")]=0

#ncol1=ncol-1

for(i in 1:ncol)
{
  data[data[,i]!=0,i]=i
}

x=1:ncol
y=1:nrow
graph.title=paste("Distribution of missing values by variable for - ",name)

image(x,y,t(data),col=c(0,topo.colors(100)),xlab="features",ylab="instances",axes=FALSE,main=(graph.title))
```

```

axis(1,labels=xaxis,at=ticks)

}
*****
parallelplot=function (x, name="",comb=0,class=0,obs=rep(0,0),col = 2, lty = 1, ...)
{

#Function to create a parallel coordinate plot for a dataset.

#x      : the dataset
#name   : name of the dataset
#comb   : integer that represents the order in which the variables will be graphed
#class  : the number of the class to be graphed
#obs    : a list of row numbers of the observations to be highlighted
#col    : color to be used when graphing only one class
#lty    : width of poly-lines
#...    : other parameters accepted by the plot() function of R

#Calculate number and size of classes in matrix
classes=x[,ncol(x)]
len.class=table(classes)
numclass=length(len.class)

#remove classes and calculate size of matrix
x=x[,-ncol(x)]
r=nrow(x)
c=ncol(x)

#find the number of distinct permutations of attributes
numgraphs=combinations(c)

class.list=as.integer(rownames(len.class))

#normalize matrix
xrnms=rownames(x)
x <- apply(x, 2, function(x) (x - min(x))/(max(x) - min(x)))
rownames(x)=xrnms

#if graph of only one class is wanted, construct submatrix
if (class!=0)
{
  same=(classes==class)
  x=x[same,]
  col=class+1
}

graphtitle=paste("Parallel Coordinate Plot for ",name)

# if more than one class, then obtain color vector for each class
if (class==0)
  col=classes+1

```

```

# if comb equals 0, user wants to see ALL graphs of distinct permutations of variables
if (comb==0)
{
  j=1
  def.par=par(mfrow=c(2,2))

  # draw the different graphs, 4 to each screen
  for (k in 1:ncol(numgraphs))
  {
    if (k %% 4==1) {win.graph();par(mfrow=c(2,2))}
    varorder=numgraphs[,j]
    par(font.lab=2,font.sub=2,cex=.75,las=2)
    subtitle=paste("Combination #",j)
    matplot(1:c, t(x[,varorder]), type = "l", col = col, lty = lty, xlab = "", ylab = "", main=
graphtitle, sub=subtitle, axes = FALSE, ...)
    axis(1, at = 1:c, labels = colnames(x[,varorder]))
    for (i in 1:c) lines(c(i, i), c(0, 1), col = "grey70")
    j=j+1
  }
}

# else if only one particular combination is desired
else
{
  varorder=numgraphs[,comb]
  def.par=par(font.lab=2,font.sub=2,cex=.75,las=2,bg=gray(.8))
  subtitle=paste("Combination #",comb)
  matplot(1:c, t(x[,varorder]), type = "l", col = col, lty = lty, xlab = "", ylab = "", main=
graphtitle, sub=subtitle, axes = FALSE, ...)
  axis(1, at = 1:c, labels = colnames(x[,varorder]))
  for (i in 1:c) lines(c(i, i), c(0, 1), col = "grey70")

  # if the user desires to highlight a particular observation
  if (length(obs)!=0)
  {
    obsers=rep(0,0)
    for(i in 1:length(obs)) obsers=c(obsers,which(rownames(x)==obs[i]))
    colors=palette()[:(numclass+2):8]
    if (length(obsers)==1) matlines(1:c,x[obsers,varorder],lty=1,lwd=3,col=colors)
    else matlines(1:c,t(x[obsers,varorder]),lty=1,lwd=3,col=colors)
    par(cex=.75)
    text(1,x[obsers,varorder[1]],rownames(x[obsers,]),pos=3)
    text(c,x[obsers,varorder[c]],rownames(x[obsers,]),pos=3)
    palette("default")
  }
}

invisible()
par(def.par)

```

```

}

combinations=function(numcol)
{
  #A function for constructing the minimal set of permutations
  #of the elements of a vector as described by
  #Wegman in Hyperdimensional Data Analysis (1999)

  combine = rep(0,0)
  variables=seq(1,numcol)
  n=variables[1]
  combine=n
  for(k in 1:(numcol-1))
  {
    ntemp=(n + (((-1)^(k+1))*k))
    if ((ntemp==0) | (ntemp==numcol)) n = numcol
    else n=(n + (((-1)^(k+1))*k)) %% numcol
    combine=c(combine,n)
  }

  combinations = combine
  repet=ceiling((numcol-1)/2)-1
  m=seq(1,repet)
  for (j in m)
  {
    combine=(combine +1) %% numcol
    combine[combine==0]=numcol
    combinations = cbind(combinations, combine)
  }

  colnames(combinations)=NULL

  return(combinations)

}

surveyplot=function(datos,dataname="",orderon=0,class=0,obs=rep(0,0),lwd=1)

{
  #Function that will create a survey plot of a dataset.
  #datos : dataset to be graphed
  #dataname : name of the dataset
  #orderon : integer from 1-(p-1) where p is the number of columns of the dataset
  # : that gives the column by which to order
  #class : number of the class to graph
  #obs : list of rownumbers of the observations to highlight
  #lwd : width of the plotting lines

  if (orderon==0) datos=datos[order(datos[,ncol(datos)],decreasing=FALSE),]
  data=datos

```

```

classes=datos[,ncol(datos)]
r1=dim(datos)[1]
c1=dim(datos)[2]-1
colors=classes+1
r=r1
c=c1

cnames=colnames(datos)[1:c]
rnames=rownames(datos)
graphtitle=paste("Survey Plot for ",dataname)

if (orderon!=0)
{
  neworder=order(data[,orderon],decreasing=T)
  data=data[neworder,]
  classes=classes[neworder]
  if (class==0) colors=colors[neworder]
}

data <- apply(data[,1:c], 2, function(data) (data - min(data))/(max(data) - min(data)))
colnames(data)=cnames

if (class !=0)
{
  data=data[(classes==class),]
  c=ncol(data)
  r=nrow(data)
  rnames=rownames(data)
  colors=4
}

width1=seq(.01, (c)*(.04), by=0.04)
x=rep(1:c,r)
x=x[order(x,decreasing=F)]
x=matrix(x,ncol=c,byrow=F)
for (i in 1:nrow(x)) x[i,2:ncol(x)]=x[i,2:ncol(x)]+width1[1:c-1]

temp=seq(from=0,to=1,length=r+1)
y=rep(temp,c)
y=matrix(y,ncol=c,byrow=F)
y=y[-1,]

op=par(bg = gray(.8),xaxs="i",yaxs="i",yaxp=c(0,1,r+1),las=2,cex.axis=.75)
plot(x,y,xlim=c(1,(c+2)),axes=F,type="n",xlab="Features",ylab="",main=graphtitle)

axislabels=c(colnames(data)," ")

width2=seq(1,c+1)
width1=c(0,width1)
width2=width2 + width1

for (i in width2) lines(c(i, i), c(0, 1), col = "dark gray")

```

```

segments(x,y,(x+data),y,col=colors,lwd=lwd)

axis(1, at = width2, labels = axislabels,pos=0)

if (length(obs)!=0)
{
  old.obs=obs
  obs=rep(0,0)
  if (orderon!=0)
  {
    if (class==0) for(i in 1:length(old.obs)) obs=c(obs,which(neworder==old.obs[i]))
    else for(i in old.obs) obs=c(obs,which(rnames==i))
  }

  else for(i in old.obs) obs=c(obs,which(rnames==i))

  axis(2,at=y[obs,1],old.obs,tick=F,cex.axis=0.5)
  segments(x[obs,1:c],y[obs,1:c],x[obs,1:c]+data[obs,1:c],y[obs,1:c],col="dark blue",lwd=lwd+1)
}

par(op)

}

outbox=function(data,nclass)
{#####
#This function detects univariate outliers simultaneously using boxplots
#data: name of the dataset
#nclass: class number
#####
ncols=dim(data)[2]
out1<-NULL
datatempo=data[data[,ncols]==nclass,1:(ncols-1)]
for(i in 1:(ncols-1)){
  blim=boxplot(datatempo)$stats
  b1=as.numeric(rownames(rbind(datatempo[datatempo[,i]<blim[1,i],],datatempo[datatempo[,i]>blim[5,i],])))
  out1=c(out1,b1)
}
sort(table(out1))
}

baysout=function(D,blocks=5,k=3,num.out=2)
{

#Function that gives the outlyingness measure for the requested number of
#observations using the algorithms developed by Bay.
#D      : the dataset
#blocks : size of block to be processed
#k      : number of nearest neighbors to compute to determine if observation

```



```

#           is a candidate for an outlier
#num.out: number of candidates for outliers and their outlyingness measure
#           to display as output. Must be less than or equal to block number.

D=as.data.frame(D)
nrows=dim(D)[1]
c=0
Out=NULL
rep=ceiling(nrows/blocks)
for (cycle in 1:rep)
{
  block.size=blocks
  if (block.size*cycle<=nrows) block=(block.size*(cycle-1)+1):(block.size*cycle)
  else {block=(block.size*(cycle-1)+1):nrows;block.size=length(block)}
  B=D[block,]
  neighbors=matrix(rep(0,(block.size*k)),block.size,k)
  rownames(neighbors)=rownames(B)
  neighbors=as.data.frame(neighbors)
  for (m in 1:nrows)
  {
    d=D[m,]
    j=1
    flag=0
    reduce = 0
    removeB=rep(0,0)
    removeN=rep(0,0)
    while ((j <= block.size)&(block.size>=1))
    {
      if (!(as.integer(rownames(D)[m])==as.integer(rownames(B)[j])))
      {
        b=B[j,]
        if ((0%in%neighbors[j,])|(distancia(b,d)<maxdist(neighbors[j,])[1]))
        {
          neighbors[j,]=closest(b,d,neighbors[j,],3)
          if (!(0%in%neighbors[j,])&(score(neighbors[j,]) < c))
          {
            removeB=cbind(removeB,j)
            removeN=cbind(removeN,j)
            reduce=reduce + 1
          }
        }
      }
      j=j+1
    }
    if (reduce==dim(B)[1]) flag=1
    else if(reduce != 0)
    {
      block.size=block.size-reduce
      B=B[-removeB,]
      neighbors=neighbors[-removeN,]
    }
  }
}

```

```

    }

    if (flag==0)
    {
        Out=top(Out,neighbors,num.out)
        c=min(Out)
    }
}

xcoord=as.integer(rownames(Out))
plot(Out,main="Instances with Greatest Median distance from K nearest neighbors",ylab="Median
Distance")
text(1:num.out,Out,rownames(Out),pos=1)
return(Out)
}

maxdist=function(dneighbors)
{
    #Function used by baysout to find the largest value of a distance vector
    #returns the value and the index number
    #dneighbors: row vector with the distance of the k nearest neighbors for a given b of B

    dneighbors=as.matrix(dneighbors)
    maxindex=which.max(dneighbors)
    max=dneighbors[maxindex]
    list(value=max,index=maxindex)
}

closest=function(b,d,neigh,k)
{
    # Function used by baysout to select the k vectors that are closest to
    # a given observation
    # b      : instance from B under study
    # neigh  : matrix containing the distance to each of the k neighbors
    #        : rownames are rownames of D
    # d      : is the instance from D under study, must have rowname
    # k      : is number of nearest neighbors

    dist=distancia(b,d)
    if(0%in%neigh) { neigh[which(neigh[]==0)[1]]=dist;new=neigh} else {new = c(neigh,dist)}
    new.sort=new[order(new,decreasing=F)]
    nearest=new.sort[1:k]
    return(nearest)
}

score=function(data)
{
    #Function to determine the score measure that will be used to determine
    #candidates for outliers
    #data    : vector containing distances to k nearest neighbors.

    s=median(as.matrix(data))
    return (s)
}

```

```

}
top=function(O,neighbors,n)
{

#Function that finds the n candidates for outliers that
#were requested by the user.

#O: n x 1 matrix with the median distance from k nearest neighbors
#   of the n top outliers up to the moment.
#   row names are equal to names from original matrix D.

#neighbors: keeps distance from k nearest neighbors of prospective outliers
#           maximum size= blocksize x k, where k is number of nearest neighbors
#

temp=as.matrix(apply(neighbors,1,median))
#rownames(temp)=rownames(ndistance)
out=rbind(O,temp)
out.sort=as.matrix(out[order(out,decreasing=T)])
outliers=as.matrix(out.sort[1:n,])
return(outliers)
}

maxlof=function(data,name="",minpts1=10,minptsu=20)
{

#Function that displays the local outlier factor of each observation in a dataset
#as a list and also as a plot.
#Calls on lofactor.

j=seq(minpts1,minptsu)
maxlofvect=rep(0,dim(data)[1])

for (i in j)
{
  temp=lofactor(data,i)
  maxlofvect=cbind(maxlofvect,temp)
  maxlofvect=apply(maxlofvect,1,max)
}

names(maxlofvect)=rownames(data)

ord.maxlofvect=order(maxlofvect,decreasing=T)
maxlofvect.ord=maxlofvect[ord.maxlofvect]

title1=paste("Plot for lof of ",name)
title2=paste("lower minpts: ",minpts1," upper minpts: ",minptsu)
par(font.sub=2)

plot(maxlofvect.ord,main=title1,sub=title2,xlab="Observation number",ylab="local outlier factor")
text(1:10,maxlofvect.ord[1:10],names(maxlofvect.ord)[1:10],pos=4)

```

```

return(maxlofvect)
}

dist.to.knn2=function(dataset,neighbors)
{

#function returns an object in which each column contains
#the indices of the first k neighbors followed by the
#distances to each of these neighbors

numrow=dim(dataset)[1]

#applies a function to find distance to k nearest neighbors
#within "dataset" for each row of the matrix "dataset"

knndist=rep(0,0)

for (i in 1:numrow)
{
  neighdist=knneigh.vect2(dataset[i,],dataset,neighbors)
  knndist=cbind(knndist,neighdist)
}

return(knndist)
}

knneigh.vect2 =function(x,data,k)
{
#Function that returns the distance from a vector "x" to
#its k-nearest-neighbors in the matrix "data"

temp=as.matrix(data)
numrow=dim(data)[1]
dimnames(temp)=NULL

#subtract rowvector x from each row of data
difference<- scale(temp, x, FALSE)

#square and add all differences and then take the square root
dtemp <- drop(difference^2 %*% rep(1, ncol(data)))
dtemp=sqrt(dtemp)

#order the distances
order.dist <- order(dtemp)
nndist=dtemp[order.dist]

#find distance to k-nearest neighbor
#uses k+1 since first distance in vector is a 0
knndist=nndist[k+1]

```

```

#find neighborhood
#eliminate first row of zeros from neighborhood
neighborhood=drop(nndist[nndist<=knndist])
neighborhood=neighborhood[-1]
numneigh=length(neighborhood)

#find indexes of each neighbor in the neighborhood
index.neigh=order.dist[1:numneigh+1]

# this will become the index of the distance to first neighbor
num1=length(index.neigh)+3

# this will become the index of the distance to last neighbor
num2=length(index.neigh)+numneigh+2

#form a vector
neigh.dist=c(num1,num2,index.neigh,neighborhood)
lvect=numrow-numneigh
extra=lvect*2
extrazeros=rep(0,extra)
neigh.dist=c(num1,num2,index.neigh,neighborhood,extrazeros)

return(neigh.dist)
}

reachability2=function(distdata,k)
{
#function that calculates the local reachability density
#of Breuing(2000) for each observation in a matrix, using
#a matrix (distdata) of k nearest neighbors computed by the function dist.to.knn2

p=dim(distdata)[2]
lrd=rep(0,p)

for (i in 1:p)
{
j=seq(3,3+(distdata[2,i]-distdata[1,i]))
# compare the k-distance from each observation to its kth neighbor
# to the actual distance between each observation and its neighbors
numneigh=distdata[2,i]-distdata[1,i]+1
temp=rbind(diag(distdata[distdata[2,distdata[j,i]],distdata[j,i]],distdata[j+numneigh,i])

#calculate reachability
reach=1/(sum(apply(temp,2,max))/numneigh)
lrd[i]=reach
}
lrd
}

```

```

robout=function(data,nclass,meth=c("mve","mcd","classical"),rep,plot=T)
{
  *****
  #This function finds out the outliers using robust versions of the
  #Mahalanobis distance
  #data: name of the dataset
  #nclass: number of the class to check for outliers
  #meth=method used to compute the Mahalanobis distance, "mve"=minimum
  #   volume estimator, "mcd"=minimum covariance determinant,
  #   "classical"=the usual Mahalanobis distance.
  #rep= number of repetitions
  #*****
  ncol=dim(data)[2]
  tempo=data[data[,ncol]==nclass,1:(ncol-1)]
  namestempo=rownames(tempo)
  nrow=dim(tempo)[1]
  roboutl=NULL
  roboutall=matrix(0,nrow,rep)
  rownames(roboutall)=namestempo
  for(i in 1:rep)
  {mcdc=cov.rob(tempo,method=meth)
   mbc=sqrt(mahalanobis(tempo,mcdc$center,mcdc$cov,to=.00000000000001))
   roboutl=c(roboutl,boxplot(mbc,plot=F)$out)
   roboutall[,i]=mbc
  }
  a=as.matrix(roboutl)
  b=apply(roboutall,1,mean)
  outme=rev(sort(b))
  topo=rev(sort(b))[1:10]
  if(plot){
    win.graph()
    plot(rev(sort(b)),ylab=paste("Mahalanobis distance(",meth,""))
    text(1:10,topo,names(topo),cex=.6,pos=4)
  }
  top=rev(sort(table(as.numeric(rownames(a))))))
  top1=top[top>5]
  cat("\nTop outliers by frequency\n")
  print(top1)
  topout=as.numeric(names(top1))
  ntops=length(topout)
  outly=rep(0,ntops)
  for(i in 1:ntops)
  {outly[i]=mean(a[as.numeric(rownames(a))==topout[i]])}
  }
  #topimp=cbind(topout,outly)
  topimp=cbind(topout,b[topout])
  topimp=topimp[order(-topimp[,2]),]
  cat("\nTop outliers by outlyingness measure\n")
  #print(topout)
  zz=as.vector(outme)
  #names(zz)="outlyingness"

```

```

print(cbind(topout,zz[order(topout)]))
list(outme=outme)
}

cv10lda=function(data)
{
# This function finds the number of instances correctly classified
# by the Linear Discriminant classifier using 10-fold cross validation
# with one repetition.
# Inputs:
# data: dataset including the classes in the last column.
# Requires the lda function of the MASS library due to Ripley.
#
# Edgar Acuna-Caroline Rodriguez, 2004
#-----
n<-dim(data)[1]
p<-dim(data)[2]

salida <- matrix(0, 1, 10)
azar <- data[rank(runif(n)), ]
parti <- floor(n/10)

for(j in 1:10)
{
cc <- ((j - 1) * parti + 1):(j * parti)
if(j == 10)
{
cc <- ((j - 1) * parti + 1):n
}
datap <- azar[cc, ]
datat <- azar[- cc, ]
tempo <- lda(as.matrix(datat[, 1:p - 1]), datat[, p])
tempo1 <- predict(tempo, as.matrix(datap[, 1:p - 1]))$class
salida[j] <- sum(tempo1 != as.numeric(datap[, p]))
}

gooderr <- n-sum(salida)

return(gooderr)
}

cv10rpart<-function(datos)
{
# This function finds the number of instances correctly classified by the
# decision tree classifier, rpart, using 10-fold cross validation
# and one repetition.
# Requiere the rpart library
# inputs:
# datos: the dataset to be used
#
# Edgar Acuna-Caroline Rodriguez, March 2004
#-----
library(rpart)

```

```

datos=as.data.frame(datos)
n <- dim(datos)[1]
p <- dim(datos)[2]
nombres<-colnames(datos)
f1<-as.formula(paste(nombres[p],".",sep=~"))

salida <- matrix(0, 1, 10)
azar <- datos[rank(runif(n)), ]
azar[, p] <- as.factor(azar[, p])
parti <- floor(n/10)
for(j in 1:10)
{
  cc <- ((j - 1) * parti + 1):(j * parti)
  if(j == 10)
  {
    cc <- ((j - 1) * parti + 1):n
  }
  datap <- azar[cc, ]
  datat <- azar[- cc, ]
  arbol <- rpart(f1, data = datat, method="class")
  pd1<-predict(arbol,datap)
  pd2=max.col(pd1)
  salida[j] <- sum(pd2!=datap[, p])
}

gooderr<- n-sum(salida)

return(gooderr)
}

cv10knn=function(data, kvec)
{
  # This function finds the number of instances correctly classified by
  # the knn classifier, using 10-fold cross validation, with one repetition.
  # It requires the knn function of the class library due to B. Ripley.
  # inputs:
  # data: dataset to be used
  # kvec: number of nearest neighbors
  #
  # Edgar Acuna-Caroline Rodriguez, March 2004
  #-----
  n <- dim(data)[1]
  p <- dim(data)[2]
  salida <- matrix(0, 1, 10)
  azar <- data[rank(runif(n)), ]
  azar[, p] <- as.factor(azar[, p])
  parti <- floor(n/10)
  for(j in 1:10)
  {
    cc <- ((j - 1) * parti + 1):(j * parti)
    if(j == 10)
    {

```



```

    cc <- ((j - 1) * parti + 1):n
  }
  datap <- azar[cc, ]
  datat <- azar[ - cc, ]
  tempo <- knn(as.matrix(datat[, 1:p - 1]), as.matrix(datap[, 1:p - 1]), datat[, p], kvec)
  salida[j] <- sum(tempo != as.numeric(datap[, p]))
}
ECV1 <- n-sum(salida)
return(ECV1)
}

```

```

relief<-function(data,nosample, threshold)
{
# *****
# This program runs Relief for multiple classes
# Uses the function near1 and distancia
# data: name of the dataset
# nosample: number of instances drawn from the original dataset
# threshold: the cutoff point to select the features. First is
# chosen as zero and later is corrected after looking at the plot
#
# Revised: June 2002, revised January 2003, February 2004
# March 03,2004
# Edgar Acuna-Caroline Rodriguez
# *****

```

```

data <- as.matrix(data)
p=dim(data)[2]
f=p-1
#Initializing acum, features, and pesototal
acum<-rep(0,f)
features <- seq(f)
ngroups=length(unique(data[,p]))
pesototal=rep(0,f)
#Number of instances
inst <- length(data[, 1])
#Computing the priors
priors <- tabulate(data[, p])/inst
#Calculating the range of each feature. range=Max-Min
dh <- rep(0, f)
for(j in 1:f)
{
  dh[j] <- diff(range(data[, j]))
}
#Here starts the loop of the 10 repetitions
for (repet in 1:10)
{
#Inilializing nearhit, pesos and tempo
nearhit <- matrix(0, nosample, f)
pesos <- rep(0, f)
tempo <- matrix(0, ngroups, f)
#Here starts the loop for updating the pesos

```

```

for(i in 1:nosample)
{
  indices <- sample(inst, 1, replace = T)
  muestra <- data[indices, ]
  datatemp <- data[ - indices, ]
  data1=split.data.frame(datatemp[,1:f],datatemp[,p])
  indg <- muestra[p]
  nearhit[i, ] <- near1(muestra[ - p], data1[[indg]])
  #Finding the nearmiss in each group distinct to the group containing the nearhit
  for(kk in 1:ngroups)
  {
    if(kk != indg)
    {
      nearmiss<- near1(muestra[ - p], data1[[kk]])
      tempo[kk, ] <- (muestra[ - p] - as.vector(nearmiss))
    }
    for(ii in 1:f)
    {
      tempo[kk, ii] <- (1/nosample)*(tempo[kk, ii]/dh[ii])^2
    }
  }
  pesomiss <- rep(0, f)
  #Updating the pesos for each feature
  for(jj in 1:f)
  {
    for(kk in 1:ngroups)
    {
      if(kk != indg)
      {
        pesomiss[jj] <- pesomiss[jj] + priors[kk] * tempo[kk, jj]
      }
    }
    pesomiss[jj] <- pesomiss[jj]/(1 - priors[indg])
  }
  for(j in 1:f)
  {
    diff <- - (1/nosample)*((muestra[j] - nearhit[i, j])/dh[j])^2 + pesomiss[j]
    pesos[j] <- pesos[j] + diff
  }
}
#print(pesos)
#Normalizing the pesos
#pesos <- pesos/nosample
#selecting the features with pesos greater than a threshold
o1 <- order( - pesos)
o2 <- pesos[o1]
o3 <- o1[o2 > threshold]
#Acumulating the pesos in each repetition

pesototal=pesototal+pesos

#Acumulating the frecuencies of the selected features
acum[o3]=acum[o3]+1

```

```

#end of repet
}
#Ordering the total pesos
pesotota=as.matrix(pesototal)
of1 <- order( - pesotota)
of2 <- pesotota[of1]/10
acum=as.matrix(acum)
#Ordering the features according to theirs weights
tabla=cbind(1:f,acum,pesotota/10)
colnames(tabla)=c("feature","frequency","weight")
tabla=tabla[order(-tabla[,3]),]
cat("Frequencies and average weights of more relevant features in 10 replicates: \n")
print(tabla[tabla[,2]>5,])
#ploting the total pesos in order to update the threshold
plot(of2,ylab="weights")
text(1:f,of2,tabla[,1],pos=4)
relevant1=which(acum>5)
#Selecting the relevant features according to their total pesos and frequencies
relevant2=which(pesotota/10>threshold)
relevant=unique(c(relevant1,relevant2))
#print(relevant)
cat("selected features", "\n")
relevant=tabla[1:length(relevant),1]
return(relevant)
}

```

```

moda<-function(x,na.rm=TRUE)
{
#Function that finds the mode of vector x

if(na.rm==TRUE) m1=rev(sort(table(x)))
else m1=rev(sort(table(x,exclude=NULL)))
moda=names(m1[m1==m1[1]])
if (is(x,"numeric")) moda=as.numeric(moda)
return(moda)
}

```

```

near1<-function(x, data)
{
#*****
# Esta funcion encuentra la observacion en el
#conjunto de datos data que esta mas cerca a
# la observacion x requiere la funcion distancia
#*****
nd <- length(data[, 1])
distall <- rep(0, nd)
for(i in 1:nd) {
distall[i] <- distancia(x, data[i, ])
}
}

```

```

#print(sort(distall))
  ind1 <- order(distall)[1]
  near1 <- data[ind1, ]
  near1
}

sfs=function(data,method=c("lda","knn","rpart"),kvec=5,repet=10)
{
# *****
# This function selects features using the sequential
# forward method with either lda, knn or rpart
# data: the data set
# method: choice of classifier
# kvec: the number of nearest neighbors to be used for the knn classifier
# repet: number of repetitions. rep=20 for small datasets and =10 for large datasets
# Required libraries: MASS, class and rpart
# Edgar Acuna- Caroline Rodriguez, March 2004
#-----
if (!(method %in% c("lda","knn","rpart")))
{
  cat("The classifier entered is not supported by this function.\n")
  return(method)
}
# n: number of instances
n=dim(data)[1]
# p: number of columns
p=dim(data)[2]
#Initializing the vector of the number of selected features in each repetition
numbersel=rep(0,repet)
#Initializing the frequencies of the features
fsel=rep(0,repet)
for(i in 1:repet)
{
# Initializing the vector that will contain the selected features
indic <- rep(0, p - 1)
output <- indic
#number of the column containing the classes
varia <- p
for(k in 1:(p-1))
{
  correct <- rep(0, p - 1) #initializing the recognition rates for each feature
  if(k > 1)
  {
    varia <- c(where, varia)
  }
  for(m in 1:(p - 1))
  {
    if(indic[m] == 0)
    {
      which <- c(m, varia)
      if (method=="lda") correct[m] <- cv10lda2(data[, which])
      else if (method=="knn") correct[m] <- cv10knn2(data[, which],kvec)
    }
  }
}
}

```

```

        else correct[m] <- cv10rpart2(data[, which])
      }
    }
    prov <- correct + runif(p - 1) #Breaking ties randomly
    where <- sum((1:(p - 1)) * as.numeric(max(prov) == prov))
    #recognition rate of the entering feature
    output[k] <- correct[where]/n
    indic[where] <- 1
    if(k > 1)
    {
      if(output[k] <= output[k - 1])
      {
        #avoids ties of recognition rates
        indic <- rep(1, p - 1)
      }
    }
  }
  which <- rev(which)
  which <- which[-1]
  which1 <- which[1:(length(which) - 1)]
  numbersel[i]=length(which1)
  fsel[which1]=fsel[which1]+1
}
bestsize=round(mean(numbersel))
rev(order(fsel))
bestsubset=rev(order(fsel))[1:bestsize]
cat("The best subset of features is:")
cat("\n")
bestsubset
}

sbs1<-function(data,indic,correct0,kvec=5,method=c("lda","knn","rpart"))
{
  # This function performs a step of the Backward Selection method using
  # one of the classifiers: LDA, knn, or rpart.
  # data: dataset to be used
  # indic: vector of 0's and 1's. 1 indicates that the variable in that position
  # has been removed and 0 that it has not been removed.
  # correct0: recognition rate of the current best subset
  # kvec: number of neighbors if knn used
  #
  # Edgar Acuna-Caroline Rodriguez, March 2004
  # -----

  # n: number of instances
  n=dim(data)[1]
  # p: number of features
  p=dim(data)[2]
  output <- indic
  varia <- 1:(p - 1)
  varia <- varia[indic > 0]
  #print(varia)

```

```

#initializing the recognition rate vector
correct <- rep(0, p - 1)

mm=0
for(m in 1:(p - 1))
{
  if(indic[m] == 1)
  {
    mm=mm+1
    #print(mm)
    #print(varia)
    temp<-varia
    which <- temp[ - mm]
    if (method=="lda") correct[m] <- cv10lda2(data[, c(which, p)])
    else if (method=="knn") correct[m] <- cv10knn2(data[, c(which, p)],kvec)
    else correct[m] <- cv10rpart2(data[, c(which, p)])
  }
}

#Breaking ties randomly
prov <- correct + runif(p - 1)

#The feature to be removed
where <- sum((1:(p - 1)) * as.numeric(max(prov) == prov))

#recognition rate of the removed feature
output <- correct[where]/n

if(output >= correct0)
{
  indic[where] <- 1
}
else
{
  output <- correct0
  where <- NULL
  which1 <- NULL
}

which <- rev(which)

which1 <- where
indic[where] <- 0

list(variaelim = which1, indic = indic, correcto = output)

}

sfs1=function(data,indic,correcto,kvec,method=c("lda","knn","rpart"))
{
  # This function carries out one "forward step" using either

```

```

# the lda, knn or rpart classifier.
# inputs:
# data: the dataset
# indic: vector of 0's and 1's. 1 indicates the the variable in that
# position has been selected and 0 that it has not been selected.
# correcto=recognition rate of the current best subset
# kvec : the number of nearest neighbors
#
# Edgar Acuna-Caroline Rodriguez, March 2004
#-----
# n: number of instances
n=dim(data)[1]
# p: number of variables
p=dim(data)[2]
output<-indic
varia <- 1:(p - 1)
varia <- varia[indic > 0]
#print(varia)

#Initializing the recognition rate vector
correct <- rep(0, p - 1)

for(m in 1:(p - 1))
{
  if(indic[m] == 0)
  {
    which <- c(m, varia, p)
    if (method=="lda") correct[m] <- cv10lda2(data[, which])
    else if (method=="knn") correct[m] <- cv10knn2(data[, which],kvec)
    else correct[m] <- cv10rpart2(data[, which])
  }
}

#Breaking ties randomly
prov <- correct + runif(p - 1)

#The entering feature
where <- which(max(prov) == prov)

#recognition rate of the entering feature
output <- correct[where]/n
#print(output)

if(output > correcto)
{
  indic[where] <- 1
}

list(indic = indic, varselec = where, accuracy = output)
}

```

```

sffs<-function(data,method=c("lda","knn","rpart"),kvec=5,repet=10)
{
# *****
# This function selects features using the sequential
# floating forward method with either the lda, knn
# or rpart classifiers
# data: the data set
# method: choice of classifier
# kvec: the number of nearest neighbors to be used for the knn classifier
# Required libraries: MASS, class, and rpart
# Caroline Rodriguez-Edgar Acuna, March 2004
#-----

if (!(method %in% c("lda","knn","rpart")))
{
  cat("The classifier entered is not supported by this function.\n")
  return(method)
}
# n: number of instances
n=dim(data)[1]

# p: number of variables
p=dim(data)[2]
grupos=data[,p]

# ngroups: number of classes
ngroups=dim(table(data[,p]))

selected=rep(0,p)
numselect=0

for (j in 1:repet)
{
  indic <- rep(0, p - 1)
  correcto <- 0

  paso1 <- sfs1(data,indic,correcto,kvec,method)
  correcto <- paso1$accuracy
  indic <- paso1$indic

  i <- 2

  while(i <= (p - 1))
  {
    paso2 <- sfs1(data,indic,correcto,kvec,method)
    if(paso2$accuracy > correcto)
    {
      correcto <- paso2$accuracy
      indic <- paso2$indic
      for(j in 1:(i - 1))
      {
        paso3 <- sbs1(data,indic,correcto,kvec,method)

```



```

correcto <- paso3$correcto
indic <- paso3$indic
    }
  }
  else
  {
    i <- p
  }
}

variables <- seq(1, (p - 1))
variables <- variables[indic == 1]
cat("Selected variables for ",method," classifier on this repetition are: \n")
print(variables)
numselect=numselect+length(variables)
selected[variables]=selected[variables]+1
}
numselect=round(numselect/repet)
fselect=order(selected,decreasing=T)[1:numselect]
cat("\n\nThe best subset of features is:\n")
return(fselect)
}

distancia<-function(x, y)
{
  #####
  # Finds the euclidean distance between
  # two vector x and y or the matrix y and the vector x
  # #####
  if(class(y)=="matrix")
  {
    distancia = drop(sqrt(colSums((x-t(y))^2)))
    distancia= t(distancia)
  }
  else distancia = sqrt(sum((x-y)^2))
  distancia
}

mmnorm<-function (data)
{
  #This is a function to apply min-mas normalization to a matrix or dataframe.
  #Min-max normalization subtracts the minimum of an attribute from each value
  #of the attribute and them divides the difference by the range of the attribute.
  #Uses stats function found in R fields package and scale function found in the R base package.
  #Input: data= The matrix or dataframe to be scaled

  library(fields)

  #store all attributes of the original data
  d=dim(data)
  c=class(data)

```

```

cnames=colnames(data)

#remove classes from dataset
classes=data[,d[2]]
data=data[, -d[2]]

minvect=stats(data)[4,]
rangevect=stats(data)[8,]-stats(data)[4,]
zdata=scale(data,center=minvect,scale=rangevect)

#remove attributes added by the function scale and turn resulting
#vector back into a matrix with original dimensions
attributes(zdata)=NULL
zdata=matrix(zdata,dim(data)[1],dim(data)[2])
zdata=cbind(zdata,classes)

if (c=="data.frame") zdata=as.data.frame(zdata)
colnames(zdata)=cnames
return(zdata)

}

decscale<-function (data)
{
#This is a function to apply decimal scaling to a matrix or dataframe.
#Decimal scaling transforms the data into a range from [-1,1] by
#finding k such that the absolute value of the maximum value of each attribute divided by 10^k
#is less than or equal to 1.
#Uses stats function found in R fields package and scale function found in the R base package.
#Input: data= The matrix or dataframe to be scaled

library(fields)

#store all attributes of the original data
d=dim(data)
c=class(data)
cnames=colnames(data)

#remove classes from dataset
classes=data[,d[2]]
data=data[, -d[2]]

maxvect=stats(abs(data))[8,]

#find k such that max/10^k is less than 1.
kvector=ceiling(log10(maxvect))
scalefactor=10^kvector
decdata=scale(data,center=FALSE,scale=scalefactor)

#remove attributes added by the function scale and turn resulting
#vector back into a matrix with original dimensions
attributes(decdata)=NULL
decdata=matrix(decdata,dim(data)[1],dim(data)[2])

```

```

decdata=cbind(decdata,classes)

if (c=="data.frame") decdata=as.data.frame(decdata)
colnames(decdata)=cnames
return(decdata)

}

signorm<-function (data)
{
#This is a function to apply sigmoidal normalization to a matrix or dataframe.
#Sigmoidal normalization transforms the data into a range from [-1,1] by
#using a sigmoid function.
#Input: data= The matrix or dataframe to be scaled

#store all attributes of the original data
d=dim(data)
c=class(data)
cnames=colnames(data)
classes=data[,d[2]]

#first step of sigmoidal normalization is to standardize data
zdata=znorm(data)

#remove classes from normalized dataset
d2=dim(zdata)
zdata=zdata[,-d2[2]]

#scaling used:  $(1 - e^{-zdata}) / (1 + e^{-zdata})$ 
sigdata=(1-exp(-zdata))/(1+exp(-zdata))

#return classes to normalized dataset
sigdata=cbind(sigdata,classes)

if (c=="data.frame") sigdata=as.data.frame(sigdata)
colnames(sigdata)=cnames
return(sigdata)

}

softmaxnorm<-function (data)
{
#This is a function to apply softmax normalization to a matrix or dataframe.
#Softmax normalization transforms the data into a range from [0,1] by
#Input: data= The matrix or dataframe to be scaled

#store all attributes of the original data
d=dim(data)

```

```

c=class(data)
cnames=colnames(data)
classes=data[,d[2]]

#first step of softmax normalization is to standardize data
zdata=znorm(data)

#remove classes from standardized dataset
d2=dim(zdata)
zdata=zdata[,-d2[2]]

#scaling used: 1/(1+e^-zdata)
softdata=1/(1+exp(-zdata))

softdata=cbind(softdata,classes)

if (c=="data.frame") softdata=as.data.frame(softdata)
colnames(softdata)=cnames
return(softdata)
}

znorm<-function (data)
{
#This is a function to apply z-Score normalization to a matrix or dataframe.
#Uses scale function found in the R base package.
#Input: data= The matrix or dataframe to be scaled

#store all attributes of the original data
d=dim(data)
c=class(data)
cnames=colnames(data)

#remove classes from dataset
classes=data[,d[2]]
data=data[,-d[2]]

zdata=scale(data)

#remove attributes added by the function scale and turn resulting
#vector back into a matrix with original dimensions
attributes(zdata)=NULL
zdata=matrix(zdata,dim(data)[1],dim(data)[2])
zdata=cbind(zdata,classes)

if (c=="data.frame") zdata=as.data.frame(zdata)
colnames(zdata)=cnames
return(zdata)
}

```

```

ce.mimp=function (w.cl,method=c("mean","median"),atr,nomatr=0,name="")
{
#find dimensions of matrix
p=dim(w.cl)

#find indexes of missing values
index.na=which(is.na(w.cl),arr.ind=TRUE)
o=order(index.na[,1], index.na[,2])
index.na=index.na[o, ]
dimnames(index.na)=NULL

#find variables with missing values
var.na=sort(as.numeric(names(table(index.na[,2]))))

#find values of var.na that are equal to relevant attributes, reduce var.na
var.na=var.na[var.na%in%atr]

if (length(var.na)==0) stop("Error: No missing values occur in relevant variables!")

#reduce rows of index.na to only (row,col) of relevant variables with missing
index.atr=matrix(index.na[index.na[,2] %in% atr[]],,2)

#find classes of rows with missing
class.na=as.matrix(w.cl[index.atr[,1],p[2]])
dimnames(class.na)=NULL
class.na=cbind(index.atr,class.na)

classes=sort(as.numeric(names(table(w.cl[index.na[,1],p[2]]))))
num.class=length(classes)

replace.na=rep(0,0)

#replace na is row with mean or median of class
for(i in 1:dim(class.na)[1])
{
#split matrix into submatrices to find mean of class
sub=w.cl[w.cl[,p[2]]==class.na[i,3],]
#method=match.arg(method)

if (class.na[i,2]%in%nomatr) imput.col=moda(sub[,class.na[i,2]])[1]
else if (method=="mean") imput.col=mean(sub[,class.na[i,2]],na.rm=TRUE)
else if (method=="median") imput.col=median(sub[,class.na[i,2]],na.rm=TRUE)

#create a vector with input value for column of class
replace.na=rbind(replace.na,imput.col)
}
dimnames(replace.na)=NULL
class.na=cbind(class.na,replace.na)

for (i in 1:dim(class.na)[1])

```

```

w.cl[class.na[i,1],class.na[i,2]]=class.na[i,4]

#Remove comments if screen view is desired
cat("\nSummary of imputations using substitution of ",method,"(mode for nominal features):\n")
colnames(class.na)=c("Row","Column","Class","Imput.value")
print(class.na)
cat("\nTotal number of imputations per class: \n")
for (i in classes)
{
  amount=sum(class.na[,3]==i)
  cat("Class ",i," : ",amount,"\n")
}
cat("\nTotal number of imputations: ",dim(class.na)[1],"\n")

#Remove comments if workspace result file is not desired
#filename=paste("Imput.rep.",method,".",name,sep="")
#yy <- textConnection(filename, "w")
#rep.title=paste("Imputation report for the matrix: ",name)
#sink(yy)
#cat("\n",rep.title,"\n\n")
#cat("\nSummary of imputations using substitution of ",method,"(mode for nominal features):\n")
#colnames(class.na)=c("Row","Column","Class","Imput.value")
#print(class.na)
#cat("\nTotal number of imputations per class: \n")
#for (i in classes)
# {
#   amount=sum(class.na[,3]==i)
#   cat("Class ",i," : ",amount,"\n")
# }
#cat("\nTotal number of imputations: ",dim(class.na)[1],"\n")
#sink()
#close(yy)
#End comments to eliminate workspace result file

return(w.cl)
}

clean<-function (w,tol.col=0.3,tol.row=0.5,name="")
{

#w: matrix that will be cleaned
#tol.col: maximum percentage of missing to be allowed for columns
#tol.row: maximum percentage of missing in relevant variables to be allowed
#attrib: matrix, mx1, containing column index of relevant variables

w=as.data.frame(w)
w=as.matrix(w)
if (sum(is.na(w))==0) cat ("No cleaning required.\n")
else

```

```

{
filename=paste("Clean.rep.",name,sep="")
zz <- textConnection(filename, "w")
rep.title=paste("Cleaning report for the matrix: ",name)
sink(zz)
cat("\n",rep.title,"\n\n")
sink()

#Find column indexes of columns with NA
sumcol=which(colSums(is.na(w))!=0,arr.ind=TRUE)

if (length(sumcol)!=0)
{
#clean columns

dr=dim(w)[1]
dc=dim(w)[2]
if (length(sumcol)==1)
{
per.miss.col=sum(is.na(w[,sumcol]))/dr

#Report of variables to be removed
colmiss=colnames(w)[sumcol]

table.miss=data.frame(cbind(Variables=colmiss,Percent.of.missing=(per.miss.col*100)),row.names=N
ULL)
print(table.miss)
cat("\n")

sink(zz)
print(table.miss)
cat("\n")
sink()

if (per.miss.col>tol.col)
{
cat("Only one variable eliminated: ",colnames(w)[above.tol],"\n\n")

sink(zz)
cat("Only one variable eliminated: ",colnames(w)[above.tol],"\n\n")
sink()

w=w[,-sumcol]
w=as.matrix(w)
}
}
else
{
#find percent of missing
per.miss.col=colSums(is.na(w[,sumcol]))/dr

#find index of columns with NA over tolerance
above.tol=sumcol[which(per.miss.col>tol.col,arr.ind=TRUE)]

```

```

#Preparing report on percents missing per variable
colmiss=colnames(w)[sumcol]

table.miss=data.frame(cbind(Variables=colmiss,Percent.of.missing=(per.miss.col*100)),row.names=N
ULL)
print(table.miss)
cat("\n")

sink(zz)
print(table.miss)
cat("\n")
sink()

if (length(above.tol)==dim(w)[2])
{
  cat("All variables have missing values above tolerance level.\n\n")

  sink(zz)
  cat("All variables have missing values above tolerance level.\n\n")
  sink()
}
else
if (length(above.tol)!=0)
{
  #Report of columns to be eliminated
  col.above.tol=matrix(colnames(w)[above.tol],length(above.tol),1)
  colnames(col.above.tol)="Variables eliminated"
  rownames(col.above.tol)=c(1:length(above.tol))
  print(col.above.tol)
  cat("\n\n")

  sink(zz)
  print(col.above.tol)
  cat("\n\n")
  sink()

  #Column elimination
  w=w[,-above.tol]
  w=as.matrix(w)
}

}

#recalculate for new w and row cleaning
dr=dim(w)[1]
dc=dim(w)[2]

}

#Find index of rows with missing values
sumrow=which(rowSums(is.na(w))!=0,arr.ind=TRUE)

```



```

if (length(sumrow)!=0)
{
  if (length(sumrow)==1)
  {
    per.miss.row=sum(is.na(w[sumrow,]))/dc
    #clean rows
    if (per.miss.row>tol.row)
    {
      cat("Number of instances eliminated: 1\n")
      cat("Instance eliminated      :",sumrow,"\n\n")

      sink(zz)
      cat("Number of instances eliminated: 1\n")
      cat("Instance eliminated      :",sumrow,"\n\n")
      sink()

      w=w[-sumrow,]
    }
  }
else
{
  #begin to clean rows
  #calculate percent of rows with missing
  per.miss.row=rowSums(is.na(w[sumrow,]))/dc

  #calculate percent of rows with missing
  #rowmiss=
  #cat("Percent of rows with missing: ",per.miss.row*100,"\n")
  #cat("Number of rows with missing: ",rowSums(is.na(w[sumrow,])), "\n")
  #sink(zz)
  #cat("Percent of rows with missing: ",per.miss.row*100,"\n")
  #cat("Number of rows with missing: ",rowSums(is.na(w[sumrow,])), "\n")
  #sink()

  #find index of rows with NA over tolerance
  above.tol=sumrow[which(per.miss.row>tol.row,arr.ind=TRUE)]
  if (length(above.tol)==dr) cat("All instances have missing values above tolerance level.\n")
  else
  if (length(above.tol)!=0)
  {
    cat("Number of instances eliminated:",length(above.tol),"\n")
    cat("Instance eliminated      :",as.numeric(above.tol),"\n\n")

    sink(zz)
    cat("Number of instances eliminated:",length(above.tol),"\n")
    cat("Instance eliminated      :",as.numeric(above.tol),"\n\n")
    sink()

    w=w[-(above.tol),]
  }
}
}

```

```

    }

w=as.matrix(w)
cat("Maximum number of values to be imputed: ",sum(is.na(w)),"\n")

sink(zz)
cat("Maximum number of values to be imputed: ",sum(is.na(w)),"\n")
sink()
close(zz)
#print(w)
return(w)
}

    w=w[,-sumcol]
    w=as.matrix(w)
  }
}
else
{
  #find percent of missing
  per.miss.col=colSums(is.na(w[,sumcol]))/dr

  #find index of columns with NA over tolerance
  above.tol=sumcol[which(per.miss.col>tol.col,arr.ind=TRUE)]

  #Preparing report on percents missing per variable
  colmiss=colnames(w)[sumcol]

table.miss=data.frame(cbind(Variables=colmiss,Percent.of.missing=(per.miss.col*100)),row.names=N
ULL)
  print(table.miss)
  cat("\n")

  sink(zz)
  print(table.miss)
  cat("\n")
  sink()

  if (length(above.tol)==dim(w)[2])
  {
    cat("All variables have missing values above tolerance level.\n\n")

    sink(zz)
    cat("All variables have missing values above tolerance level.\n\n")
    sink()
  }
  else
  if (length(above.tol)!=0)
  {
    #Report of columns to be eliminated
    col.above.tol=matrix(colnames(w)[above.tol],length(above.tol),1)
    colnames(col.above.tol)="Variables eliminated"
    rownames(col.above.tol)=c(1:length(above.tol))
  }
}

```

```

print(col.above.tol)
cat("\n\n")

sink(zz)
print(col.above.tol)
cat("\n\n")
sink()

#Column elimination
w=w[,-above.tol]
w=as.matrix(w)
}

}

#recalculate for new w and row cleaning
dr=dim(w)[1]
dc=dim(w)[2]

}

#Find index of rows with missing values
sumrow=which(rowSums(is.na(w))!=0,arr.ind=TRUE)

if (length(sumrow)!=0)
{
  if (length(sumrow)==1)
  {
    per.miss.row=sum(is.na(w[sumrow,]))/dc
    #clean rows
    if (per.miss.row>tol.row)
    {
      cat("Number of instances eliminated: 1\n")
      cat("Instance eliminated      :",sumrow,"\n\n")

      sink(zz)
      cat("Number of instances eliminated: 1\n")
      cat("Instance eliminated      :",sumrow,"\n\n")
      sink()

      w=w[-sumrow,]
    }
  }
  else
  {
    #begin to clean rows
    #calculate percent missing of rows with missing
    per.miss.row=rowSums(is.na(w[sumrow,]))/dc

    #find index of columns with NA over tolerance
    above.tol=sumrow[which(per.miss.row>tol.row,arr.ind=TRUE)]
    if (length(above.tol)==dr) cat("All instances have missing values above tolerance level.\n")
  }
}

```

```

else
if (length(above.tol)!=0)
{
cat("Number of instances eliminated:",length(above.tol),"\n")
cat("Instance eliminated      :",as.numeric(above.tol),"\n\n")

sink(zz)
cat("Number of instances eliminated:",length(above.tol),"\n")
cat("Instance eliminated      :",as.numeric(above.tol),"\n\n")
sink()

w=w[-(above.tol),]
}
}
}
}

w=as.matrix(w)
cat("Maximum number of values to be imputed: ",sum(is.na(w)), "\n")

sink(zz)
cat("Maximum number of values to be imputed: ",sum(is.na(w)), "\n")
sink()
close(zz)
#print(w)
return(w)
}

ec.knnimp<-function(data,nomatr,k = 10)
{
#xnom: vector containing the indexes of the nominal variables
#data: matrix containing data
x <- data
N <- dim(x)[1]
p <- dim(x)[2]

#Checking if a row has a missing value
nas <- is.na(drop(x %*% rep(1, p)))
if(sum(nas)==N) stop("Error: All cases have missing values. Cannot compute neighbors.")

#submatrix with complete rows
#matrix needed in case xcomplete has only one row
xcomplete <- matrix(x[!nas, ],p)
colnames(xcomplete)=seq(p)

#submatrix of rows with at least one missing value
xbad <- x[nas,,drop=FALSE ]

#forming logical vector of nominal variables
xnom=seq(p) %in% nomatr

#Locating the missing values in the missing submatrix

```

```

xnas <- is.na(xbad)
xbadhat <- xbad
cat(nrow(xbad), fill = TRUE)
for(i in seq(nrow(xbad)))
{
  cat(i, fill = TRUE)
  xinas <- xnas[i, ]
  xbadhat[i, ] <- nnmiss(xcomplete, xbad[i, ],xinas,xnom, K = k)
}
x[nas, ] <- xbadhat
data2 <-x
return(data2)
}

```

```

nnmiss<-function(x, xmiss, ismiss,xnom, K=1)
{
  #x:submatrix of complete rows from original matrix
  #xmiss: a row with a missing value
  #issmiss: vector that indicates whether a value in xmiss is missing or not
  #xnom: vector with indexes of nominal variables

  #Find distance between xmiss (not NA) and each row of x
  xd <- scale(x, xmiss, FALSE)[, !issmiss]
  col=length(xmiss)-sum(is.na(xmiss))
  xd=matrix(xd,col)
  dd <- drop(xd^2 %*% rep(1, ncol(xd)))

  #order of the rows of x according to their closeness to xmiss
  od <- order(dd)[seq(K)]
  #if column of ismiss is nominal, find mode if not find mean of KNN

  ismiss.nom=issmiss[!xnom]
  ismiss.con=issmiss[xnom]
  xmiss[ismiss.nom] <- as.numeric(moda(x[od, ismiss.nom, drop = FALSE]))[1])
  xmiss[ismiss.con] <- drop(rep(1/K, K) %*% x[od, ismiss.con, drop = FALSE])
  xmiss
}

```

```

ce.knn.imp=function(m,k1)
{

  #Function that calls ec.knnimp to perform knn imputation
  #m : matrix to be tested with relevant variables and classes

  m=as.matrix(m)

  dr=dim(m)[1]
  dc=dim(m)[2]

  classes=tabulate(m[,dc])

```

```
no.classes=length(classes)

r=NULL

for(i in 1:no.classes)
{
  m.imp=ec.knnimp(m[m[,dc]==i,],k1)
  r=rbind(r,m.imp)
}
s2=sum(is.na(r))
cat(s2,"\n")
return(r)
}
```

Chapter 10 References

1. Acuña, E., Rojas, A. and Coaquira, F. (2002). The effect of feature selection on combining classifiers based on kernel density estimates. In K.Jajuga, A. Sokodowski, H.-H. Bock (Eds). Classification, Clustering and Data Analysis. Springer-Verlag, Berlin, 161-168.
2. Acuña, E., Coaquira, F. and Gonzalez, M. (2003). A comparison of feature selection procedures for classifiers based on kernel density estimation. Proc. of the Int. Conf. on Computer, Communication and Control technologies, CCCT'03. Vol I. p. 468-472. Orlando, Florida.
3. Atkinson, A. (1994). Fast very robust methods for the detection of multiple outliers. Journal of the American Statistical Association, 89:1329-1339.
4. Azzopardi, L. (2002). "Am I Right?" asked the Classifier: Preprocessing Data in the Classification Process. Computing and Information Systems, 9: 37-44.
5. Barnett, V. and Lewis, T. (1994). Outliers in Statistical Data. John Wiley, New York.
6. Batista, G. and Monard, M. C. (2002). K-Nearest Neighbour as Imputation Method: Experimental Results. Tech. Report 186, ICMC-USP.
7. Bay, S.D., and Schwabacher, M. (2003). Mining distance-based outliers in near linear time with randomization and a simple pruning rule. Proceedings from the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

8. Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988). *The New S Language: A programming environment for data analysis and graphics*. Chapman and Hall, New York.
9. Bello, A. L. (1995). Imputation techniques in regression analysis: Looking closely at their implementation. *Computational Statistics and Data Analysis*, 20: 45-57.
10. Blake, C.L. and Mertz, C.J. (1998). UCI Repository of machine learning databases [<http://www.ics.uci.edu/~mlearn/MLRepository.html>]. Irvine, CA: University of California, Department of Information and Computer Science.
11. Breuning, M., Kriegel, H., Ng, R.T, and Sander. J. (2000). LOF: Identifying density-based local outliers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.
12. Coaquira, F. (2002). Selección de variables para clasificación supervisada. Tesis MS. Universidad de Puerto Rico, Mayagüez.
13. Dash, M. and Liu, H. (1997). Feature Selection for classification. *Intelligent Data Analysis I*. 131-156.
14. Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. *Proceedings of the Twelfth International Conference in Machine Learning*, Morgan Kaufmann. San Francisco, 194-202.
15. Engels, R., Theusinger, C. (1998). Using a Data Metric for Preprocessing Advice for Data Mining Applications. *Proceedings of 13th European Conference on Artificial Intelligence*. , 430-434.

16. Fayyad, U., Grinstein, G, and Wierse, A. (2001). *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann, San Francisco.
17. Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth,P. (1996). From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, AAAI Press and the MIT Press, Chapter 1, 1-34.
18. Ferreira de Oliveira, M., Levkowitz, H. (2003). From Visual Data Exploration to Visual Data Mining: A Survey. *IEEE Transactions on Visualization and Computer Graphics* 9(3): 378-394.
19. Graham, M. and Kennedy, J. (2003). Using Curves to Enhance Parallel Coordinate Visualizations. In *IV 2003 - 7th International Conference on Information Visualization*, 10-16. London, UK: IEEE Computer Society Press.
20. Grinstein, G., Trutschl, M. and Cvek, U. (2001). High-Dimensional Visualizations. *Proceedings of the Visual Data Mining workshop, KDD'2001*.
21. Grzymala-Busse, J.W. and Hu, M. (2000). A Comparison of Several Approaches to Missing Attribute Values in Data Mining. In *RSCTC'2000*, pages 340-347.
22. Hadi, A. (1992). Identifying multiple outliers in multivariate data. *Journal of the Royal Statistical Society B*, 54:761-771.
23. Hall, M.A. (2000). Feature Selection for Discrete and Numeric Class Machine Learning. *Proc. Seventeenth International conference on Machine Learning*. Morgan Kaufmann, San Francisco, CA: 359-366.

24. Han, J., and Kamber, M. (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufman Publishers.
25. Hastie, T., Tibshirani, R., Sherlock, G., Eisen, M, Brown, P. and Bolstein, D. (1999). Imputing missing data por gene expression arrays. Technical Report. Division of Biostatistics, Stanford University.
26. Hawkins, D. (1980). *Identification of Outliers*. Chapman and Hall. London.
27. Healey, C. (1996). Effective Visualization of Large Multidimensional Datasets, PhD thesis., University of British Columbia.
28. Ihaka, R. and Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*: 5, 299-314.
29. Inselberg, A. (1985). The Plane with Parallel Coordinates, Special Issue on Computational Geometry. *The Visual Computer*, 1: 69-97.
30. Inselberg, A. and Dimsdale, B. (1990). Parallel coordinates: A tool for visualizing multidimensional geometry. *Proc. of Visualization '90*, p. 361-78.
31. Kalton, G. and Kasprzyk, D. (1986). The treatment of missing survey data. *Survey Methodology*, 12: 1-16.
32. Kaufman, L. and Rousseeuw, P.J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York.
33. Keim, D. (2001). Visual Data Mining and Exploration of Large Databases. Tutorial 12th European Conference on Machine Learning (ECML'01).

34. Kira, K. and Rendel, L. (1992). The Feature Selection Problem : Traditional Methods and a new algorithm. Proc. Tenth National Conference on Artificial Intelligence, MIT Press, 129-134.
35. Knorr, E. and Ng, R. (1997). A unified approach for mining outliers. Proc. KDD: 219–222.
36. Knorr, E., and Ng. R. (1998). Algorithms for mining distance-based outliers in large datasets. Proc. 24th Int. Conf. Very Large Data Bases, VLDB, 392–403, 24–27.
37. Knorr., E., R. Ng, and V. Tucakov. (2000). Distance-based outliers: Algorithms and applications. VLDB Journal: Very Large Data Bases, 8(3–4):237–253.
38. Kohavi, R and John, G. H. (1997). Wrappers for feature subset selection. Artificial Intelligence Journal, 97, 1-2 273-324.
39. Kononenko, I., Simec, E., and Robnik-Sikonja, M. (1997). Overcoming the myopia of induction learning algorithms with RELIEFF. Applied Intelligence Vol 7: 1, 39-55.
40. Kononenko, I. (1994). Estimating Attributes: Analysis and Extension of Relief. In F. Bergadano y L. D.Raedt, eds. Proc. seventh European Conference on Machine Learning, 171-182.
41. Kudo, M. and Sklansky, J. (2000). Comparison of algorithms that select features for pattern classifiers. Pattern recognition 33(1) 25-41.
42. Little, R. J. and Rubin, D.B. (2002). *Statistical Analysis with Missing Data*. Second Edition. John Wiley and Sons, New York.

43. Liu, H., and Setiono, R. (1996). A probabilistic approach to feature selection- a filter solution. Proc. of the thirteenth International Conference of Machine Learning, 319-337.
44. Lohninger, H. (1994). "INSPECT: A Program System to Visualize and Interpret Chemical Data". Chemomet. Intell. Lab. Syst. 22,147-153.
45. Lu, H., Sun, S., Lu, Y. (1996). On Preprocessing Data for Effective Classification. ACM SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery, Montreal, Canada.
46. Merz, C., Murphy, P. (1998). UCI Repository of machine learning databases. <http://www.ics.uci.edu/~mlearn/MLRepository.html>
47. Müller, H., Freytag, J., (2003). Problems, Methods, and Challenges in Comprehensive Data Cleansing Technical Report HUB-IB-164, Humboldt University Berlin.
48. Mundfrom, D.J and Whitcomb, A. (1998). Imputing missing values: The effect on the accuracy of classification. Multiple Linear Regression Viewpoints, 25(1), 13-19.
49. Ng, R.T. and Han, J. (1994). Efficient and effective clustering methods for spatial data mining. Proc. 20th Int.Conf. on Very Large Data bases. Morgan and Kaufmann Publishers, San Francisco, 144-155.
50. Poulet, F., (1999). Visualization in data mining and knowledge discovery. Proc. HCP'99, 10th Mini Euro Conference on Human Centered Processes.

50. Pudil, P., Ferri, J., Novovicová, J., and Kittler, J. (1994). Floating search methods for feature selection with nonmonotonic criterion function. 12th International Conference on Pattern Recognition, 279-283.
51. Pyle, D. (1999). *Data Preparation for Data Mining*. Morgan Kaufmann, San Francisco.
52. Ramaswamy, S., Rastogi, R., and Shim, K. (2000). Efficient algorithms for mining outliers from large datasets. In Proceedings of the ACM SIGMOD International Conference on Management of Data.
53. Rocke, D. and Woodruff, D. (1996). Identification of outliers in multivariate data. *Journal of the American Statistical Association*, 91:1047-1061.
54. Rousseeuw, P. (1985). Multivariate estimation with high breakdown point. *Mathematical statistics and applications*.
55. Rousseeuw, P. and Leroy, A. (1987). *Robust Regression and Outlier Detection*. John Wiley, New York.
56. Rousseeuw, P. and Van Zomeren, B. (1990). Unmasking multivariate outliers and leverage points. *Journal of the American Statistical Association*, 85:633-639.
57. Rousseeuw, P. J. and Van Driessen, K. (1999). A Fast Algorithm for the Minimum Covariance Determinant Estimator. *Technometrics*, 41, 212-223.
58. Sahling, G. (2002). Interactive 3D Scatterplots - From High-Dimensional Data to Insight. <http://www.VRVis.at/vis/resources/DA-NSahling/>. (29-jan-2004)

59. Schafer, J.L. (1997). *Analysis of Incomplete Multivariate Data*. Chapman and Hall, London.
60. Schmid, C, and Hinterberger, H. (1994). Comparative Multivariate Visualization Across Conceptually Different Graphic Displays. Proceedings of the Seventh International Working Conference on Statistical and Scientific Database Management, SSDBM 94, Charlottesville, Virginia, September 28 - 30.
61. Swayne, D.F., Cook, D. and Buja, A. (1991). XGobi: interactive dynamic graphics in the X window system with a link to S. In Proceedings of the ASA Section on Statistical Graphics, pages 1–8, Alexandria, VA.
62. Temple, D., Swayne, D. (2001). GGobi meets R: an extensible environment for interactive dynamic data visualization. DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing March 15-17, Vienna, Austria
63. Tresp, V., Neuneier, R. and Ahmad, S. (1995). Efficient methods for dealing with missing data in supervised learning. In G. Tesauro, D. S. Touretzky, and Leen T. K., editors, Advances in NIPS 7. MIT Press.
64. Tukey, J.W. (1977). *Exploratory Data Analysis*. Addison-Wesley, Reading, MA.
65. Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P. Hastie, T., Tibshirani, R., Bostein, D. and Altman, R.B. (2001). Missing value estimation methods for dna microarrays. Bioinformatics, 17(6), 520-525.
66. Venables, W.N. and Ripley, B.D. (1997). *Modern Applied Statistics with S-PLUS*. Second Edition. Springer-Verlag, New York.

67. Wegman, H. (1990). Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, Vol. 411(85), p. 664-675.
68. Wegman, E.J. and Carr, D.B. (1993). Statistical graphics and visualization, in *Handbook of Statistics 9: Computational Statistics*, (Rao, C. R., ed.), Amsterdam: North Holland, 857-958.
69. Wegman, E. and Luo, Q. (1997). High dimensional clustering using parallel coordinates and the grand tour. *Computing Science and Statistics*, Vol 28, p. 352-360.
70. Yang, J., Ward, M. O., Rundensteiner, E. A., and Huang, S. (2003). Visual hierarchical dimension reduction for exploration of high dimensional datasets. *VisSym 2003*, p. 19-28.
71. Yang, L. (2000). Interactive exploration of very large relational datasets through 3D dynamic projections. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, 236 - 243. Boston, Massachusetts.