Tribhuvan University

Institute of Science and Technology

PERFORMANCE ANALYSIS OF FILTER BASED FEATURE SELECTION TECHNIQUES IN TREE BASED CLASSIFICATION METHODS

A Dissertation

Submitted to:

Central Department of Computer Science and Information Technology
Tribhuvan University, Kirtipur

In partial fulfillment of the requirements
for the Master's Degree in Computer Science & Information Technology

Submitted by:

Meghraj Sapkota
June, 2017

Supervisor

Prof. Dr. Shashidhar Ram Joshi

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than the listed here have been used in this work.

… … … … …

Meghraj Sapkota

Date: 7th June, 2017

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

Supervisor's Recommendation

I hereby recommend that the dissertation prepared under my supervision by **Mr. Meghraj Sapkota** entitled "PERFORMANCE ANALYSIS OF FILTER BASED FEATURE SELECTION TECHNIQUES IN TREE BASED CLASSIFICATION METHODS" be accepted as in fulfilling partial requirement for the completion of Masters Degree of Science in Computer Science & Information Technology.

Prof. Dr. Shashidhar Ram Joshi

Department of Electronics & Computer Engineering,

Institute of Engineering,

Pulchowk, Nepal

Date: 27th June, 2017

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

LETTER OF APPROVAL

We certify that we have read this dissertation work and in our opinion it is appreciable for the scope and quality as a dissertation in the partial fulfillment of the requirements of Masters Degree of Science in Computer Science & Information Technology.

Evaluation Committee

Asst. Prof. Nawaraj Paudel
Head of Department
Central Department of Computer Science
& Information Technology
Tribhuvan University
Kirtipur, Nepal

Prof. Dr. Shashidhar Ram Joshi
Department of Electronics & Computer Engineering,
Institute of Engineering,
Pulchowk, Nepal

(External Examiner)

(Internal Examiner)

Date: 3rd July, 2017

Acknowledgement

I would never have been able to finish my dissertation without the guidance, support and encouragement of numerous people including my supervisor, my friends, colleagues and support from my family. At the end of my thesis I would like to thank all those people who made this thesis possible and an unforgettable experience for me.

First, I would like to express my gratitude to my supervisor **Professor Dr. Shashidhar Ram Joshi,** Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk, Kathmandu for his continuous support without which the thesis wouldn't have been possible to complete. His suggestions, guidance, thorough knowledge and expertise helped me immensely in understanding and developing this thesis. I thank him immensely for his patience and generous time spent to guide me through the entire process.

Most importantly I would like to thank to respected Head of Department of Central Department of Computer Science and Information Technology, Asst. Prof. Nawaraj Paudel for his kind support, help and constructive suggestions. I am very much grateful and thankful to all the respected teachers Prof. Dr. Subarna Sakya, Dr. Arun Kumar Timilsina, Mr. Min Bahadur Khati, Mr. Dheeraj Kedar Pandey, Mr. Jagdish Bhatta, Mr. Sarbin Sayami, Mrs. Lalita Sthapit, Mr. Arjun Singh Saud, Mr. Bikash Balami and Tej Bahadur Shahi for providing me such a broad knowledge and inspirations.

Special thanks to my family for their endless motivation, constant mental support and love which have been influential in whatever I have achieved so far. All my class fellows are worthy of my gratefulness for their direct or indirect support in completion of my dissertation.

I have done my best to complete this research work. Suggestions from the readers are always welcomed, which will improve this work.

Abstract

Classification has been called the most influential development in Data Mining and Machine Learning in the past decade. The idea of classification is to find the class of the unknown objects based on their attributes.

In this thesis, the performance of decision tree based classification methods is analysed with feature selection methods; Chi-square and Relief. The feature selection process chooses optimal subset of features according to objective function. These feature selection method helps to remove unnecessary attributes from the high dimensional dataset, thus improves the efficiency of the classification algorithms. The performance of feature section methods; Chi-square and Relief were compared in Tree based classification methods; C4.5, CART, LMT and Random Tree. The study shows that the Chi-square feature selection method is more suitable while using with LMT followed by C4.5, Random Tree and CART respectively. In case Relief based feature selection method LMT gives the best result followed by CART, C4.5 and Random Tree respectively.

Keywords: Feature selection, Chi-square, Relief, C4.5, CART, LMT, Random Tree.

List of Figures

List of Tables

List of Abbreviations

Abbreviations	Full Form
MDL	Minimum Description Length
CART	Classification and regression tree
LMT	Logistic Model Tree
WEKA	Waikato Environment for Knowledge Analysis
SDK	Software development kit
IDE	Integrated development Environment
SWT	Standard Widget toolkit

Chapter 1 Introduction

1.1 Introduction

Data mining is defined as the process of discovering patterns in data. One of the most used task in data mining is classification. Classification is the most powerful technique used for data analysis. Classification is supervised learning paradigm in which object are assigned into a predefined group or class based on a number of observed attributes related to that object. There are many industrial problems identified as classification problems such as Stock market prediction, Weather forecasting, Bankruptcy prediction, Medical diagnosis, Speech recognition, Character recognitions [1]. There are no. of classification technique which can be categorize as follows:

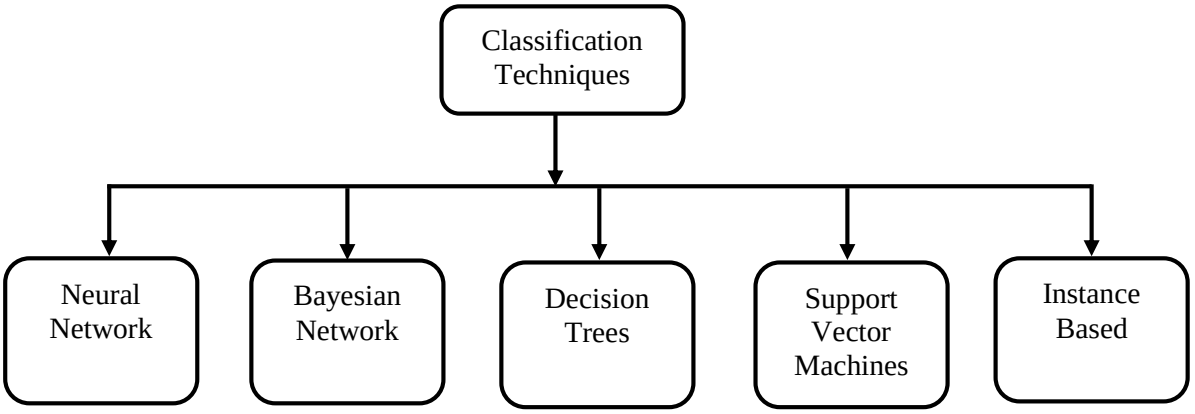

Figure 1.1: Classification techniques

The dataset used for classification might have large dimension with noisy data. When the data analysis task such as classification is directly applied into the large dataset having high dimension and noisy data the performance of classification method degrades. In order to increase performance, the high dimensions of the dataset are first reduced into lower dimension using feature selection method then apply classification algorithms [2].

Feature selection has been an active and fruitful field of research area in pattern recognition, machine learning, statistics and data mining communities [3]. The main objective of feature selection is to choose a subset of input variables by eliminating features, which are irrelevant or of no predictive information. Feature selection has proven in both theory and practice to be effective in enhancing learning efficiency, increasing predictive accuracy and reducing complexity of learned results [4, 5]. Feature selection in supervised learning has a main goal of finding a feature subset that produces higher classification accuracy. Even though several models exist for feature selection process only few will be suitable for an environment of the application. Thus it is necessary to study the suitability for attribute selection methods.

1.2 Thesis Organisation

Introduction part of this dissertation work focuses on Classification techniques along with feature selection.

The rest of the material in this study is organized into subsequent five chapters.

Chapter 2 provides background study required for dissertation. In this chapter problem of using classification techniques without using feature selection is given, problem statement is formulated and how this study response those issues is mentioned.

Chapter 3 contains previous literature allied to this work in detail under literature review. In this chapter detailed description about classification method and feature selection techniques are discussed.

Chapter 4 provides an implementation overview of the work using WEKA and Eclipse tool.

Chapter 5 includes the performance measure of HFEE method with different other ensemble methods. The result of the study is shown in tabular form as well as in graphs.

Finally, the concluding remarks and further recommendations are outlined in chapter 6.

Chapter 2 Background study and Problem Formulation

2.1 Classification

Classification is a data mining function that assigns items in a collection to target categories or classes. The goal of classification is to accurately predict the target class for each case in the data. For example, a classification model could be used to identify loan applicants as low, medium, or high credit risks.

A classification task begins with a data set in which the class assignments are known. For example, a classification model that predicts credit risk could be developed based on observed data for many loan applicants over a period of time. In addition to the historical credit rating, the data might track employment history, home ownership or rental, years of residence, number and type of investments, and so on. Credit rating would be the target, the other attributes would be the predictors, and the data for each customer would constitute a case.

Formally, A typical supervised classification problem has a database of the form: $D = (\mathbf{x}1, y1), (\mathbf{x}2, y2), \ldots, (\mathbf{x}n, yn)$. Here $\mathbf{x}$ values are typically vectors of the form: $\mathbf{x} = <x1, \ldots, xn>$whose components can be discrete or real valued. These components are the attributes (or features) of the database. The objective is to infer the unknown function (or relation) $y = f(\mathbf{x})$, where the y value is drawn from a discrete set of classes $C = \{C1, \ldots, Ck\}$ that characterize the given data.(taken from Computational Intelligence and Feature selection.)

2.2 Dimension Reduction

Data mining algorithms search for meaningful patterns in raw data sets. The Data Mining process requires high computational cost when dealing with large data sets. Reducing dimensionality (the number of attributed or the number of records) can effectively cut this cost. These techniques reduce the higher dimensional dataset into the lower dimensional dataset. The low-dimensional representation is referred to as the embedding of the dataset. Furthermore, an effective dimension reduction method also removes noisy features and inter-features correlations [6].

Thus dimension reduction is the pre-processing step which reduces the dimension of the dataset so that the data mining algorithms performs efficiently.

It can be seen that there are four major reasons for performing dimension reduction:

1. Decreasing the learning (model) cost;

2. Increasing the learning (model) performance;
3. Reducing irrelevant dimensions;
4. Reducing redundant dimensions.

There are two techniques for reducing the dimension of the datasets:

1. Feature selection
2. Feature extraction

2.2.1 Feature Selection

Feature selection is the process which chooses the subset of features from the total number of available features that are relevant. Feature selection is studied intensively in the theoretical field such as machine learning for its vast applications in gene expression microarray analysis, image analysis and text processing [7]. Feature selection is of crucial importance in above areas, since it helps improve the prediction performance of machine learning models by eliminating noisy variables, provide simpler models that facilitate better interpretation.

Generally, the approaches of feature selection can be divided into three types: **filters, wrappers and embedded methods** [8].

2.2.1.1 Filters

Filters estimate a relevance index for each feature to measure how relevant a feature is to the target. Then filters rank features by their relevance indices and perform search according to the ranks or based on some statistical criterion e.g. significance level. The most distinguishing characteristic of filters is that the relevance index is calculated based solely on a single feature without considering the values of other features. Such implementation implies that filters assume orthogonally between features which usually is not true in practice. Therefore, filters omit any conditional dependence (or independence) that might exist, which is known to be one of the weaknesses of filters, since they might miss optimal subset of features.

There are various heuristics to design relevance indices for filters, including univariate prediction error rate (i.e. evaluate the relevance of a feature as how accurate the prediction is using only itself), correlation-based (e.g. Pearson coefficient, signal to noise ratio), distances between distributions (K-L divergence, Jeffreys-Matusita distance), information theory (mutual information, Minimum Description Length (MDL)), decision trees (C45, CART), Relief (a class of filters incorporating sample relations into feature selection).

2.2.1.2 Wrappers

Instead of ranking every single feature, wrappers rank feature subsets by the prediction performance of a classifier on the given subset, which were first proposed by Kohavi and John (1997). Unlike filters, wrappers can be used to search through all possible subsets of features and explore the mutual information between features. After choosing a classifier (preferably consistent), wrappers evaluate the prediction performance either by cross-validation or theoretical performance bounds. Other than the choices of classifiers, wrappers differ in the underlying search strategies. Exhaustively searching combinatorial subsets is NP-hard and is prone to over fitting. Therefore, greedy search strategies are generally preferred, such as sequential forward selection or backward elimination.

2.2.1.3 Embedded

Embedded methods select features based on criterions that are generated during the learning process of a specific classifier. In contrast to wrappers, they do not separate the learning from the feature selection part, i.e. the selected features are sensitive to the structures of the underlying classifiers. For this reason, in most cases, the feature selected by one embedded methods might not be suitable for others.

2.2.2 Feature Extraction

Feature reduction refers to the mapping of the original high-dimensional data onto a lower dimensional space. In mathematical terms, the problem can be stated as follows: given the p-dimensional random variable $x = (x_{1,\ldots\ldots\ldots\ldots,} x_p)^T$ find a lower dimensional representation of it, $s = (s_1,\ldots\ldots\ldots, s_k)^T$ with $k <= p$, that captures the content in the original data, according to some criterion.

Taxonomy of dimensionality reduction algorithm is divided into convex and non-convex technique. Convex techniques optimize an objective function that does not contain any local optima, whereas non-convex techniques optimize objective functions that do contain local optima [8].

a. Convex
    - PCA
    - Isomap
    - Kernel PCA
    - Diffusion maps
b. Non-convex
    - Sammon mapping
    - Autoencoder

2.3 Problem Formulation

2.3.1 Problem Statement

Decision tree induction is the learning of decision trees from class-labelled training tuples. A decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node holds a class label [9]. The structure of Decision tree mainly depends upon the selection of attributes during the construction of the branches of tree. When feature selection method applied to the dataset before decision tree based classification algorithms gives different decision tree structure. Hence, the classification result may vary. Also different decision tree based algorithms gives different decision tree and different feature selection methods gives different subsets of features. Thus the problem is to find which feature selection method performs well on which decision tree based classification method.

2.3.2 Objectives

The main objectives of this thesis are

- To analyse the performance of feature selection methods in Decision tree based classification methods.
- To determine the best combination of feature selection methods with decision tree based classification method.

Chapter 3 Literature Review & Methodology

3.1 Literature Review

3.1.1 Decision Tree Induction

A decision tree is a classifier expressed as a recursive partition of the instance space [10]. The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called "root" that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an internal or test node. All other nodes are called leaves (also known as terminal or decision nodes). In a decision tree, each internal node splits the instance space into two or more subspaces according to a certain discrete function of the input attributes values. In the simplest and most frequent case, each test considers a single attribute, such that the instance space is partitioned according to the attribute's value. In the case of numeric attributes, the condition refers to a range.

Each leaf is assigned to one class representing the most appropriate target value. Alternatively, the leaf may hold a probability vector indicating the probability of the target attribute having a certain value. Instances are classified by navigating them from the root of the tree down to a leaf, according to the outcome of the tests along the path.

Attribute Selection measures

During tree construction, attribute selection measures [9] are used to select the attribute that best partitions the tuples into distinct classes. Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split. The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure is chosen as the *splitting attribute* for the given tuples. Popular measures of attribute selection are given as follows:

- **Information gain**
- **Gain ratio**
- **Gini index**

The notation used herein is as follows. Let D, the data partition, be a training set of class-labelled tuples. Suppose the class label attribute has m distinct values defining m distinct classes, C_i (for i = 1, ...,m). Let $C_{i,D}$ be the set of tuples of class C_i in D. Let $|D_j|$ and $| C_{i,D}|$ denote the number of tuples in D and $C_{i,D}$, respectively.

Information gain

This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or "information content" of messages. Let node *N* represents or hold the tuples of partition *D*. The attribute with the highest information gain is chosen as the splitting attribute for node *N*. This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple tree is found.

The expected information needed to classify a tuple in *D* is given by

$$Info(D) = -\sum_{i=1}^{m} p_i log_2(p_i) \quad \text{.....(3.1)}$$

Where p_i is the probability that an arbitrary tuple in *D* belongs to class C_i and is estimated by $|C_{i,D}|/|D|$.

Let us consider Attribute *A* can be used to split *D* into *v* partitions or subsets, $\{D_1, D_2, ..., D_v\}$, where D_j contains those tuples in *D* that have outcome a_j of *A*. These partitions would correspond to the branches grown from node *N*. The amount of extra information needed to after the partitioning in order to arrive at an exact classification is given by

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} * Info(D_j) \quad \text{.....(3.2)}$$

Information gain is defined as the difference between the original information requirement and the new requirement. That is,

$$Gain(A) = Info(D) - Info_A(D) \quad \text{.....(3.3)}$$

Gain ratio

Gain ratio is an extension of information gain, which attempts to overcome the biasness to select the attribute having many outcomes. It applies a kind of normalization to information gain using a "split information" value defined analogously with *Info*(*D*) as

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} * log_2\left(\frac{|D_j|}{|D|}\right) \text{.....(3.4)}$$

The gain ratio is defined as

$$GainRatio(A) = Gain(A) / SplitInfo(A) \text{.....(3.5)}$$

The attribute with the maximum gain ratio is selected as the splitting attribute.

Gini Index

The Gini index considers a binary split for each attribute. The Gini index measures the impurity of *D*, a data partition or set of training tuples, as

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2 \quad \text{.....(3.6)}$$

Where p_i is the probability that a tuple in D belongs to class C_i and is estimated by $|C_{i,D}|/|D|$. The sum is computed over m classes.

Also, let a binary split on A partitions D into D_1 and D_2, the gini index of D given that partitioning is

$$\text{Gini}_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2) \quad \text{.....(3.7)}$$

Then, the reduction in impurity that would be incurred by a binary split on a discrete- or continuous-valued attribute A is

$$\Delta Gini(A) = Gini(D) - Gini_A(D) \quad \text{.....(3.8)}$$

The attribute that maximizes the reduction in impurity is selected as the splitting attribute.

Tree pruning

When decision trees are built, many of the branches may reflect noise or outliers in the training data. Tree pruning attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data. There are two common approaches to tree pruning: **Pre pruning**, and **Post pruning**

In the **pre pruning** [9] approach, a tree is "pruned" by halting its construction early (e.g., by deciding not to further split or partition the subset of training tuples at a given node). Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.

The second and more common approach is **post pruning** [9], which removes sub trees from a "fully grown" tree. A sub tree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labelled with the most frequent class among the sub tree being replaced. Examples of post pruning methods are: Cost complexity, pessimistic pruning, minimum description length principle etc.

Cost-complexity

This approach considers the cost complexity of a tree to be a function of the number of leaves in the tree and the error rate of the tree. It starts from the bottom of the tree. For each internal node, N, it computes the cost complexity of the sub tree at N, and the cost complexity of the sub tree at N if it were to be pruned (i.e., replaced by a leaf node). The two values are compared. If pruning the sub tree at node N would result in a smaller cost complexity, then the sub tree is pruned. Otherwise, it is kept. A pruning set of class-labelled tuples is used to estimate cost complexity. This set is independent of the training set used to build the unpruned tree and of

any test set used for accuracy estimation. The algorithm generates a set of progressively pruned trees. In general, the smallest decision tree that minimizes the cost complexity is preferred.

Pessimistic pruning

It is similar to the cost complexity method in that it also uses error rate estimates to make decisions regarding sub-tree pruning. Pessimistic pruning, however, does not require the use of a prune set. Instead, it uses the training set to estimate error rates.

3.1.1.1 C4.5

C4.5 [11] is an evolution of ID3, presented by Quinlan in 1993. It uses gain ratio as splitting criteria. Pessimistic pruning is performed after the growing phase.

3.1.1.2 CART

CART [12] stands for Classification and Regression Trees. It is characterized by the fact that it constructs binary trees, namely each internal node has exactly two outgoing edges. The splits are selected using the Gini index criteria and the obtained tree is pruned by cost–complexity Pruning.

3.1.1.3 LMT

A logistic model tree [13] basically consists of a standard decision tree structure with logistic regression functions at the leaves. More formally, a logistic model tree consists of a tree structure that is made up of a set of inner or non-terminal nodes *N* and a set of leaves or terminal nodes *T*. Let *S* denote the whole instance space, spanned by all attributes that are present in the data. Then the tree structure gives a disjoint subdivision of *S* into regions S_t, and every region is represented by a leaf in the tree:

$$S = \bigcup_{t \in T} S_t, \quad S_t \cap S_{t'} = \emptyset \quad for\ t \neq t'$$

Unlike ordinary decision trees, the leaves $t \in T$ have an associated logistic regression function f_t instead of just a class label. The regression function f_t takes into account a subset $V_t \in V$ of all attributes present in the data (where we assume that nominal attributes have been binarized for the purpose of regression), and models the class membership probabilities as

$$Pr(G = j \mid X = x) = \frac{e^{Fj(x)}}{\sum_{k=1}^{J} e^{Fk(x)}} \quad \text{.....(3.9)}$$

where,

$$F_j(x) = \alpha_0^j + \sum_{v \in V_t} \alpha_v^j * v \quad \text{..... (3.10)}$$

And J represents the no. of classes. Given estimates for the class probabilities, LMT classify unseen instances by

$$j^* = \underset{j}{argmax}\, Pr(G = j|X = x) \quad \text{.....(3.11)}$$

The model represented by the whole logistic model tree is then given by

$$f(x) = \sum_{t \in T} f_t(x) * I(x \in S_t) \quad \text{.....(3.12)}$$

where, $I(x \in S_t)$ is 1 if $x \in S_t$ and 0 otherwise.

Fitting a logistic regression model means estimating the parameter vectors α^j. Friedman et al. propose the LogitBoost [14] algorithm for fitting logistic regression models by maximum likelihood.

3.1.1.4 Random Tree

Random tree construct a decision tree by randomly choosing an attribute for each node. It does not employ pruning method. Choosing a random attribute is done using linear congruential method.

3.1.2 Feature Selection

Feature selection is the process which chooses the subset of features from the total number of available features that are relevant. Feature selection is studied intensively in the theoretical field such as machine learning for its vast applications in gene expression microarray analysis, image analysis and text processing [15]. Generally, the approaches of feature selection can be divided into three types: **filters, wrappers and embedded methods**.

L.Ladha et al. [8] published a paper "Feature selection methods and Algorithms" which describes different types of features selection methods. They present an empirical comparison of feature selection methods and its algorithms.

3.1.2.1 Filters

Filters estimate a relevance index for each feature to measure how relevant a feature is to the target. Then filters rank features by their relevance indices and perform search according to the ranks or based on some statistical criterion e.g. significance level. There are various heuristics to design relevance indices for filters, including univariate prediction error rate (i.e. evaluate the relevance of a feature as how accurate the prediction is using only itself), correlation-based (e.g. Pearson coefficient, signal to noise ratio), distances between distributions (K-L divergence, Jeffreys-Matusita distance), information theory (mutual information, Minimum Description Length (MDL)), Relief (a class of filters incorporating sample relations into feature selection).

Chi-square (χ^2) method

This method measure the lack of independence between a term and the category. Chi-Squared [1] is the common statistical test that measures divergence from the distribution expected if one assumes the feature occurrence is actually independent of the class value. In statistics, the $\chi 2$ test is applied to test the independence of two events. In feature selection, the two events are occurrence of the term and occurrence of the class. Feature selection using the $\chi 2$ statistic is analogous to performing a hypothesis test on the distribution of the class as it relates to the values of the feature in question. The null hypothesis is that there is no dependency; each value is as likely to have instances in any one class as any other class. The $\chi 2$ statistic quantifies the difference between observed and expected counts for each pair of values; it is defined as follows:

Let under the null hypothesis X_1 and X_2 are assumed to be independent, then the expected frequency for each pair of values is given as

$$e_{i,j} = \frac{n_i^1 n_j^2}{n} \quad \text{.....(3.13)}$$

where,

n_i^1 represents the no. of counts that have the values i in X_1

n_j^2 represents the no.of counts that have the values j in X_2

i represents the possible values in X_1

j represents the possible values in X_2

n represents the total no. of counts.

The $\chi 2$ statistic quantifies the difference between observed and expected counts for each pair of values; it is defined as follows:

$$\chi 2 = \sum_i \sum_j \frac{(n_{ij} - e_{ij})^2}{e_{ij}} \quad \text{..... (3.14)}$$

The larger this chi-squared statistic, the more unlikely it is that the distribution of values and classes are independent; that is, they are related, and the feature in question is relevant to the class.

RELIEF

Kira and Rendell describe an algorithm called RELIEF [16] that uses instance based learning to assign a relevance weight to each feature. Each feature's weight reflects its ability to distinguish among the class values. Features are ranked by weight and those that exceed a user-specified threshold are selected to form the final subset. The algorithm works by randomly sampling instances from the training data. For each instance sampled the nearest instance of the same class (nearest hit) and opposite class (nearest miss) is found. An attribute's weight is updated according to how well its values distinguish the sampled instance from its nearest hit and nearest miss. An attribute will receive a high weight if it differentiates between instances from different classes and has the same value for instances of the same class. However, this method only works for binary classification problem. RELIEF is further extended by Igor Kononenko [17] to support multi-classification problem.

Consider training data S with size n having features $\{f_1, f_2, f_3, \ldots\ldots\ldots, f_p\}$. An instance X is denoted by p-dimensional vector $\{x_1, x_2, x_3, \ldots\ldots\ldots, x_p\}$, where x_j denotes the value of feature f_j of instance X.

Given training data S, sample size m, and a threshold of relevancy τ, Relief detects those features which are statistically relevant to the target concept. τ encodes a relevance threshold ($0 \leq \tau \leq 1$). The value of τ should be chosen such that $\tau \leq 1/\sqrt{\alpha m}$, where **α** is the probability of rejecting the hypothesis when it is true. It assumes the scale of every feature is either nominal (including boolean) or numerical (integer or real). Differences of feature values between two instances X and Y are defined by the following function diff.

When x_k and y_k are nominal,

$$\text{diff}(x_k, y_k) = \begin{cases} 0 & \text{if } x_k \text{ and } y_k \text{ are the same} \\ 1 & \text{if } x_k \text{ and } y_k \text{ are the different} \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.15)$$

When x_k and y_k are numerical,

$$\text{diff}(x_k, y_k) = (x_k, - y_k) / nu_k \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.16)$$

where, nu_k is a normalization unit to normalize the values of diff into the interval [0, l].

The weight of the feature is updated by the following function:

$$W_i = W_i - \text{diff}(x_i, \text{near-hit})^2 + \text{diff}(x_i, \text{near-miss})^2 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.17)$$

Algorithm:

RELIEF(*S*, p, *m*, τ)

Input: *S*, training set; p, the set of conditional

features; *m*, sample size; τ, weight threshold value

Output: *R*, the feature subset

(1) $R \leftarrow \{\}$

(2) **foreach** *Wa*, $Wa \leftarrow 0$

(3) **foreach** $i = 1$ to m

(4) choose an object *X* from Srandomly

(5) calculate *X*'s nearHit (*nH*) and nearMiss (*nM*)

(6) **foreach** $j = 1$ to p

(7) $Wj \leftarrow Wj - d(x_j, nH_j)/m + d(x_j, nM_j)/m$

(8) **foreach** $j = 1...|C|$

(9) **if** $Wj \geq \tau$; $R \leftarrow R \cup \{ j \}$ OR select the k top most features having high relevance

(10) **return** *R*

Gopala Krishna Murthy Nookalaet al.[18] performed comparative analysis of 14 different classification algorithms and their performance has been evaluated by using 3 different cancer data sets. The results indicate that none of the classifiers outperformed all others in terms of the accuracy when applied on all the 3 data sets. Most of the algorithms performed better as the size of the data set is increased.

D. L. Gupta et al. [19] analyse the different tree based classification method; 48, Random Forest (RF), Reduce Error Pruning (REP) and Logistic Model Tree (LMT) to classify the "WEATHER NOMINAL" open source Data Set. It is found that RF had highest accuracy followed by REM and LMT and then J48 respectively.

M. Vasantha and Subbiah Bharathy [20] published a paper which analyse the performance of correlation and consistency based feature selection methods with tree based classifications methods in Mammogram dataset.

3.2 Methodology

3.2.1 Research Methodology

Research is a careful study performed to find out new things in a systematic way. In a scientific method of research at first problem is formulated then output information is generated from collected input data and output is analyzed and finally the result is generalized. This dissertation work is truly scientific and flows in the same way. The main exploration of this dissertation focuses on determining the best tree based classification method for filter feature selection techniques: Chi-square method and Relief. In this dissertation first Chi-square and Relief methods will applied to datasets to reduce the dimension then C4.5, CART, LMT, Random tree are used for classification. The data needed to conduct the experiment will be taken UCI [21] machine learning repository. Output information gathered is analyzed in a quantitative approach. Finally, conclusion will be drawn using the empirical analysis of captured datasets.

3.2.2 Evaluation metrics

Let D = {D1,D2, . . . ,Dk } denote a partitioning of the testing points based on their true class labels, where

$$D_j = \{x_i \in D \mid y_i = C_j\}$$

Let $n_i = |D_i|$ denote the size of true class C_i.

Let R = {R1,R2, . . . ,Rk } denote a partitioning of the testing points based on the predicted labels, that is,

$$Rj = \{x_i \in D \mid \hat{y}_i = C_j\}$$

Let $m_j = |R_j|$ denote the size of the predicted class C_j.

R and D induce a k × k contingency table N, also called a confusion matrix, defined as follows:

$$N(i, j) = n_{i\,j} = |R_i \cap Dj| = |\{x_a \in D \mid \hat{y}_a = C_i \text{ and } y_a = Cj\}|$$

where $1 \leq i, j \leq k$. The count $n_{i\,j}$ denotes the number of points with predicted class ci whose true label is Cj. Thus, $n_{i\,i}$ (for $1 \leq i \leq k$) denotes the number of cases where the classifier agrees on the true label C_i . The remaining counts $n_{i\,j}$, with $i \neq j$, are cases where the classifier and true labels disagree.

Following parameters were used for the validation classification algorithm:

3.2.2.1 Precision

The class-specific precision of the classifier M for class C_i is given as the fraction of correct predictions over all points predicted to be in class C_i

$$acc_i = prec_i = n_{i\,i} / m_i \qquad \text{.....(3.18)}$$

where m_i is the number of examples predicted as C_i by classifier M. The higher the precision on class C_i the better the classifier.

3.2.2.2 Recall

The class-specific coverage or recall of M for class C_i is the fraction of correct predictions over all points in class C_i :

$$coverage_i = recall_i = n_{i\,i} / n_i \quad \text{.....(3.19)}$$

where n_i is the number of points in class C_i . The higher the recall the better the classifier.

3.2.2.3 F-measure

The class-specific F-measure tries to balance the precision and recall values, by computing their harmonic mean for class c_i :

$$Fi = 2n_{i\,i} / n_{i+} m_i \quad \text{.....(3.20)}$$

Chapter 4 Implementation

4.1 Tools used

All the algorithms are implemented in Java language using Eclipse IDE with the partial use of WEKA's libraries.

4.1.1 Programming language

For the implementation of proposed algorithm Java Programming Language is used. Java is a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any hardware/operating-system platform. This is achieved by compiling the Java language code to an intermediate representation called Java bytecode, instead of directly to platform-specific machine code. Java bytecode instructions are analogous to machine code, but they are intended to be interpreted by a virtual machine written specifically for the host hardware. End-users commonly use a Java Runtime Environment installed on their own machine for standalone Java applications, or in a Web browser for Java applets.

Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types. It is designed as a garbage-collected language ease the programmers virtually all memory management problems. Java also incorporates the concepts of exception handling which captures series errors and eliminates any risk of crashing the system.

4.1.2 Eclipse IDE

Eclipse is an integrated development environment which contains base workspace and an extensible plug-in system for customizing the environment. Eclipse SDK is free and open source software mostly written in Java. The initial software development can extend its ability by installing plug-ins written for Eclipse Platform, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.

The Eclipse SDK includes the Eclipse Java development tools, offering an IDE with a built-in incremental Java compiler and a full model of the Java source files. This allows advanced refactoring techniques and analysis. Eclipse implements the graphical elements of the Java toolkit called SWT. It provides the Rich client platform for developing general purpose applications.

4.1.3 WEKA Workbench

The WEKA workbench is a collection of state-of-the-art machine learning algorithms and data pre-processing tools [22]. It includes virtually all the ML algorithms. It provides extensive support for the whole process of experimental data mining, including preparing the input data, evaluating learning schemes statistically, and visualizing the input data and the result of learning. As well as a variety of learning algorithms, it includes a wide range of pre-processing tools. This diverse and comprehensive toolkit is accessed through a common interface so that its users can compare different methods and identify those that are most appropriate for the problem at hand.

WEKA was developed at the University of Waikato in New Zealand; the name stands for *Waikato Environment for Knowledge Analysis*. The system is written in Java and distributed under the terms of the GNU General Public License. It runs on almost any platform and has been tested under Linux, Windows, and Macintosh operating systems—and even on a personal digital assistant. It provides a uniform interface to many different learning algorithms, along with methods for pre- and post-processing and for evaluating the result of learning schemes on any given dataset.

4.2 Chi-square module

```
for (int i = 0; i < numClasses; i++)
{
        for (int j = 0; j  < numValues; j++)
        {
                additions[j][i] + = (columnSums[i] / sum) *  counts[k][j][numClasses];
    }
}
for (int i = 0; i < numClasses; i++)
{
        for (int j = 0; j  < numValues; j++)
        {
                additions[j][i]+= (counts[k][j][i] / sum) * counts[k][numValues][numClasses];
    }
 }

// Make new contingency table
```

```
double[][] newTable = new double[numValues][numClasses];
for (int i = 0; i < numValues; i++)
{
for (int j = 0; j < numClasses; j++)
        {
        newTable[i][j] = counts[k][i][j] + additions[i][j];
         }
}
counts[k] = newTable;
// Compute chi-squared values
m_ChiSquareds = new double[data.numAttributes()];
for (int i = 0; i < data.numAttributes(); i++) {
if (i != classIndex) {
      m_ChiSquareds[i] = ContingencyTables.
chiVal(ContingencyTables.reduceMatrix(counts[i]), false);
    }
  }
 }
```

4.3 Relief module

```
for (int i = 0; i < totalInstances; i++) {
if (totalInstances == m_numInstances) {
      z = i;
    }
else {
      z = r.nextInt()%m_numInstances;
    }

if (z < 0) {
z *= -1;
    }

if (!(m_trainInstances.instance(z).isMissing(m_classIndex))) {
      // first clear the knn and worst index stuff for the classes
```

```
for (int j = 0; j < m_numClasses; j++) {
        m_index[j] = m_stored[j] = 0;

for (int k = 0; k < m_Knn; k++) {
          m_karray[j][k][0] = m_karray[j][k][1] = 0;
        }
       }
findKHitMiss(z);
updateWeightsDiscreteClass(z);
     }
   }

   // now scale weights by 1/m_numInstances (nominal class) or
   // calculate weights numeric class
for (int i = 0; i < m_numAttribs; i++) {if (i != m_classIndex) {
if (m_numericClass) {
       m_weights[i] = m_ndcda[i]/m_ndc -
         ((m_nda[i] - m_ndcda[i])/((double)totalInstances - m_ndc));
     }
else {
       m_weights[i] *= (1.0/(double)totalInstances);
     }
   }
   }
```

Chapter 5 Data collection and Analysis

5.1 Data Collection

All the data used for this research are primary data taken from UCI [23] machine learning repository. Table below summarizes the benchmark dataset used for the analysis of the algorithm.

S.N	Name	Instances	Attributes	Classes
1	Breast-cancer	286	9	2
2	Car	1728	6	4
3	Cmc	1473	9	3
4	Credit	690	15	2
5	Credit-g	1000	20	2
6	Dermatology	362	34	6
7	Diabetes	768	8	2
8	Glass	210	13	7
9	Hepatitis	155	19	2
10	House-votes-84	435	16	2
11	Ionosphere	351	34	2
12	Iris	150	4	3
13	Labor	57	16	2
14	Mammographic_masses	980	6	2
15	Optdigits	1797	64	10
16	Segment-challenge	1500	19	7
17	Soybean	683	35	19
18	Spect	267	22	2
19	Wine	178	13	3
20	Zoo	101	17	7

Table 5.1: List of Datasets

5.1.1 Training & Testing data

For each dataset 30% of instances of dataset are used for training and rest of the instances are used for testing.

Sample of Training data

1, 0, 0.99539, -0.05889, 0.85243, 0.02306, 0.83398, 0.37708, 1, 0.03760, 0.85243, -0.17755, 0.59755, -0.44945, 0.60536, 0.38223, 0.84356, -0.38542, 0.58212, -0.32192, 0.56971, -0.29674, 0.36946, 0.47357, 0.56811, -0.51171, 0.41078, -0.46168, 0.21266, 0.34090, 0.42267, -0.54487, 0.18641, -0.45300, g

1, 0, 1, -0.18829, 0.93035, -0.36156, -0.10868, -0.93597, 1, 0.04549, 0.50874, -0.67743, 0.34432, -0.69707, -0.51685, -0.97515, 0.05499, -0.62237, 0.33109, -1, -0.13151, -0.45300, -0.18056, 0.35734, -0.20332, -0.26569, -0.20468, -0.18401, -0.19040, 0.11593, -0.16626, -0.06288, -0.13738, -0.02447, b

1,0,1,-0.03365,1,0.00485,1,-0.12062,0.88965,0.01198,0.73082,0.05346,0.85443,0.00827,0.54591,0.00299,0.83775,-0.13644,0.75535,-0.08540,0.70887,-0.27502,0.43385,-0.12062,0.57528,-0.40220,0.58984,-0.22145,0.43100,-0.17365,0.60436,-0.24180,0.56045,-0.38238,g

1,0,1,-0.45161,1,1,0.71216,-1,0,0,0,0,0,0,-1,0.14516,0.54094,-0.39330,-1,-0.54467,-0.69975,1,0,0,1,0.90695,0.51613,1,1,-0.20099,0.25682,1,-0.32382,1,b

Figure 5.1.1 (a): Sample data of Ionosphere dataset used for training

b, 30.83, 0, u, g, w, v, 1.25, t, t, 01, f, g, 00202, 0, +
a, 58.67, 4.46, u, g, q, h, 3.04, t, t, 06, f, g, 00043, 560, +
a, 24.50, 0.5, u, g, q, h, 1.5, t, f, 0, f, g, 00280, 824, +
b, 20.67, 5.29, u, g, q, v, 0.375, t, t, 01, f, g, 00160, 0, -
b, 34.08, 6.5, u, g, aa, v, 0.125, t, f, 0, t, g, 00443, 0, -

Figure 5.1.1 (b): Sample data of Credit dataset used for training

Sample of Testing data

1,0,1,-0.14754,1,0.04918,0.57377,-0.01639,0.65574,0.01639,0.85246,-0.03279,0.72131,0,0.68852,-0.16393,0.19672,-0.14754,0.65558,-0.17176,0.67213,0.03279,1,-0.29508,0.31148,-0.34426,0.52385,-0.20325,0.32787,-0.03279,0.27869,-0.44262,0.49180,-0.06557,b

1,0,0.98182,0,0.88627,0.03131,0.86249,0.04572,0.80000,0,0.69091,0.04545,0.79343,0.0843
6,0.77118,0.09579,0.62727,0.25455,0.68182,0.12727,0.70674,0.12608,0.68604,0.13493,0.74
545,0.22727,0.64581,0.15088,0.67273,0.02727,0.60715,0.16465,0.58840,0.17077,g
1,0,1,0.06843,1,0.14211,1,0.22108,1,-0.12500,1,0.39495,1,0.48981,1,0.58986,-
0.37500,1,1,0,1,0.92001,1,1,1,1,1,1,1,0.25000,1,1,1,1,g
0,0,-1,-1,0,0,0,0,0,0,0,0,0,1,-1,0,0,-1,-1,0,0,1,1,1,-1,1,-1,0,0,0,0,0,0,b

Figure 5.1.1 (c): Sample data of Ionosphere dataset used for testing

b, 31.67, 16.165, u, g, d, v, 3, t, t, 09, f, g, 00250, 730, +
a, 23.42, 0.79, y, p, q, v, 1.5, t, t, 02, t, g, 00080, 400, +
b, 21.50, 9.75, u, g, c, v, 0.25, t, f, 0, f, g, 00140, 0, -
b, 49.58, 19, u, g, ff, ff, 0, t, t, 01, f, g, 00094, 0, -
a, 27.67, 1.5, u, g, m, v, 2, t, f, 0, f, s, 00368, 0, -

Figure 5.1.1 (d): Sample data of Credit dataset used for testing

5.2 Experiment & Result

5.2.1 Experimental setup

The aim is to experimentally determine the effectiveness of filter based feature selection method; chi-square and Relief in Decision tree based classification methods.

The experiments were performed using Intel (R) Core (TM) i5-3230M CPU @ 2.60GHz 2.60 GHz with 4.00 GB RAM in 64-bit Windows 8 Operating System.

The feature selection method; chi-square and relief are applied to above 20 enlisted datasets shown in Table 5.1 taken from the UCI Repository of Machine learning Database to reduce their dimension. The decision tree based classification method is then applied to these reduced dataset and their performance is measured in terms of accuracy, precision, recall and F-measure.

The result of the experiment is shown in following tables.

5.2.2 Evaluation metrics Result

5.2.2.1 Precision

S.N	Algorithm / Dataset	C4.5	Cart	LMT	Random Tree
1	Breast-cancer	0.683	0.712	0.727	0.689
2	Car	0.763	0.744	0.790	0.766
3	Cmc	0.514	0.564	0.541	0.481
4	Credit	0.840	0.826	0.840	0.803
5	Credit-g	0.696	0.693	0.730	0.634
6	Dermatology	0.799	0.831	0.836	0.818
7	Diabetes	0.742	0.730	0.761	0.679
8	Glass	0.614	0.62	0.587	0.609
9	Hepatitis	0.313	0.716	0.849	0.828
10	House-votes-84	0.959	0.957	0.959	0.942
11	Ionosphere	0.872	0.874	0.844	0.844
12	Iris	0.959	0.952	0.959	0.952
13	Labor	0.820	0.85	0.9	0.9
14	Mammographic_masses	0.825	0.83	0.831	0.834
15	Optdigits	0.832	0.759	0.932	0.759
16	Segment-challenge	0.926	0.944	0.941	0.915
17	Soybean	0.754	0.839	0.910	0.807
18	Spect	0.724	0.701	0.662	0.715
19	Wine	0.861	0.912	0.984	0.917
20	Zoo	0.734	0.437	0.737	0.639
	Average	0.757	0.773	0.815	0.775

Table: 5.2.2.1(a): Precision Result in Chi-square reduced dataset

S.N	Algorithm / Dataset	C4.5	Cart	LMT	Random Tree
1	Breast-cancer	0.699	0.686	0.695	0.633
2	Car	0.763	0.744	0.790	0.766
3	Cmc	0.504	0.534	0.549	0.455
4	Credit	0.836	0.840	0.840	0.836
5	Credit-g	0.685	0.694	0.727	0.673
6	Dermatology	0.914	0.909	0.916	0.839
7	Diabetes	0.742	0.743	0.776	0.645
8	Glass	0.631	0.615	0.579	0.608
9	Hepatitis	0.778	0.778	0.806	0.767
10	House-votes-84	0.954	0.959	0.959	0.934
11	Ionosphere	0.872	0.876	0.854	0.873
12	Iris	0.959	0.952	0.959	0.952
13	Labor	0.820	0.9	0.9	0.9
14	Mammographic_masses	0.832	0.83	0.829	0.795
15	Optdigits	0.810	0.834	0.925	0.754
16	Segment-challenge	0.930	0.940	0.950	0.939
17	Soybean	0.772	0.800	0.922	0.798
18	Spect	0.713	0.719	0.725	0.702
19	Wine	0.861	0.915	0.984	0.891
20	Zoo	0.762	0.685	0.737	0.720
	Average	0.789	0.795	0.820	0.771

Table: 5.2.2.1(b): Precision Result in Relief reduced dataset

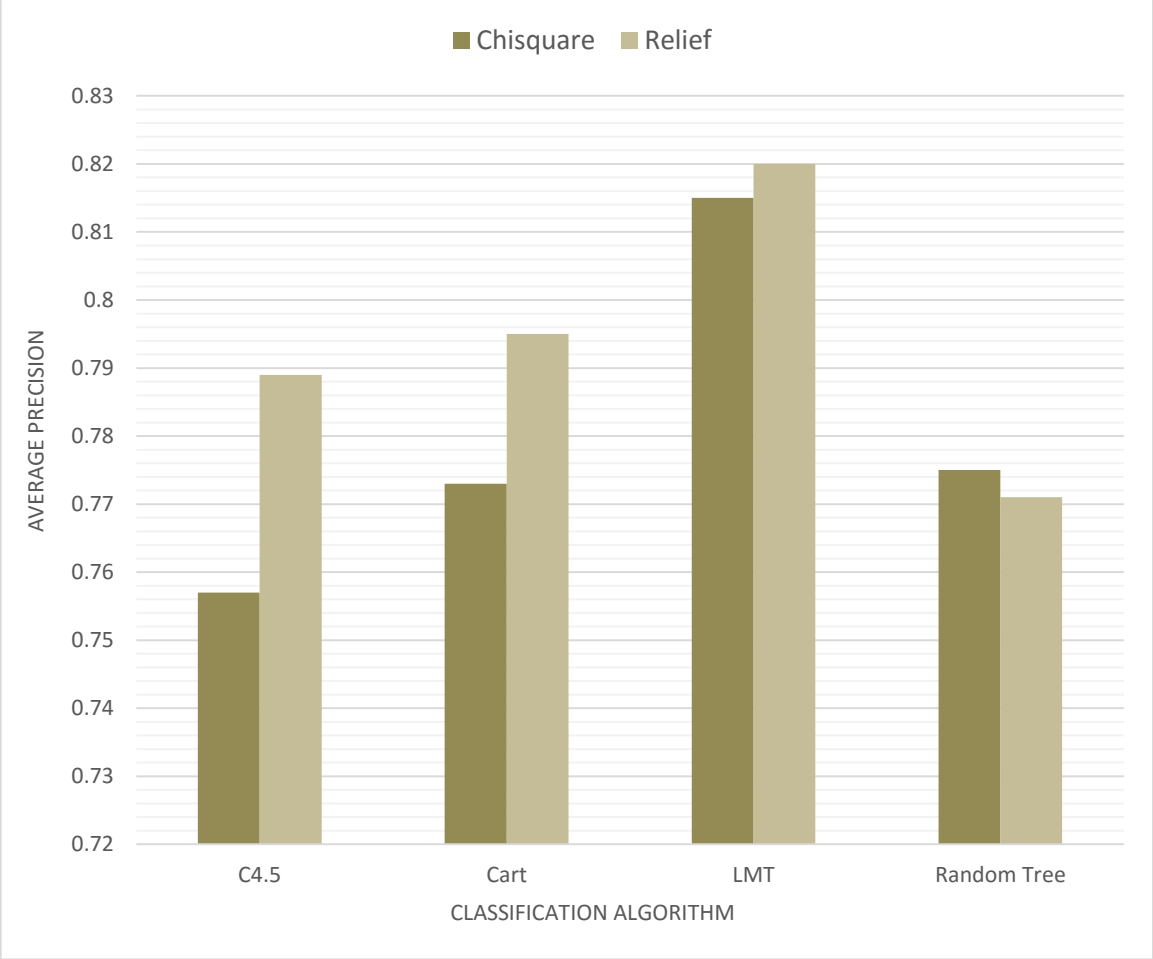

Fig: 5.2.2.1(c) Graph showing average precision

The above graph shows that the average precision value of LMT method is higher than all the other methods. When Chi-square feature selection is applied LMT method gives the average precision value of 0.815 which is 5.5%, 4.2%, and 4%, greater than C4.5, CART, and Random Tree method respectively. Similarly, when Relief feature selection method is used LMT gives the average precision value of 0.82 which is 3.1%, 2.2%, and 4.9 % greater than C4.5, CART, and Random Tree method respectively.

5.2.2.2 Recall

S.N	Algorithm / Dataset	C4.5	Cart	LMT	Random Tree
1	Breast-cancer	0.712	0.726	0.741	0.716
2	Car	0.768	0.794	0.807	0.780

3	Cmc	0.516	0.513	0.536	0.478
4	Credit	0.826	0.826	0.826	0.803
5	Credit-g	0.697	0.693	0.731	0.617
6	Dermatology	0.790	0.831	0.835	0.815
7	Diabetes	0.740	0.730	0.766	0.686
8	Glass	0.605	0.62	0.52	0.533
9	Hepatitis	0.739	0.716	0.835	0.807
10	House-votes-84	0.958	0.957	0.959	0.941
11	Ionosphere	0.870	0.874	0.841	0.841
12	Iris	0.952	0.952	0.952	0.943
13	Labor	0.825	0.85	0.9	0.9
14	Mammographic_masses	0.819	0.830	0.829	0.832
15	Optdigits	0.826	0.759	0.93	0.756
16	Segment-challenge	0.926	0.944	0.94	0.913
17	Soybean	0.803	0.839	0.904	0.785
18	Spect	0.717	0.701	0.647	0.706
19	Wine	0.856	0.912	0.984	0.912
20	Zoo	0.845	0.437	0.817	0.704
	Average	0.789	0.774	0.815	0.773

Table 5.2.2.2(a): Recall Result in Chi-square reduced dataset

S.N	Algorithm / Dataset	C4.5	Cart	LMT	Random Tree
1	Breast-cancer	0.701	0.701	0.716	0.652
2	Car	0.768	0.794	0.807	0.786
3	Cmc	0.491	0.518	0.531	0.453
4	Credit	0.832	0.826	0.826	0.834
5	Credit-g	0.693	0.701	0.721	0.634

6	Dermatology	0.902	0.902	0.906	0.823
7	Diabetes	0.697	0.727	0.781	0.647
8	Glass	0.627	0.593	0.593	0.58
9	Hepatitis	0.697	0.697	0.752	0.752
10	House-votes-84	0.954	0.957	0.957	0.934
11	Ionosphere	0.870	0.874	0.854	0.873
12	Iris	0.952	0.952	0.952	0.943
13	Labor	0.825	0.9	0.9	0.9
14	Mammographic_masses	0.830	0.830	0.829	0.795
15	Optdigits	0.808	0.831	0.924	0.750
16	Segment-challenge	0.930	0.94	0.950	0.936
17	Soybean	0.804	0.816	0.919	0.8
18	Spect	0.692	0.717	0.711	0.679
19	Wine	0.856	0.912	0.984	0.888
20	Zoo	0.845	0.746	0.817	0.732
	Average	0.789	0.797	0.822	0.770

Table 5.2.2.2(b): Recall Result in Relief reduced dataset

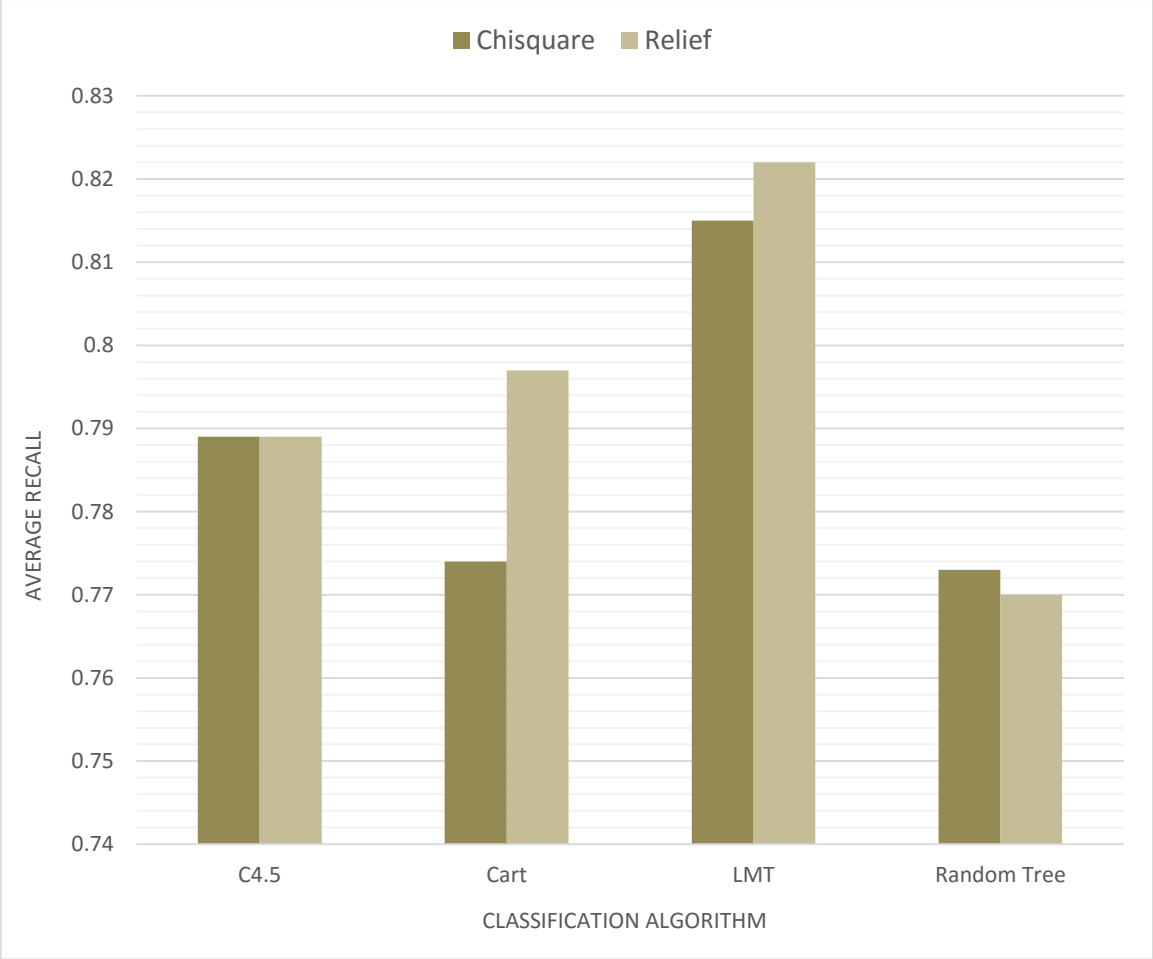

Fig: 5.2.2.1(c) Graph showing average recall

Above graph shows the recall value obtained by different referenced algorithms. The average recall value of LMT method after applying chi-square feature selection algorithm is 0.815 which is 2.6%, 4.1%, and 4.2% greater than C4.5, CART, and Random Tree method respectively. When Relief feature selection method is applied to the dataset and then above decision tree based classification method are used for classification, LMT gives the recall value 0.822 which is 3.3%, 2.5%, and 5.2% greater than C4.5, CART, and Random Tree respectively.

5.2.2.3 F-measure

S.N	Algorithm / Dataset	C4.5	Cart	LMT	Random Tree
1	Breast-cancer	0.678	0.670	0.705	0.689
2	Car	0.763	0.767	0.797	0.766

3	Cmc	0.514	0.517	0.538	0.481
4	Credit	0.826	0.826	0.826	0.803
5	Credit-g	0.697	0.703	0.730	0.634
6	Dermatology	0.790	0.811	0.820	0.818
7	Diabetes	0.740	0.728	0.763	0.679
8	Glass	0.605	0.587	0.511	0.609
9	Hepatitis	0.739	0.746	0.841	0.828
10	House-votes-84	0.958	0.958	0.958	0.948
11	Ionosphere	0.870	0.874	0.842	0.844
12	Iris	0.952	0.952	0.952	0.952
13	Labor	0.817	0.855	0.9	0.9
14	Mammographic_masses	0.816	0.83	0.829	0.834
15	Optdigits	0.826	0.759	0.930	0.759
16	Segment-challenge	0.925	0.944	0.940	0.915
17	Soybean	0.767	0.828	0.905	0.807
18	Spect	0.719	0.706	0.652	0.715
19	Wine	0.855	0.911	0.984	0.917
20	Zoo	0.784	0.265	0.737	0.639
	Average	0.782	0.769	0.808	0.777

Table 5.2.2.3(a): F-measure Result in Chi-square reduced dataset

S.N	Algorithm / Dataset	C4.5	Cart	LMT	Random Tree
1	Breast-cancer	0.700	0.691	0.697	0.640
2	Car	0.763	0.767	0.797	0.768
3	Cmc	0.495	0.522	0.536	0.452
4	Credit	0.833	0.826	0.826	0.835
5	Credit-g	0.689	0.697	0.724	0.649

6	Dermatology	0.901	0.900	0.903	0.826
7	Diabetes	0.705	0.732	0.776	0.646
8	Glass	0.625	0.561	0.570	0.582
9	Hepatitis	0.726	0.726	0.772	0.759
10	House-votes-84	0.954	0.958	0.958	0.934
11	Ionosphere	0.871	0.874	0.854	0.873
12	Iris	0.952	0.952	0.952	0.952
13	Labor	0.817	0.9	0.9	0.9
14	Mammographic_masses	0.830	0.83	0.829	0.794
15	Optdigits	0.807	0.831	0.924	0.747
16	Segment-challenge	0.930	0.94	0.951	0.937
17	Soybean	0.777	0.800	0.919	0.784
18	Spect	0.700	0.718	0.715	0.684
19	Wine	0.855	0.911	0.984	0.887
20	Zoo	0.791	0.704	0.767	0.712
	Average	0.786	0.792	0.818	0.768

Table 5.2.2.3(b): F-measure Result in Relief reduced dataset

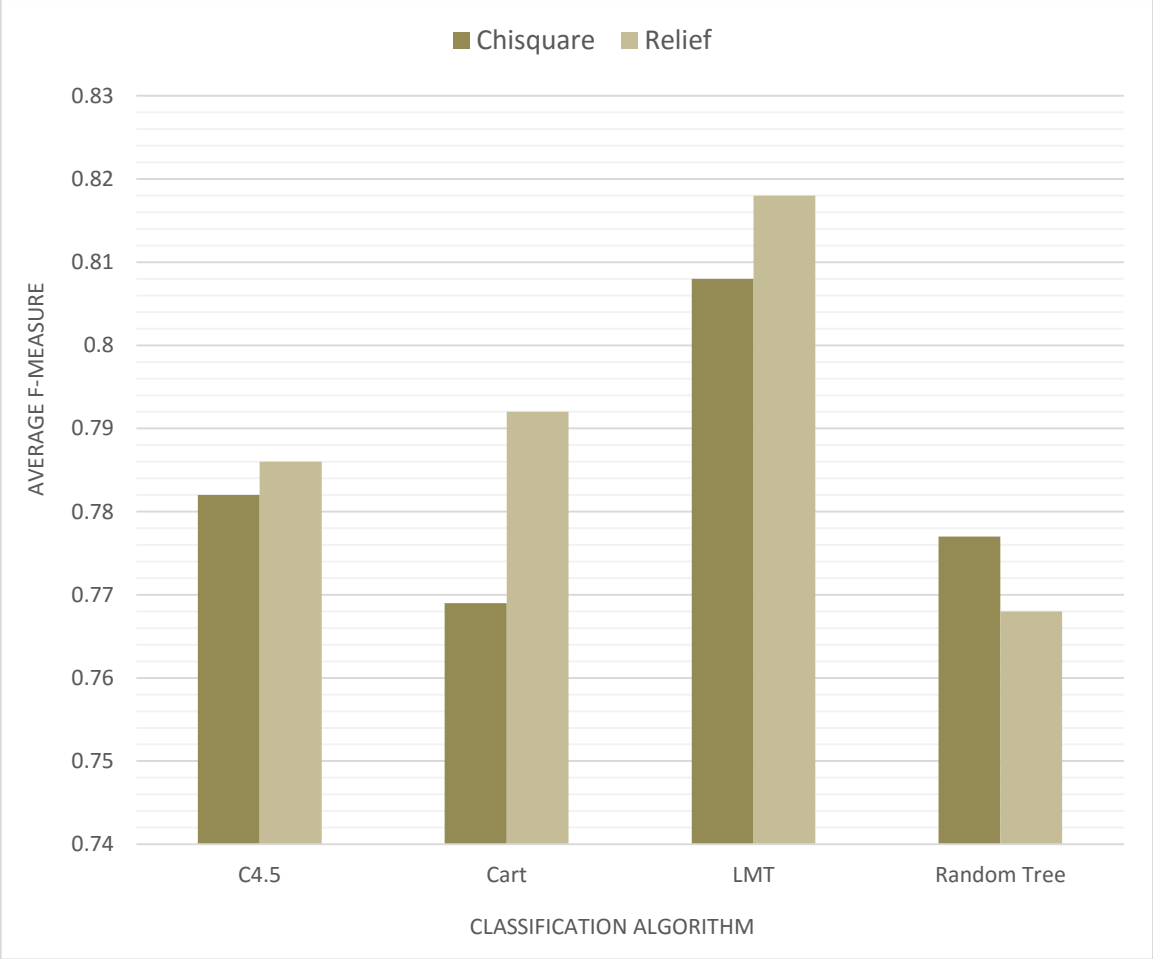

Figure 5.2.2.3(c): Graph showing Average F-measure

Graph 5.2.2.3(c) shows that the average F-measure value of the LMT method is higher than all other referenced algorithms. LMT method gives the average F-measure value 0.808 which is 2.6%, 3.9%, and 3.1% greater than the C4.5, CART, and Random Tree algorithm when Chi-square is used as feature selection technique. When Relief is used as feature selection method, LMT gives 0.818 average f-measure value which is 3.2%, 2.6%, and 5% greater than C4.5, CART, and Random Tree respectively.

Chapter 6 Conclusion and Future Work

6.1 Conclusion

The result of the experimental study indicates that Chi-square and Relief based feature selection methods are more suitable when applied with the LMT classification method. Chi-square feature selection method gives the best result when applied with LMT classification method followed by C4.5, Random Tree and CART respectively. In case Relief based feature selection method LMT gives the best result followed by CART, C4.5 and Random Tree respectively.

6.2 Future Work

The result of the classification method also depends upon the nature of the datasets. Thus, furthermore analysis can be done on these feature selection method using nature of the datasets.

References

1. Z.J. Mohammed, M. Wakner, *"Data mining and Analysis fundamental concepts and Analysis".*
2. I. Guyon, S. Gunn, N. Masoud, L.A. Zadeh, *"Feature Extraction, Foundations and Applications".*
3. M. Ramaswami and R. Bhaskaran*, "A Study on Feature Selection Techniques in Educational Data Mining", Journal of Computing, Volume 1, Issue 1, Decenber 2009.*
4. H. Almuallim and T. G. Dietterich. "*Learning boolean concepts in the presence of many irrelevant features*," *Artificial Intelligence*, vol. 69, no. 1-2, pp. 279–305, 1994.
5. D. Koller and M. Sahami, "*Toward optimal feature selection*," In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 284–292, 1996.
6. I. Guyon, S. Gunn, N. Masoud, L.A. Zadeh, *"Feature Extraction, Foundations and Applications".*
7. Y. kaung*, "A Comparative Study on Feature Selection Methods and Their Applications in Causal Inference", 2009.*
8. L. Ladha and T. Deep,*. "Feature selection methods and Algorithms", IJCSE.*
9. J. Han and M. Kamber, *"Data mining concepts and techniques", 2nd Edition.*
10. O. Maion and L. Rokach*, "Data mining and Knowledge discovery handbook", Springer, 2nd Edition.*
11. J.R. Quinlan, *"C4.5: Programs for Machine Learning. Morgan Kaufmann", 1993*.
12. Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J, *"Classification and regression trees", 1984.*
13. N. Landwehr, M. Hall, and E. Frank, *" Logistic model trees". for Machine Learning.,Vol. 59(1-2),pp.161-205, 2005.*
14. Friedman, J., T. Hastie, and R. Tibshirani: 2000, *"Additive Logistic Regression: a Statistical View of Boosting". The Annals of Statistic* ***38****(2), 337–374.*
15. Y. kaung*, "A Comparative Study on Feature Selection Methods and Their Applications in Causal Inference", 2009.*
16. K. Kenji, L.A. Rendell*, "The Feature Selection Problem: Traditional methods and new Algorithm", AAAI, 1992.*
17. Kononenko Igor, "*Estimating Attributes: Analysis and Extensions of RELIEF", 1994.*

18. D.L Gupta et al., *"Performance Analysis of Classification Tree Learning Algorithms", 2012.*
19. Gopala Krishna Murthy Nookala et al. *"Performance Analysis and Evaluation of Different Data Mining Algorithms used for Cancer Classification", 2013.*
20. M. Vasantha, V.Subbiah Bharathy, *"Evaluation of Attribute selection methods with tree based supervised classification – A case study with Mammogram", International Jounal of Computer Applicaitons, October 2010.*
21. C. Blake, E. Keogh, and C.J. Merz, *"UCI repository of machine learning databases", 1998.*
22. Weka web site, *http://www.cs.waikato.ac.nz/ml/WEKA/*

Bibliography

1. I.H. Witten, E. Frank, and M.A. Hall*, "Data mining practical machine learning tools and techniques", 3rd Edition.*
2. O. Maion and L. Rokach*, "Data mining and Knowledge discovery handbook", Springer, 2nd Edition.*
3. R. Jensen and Q. Shen, *"Computational Intelligence and Feature selection".*
4. S. Marshall, *"Machine learning an algorithmic perspective", CRC Press.*

Appendix
(Code for Implementation)

```
import java.util.*;
import java.io.*;
import weka.classifiers.trees.J48;
import weka.classifiers.trees.LMT;
import weka.classifiers.trees.RandomTree;
import weka.classifiers.trees.SimpleCart;
import weka.core.*;
import weka.attributeSelection.*;
import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.MultipleClassifiersCombiner;
import weka.filters.Filter;
import weka.filters.SimpleFilter;
import weka.filters.unsupervised.attribute.RandomSubset;
public class AlgoComparison {
	public static int noofinstance;
	public static int traininstance;
	public static int testinstance;

	public Instances gettraindata(Instances reducedataset)
	{
		noofinstance =reducedataset.numInstances();
		//System.out.println("Total instances:"+noofinstance);
		traininstance=(noofinstance*30)/100;
		//System.out.println("Number of trainig instance:"+traininstance);
		testinstance=noofinstance-traininstance;
		Instances traindata=new Instances(reducedataset,0,traininstance);
		//System.out.println(traindata);
		return(traindata);

	}
	public Instances gettestdata(Instances reducedataset)
	{
		Instances testdata=new Instances(reducedataset,traininstance,testinstance);
		return(testdata);
	}
	public  Instances executeRelief(Instances randdataset) throws Exception
	{
		int noofattriselect=randdataset.numAttributes()*50/100;
		ReliefFAttributeEval eval=new ReliefFAttributeEval();
```

```
		AttributeSelection attri=new AttributeSelection();
		eval.setNumNeighbours(10);
		eval.setSampleSize(-1);
		eval.setSeed(1);
		//eval.setSigma(2);
		//eval.setWeightByDistance(false);
		Ranker ran=new Ranker();
		ran.setGenerateRanking(true);
		ran.setNumToSelect(noofattriselect);

		ran.setThreshold(0.0);
		attri.setEvaluator(eval);
		attri.setSearch(ran);
		attri.SelectAttributes(randdataset);
		//System.out.println(attri.toResultsString());
		Instances reducedataset=new
Instances(attri.reduceDimensionality(randdataset));
		return reducedataset;

	}
	public Instances executeChisquare(Instances randdataset) throws Exception
	{
		int noofattriselect=randdataset.numAttributes()/2;
		ChiSquaredAttributeEval eval=new ChiSquaredAttributeEval();
		AttributeSelection attri=new AttributeSelection();
		Ranker ran=new Ranker();
		ran.setGenerateRanking(true);
		ran.setNumToSelect(noofattriselect);

		ran.setThreshold(0.0);
		attri.setEvaluator(eval);
		attri.setSearch(ran);
		attri.SelectAttributes(randdataset);
		//System.out.println(attri.toResultsString());
		Instances reducedataset=new
Instances(attri.reduceDimensionality(randdataset));
		return reducedataset;

	}
	public J48 trainC45Classifier(Instances traindata) throws Exception
	{
		J48 C45=new J48();
		C45.setConfidenceFactor(0.25f);
		C45.setMinNumObj(2);
```

```
        C45.setDebug(false);
        C45.setNumFolds(3);
        C45.setSeed(1);
        C45.buildClassifier(traindata);
        return(C45);
}
public SimpleCart trainCartClassifier (Instances traindata) throws Exception
{
        SimpleCart Cart =new SimpleCart();
        Cart.setDebug(false);
        Cart.setMinNumObj(2.0);
        Cart.setNumFoldsPruning(3);
        Cart.setSeed(1);
        Cart.setSizePer(1.0);
        Cart.buildClassifier(traindata);
        return(Cart);
}
public LMT trainLMTClassifier (Instances traindata)throws Exception
{
        LMT Lmt= new LMT();
        Lmt.setDebug(false);
        Lmt.setNumBoostingIterations(-1);
        Lmt.setMinNumInstances(15);
        Lmt.setWeightTrimBeta(0.0);
        Lmt.buildClassifier(traindata);
        return(Lmt);
}
public RandomTree trainRandomClassifier(Instances traindata)throws Exception
{
        RandomTree RT = new RandomTree();
        RT.setKValue(0);
        RT.setMaxDepth(0);
        RT.setMinNum(1.0);
        RT.setNumFolds(0);
        RT.setSeed(1);
        RT.buildClassifier(traindata);
        return (RT);
}
public int[] predictClass_C45(Instances reducedataset,J48 C45) throws Exception
{

        Instances testdata=new Instances(gettestdata(reducedataset));

        double[] pred=new double[testinstance];
```

```
                int[] prediction=new int[testinstance];

                for(int j=0;j<testdata.numInstances();j++)
                {
                        pred[j]=C45.classifyInstance(testdata.instance(j));

                        // Converting the doubel type result into integer type
                        prediction[j]=(int)pred[j];

                        //System.out.println("Classified as:"+(int)pred[j]);
                        //System.out.println("Correct
class:"+(int)testdata.instance(j).value(testdata.classIndex));
                        //System.out.println("Classified
as:"+testdata.classAttribute().value((int)pred[j]));

                }
                return(prediction);
        }
        public int[] predictClass_CART(Instances reducedataset,SimpleCart Cart) throws
Exception
        {

                Instances testdata=new Instances(gettestdata(reducedataset));

                double[] pred=new double[testinstance];
                int[] prediction=new int[testinstance];

                for(int j=0;j<testdata.numInstances();j++)
                {
                        pred[j]=Cart.classifyInstance(testdata.instance(j));

                        // Converting the doubel type result into integer type
                        prediction[j]=(int)pred[j];

                        //System.out.println("Classified as:"+(int)pred[j]);
                        //System.out.println("Correct
class:"+(int)testdata.instance(j).value(testdata.classIndex));
                        //System.out.println("Classified
as:"+testdata.classAttribute().value((int)pred[j]));

                }
                return(prediction);
        }
        public int[] predictClass_LMT(Instances reducedataset,LMT Lmt) throws Exception
```

```
	{

		Instances testdata=new Instances(gettestdata(reducedataset));

		double[] pred=new double[testinstance];
		int[] prediction=new int[testinstance];

		for(int j=0;j<testdata.numInstances();j++)
		{
			pred[j]=Lmt.classifyInstance(testdata.instance(j));

			// Converting the doubel type result into integer type
			prediction[j]=(int)pred[j];

			//System.out.println("Classified as:"+(int)pred[j]);
			//System.out.println("Correct
class:"+(int)testdata.instance(j).value(testdata.classIndex));
			//System.out.println("Classified
as:"+testdata.classAttribute().value((int)pred[j]));

		}
		return(prediction);
	}
	public int[] predictClass_RandomTree(Instances reducedataset,RandomTree RT)
throws Exception
	{

		Instances testdata=new Instances(gettestdata(reducedataset));

		double[] pred=new double[testinstance];
		int[] prediction=new int[testinstance];

		for(int j=0;j<testdata.numInstances();j++)
		{
			pred[j]=RT.classifyInstance(testdata.instance(j));

			// Converting the doubel type result into integer type
			prediction[j]=(int)pred[j];

			//System.out.println("Classified as:"+(int)pred[j]);
			//System.out.println("Correct
class:"+(int)testdata.instance(j).value(testdata.classIndex));
```

```
                    //System.out.println("Classified
as:"+testdata.classAttribute().value((int)pred[j]));

                }
                return(prediction);
        }

        public void printPrediction(int[] pred,Instances reducedataset,Classifier CL) throws
Exception
        {
                System.out.println("Number of testing instances:"+testinstance);
                Instances testdata=new Instances(gettestdata(reducedataset));
                Evaluation eval=new Evaluation(reducedataset);
                eval.evaluateModel(CL, testdata);
                double                                                  [][]conmatrix=new
double[reducedataset.numClasses()][reducedataset.numClasses()];
                conmatrix=eval.confusionMatrix();
                for(int a=0;a<reducedataset.numClasses();a++)
                {
                        for(int b=0;b<reducedataset.numClasses();b++)
                        {
                                System.out.print((int)conmatrix[a][b]);
                                System.out.print("   ");
                        }
                        System.out.println();
                }
                System.out.println(conmatrix.toString());
                //System.out.println(eval.toSummaryString());
                System.out.println("Precision:"+eval.weightedPrecision());
                System.out.println("Recall:"+eval.weightedRecall());
                System.out.println("F-measure:"+eval.weightedFMeasure());
                System.out.println("Accuracy:"+eval.pctCorrect());
                //System.out.println(eval.toClassDetailsString());

                /*for(int j=0;j<testdata.numInstances();j++)
                {
                        System.out.print((j+1));
                        System.out.print(" - ");
                        System.out.print(testdata.instance(j).toString(testdata.classIndex()));
                        System.out.print(" - ");
                        System.out.print(testdata.classAttribute().value((int)pred[j]));
                        System.out.print(" - ");
                        if(pred[j]==testdata.instance(j).classValue())
                                System.out.print("Yes");
```

```
                else
                        System.out.print("No");
                System.out.println();
        }*/
}
public void Relief_C45(Instances randdataset) throws Exception
{
        int[]prediction=new int[testinstance];
        Instances reducedataset=new Instances(executeRelief(randdataset));
        Instances traindata=new Instances(gettraindata(reducedataset));
        J48 C45 = new J48();
        C45=trainC45Classifier(traindata);
        prediction=predictClass_C45(reducedataset,C45);
        printPrediction(prediction,reducedataset,C45);
}

public void Relief_CART(Instances randdataset) throws Exception
{
        int[]prediction=new int[testinstance];
        Instances reducedataset=new Instances(executeRelief(randdataset));
        Instances traindata=new Instances(gettraindata(reducedataset));
        SimpleCart Cart=new SimpleCart();
        Cart=trainCartClassifier(traindata);
        prediction=predictClass_CART(reducedataset,Cart);
        printPrediction(prediction,reducedataset,Cart);
}
public void Relief_LMT(Instances randdataset) throws Exception
{
        int[]prediction=new int[testinstance];
        Instances reducedataset=new Instances(executeRelief(randdataset));
        Instances traindata=new Instances(gettraindata(reducedataset));
        LMT Lmt=new LMT();
        Lmt=trainLMTClassifier(traindata);
        prediction=predictClass_LMT(reducedataset,Lmt);
        printPrediction(prediction,reducedataset,Lmt);
}

public void Relief_RandomTree(Instances randdataset) throws Exception
{
        int[]prediction=new int[testinstance];
        Instances reducedataset=new Instances(executeRelief(randdataset));
        Instances traindata=new Instances(gettraindata(reducedataset));
        RandomTree RT=new RandomTree();
        RT=trainRandomClassifier(traindata);
```

```
        prediction=predictClass_RandomTree(reducedataset,RT);
        printPrediction(prediction,reducedataset,RT);
}

public void Chisquare_C45(Instances randdataset) throws Exception
{
        int[]prediction=new int[testinstance];
        Instances reducedataset=new Instances(executeChisquare(randdataset));
        Instances traindata=new Instances(gettraindata(reducedataset));
        J48 C45 = new J48();
        C45=trainC45Classifier(traindata);
        prediction=predictClass_C45(reducedataset,C45);
        printPrediction(prediction,reducedataset,C45);
}
public void chisquare_CART(Instances randdataset) throws Exception
{
        int[]prediction=new int[testinstance];
        Instances reducedataset=new Instances(executeChisquare(randdataset));
        Instances traindata=new Instances(gettraindata(reducedataset));
        SimpleCart Cart=new SimpleCart();
        Cart=trainCartClassifier(traindata);
        prediction=predictClass_CART(reducedataset,Cart);
        printPrediction(prediction,reducedataset,Cart);
}
public void chisquare_LMT(Instances randdataset) throws Exception
{
        int[]prediction=new int[testinstance];
        Instances reducedataset=new Instances(executeChisquare(randdataset));
        Instances traindata=new Instances(gettraindata(reducedataset));
        LMT Lmt=new LMT();
        Lmt=trainLMTClassifier(traindata);
        prediction=predictClass_LMT(reducedataset,Lmt);
        printPrediction(prediction,reducedataset,Lmt);
}
public void chisquare_RandomTree(Instances randdataset) throws Exception
{
        int[]prediction=new int[testinstance];
        Instances reducedataset=new Instances(executeChisquare(randdataset));
        Instances traindata=new Instances(gettraindata(reducedataset));
        RandomTree RT=new RandomTree();
        RT=trainRandomClassifier(traindata);
        prediction=predictClass_RandomTree(reducedataset,RT);
        printPrediction(prediction,reducedataset,RT);
}
```

```
	public static void main(String [] args) throws Exception{
		BufferedReader inputFile = new BufferedReader(new
FileReader("C:/Users/Rajesh/Desktop/New folder (2)/diabetes.arff"));
		Instances dataset=new Instances(inputFile);
		int i=dataset.classIndex();
		dataset.setClassIndex(i);

		//Randomizing the dataset
		int seed=1;
		Random rand=new Random(seed);
		Instances randdataset=new Instances(dataset);
		randdataset.randomize(rand);

		AlgoComparison algo=new AlgoComparison();
		algo.Chisquare_C45(randdataset);
		algo.chisquare_CART(randdataset);
		algo.chisquare_LMT(randdataset);
		algo.chisquare_RandomTree(randdataset);
		algo.Relief_C45(randdataset);
		algo.Relief_CART(randdataset);
		algo.Relief_LMT(randdataset);
		algo.Relief_RandomTree(randdataset);
	}
}
```

Code ReliefFAttributeEval.java

```
	import java.util.Enumeration;
	import java.util.Random;
	import java.util.Vector;
	import weka.attributeSelection.ASEvaluation;
	import weka.attributeSelection.AttributeEvaluator;

	import weka.core.Instance;
	import weka.core.Instances;
	import weka.core.Option;
	import weka.core.OptionHandler;

	import weka.core.TechnicalInformationHandler;
	import weka.core.Utils;
publicabstractclassReliefFAttributeEvalextends ASEvaluation implements
AttributeEvaluator, OptionHandler, TechnicalInformationHandler {
```

```
/** The training instances */
private Instances m_trainInstances;

/** The class index */
privateintm_classIndex;

/** The number of attributes */
privateintm_numAttribs;

/** The number of instances */
privateintm_numInstances;

/** Numeric class */
privatebooleanm_numericClass;

/** The number of classes if class is nominal */
privateintm_numClasses;

/**
  * Used to hold the probability of a different class val given nearest
  * instances (numeric class)
  */
privatedoublem_ndc;

/**
  * Used to hold the prob of different value of an attribute given
  * nearest instances (numeric class case)
  */
privatedouble[] m_nda;

/**
  * Used to hold the prob of a different class val and different att
  * val given nearest instances (numeric class case)
  */
privatedouble[] m_ndcda;

/** Holds the weights that relief assigns to attributes */
privatedouble[] m_weights;

/** Prior class probabilities (discrete class case) */
privatedouble[] m_classProbs;

/**
  * The number of instances to sample when estimating attributes
  * default == -1, use all instances
  */
privateintm_sampleM;

/** The number of nearest hits/misses */
```

```
privateintm_Knn;

/** k nearest scores + instance indexes for n classes */
privatedouble[][][] m_karray;

/** Upper bound for numeric attributes */
privatedouble[] m_maxArray;

/** Lower bound for numeric attributes */
privatedouble[] m_minArray;

/** Keep track of the farthest instance for each class */
privatedouble[] m_worst;

/** Index in the m_karray of the farthest instance for each class */
privateint[] m_index;

/** Number of nearest neighbours stored of each class */
privateint[] m_stored;

/** Random number seed used for sampling instances */
privateintm_seed;

privatedouble[] m_weightsByRank;
privateintm_sigma;

/** Weight by distance rather than equal weights */
privatebooleanm_weightByDistance;



public ReliefFAttributEval () {
resetOptions();
  }

public Enumeration<Option> listOptions () {
   Vector<Option> newVector = new Vector<Option>(4);
newVector
     .addElement(new Option("\tSpecify the number of instances to\n"
                                + "\tsample when estimating attributes.\n"
                                + "\tIf not specified, then all instances\n"
                                + "\twill be used.", "M", 1
                                , "-M <num instances>"));
newVector.
addElement(new Option("\tSeed for randomly sampling instances.\n"
                               + "\t(Default = 1)", "D", 1
                               , "-D <seed>"));
newVector.
addElement(new Option("\tNumber of nearest neighbours (k) used\n"
                               + "\tto estimate attribute relevances\n"
```

```
                                + "\t(Default = 10).", "K", 1
                                , "-K <number of neighbours>"));
newVector.
addElement(new Option("\tWeight nearest neighbours by distance\n", "W"
                                , 0, "-W"));
newVector.
addElement(new Option("\tSpecify sigma value (used in an exp\n"
                                + "\tfunction to control how quickly\n"
                                + "\tweights for more distant instances\n"
                                + "\tdecrease. Use in conjunction with -W.\n"
                                + "\tSensible value=1/5 to 1/10 of the\n"
                                + "\tnumber of nearest neighbours.\n"
                                + "\t(Default = 2)", "A", 1, "-A <num>"));
return  newVector.elements();
  }



publicvoid setOptions (String[] options)
throws Exception
  {
    String optionString;
resetOptions();
setWeightByDistance(Utils.getFlag('W', options));
optionString = Utils.getOption('M', options);

if (optionString.length() != 0) {
setSampleSize(Integer.parseInt(optionString));
    }

optionString = Utils.getOption('D', options);

if (optionString.length() != 0) {
setSeed(Integer.parseInt(optionString));
    }

optionString = Utils.getOption('K', options);

if (optionString.length() != 0) {
setNumNeighbours(Integer.parseInt(optionString));
    }

optionString = Utils.getOption('A', options);

if (optionString.length() != 0) {
setWeightByDistance(true); // turn on weighting by distance
setSigma(Integer.parseInt(optionString));
    }
  }
```

```
publicvoid setSigma (int s)
throws Exception
  {
if (s <= 0) {
thrownew Exception("value of sigma must be > 0!");
   }

m_sigma = s;
  }

publicint getSigma () {
returnm_sigma;
  }
publicvoid setNumNeighbours (int n) {
m_Knn = n;
  }

publicint getNumNeighbours () {
returnm_Knn;
  }


publicvoid setSeed (int s) {
m_seed = s;
  }

publicint getSeed () {
returnm_seed;
  }


publicvoid setSampleSize (int s) {
m_sampleM = s;
  }


publicint getSampleSize () {
returnm_sampleM;
  }


publicvoid setWeightByDistance (boolean b) {
m_weightByDistance = b;
  }

publicboolean getWeightByDistance () {
returnm_weightByDistance;
  }
```

```
public String[] getOptions () {
String[] options = new String[9];
int current = 0;

if (getWeightByDistance()) {
options[current++] = "-W";
   }

options[current++] = "-M";
options[current++] = "" + getSampleSize();
options[current++] = "-D";
options[current++] = "" + getSeed();
options[current++] = "-K";
options[current++] = "" + getNumNeighbours();
options[current++] = "-A";
options[current++] = "" + getSigma();

while (current < options.length) {
options[current++] = "";
   }

return  options;
  }



public String toString () {
   StringBuffer text = newStringBuffer();

if (m_trainInstances == null) {
text.append("ReliefF feature evaluator has not been built yet\n");
   }
else {
text.append("\tReliefF Ranking Filter");
text.append("\n\tInstances sampled: ");

if (m_sampleM == -1) {
text.append("all\n");
     }
else {
text.append(m_sampleM + "\n");
     }

text.append("\tNumber of nearest neighbours (k): " + m_Knn + "\n");

if (m_weightByDistance) {
text.append("\tExponentially decreasing (with distance) "
                       + "influence for\n"
                       + "\tnearest neighbours. Sigma: "
                       + m_sigma + "\n");
```

```
    }
else {
text.append("\tEqual influence nearest neighbours\n");
    }
   }

return  text.toString();
 }



publicvoid buildEvaluator (Instances data)
throws Exception
 {
int z, totalInstances;
   Random r = newRandom(m_seed);

if (data.checkForStringAttributes()) {
thrownew Exception("Can't handle string attributes!");
   }

m_trainInstances = data;
m_classIndex = m_trainInstances.classIndex();
m_numAttribs = m_trainInstances.numAttributes();
m_numInstances = m_trainInstances.numInstances();

if (m_trainInstances.attribute(m_classIndex).isNumeric()) {
m_numericClass = true;
   }
else {
m_numericClass = false;
   }

if (!m_numericClass) {
m_numClasses = m_trainInstances.attribute(m_classIndex).numValues();
   }
else {
m_ndc = 0;
m_numClasses = 1;
m_nda = newdouble[m_numAttribs];
m_ndcda = newdouble[m_numAttribs];
   }

if (m_weightByDistance) // set up the rank based weights
    {
        m_weightsByRank = newdouble[m_Knn];

        for (int i = 0; i <m_Knn; i++) {
        m_weightsByRank[i] =
        Math.exp(-((i/(double)m_sigma)*(i/(double)m_sigma)));
```

```
        }
    }

// the final attribute weights
m_weights = newdouble[m_numAttribs];
// num classes (1 for numeric class) knn neighbours,
// and 0 = distance, 1 = instance index
m_karray = newdouble[m_numClasses][m_Knn][2];

if (!m_numericClass) {
m_classProbs = newdouble[m_numClasses];

for (int i = 0; i <m_numInstances; i++) {
m_classProbs[(int)m_trainInstances.instance(i).value(m_classIndex)]++;
    }

for (int i = 0; i <m_numClasses; i++) {
m_classProbs[i] /= m_numInstances;
    }
   }

m_worst = newdouble[m_numClasses];
m_index = newint[m_numClasses];
m_stored = newint[m_numClasses];
m_minArray = newdouble[m_numAttribs];
m_maxArray = newdouble[m_numAttribs];

for (int i = 0; i <m_numAttribs; i++) {
m_minArray[i] = m_maxArray[i] = Double.NaN;
   }

for (int i = 0; i <m_numInstances; i++) {
updateMinMax(m_trainInstances.instance(i));
   }

if ((m_sampleM>m_numInstances) || (m_sampleM< 0)) {
totalInstances = m_numInstances;
   }
else {
totalInstances = m_sampleM;
   }

// process each instance, updating attribute weights
for (int i = 0; i < totalInstances; i++) {
if (totalInstances == m_numInstances) {
      z = i;
    }
else {
      z = r.nextInt()%m_numInstances;
    }
```

```
if (z < 0) {
z *= -1;
      }

if (!(m_trainInstances.instance(z).isMissing(m_classIndex))) {
// first clear the knn and worst index stuff for the classes
for (int j = 0; j <m_numClasses; j++) {
m_index[j] = m_stored[j] = 0;

for (int k = 0; k <m_Knn; k++) {
m_karray[j][k][0] = m_karray[j][k][1] = 0;
         }
       }

findKHitMiss(z);

if (m_numericClass) {
updateWeightsNumericClass(z);
       }
else {
updateWeightsDiscreteClass(z);
       }
     }
   }

for (int i = 0; i <m_numAttribs; i++) {if (i != m_classIndex) {
if (m_numericClass) {
m_weights[i] = m_ndcda[i]/m_ndc -
            ((m_nda[i] - m_ndcda[i])/((double)totalInstances - m_ndc));
     }
else {
m_weights[i] *= (1.0/(double)totalInstances);
     }

   }
   }
  }

publicdouble evaluateAttribute (int attribute)
throws Exception
  {
returnm_weights[attribute];
  }


/**
   * Reset options to their default values
   */
protectedvoid resetOptions () {
```

```
m_trainInstances = null;
m_sampleM = -1;
m_Knn = 10;
m_sigma = 2;
m_weightByDistance = false;
m_seed = 1;
  }



privatedouble norm (double x, int i) {
if (Double.isNaN(m_minArray[i]) ||
        Utils.eq(m_maxArray[i], m_minArray[i])) {
return  0;
   }
else {
return  (x - m_minArray[i])/(m_maxArray[i] - m_minArray[i]);
   }
  }


/**
   * Updates the minimum and maximum values for all the attributes
   * based on a new instance.
   *
   * @param instance the new instance
   */
privatevoid updateMinMax (Instance instance) {
for (int j = 0; j <m_numAttribs; j++) {
if ((m_trainInstances.attribute(j).isNumeric()) &&
        (!instance.isMissing(j))) {
        if (Double.isNaN(m_minArray[j])) {
        m_minArray[j] = instance.value(j);
        m_maxArray[j] = instance.value(j);
        }
        else {
        if (instance.value(j) <m_minArray[j]) {
        m_minArray[j] = instance.value(j);
          }
        else {
        if (instance.value(j) >m_maxArray[j]) {
        m_maxArray[j] = instance.value(j);
           }
         }
        }
    }
   }
  }
```

```
privatedouble attributeDiff (int attrib, int first, int second) {
double temp, d;

// Nominal attribute
if (m_trainInstances.attribute(attrib).isNominal()) {
if (m_trainInstances.instance(first).isMissing(attrib) ||
        m_trainInstances.instance(second).isMissing(attrib)) {
        temp = (1.0 - (1.0/((double)m_trainInstances.
                        attribute(attrib).numValues())));
    }
else {
        if (m_trainInstances.instance(first).value(attrib) !=
        m_trainInstances.instance(second).value(attrib)) {
        temp = 1.0;
        }
        else {
        temp = 0.0;
        }
    }
   }
else
// Numeric attribute
    {
        if (m_trainInstances.instance(first).isMissing(attrib) &&
        m_trainInstances.instance(second).isMissing(attrib)) {
        temp = 1.0; // maximally different
        }
        else {if (m_trainInstances.instance(first).isMissing(attrib)) {
          d = norm(m_trainInstances.instance(second).value(attrib), attrib);

        if (d < 0.5) {
           d = 1.0 - d;
          }

        temp = d;
        }
        else {if (m_trainInstances.instance(second).isMissing(attrib)) {
          d = norm(m_trainInstances.instance(first).value(attrib), attrib);

        if (d < 0.5) {
           d = 1.0 - d;
          }

        temp = d;
        }
        else {
          d = norm(m_trainInstances.instance(first).value(attrib), attrib) -
        norm(m_trainInstances.instance(second).value(attrib), attrib);

        if (d < 0.0) {
```

```
            d *= -1.0;
              }

            temp = d;
            }
            }
            }
        }

return  temp;
  }

privatedouble diff (int first, int second) {
int i, j;
double temp = 0;

for (i = 0; i <m_numAttribs; i++) {
if (i != m_classIndex) {
            temp += attributeDiff(i, first, second);
        }
    }

return  temp;
  }

privatevoid updateWeightsNumericClass (int instNum) {
int i, j;
double temp;
int[] tempSorted = null;
double[] tempDist = null;
double distNorm = 1.0;

// sort nearest neighbours and set up normalization variable
if (m_weightByDistance) {
tempDist = newdouble[m_stored[0]];

for (j = 0, distNorm = 0; j <m_stored[0]; j++) {
            // copy the distances
            tempDist[j] = m_karray[0][j][0];
            // sum normalizer
            distNorm += m_weightsByRank[j];
        }

tempSorted = Utils.sort(tempDist);
    }

for (i = 0; i <m_stored[0]; i++) {
// P diff prediction (class) given nearest instances
if (m_weightByDistance) {
            temp = attributeDiff(m_classIndex, instNum,
```

```
                                (int)m_karray[0][tempSorted[i]][1]);
            temp *= (m_weightsByRank[i]/distNorm);
        }
else {
            temp = attributeDiff(m_classIndex, instNum, (int)m_karray[0][i][1]);
            temp *= (1.0/(double)m_stored[0]); // equal influence
        }

m_ndc += temp;

// now the attributes
for (j = 0; j <m_numAttribs; j++) {
            if (j != m_classIndex) {
            // P of different attribute val given nearest instances
            if (m_weightByDistance) {
            temp = attributeDiff(j, instNum,
                                    (int)m_karray[0][tempSorted[i]][1]);
            temp *= (m_weightsByRank[i]/distNorm);
              }
            else {
            temp = attributeDiff(j, instNum, (int)m_karray[0][i][1]);
            temp *= (1.0/(double)m_stored[0]); // equal influence
              }

            m_nda[j] += temp;

            // P of different prediction and different att value given
            // nearest instances
            if (m_weightByDistance) {
            temp = attributeDiff(m_classIndex, instNum,
                                    (int)m_karray[0][tempSorted[i]][1]) *
            attributeDiff(j, instNum, (int)m_karray[0][tempSorted[i]][1]);
            temp *= (m_weightsByRank[i]/distNorm);
              }
            else {
            temp = attributeDiff(m_classIndex, instNum,
                                    (int)m_karray[0][i][1]) *
            attributeDiff(j, instNum, (int)m_karray[0][i][1]);
            temp *= (1.0/(double)m_stored[0]); // equal influence
              }

            m_ndcda[j] += temp;
            }
        }
      }
    }

privatevoid updateWeightsDiscreteClass (int instNum) {
int i, j, k;
int cl;
```

```
doublecc = m_numInstances;
double temp, temp_diff, w_norm = 1.0;
double[] tempDistClass;
int[] tempSortedClass = null;
double distNormClass = 1.0;
double[] tempDistAtt;
int[][] tempSortedAtt = null;
double[] distNormAtt = null;
// get the class of this instance
cl = (int)m_trainInstances.instance(instNum).value(m_classIndex);

// sort nearest neighbours and set up normalization variables
if (m_weightByDistance) {
// do class (hits) first
// sort the distances
tempDistClass = newdouble[m_stored[cl]];

for (j = 0, distNormClass = 0; j <m_stored[cl]; j++) {
        // copy the distances
        tempDistClass[j] = m_karray[cl][j][0];
        // sum normalizer
        distNormClass += m_weightsByRank[j];
    }

tempSortedClass = Utils.sort(tempDistClass);
// do misses (other classes)
tempSortedAtt = newint[m_numClasses][1];
distNormAtt = newdouble[m_numClasses];

for (k = 0; k <m_numClasses; k++) {
        if (k != cl) // already done cl
          {
        // sort the distances
        tempDistAtt = newdouble[m_stored[k]];

        for (j = 0, distNormAtt[k] = 0; j <m_stored[k]; j++) {
        // copy the distances
        tempDistAtt[j] = m_karray[k][j][0];
        // sum normalizer
        distNormAtt[k] += m_weightsByRank[j];
           }

        tempSortedAtt[k] = Utils.sort(tempDistAtt);
          }
    }
  }

if (m_numClasses> 2) {
    w_norm = (1.0 - m_classProbs[cl]);
  }
```

```
for (i = 0; i <m_numAttribs; i++) {
if (i != m_classIndex) {
        // first do k nearest hits
        for (j = 0, temp_diff = 0.0; j <m_stored[cl]; j++) {
        if (m_weightByDistance) {
           temp_diff +=
        attributeDiff(i, instNum,
                        (int)m_karray[cl][tempSortedClass[j]][1])*
          (m_weightsByRank[j]/distNormClass);
         }
        else {
           temp_diff += attributeDiff(i, instNum, (int)m_karray[cl][j][1]);
         }
        }

        // average
        if ((!m_weightByDistance) && (m_stored[cl] > 0)) {
         temp_diff /= (double)m_stored[cl];
        }

        m_weights[i] -= temp_diff;
        // now do k nearest misses from each of the other classes
        temp_diff = 0.0;

        for (k = 0; k <m_numClasses; k++) {if (k != cl) // already done cl
         {
        for (j = 0, temp = 0.0; j <m_stored[k]; j++) {
        if (m_weightByDistance) {
                temp +=
                attributeDiff(i, instNum,
                                (int)m_karray[k][tempSortedAtt[k][j]][1])*
                 (m_weightsByRank[j]/distNormAtt[k]);
            }
        else {
                temp += attributeDiff(i, instNum, (int)m_karray[k][j][1]);
            }
           }

        if ((!m_weightByDistance) && (m_stored[k] > 0)) {
        temp /= (double)m_stored[k];
           }

        // now add temp to temp_diff weighted by the prob of this
        // class
        if (m_numClasses> 2) {
            temp_diff += (m_classProbs[k]/w_norm)*temp;
           }
        else {
            temp_diff += temp;
```

```
            }
          }
        }

        m_weights[i] += temp_diff;
    }
   }
  }


/**
  * Find the K nearest instances to supplied instance if the class is numeric,
  * or the K nearest Hits (same class) and Misses (K from each of the other
  * classes) if the class is discrete.
  *
  * @param instNum the index of the instance to find nearest neighbours of
  */
privatevoid findKHitMiss (int instNum) {
int i, j;
int cl;
double ww;
double temp_diff = 0.0;

for (i = 0; i <m_numInstances; i++) {if (i != instNum) {
    temp_diff = diff(i, instNum);

// class of this training instance or 0 if numeric
if (m_numericClass) {
        cl = 0;
    }
else {
        cl = (int)m_trainInstances.instance(i).value(m_classIndex);
    }

// add this diff to the list for the class of this instance
if (m_stored[cl] <m_Knn) {
        m_karray[cl][m_stored[cl]][0] = temp_diff;
        m_karray[cl][m_stored[cl]][1] = i;
        m_stored[cl]++;

        // note the worst diff for this class
        for (j = 0, ww = -1.0; j <m_stored[cl]; j++) {
        if (m_karray[cl][j][0] > ww) {
        ww = m_karray[cl][j][0];
        m_index[cl] = j;
          }
        }

        m_worst[cl] = ww;
    }
```

```
else
        /* if we already have stored knn for this class then check to
        see if this instance is better than the worst */
        {
        if (temp_diff <m_karray[cl][m_index[cl]][0]) {
        m_karray[cl][m_index[cl]][0] = temp_diff;
        m_karray[cl][m_index[cl]][1] = i;

        for (j = 0, ww = -1.0; j <m_stored[cl]; j++) {
        if (m_karray[cl][j][0] > ww) {
                ww = m_karray[cl][j][0];
                m_index[cl] = j;
            }
           }
        m_worst[cl] = ww;
          }
        }
   }
   }
 }

}
```