Data Classification and Its Application in Credit Card Approval

by

Thai Vinh Tuan (2441)

submitted in partial fulfilment of
the requirements for the
Bachelor of Technology (Hons)
(Information Technology)

JULY 2004

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Data Classification and Its Application in Credit Card Approval

by

Thai Vinh Tuan

Dissertation submitted in partial fulfilment of
the requirements for the
Bachelor of Technology (Hons)
(Information Technology)

Approved by,

[illegible]

(Mr. Helmi Md Rais)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

December 04

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

THAI VINH TUAN

ABSTRACT

We are all now living in the information age. The amount of data being collected by businesses, companies and agencies is large. Recent advances in technologies to automate and improve data collection have increased the volumes of data. Lying hidden in all this data is potentially useful information that is rarely made explicit or taken advantage of. In this context, data mining has arisen as an important research area that helps to reveal the hidden useful information from the raw data collected. Many intensive researches have been conducted to enhance the capability of data mining solution in providing the intelligence so that different types of businesses can make informed decisions.

This project demonstrates how data mining can address the need of business intelligence in the process of decision-making. An analysis on the field of data mining is done to show how data mining, especially data classification, can help in businesses such as targeted marketing, credit card approval, fraud detection, medical diagnosis, and scientific work. This project is involved with identification of the available algorithms used in data classification and the implementation of C4.5 decision tree induction algorithm in solving the data classifying task. Sample credit card approval dataset is used to demonstrate the functionality of a data mining solution prototype, which includes the typical tasks of a decision tree induction process: data selection, data preprocessing, decision tree induction, tree pruning, rules generation and validation.

The result of this application using the sample credit card approval dataset includes a decision tree, a set of rules derived from the decision tree and its accuracy. These outputs help to identify the pattern of applicants who are more likely to be accepted or rejected. The set of rules can be used as part of the knowledge base in expert system or decision support system for financial institutions.

ACKNOWLEDGEMENT

With the help and support from many people while working on this project, I would like to relish this opportunity to express my gratitude to all who has in one way or another helped me to complete this work.

First of all, I would like to thank Mr. Helmi Md. Rais, the project supervisor who has been given many supportive and constructive advices. His suggestions have guided me throughout this project.

I also gratefully acknowledge all my friends who have given invaluable review and feebacks on the project. Special thanks go to Cong Phuc for his time discussing about machine learning as well as Java debugging.

Last, and most of all, I am thankful for the endless encouragement and care from my Dad, Mum and my lovely sister. They have given me the strength, motivation and determination in all my undertakings.

TABLE OF CONTENTS

CERTIFICATION i

ABSTRACT ii

ACKNOWLEDGEMENT iii

CHAPTER 1: INTRODUCTION 1

1.1 Background of Study 1

1.2 Problem Statement 3

1.2.1 Problem Identification. . . 3

1.2.2 Significance of project. . . 3

1.3 Objectives and Scope of Study . . 4

CHAPTER 2: THEORY / LITERATURE REVIEW . . 5

2.1 Data, Information and Knowledge. . . 5

2.2 Data Mining 5

2.3 Data Mining Process 8

2.4 Data Mining Components. . . . 9

2.4.1 Association Analysis . . . 9

2.4.2 Classification/ Clustering . . 10

2.4.3 Sequence Analysis . . . 10

2.5 Data Classification Algorithms. . . 10

2.5.1 Decision Tree Induction . . 11

2.5.2 Naïve-Bayes classifier . . 12

2.5.3 Neural Network . . . 13

2.5.4 k-Nearest Neighbor Classifiers. . 13

2.5.5 Fuzzy set approach . . . 13

2.6 Classification Performance Evaluation. . 13

2.7 Data mining problems/issues. . . 14

2.7.1 Limited Information . . . 14

2.7.2 Noise and missing values . . 14

2.7.3 Uncertainty 15

2.7.4 Size, updates, and irrelevant fields. . 15

2.8 ID3 and C4.5 Algorithms . . . 16

2.8.1 ID3 algorithm. . . . 16

2.8.2 C4.5 algorithm . . . 19

2.9 Decision Tree Pruning. . . . 22

2.9.1 Pre-pruning approach . . . 22

2.9.2 Post-pruning approach. . . 22

2.10 Decision Tree Validation. . . . 24

2.10.1 Holdout method . . . 25

2.10.2 k-fold cross validation method. . 26

CHAPTER 3:	**METHODOLOGY**		28
	3.1	Procedure Identification	28
		3.1.1 Requirements analysis and definition	29
		3.1.2 System and software design	29
		3.1.3 Implementation and unit testing.	30
		3.1.4 Integration and system testing.	30
		3.1.5 Operation and maintenance	31
	3.2	Tools Required	31
CHAPTER 4:	**RESULTS AND DISCUSSION**		32
	4.1	Software Architectural Design	32
		4.1.1 Data Flow Diagram	32
		4.1.2 Class Diagram.	38
	4.2	Implementation	40
		4.2.1 Credit Card Approval Dataset.	40
		4.2.2 Results.	41
		4.2.3 Performance Analysis	50
	4.2	Social, Ethical Issues of Data Mining.	53
CHAPTER 5:	**CONCLUSION AND RECOMMENDATION**		55
	5.1	Conclusion	55
	5.2	Recommendations	56
REFERENCES			58
APPENDICES			61
	A1	Flat text file format	61
	A1	DTree class structure	62
	A1	Program Code	63

LIST OF FIGURES

Figure 2.1 Data Mining (KDD) Process 8
Figure 2.2 ID3 algorithm 16
Figure 2.3 C4.5 algorithm 22
Figure 2.4 Estimating classifier accuracy with Holdout method 25
Figure 2.5 Estimating classifier accuracy with k-fold cross validation method 25
Figure 3.1 Waterfall model 28
Figure 4.1 Data Flow Diagram 32
Figure 4.2 Class Diagram 38
Figure 4.3 Credit card dataset 42
Figure 4.4 Credit card dataset preprocessed 43
Figure 4.5 Decision Tree with Graphical Display 44
Figure 4.6 Decision Tree with Hierarchical Display 45
Figure 4.7 Decision Rules 46
Figure 4.8 Pruned Decision Tree 47
Figure 4.9 Holdout Validation 48
Figure 4.10 10-fold Cross Validation 50

LIST OF TABLES

Table 4.1 Credit Card Approval Dataset Description 41
Table 4.2 Unpruned and Pruned Trees' accuracy comparison 49
Table 4.3 Experimental results 51
Table 4.4 Summarized accuracies 52

CHAPTER 1
INTRODUCTION

1. INTRODUCTION

1.1 Background of study

We are now living in the information age. The amount of data being collected by businesses, companies and agencies is really huge. Recent advances in technology to automate and improve data collection have increased the volumes of data. Ubiquitous electronics record our decisions, our choices in the supermarket, our financial habits, our comings and goings. The World Wide Web overwhelms us with information and every choice we make is recorded. With the substantial decrease of hardware cost, organizations are able to store an enormous amount of raw data. Some of the data are essential than others. However, as the volume of collected data increases, the proportion of people who can understand it decreases. Real-time collection of data, based on thousands of variables, is practically impossible for anyone to analyze today without the aid of information technology. It is likely that hidden patterns or potentially useful information exists and is not taken advantage of from the data collected. It is important to develop powerful tool for analysis and interpretation of such data and for the extraction of interesting knowledge that could help in decision-making.

Data mining is a scientific field that helps extracting knowledge or useful information from a large amount of raw data that the user did not anticipate. Turning data into useful information is very meaningful in many areas including medical diagnosis, scientific research, market analysis, etc. Financial institutions need to classify their customers based on their personal profile to grant loan or approve credit card applications.

Insurance company would need some intelligence to assist in investigating the incidents before making payment to claims. Advertising company would need to know which characteristics of customers suit its campaign to make effective marketing (targeted marketing). Medical field requires assistance from decision support system to give quick diagnosis. Scientific research also needs to have a mechanism to perform effective separation and classification of objects and identify the relationships between them. Those are examples of real-life situations that necessitate the use of data mining to gain beneficial information.

The three main components of data mining are: mining association rules, data classification/prediction, and sequence analysis. There have been intensive researches on improving the effectiveness of each of the above branches of data mining as well as how to apply in real-life applications to assist business operations. In principle, data mining is not specific to one type of media or data. Data mining should be applicable to any kind of information repository. However, algorithms and approaches may differ when applied to different types of data.

Data mining's objective is to discover hidden and potentially useful knowledge, which may be represented as patterns or rules. However, the amount of rules found can sometimes be a burden itself to users seeking useful information. This leads to the problem of evaluating which patterns or rules are interesting and useful among the myriad of rules mined from the data. Different users would perceive the mined patterns differently, which would lead to different values for a particular set of rules/patterns. It is therefore necessary that users specify some predefined threshold in order to evaluate the interestingness and reliability of the knowledge obtained and to select which one to make use of.

1.2. Problem Statement

1.2.1 Problem Identification

In many cases, a classification model is required to find a pattern that maps a data item into one of several predefined classes and can be used to predict the class for some entity based on the values of the available input attributes. Data classification is therefore the typical methodology of data mining that can be applied in many real-world applications as a powerful solution to classification problems. Depending on the nature of the data and their relationships, different methodologies of data classification need to be applied on the original dataset to obtain the pattern underlying in the data. Among different classification methodologies available today, decision tree has been proven to be the most popular classifier due to its advantage of being easy to interpret, fast to build and highly accurate. Therefore, building a decision tree from the raw data to represent the underlying hidden pattern from the data can be a powerful tool to support making informed decision in many businesses.

1.2.2 Significance of the project

This project, once completed, would serve as a generic decision tree induction tool that allows users to perform typical steps of a data mining solution. From the raw data set, users can load the data into the tool and then perform the necessary basic preprocessing (cleaning and transformation) of data and then the algorithm provided within the tool will help classify the data based on the provided attributes. The outcome of the process is an easily interpretable decision tree and its set of decision rules that would help in making decisions, i.e. applying to expert system or decision support system as part of the knowledge base. The project is demonstrated on a sample credit card approval dataset to help making decisions in approving credit card applications. However, the application has a further and wider application as it can be applied to any dataset that conforms to its required format and provided that the memory is large enough that the dataset can reside in it each run.

1.3. Objective and scope of the study

This project aims at an overview of available data classification algorithms and an implementation of decision tree learning algorithm called C4.5 developed by Quinlan. The prototypical application of this project is able to perform data selection, basic data preprocessing, decision tree induction, tree pruning and validation. The application will assist users to extract hidden knowledge (patterns that differentiate the classes that a sample object belongs to) from raw data collected.

A sample credit card approval dataset is used to demonstrate the capability of the decision tree classifier to support making informed decisions on credit card applications. The output of the application on this dataset is the decision tree together with its set of rules and accuracy. Depending on the parameters chosen, acceptably accurate decision rules can be used as part of the knowledge base for an expert system.

CHAPTER 2
THEORY/LITERATURE REVIEW

2. LITERATURE REVIEW

2.1 Data, Information, and Knowledge

Palace (1996) differentiated data, information and knowledge as follows:

- ***Data***

Data are any facts, numbers, or text that can be processed by a computer. Today, organizations are accumulating vast and growing amounts of data in different formats and different databases. This includes:

- operational or transactional data such as, sales, cost, inventory, payroll, and accounting
- non-operational data, such as industry sales, forecast data, and macro economic data
- meta data - data about the data itself, such as logical database design or data dictionary definitions

- ***Information***

The patterns, associations, or relationships among all this data can provide information.

- ***Knowledge***

Information can be converted into knowledge about historical patterns and future trends.

2.2 Data mining

Data mining, or **K**nowledge **D**iscovery in **D**atabases (**KDD**), is defined as *"the automated or convenient extraction of patterns representing knowledge implicitly stored in large databases, data warehouses, and other massive information repositories."* (Han, 2001). Witten and Frank (1999) refer to data mining as *"the* *process of discovering patterns in data. The process must be automatic or semi-automatic. The pattern discovered must be meaningful in that they lead to some advantage, usually economic advantage."* Another definition states that data mining involves extracting hidden predictive information from databases to solve business problems (Brandel, 2001).

The benefit brought by data mining has led to an enormous amount of researches in order to solve different issues related to this field. Data mining research has drawn on a number of other fields such as inductive learning, machine learning and statistics etc. An overview of these fields is given by Palace (1996):

- ***Inductive learning***

Induction is the inference of information from data and inductive learning is the model building process where the environment i.e. database is analyzed with a view to finding patterns. Similar objects are grouped in classes and rules formulated whereby it is possible to predict the class of unseen objects. This process of classification identifies classes such that each class has a unique pattern of values which forms the class description. The nature of the environment is dynamic hence the model must be adaptive i.e. should be able to learn.

Inductive learning where the system infers knowledge itself from observing its environment has two main strategies:

- <u>Supervised learning</u> - this is learning from examples where classes are defined and examples of each class are supplied. The system has to find a description of each class i.e. the common properties in the examples. Once the description has been formulated, the description and the class form a classification rule which can be used to predict the class of unseen objects
- <u>Unsupervised learning</u> - this is learning from observation and discovery. The data mining system is supplied with objects but no classes are defined so it has to observe the examples and recognize patterns (i.e. class description) by itself.

This system results in a set of class descriptions, one for each class discovered in the environment.

Induction is therefore the extraction of patterns. The quality of the model produced by inductive learning methods is such that the model could be used to predict the outcome of future situations. The problem is that most environments have different states, i.e. changes within, and it is not always possible to verify a model by checking it for all possible situations.

Given a set of examples the system can construct multiple models, some of which will be simpler than others. The simpler models are more likely to be correct if we adhere to Ockham's razor, which states that if there are multiple explanations for a particular phenomenon it makes sense to choose the simplest because it is more likely to capture the nature of the phenomenon.

- ***Machine Learning***

Machine learning is the automation of a learning process and learning is tantamount to the construction of rules based on observations of environmental states and transitions. This is a broad field which includes not only learning from examples, but also reinforcement learning, learning with teacher, etc. A learning algorithm takes the dataset and its accompanying information as input and returns a statement e.g. a concept representing the results of learning as output. Machine learning examines previous examples and their outcomes and learns how to reproduce these and make generalizations about new cases.

- ***Statistics***

Statistics has a solid theoretical foundation but the results from statistics can be overwhelming and difficult to interpret as they require user guidance as to where and how to analyze the data. Data mining however allows the expert's knowledge of the data and the advanced analysis techniques of the computer to work together.

Statistical analysis systems such as SAS and SPSS have been used by analysts to detect unusual patterns and explain patterns using statistical models such as linear models. Statistics have a role to play and data mining will not replace such analyses but rather they can act upon more directed analyses based on the results of data mining.

2.3 Data mining process

Data mining is a process consisting of iterative sequence of steps proposed by Han (2001) as illustrated in Figure 2.1:

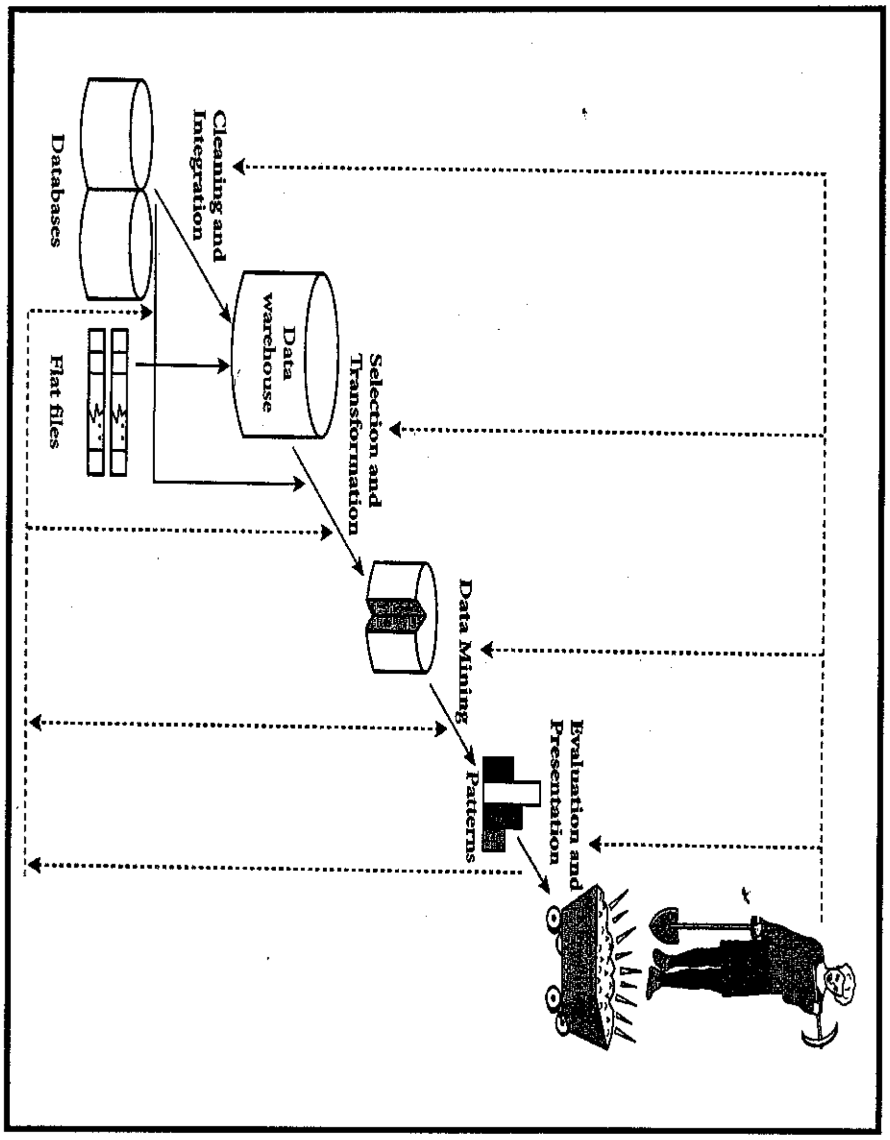


Figure 2.1: Data Mining (KDD) Process

- ***Data cleaning***: to remove noise and inconsistent data
- ***Data integration***: where multiple data sources may be combined
- ***Data selection***: where data relevant to the analysis task are retrieved from the database
- ***Data transformation***: where data are transformed or consolidated into forms appropriate for mining by performing summary or aggregation operations
- ***Data mining***: an essential process where intelligent methods are applied in order to extract data patterns
- ***Pattern evaluation***: to identify the truly interesting patterns representing knowledge based on some interestingness measures
- ***Knowledge presentation***: where visualization and knowledge representation techniques are used to present the mined knowledge to the user

It is common to combine some of these steps together. For instance, data cleaning and data integration can be performed together as a pre-processing phase to generate a data warehouse. Data selection and data transformation can also be combined where the consolidation of the data is the result of the selection, or, as for the case of data warehouses, the selection is done on transformed data

The KDD is an iterative process. Once the discovered knowledge is presented to the user, the evaluation measures can be enhanced, the mining can be further refined, new data can be selected or further transformed, or new data sources can be integrated, in order to get different, more appropriate results.

2.4 Data mining components

Data mining has three major components: Association analysis, Classification/ Clustering, and Sequence Analysis (Joshi, 1997)

2.4.1 Association analysis

Association analysis is the discovery of certain association relationship among a set of objects. It has attracted attention in research as it may disclose some useful patterns for decision support (Joshi, 1997). The discovery of interesting association relationships among huge amounts of business transaction records can help in many business decision making processes, such as catalog design, cross-marketing, and loss-leader analysis. A typical type of association rule mining is market basket analysis. This process analyzes customer buying habits by finding associations between the different items that customers place in their "shopping basket". The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. However, association rules are not to be used directly for prediction without further analysis or domain knowledge. They do not necessarily indicate causation, but only help to understand the data (Han, 2001).

2.4.2 Classification

Classification/ Clustering is the process of finding a set of models (or functions) that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown (Han, 2001). Classification is also defined by Tu *et al* (1992) as a general problem-solving approach that takes a large collection of examples from multiple groups as inputs and identifies the characteristic pattern or property for each group.

2.4.3 Sequence analysis

Sequence analysis is the discovery of patterns that occur in sequence or time. Since many business transactions, telecommunications records, weather data, and production processes are time sequence data, sequential pattern mining is useful in the analysis of such data for targeted marketing, customer retention, weather prediction, etc.

2.5 Data classification methods

The widespread application of data classification has led to intensive research in this particular branch of data mining. Many classification methods have been proposed by researchers in machine learning, expert systems, statistics and neurobiology:

2.5.1 Decision Tree Induction

One common approach to classification is to use decision trees. The decision tree itself is a set of decision rules which describe each group's patterns learned from these given examples. Decision-tree classification method has emerged as the essential knowledge acquisition procedure which follows one machine learning strategy, learning from examples (Michalski, 1983). The most important feature of decision trees is their capability to break down a complex decision-making process into a collection of simpler decisions and therefore provides an easily interpretable solution. The advantages of learning a decision tree are:

- a program, rather than a knowledge engineer, elicits knowledge from expert.
- inexpensive to construct
- easy to interpret
- easy to integrate with database systems
- comparable or better accuracy in many applications

A few well-known decision-tree induction methods are outlined by Han (2001):

- ***ID3 (Interactive Dichotomizer 3):***

ID3 is a popular and efficient method proposed by J. Ross Quinlan in 1979, which makes a decision tree for classification from symbolic data. J. Ross Quinlan originally developed ID3 at the University of Sydney. He first presented ID3 in 1975 in a book, *Machine Learning*, vol. 1, no. 1. ID3 is based off the Concept Learning System (CLS) algorithm. ID3 is a greedy algorithm that selects the next attribute to test based on the information gain associated with this attribute.

- *C4.5:*

C4.5 is the successor algorithm to ID3 with several enhancements. It allows for classification with numeric attributes, using gain ratio to reduce the bias for attributes with many values, and handle missing values. The description for ID3 and C4.5, the algorithms being used in this project, is detailed in Section 2.8.

- ***SLIQ (Supervised Learning In Quest) and SPRINT (Scalable PaRallelizable INduction of decision Trees):***

They were developed by IBM's Quest project team at IBM Almaden Research Center to address the scalability issue of the above algorithms by proposing presorting techniques on disk-resident data sets that are too large to fit in memory. New data structures defined by SLIQ are disk-resident *attribute lists* and a single memory-resident *class-list*. A sample record (tuple) is represented by a link between one entry in each attribute list to an entry in the class list. During classification process, only the class list is resident in memory. However, as its size grows proportionally with the size of the samples set, SLIQ has its own limitation when dealing with very large dataset. SPRINT overcomes the size limitation of SLIQ by using different data structure (the attribute list) and distributing the processing tasks evenly to parallel processors. Therefore, SPRINT removes all the memory restrictions.

- ***Naïve-Bayes classifier***

Naïve Bayes classifier is a simple induction statistical algorithm that assumes a conditional independence model of attributes given the label to simplify the computations involved (Han, 2001). In other words, Naïve Bayes classifiers assume that the effect of a variable value on a given class is independent of the values of other variables. This assumption is called class conditional independence. It is made to simplify the computation and in this sense considered to be "Naïve". This assumption is a fairly strong assumption and is often not applicable. However, bias in estimating probabilities often may not make a difference in practice. It is the order of the probabilities, not their exact values, which determine the classifications. Studies comparing classification algorithms have found the Naïve Bayesian classifier to be comparable in performance with classification trees and with neural network classifiers. They have also exhibited high accuracy and speed when applied to large databases.

- *Neural Network*

When used for data classification, neural network is typically a collection of neuron-like processing units with weighted connections between the units. During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input samples. Neural network learning is also referred to as *connectionist learning* due to the connections between units (Han, 2001).

According to Stergiou (1996), neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze.

- ***k-Nearest Neighbor Classifiers***

When given an unknown sample, this classifier searches the pattern space for the k training samples that are closest to the unknown sample. This classifier is called instance-based or lazy learners in that they store all the training samples and do not build a classifier until a new (unlabeled) sample needs to be classified.

- ***Fuzzy set approach***

This approach is used for classification of data with numerical attributes. It solves the question of where to set the cut-points in the discretisation process of continuous-valued attributes. This approach handles missing value or inconsistent data well by using the membership function to define the membership degree of each tuple to a class.

2.6 Classification Methods Performance Evaluation

Han (2001) states that the performance of a classification method is evaluated based on the following criteria:

- ***Predictive accuracy***: the ability of the model to correctly predict the class label of new or previously unseen data

- ***Speed***: the computation costs involved in generating and using the model
- ***Robustness***: ability of the model to make correct predictions given noisy data or data with missing values
- ***Scalability***: ability to construct the model efficiently given large amount of data
- ***Interpretability***: level of understanding and insight provided by the model

2.7 Data mining problems/issues

Dilly (1995) has highlighted a few problems and issues associated with data mining. Data mining systems rely on databases to supply the raw data for input and this raises problems in that databases tend be dynamic, incomplete, noisy, and large. Other problems arise as a result of the adequacy and relevance of the information stored.

2.7.1 Limited Information

A database is often designed for purposes different from data mining and sometimes the properties or attributes that would simplify the learning task are not present nor can they be requested from the real world. Inconclusive data causes problems because if some attributes essential to knowledge about the application domain are not present in the data it may be impossible to discover significant knowledge about a given domain. For example cannot diagnose malaria from a patient database if that database does not contain the patient's red blood cell count.

2.7.2 Noise and missing values

Databases are usually contaminated by errors so it cannot be assumed that the data they contain is entirely correct. Attributes which rely on subjective or measurement judgments can give rise to errors such that some examples may even be misclassified. Errors in either the values of attributes or class information are known as noise. Obviously where possible it is desirable to eliminate noise from the classification information as this affects the overall accuracy of the generated rules.

Missing data can be treated by discovery systems in a number of ways such as:

- infer missing values from known values
- treat missing data as a special value to be included additionally in the attribute domain
- average over the missing values using Bayesian techniques
- simply disregard missing values
- omit the corresponding records

Noisy data in the sense of being imprecise is characteristic of all data collection and typically fit a regular statistical distribution such as Gaussian while wrong values are data entry errors. Statistical methods can treat problems of noisy data, and separate different types of noise.

2.7.3 Uncertainty

Uncertainty refers to the severity of the error and the degree of noise in the data. Data precision is an important consideration in a discovery system.

2.7.4 Size, updates, and irrelevant fields

Databases tend to be large and dynamic in that their contents are ever-changing as information is added, modified or removed. The problem with this from the data mining perspective is how to ensure that the rules are up-to-date and consistent with the most current information. Also the learning system has to be time-sensitive as some data values vary over time and the discovery system is affected by the `timeliness' of the data.

Another issue is the relevance or irrelevance of the fields in the database to the current focus of discovery, for example post codes are fundamental to any studies trying to establish a geographical connection to an item of interest such as the sales of a product.

2.8 ID3 and C4.5 algorithms

2.8.1 ID3 algorithm

The ID3 algorithm developed by Quinlan is a greedy algorithm that constructs decision tree in a top-down recursive divide-and-conquer manner. The algorithm was summarized by Han (2001) as follows:

```
Algorithm: Generate_decision_tree. Generate a decision tree from the
given training data
Input: The training samples, samples, represented by discrete-valued
attributes; the set of candidate attributes, attribute-list.
Output: A decision tree
Method:
1) create a node N
2) if samples are all of the same class, C then
3)     return N as a leaf node labeled with the class C;
4) if attribute-list is empty then
5)     return N as a leaf node labeled with the most common class in
                                       samples;// majority voting
6) select test-attribute, the attribute among attribute-list with the
                                            highest information gain;
7) label node N with test-attribute;
8) for each known value $a_i$ of test-attribute //partition the samples
9)      grow a branch from node N for the condition test-attribute = $a_i$;
10)     let $s_i$ be the set of samples in samples for which test-
                                       attribute= $a_i$; //a partition
11)     if $s_i$ is empty then
12)         attach a leaf labeled with the most common class in samples;
13)     else attach the node returned by
            Generate_decision_tree($s_i$,attribute-list - test-attribute);
```

Figure 2.2: ID3 algorithm

Basic strategy is as follows:

- The tree starts as a single node representing the training samples, usually called root node (step 1)

- If the samples are all of the same class, then the node becomes a leaf and is labeled with that class (steps 2 and 3). This is only the most obvious stopping criterion that the recursive-partitioning process must be stopped. In addition to this criterion, which can be considered as mandatory, several other stopping criteria can be used. One example is to stop the recursive-partitioning process when the number of data instances is below a small pre-specified threshold, on the grounds that there would be too few data instances to justify the creation of any new tree node. The algorithm would then label the leaf node with the most frequent class occurring in that node (Freitas, 2002).
- Otherwise, the algorithm uses an entropy-based measure known as **information gain** as a heuristic for selecting the attribute that will best separate the samples into individual classes (class 6). This attribute becomes the "test" or "decision" attribute at the node (step 7). In this algorithm, all attributes are categorical, i.e. discreet-valued. Continuous-valued attributes must be discretized.
- A branch is created for each known value of the test attribute, and the samples are partitioned accordingly (steps 8-10). Each attribute can only be used to split the samples once. Therefore, in case of ID3 algorithm used in categorical data, an attribute only appears once in any path from the root node to the leaf node. This method of partitioning is sometimes called subsetting in the literature (Freitas, 2002).
- The algorithm uses the same procedure recursively to form a decision tree for the samples at each partition. Once an attribute has occurred at a node, it need not be considered in any of the node's descendents (step 13).
- The recursive partitioning stops only when any one of the following conditions is true:
    - All samples for a given node belong to the same class (steps 2 and 3) , or
    - There are no remaining attributes on which the samples may be further partitioned (step 4). In this case, **majority voting** is employed (step 5). This involves converting the given node into a leaf and labeling it with the class in majority among *samples*. Alternatively, the class distribution of the node samples may be stored.

- There are no samples for the branch *test-attribute*= a_i (step 11). In this case, a leaf is created with the majority class in *samples* (step 12).

- Attribute Selection Measure

The information gain measure is used to select the test attribute at each node in the tree, which is called measure of the goodness of split. The attribute with the highest information gain (or greatest entropy reduction) is chosen as the test attribute for the current node. This attribute minimizes the information needed to classify the samples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. This information-theoretic approach minimizes the expected number of tests needed to classify an object and guarantees that a simple tree is found. The information gain (measured in bits) is calculated as follows:

Let S be a set of training samples, where the class label of each sample is known. Suppose there are m output classes. S contains s_i samples of class C_i, for i=1,..,m. An arbitrary sample belongs to class C_i with probability s_i/s, where s is the total number of samples in set S.

The expected information needed to classify a given sample is

$$\boxed{I(s_1,s_2,..,s_m) = -\sum_{i=1}^{m} \frac{s_i}{s} \log_2 \frac{s_i}{s}} \qquad (1)$$

An attribute A with values $\{a_1,a_2,..,a_v\}$ can be used to partition S into the subsets $\{S_1,S_2,..S_v\}$, where S_j contains those samples in S that have value a_j of A. Let S_j contains s_{ij} samples of class C_i. The expected information based on this partitioning by A is known as the entropy of A.

$$\boxed{E(A)= \sum_{j=1}^{v} \frac{s_{1j} + \ldots + s_{mj}}{s} I(s_{1j},..,s_{mj})} \qquad (2)$$

Entropy measures the amount of randomness or uncertainty. The goal in classification is to minimize uncertainty, or to minimize the entropy.

The information gain obtained by this partitioning on A is defined by:

$$\boxed{\text{Gain}(S,A) = I\ (s_1,s_2,..,s_m) - E(A)} \tag{3}$$

The attribute with the highest information gain is chosen as the test attribute for the given set S. A node is created and labeled with the attribute, branches are created for each value of the attribute, and the samples are partitioned accordingly.

The ID3 algorithm uses a greedy search. It selects a test using the information gain criterion, and then never explores the possibility of alternate choices. This makes it a very efficient algorithm, in terms of processing time. However, a disadvantage is that the information gain criterion has a strong bias in favor of tests with more outcomes over tests with fewer outcomes. This can produce unnecessarily wide decision-trees that have many leaves containing single instances, which may lead to over-training. The ID3 algorithm also cannot deal with numeric attributes, missing values and noisy data.

2.8.2 C4.5 algorithm

The C4.5 was developed by Quinlan in 1986 as an enhancement to ID3 algorithm so that it can classify data with numeric-valued attributes, and handle missing values. The C4.5 algorithm generates a decision tree for the given data by recursively splitting the node containing the data. The decision tree grows using Depth-first strategy. The C4.5 algorithm considers all the possible tests that can split the data and selects a test that gives the best information gain (i.e. highest gain ratio). This test removes ID3's bias in favor of attributes with a large number of values (v is large in equation (2)). The gain ratio takes number and size of branches into account when choosing an attribute. This means the intrinsic information of a split is taken into account. The intrinsic information, also called split ratio, represents how much information we need to tell which branch an instance belongs to, i.e. the entropy of distribution of instances into branches.

The gain ratio is calculated as follows:

Let samples set S be split by attribute A having values $a_1, a_2, .. a_v$ into subsets $s_1, s_2, .., s_v$:

$$\boxed{\text{SplitInfo(S,A)} = - \sum_{i=1}^{v} \frac{s_i}{s} \log_2 \frac{s_i}{s}} \qquad (4)$$

$$\boxed{\text{Gain ratio(S,A)} = \frac{Gain(S,A)}{SplitInfo(S,A)}} \qquad (5)$$

However, using gain ratio may lead to the problem of overcompensation, which means that the algorithm may choose an attribute just because its split information is very low. The standard way to work around this problem is to first consider attributes with **greater than average** information gain. Only then qualified attributes will be compared on gain ratio.

For each discrete attribute, one test is used to produce many outcomes as the number of distinct values of the attribute. However, discreet-valued attributes will be used for splitting only once during classification. Continuous-valued attributes are used many times in a path for splitting. For each continuous attribute, the data is sorted in ascending order, and the information gain is calculated based on **binary** cuts on each distinct value in one scan of the sorted data. From the calculated information gains, the best split point is the point with the highest gain. The best split point's gain is the gain for the attribute. This process is repeated for all continuous attributes. This leads the algorithm to be more computationally demanding.

The splitting process of C4.5 is stopped under the following conditions:

- When all cases have the same class. The leaf node is labeled by this class
- When there is no available attribute. The leaf node is labeled by the majority class.
- When the number of cases is less than a specified threshold. The leaf node is labeled by the majority class.

Overall, the modified algorithm is as follows:

```
Algorithm: Generate_decision_tree. Generate a decision tree from the
given training data
Input: The training samples, samples, represented by discrete-valued
or numerical-valued attributes; the set of candidate attributes,
attribute-list.
Output: A decision tree
Method:
1) create a node N
2) if samples are all of the same class, C then
3)     return N as a leaf node labeled with the class C;
4) if attribute-list is empty then
5)     return N as a leaf node labeled with the most common class in
                                            samples;// majority voting
6) If number of samples in samples is less than a threshold then
7)     return N as a leaf node labeled with the most common class in
                                            samples;// majority voting
8) select test-attribute, the attribute among attribute-list with the
   highest information gain ratio among the attributes has information
                                                  gain above average;
9) label node N with test-attribute;
10)    If test-attribute is discreet-valued attribute then
11)    for each known value $a_i$ of test-attribute //partition the samples
12)      grow a branch from node N for the condition test-attribute = $a_i$;
13)      let $s_i$ be the set of samples in samples for which test-
                                            attribute= $a_i$; //a partition
14)      if $s_i$ is empty then
15)          attach a leaf labeled with the most common class in samples;
16)      else attach the node returned by
                 Generate_decision_tree($s_i$,attribute-list - test-attribute);
17)    Else (i.e. test-attribute is numerical-valued attribute, and cut
                                                  point value is $a_i$)
18)      grow a branch from node N for the condition test-attribute <= $a_i$
           and a branch from node N for the condition test-attribute > $a_i$
19)      let $s_1$, $s_2$ be the set of samples in samples for which test-
                     attribute <= $a_i$ and test-attribute > $a_i$ respectively
20)       if $s_1$, $s_2$ is empty then
```

```
21)      attach a leaf labeled with the most common class in samples;
22)    else attach the node returned by
         Generate_decision_tree($s_i$, attribute-list - test-attribute);
  // $s_i$ here is $s_1$, $s_2$
```

Figure 2.3: C4.5 algorithm

2.9 Decision tree pruning

The C4.5 algorithm allows pruning of the resulting decision trees. This increases the error rates on the training data, but importantly, decreases the error rates on the unseen testing data. Pruning methods simplify the decision tree to prevent overfitting to noise in the data. Such methods typically use statistical measures to remove the least reliable branches, generally resulting in faster classification and improvement in the ability of the tree to correctly classify independent test data. There are two main strategies:

2.9.1 Pre-pruning approach

A tree is pruned by halting its construction early, i.e. not to further split or partition the subset of training samples at a given node. The node then becomes a leaf. The class of the leaf may be decided using majority voting of the probability distribution of samples within that node. This method is also called *forward* pruning.

Overall, this method involves trying to decide during the tree-building process when to stop developing sub-trees- quite an attractive prospect because that would avoid all the work of developing sub-trees only to throw them away afterward (Witten and Frank, 1999). In this approach, measures such as statistical significance (using chi-squared test), information gain are used to assess the goodness of a split. If the result of splitting falls below a pre-specified threshold, then the split is halted.

2.9.2 Post-pruning approach

This approach removes branches from a fully constructed tree. The cost complexity calculation is performed. For each non-leaf node in the tree, the expected error rate is calculated if the sub-tree at that node was pruned. Then the expected error rate if the node was not pruned is calculated. If the later rate is greater than the former one then the sub-tree is pruned. This method is also called *backward* pruning.

This approach offers some advantage over pre-pruning. For example, some situations occur in which two attributes individually seem to have nothing to contribute, but are powerful predictors when combined- a sort of combination-lock effect where the correct combination of the two attribute values is very informative whereas the attributes taken individually are not. Most decision tree builders use post-pruning. Possible strategies to be used: error estimation, significance testing, Minimum Description Length (MDL) principle.

There are two operations involved in this approach; users can choose either *sub-tree replacement*, i.e. to replace an internal node by a leaf node or *sub-tree raising*, i.e. to replace an internal node by one of the nodes below. In order to decide to perform either operation at a node, it is necessary to estimate the error rate that would be expected given an independently chosen test set. Error rate is estimated at internal nodes as well as leaf nodes then it would be clear whether to replace, or raise, a particular sub-tree simply by comparing the estimated error of the sub-tree with that of its proposed replacement.

However, the training set is not used for error estimation as the tree has been constructed expressly for that particular training set. The reduced-error pruning uses the standard verification technique: hold back some of the data originally given and use it as an independent test set to estimate the error at each node. It suffers from the disadvantage that the actual tree is based on less data.

C4.5 algorithm uses an alternative, making estimate of error based on the training data itself, which is a heuristic based on some statistical reasoning. Taking the set of instances reaching a node into consideration, with the majority class represents the node. A certain number of errors, E, is counted out of the total number of N instances. If true probability of error at the node is q, then the N instances are generated by a Bernoulli process with parameter q, of which E turn out to be errors.

Given a particular confidence level c (the default figure used in C4.5 is c=25%), the confidence limit z is such that

$$\boxed{\Pr\left[\frac{f-q}{\sqrt{q(1-q)/N}} > z\right] = c} \tag{6}$$

Where f=E/N is the observed error rate, and q is the true error rate. This leads to an upper confidence limit for q, which is used as a pessimistic estimate for the error rate e at the node:

$$\boxed{e = \frac{f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}}} \tag{7}$$

Where z is the number of standard deviations corresponding to the confidence c, which for c=25% is z=0.69

This is done with the assumption that the data is normally distributed and statistics from the training set are used. However, the error formula is accepted since it works reasonably well in practice (Witten and Frank, 1999).

2.10 Decision Tree Validation

Normally, the error rate, i.e. the ratio of the number of wrongly classified samples over the total number of samples, is used to measure the classifier's overall performance. However, we are only interested in the performance of the classifier on the unseen or future data, not the data already used for training. The reason is the classifier has learned from that training data, therefore any estimate of performance based on that data will be optimistic. Some learning schemes would involve two stages, one to create the basic model and one to optimize the parameters involved in that model. Three independent sets of data are required in such case: the training set, the validation set and the test set. The training set is used to come up with the classifier; the validation set is used to optimize parameters of the classifier, or to select the best combination of parameters; the test set is used to calculate the error rate of the final, optimized scheme (Witten and Frank, 1999).

According to Han (2001), the two common techniques for estimating classifier accuracy are Holdout method and k-fold cross validation method; both are based on randomly-partitioned partitions of the given data.

2.10.1 Holdout method

In this method, the original population of samples is partitioned into training set and test set. Typically, two thirds of the population is used for training and one third is used for testing. The training set is used to build the classifier, and its accuracy will be estimated with the test set as illustrated in Figure 2.4:

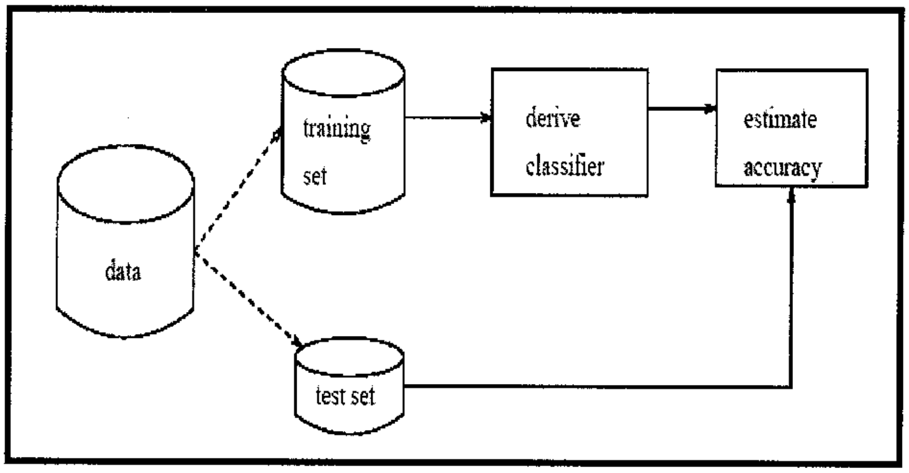


Figure 2.4: Estimating classifier accuracy with Holdout method

The estimate is pessimistic because only a portion of the original population is used for deriving the classifier. Random sub-sampling is a variation of the holdout method in which the holdout method is repeated k times. The overall accuracy estimate is taken as the average of the accuracies obtained from each run.

2.10.2 k-fold cross validation method

This method is illustrated in Figure 2.5:

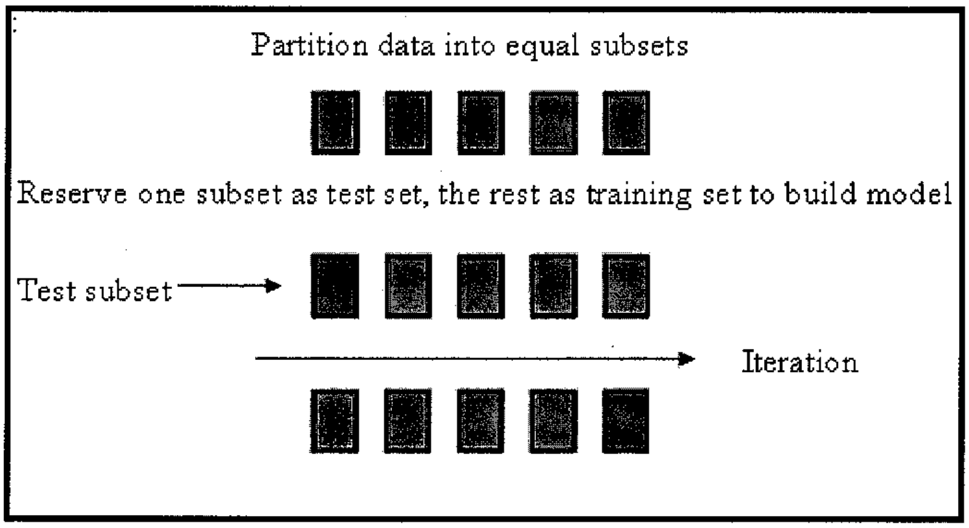

Partition data into equal subsets
Reserve one subset as test set, the rest as training set to build model
Test subset
Iteration

Figure 2.5: Estimating classifier accuracy with k-fold cross validation method

The initial data are randomly partitioned into k mutually exclusive subsets or "folds", S_1, S_2,...,S_k, each of approximately equal size. Training and testing is performed k times. In iteration i, the subset S_i is reserved as the test set, and the remaining subsets are collectively used to train the classifier. That is, the classifier of the first iteration is trained on subsets S_2,...,S_k, and tested on S_1; the classifier of the section iteration is trained on subsets S_1,S_3,...,S_k, and tested on S_2; and so on.

The accuracy estimate is the overall number of correct classifications from the k iterations, divided by the total number of samples in the initial data. In stratified cross-validation, the folds are stratified so that the class distribution of the samples in each fold is approximately the same as that in the initial data.

Other methods of estimating classifier accuracy include **bootstrapping**, which samples the given training instances uniformly with replacement, and **leave-one-out**, which is k-fold cross validation with k set to s, the number of initial samples. In general, stratified **10-fold cross-validation** is recommended for estimating classifier accuracy (even if computation power allows using more folds) due to its relatively low bias and variance. The use of such techniques to estimate classifier accuracy increases the overall computation time, yet is useful for selecting among several classifiers.

CHAPTER 3
METHODOLOGY

3. METHODOLOGY

3.1. Procedure Identification

The "**waterfall model**" is chosen as the methodology of this project. The phases of waterfall model are illustrated in Figure 3.1:

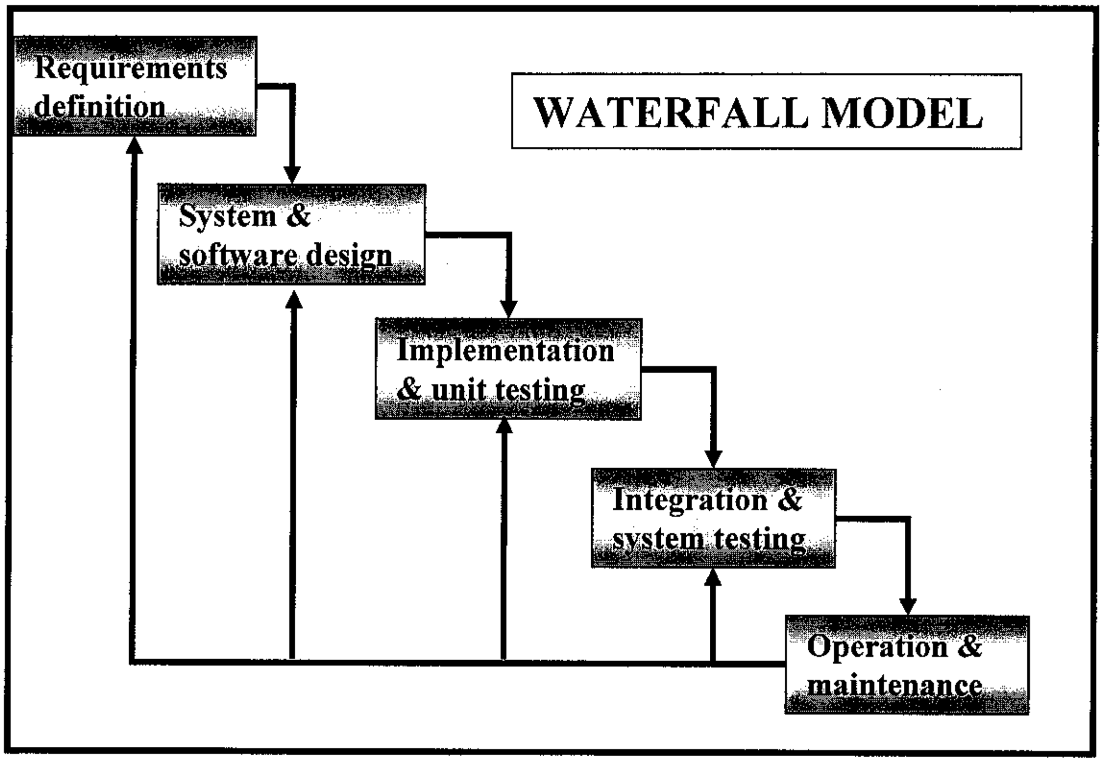


Figure 3.1: Waterfall model

The waterfall approach emphasizes a structured progression between defined phases. Each phase consists of a definite set of activities and deliverables that must be accomplished before the following phase can begin. The first phase tries to capture *What* the system will do (its requirements), the second determines *How* it will be designed, in the middle is the actual programming, the fourth phase is the full system *Testing*, and the final phase is focused on *Implementation* tasks such as go-live, training, and documentation. (Marks, 2002). The details of each of the phases are discussed below:

3.1.1 Requirements analysis and definition

It is important that the requirements of the project be clearly analyzed and defined in detail so that they can serve as project specification. This phase involves with the identification of the background of the study and the definition of the objective and scope of the project.

From this specification, a literature review is conducted to identify the available data classification methods and suitable decision tree induction algorithm to tackle the identified requirements. Requirements specification also helps in the design and planning of the software development.

3.1.2 System and software design

The system design process normally partitions the requirements to either hardware or software systems. It establishes overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships (Sommerville, 2001).

Based on the detailed requirements obtained from the first phase of this project, the software architectural design task is performed, which includes identifying the data flow, the class diagram of this application and also the test plan. The data flow diagram depicts at high level the flow of data and the processes performed in the application. The class diagram is used to illustrate the objects, their properties and methods and the relationships between different objects. The data flow diagram and the class diagram together serve as the solid framework from which the implementation task is based upon. Typically, projects using waterfall methodology would result in 20%-40% of the time budget allocated for the first two phases of the development. This considerable amount of time spent on assembling, documenting the requirements and software design results in better segregation of work into modules and also in better control of the modules in later phase. The detailed design using this waterfall approach also makes it easier to ensure that the modules will be easier to integrate when the project nears the end of the implementation phase.

3.1.3 Implementation and unit testing

Based on the architectural design framework from the previous phase, the implementation task is performed. The whole application is partitioned into a set of program units or modules. Depending on the resources available, modules will be developed by a team or a person. Unit testing involves verifying that each unit/ module meets its specification. Typically, 30%-40% of project's time budget is allocated to this phase.

Based on the data flow diagram, the implementation of this project is divided into modules consisting of the typical processes or phases of data mining, namely Data Selection, Data Preprocessing, Decision Tree Induction, Decision Tree Pruning, and Validation. Each of these processes is considered as a module of the application and is developed in sequence due to the linear nature in this tool. After modules are coded, they will be tested to ensure their conformance to the requirements specification.

3.1.4 Integration and system testing

The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer (Sommerville, 2001). This is the phase whereby different modules developed and tested in this project are integrated into a complete application. An overall testing on the application is performed to ensure that the integrated modules work according to the specification. At this stage, it may lead to detection of incompatibility between developed modules and modifications are necessary to ensure that the end product is error-free.

3.1.5 Operation and maintenance

Normally this is the longest life-cycle phase. The application is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's functionalities as new requirements are discovered. The documentation of the project is one important deliverable in this phase.

In summary, with waterfall methodology, project tasks are divided into phases, typically the previous phase's deliverables have to be completed before the next phase can be started. However, there are also iterations at the end of the development process that allows for modifications on the previous phase's deliverables to ensure compliance to the software specifications. The reason for choosing the waterfall model in this project is because of its structured processes with focus on a detailed set of requirements and detailed design of the whole application that would make the development and verification tasks easier. This approach also allows for a good estimation of the time allocated for each of the major phase of the project. The linear nature of this approach requires that the deliverables of the previous phase be completed and tested before the next phase can be carried out, which reduces the amount of rework and debugging when all the modules are integrated.

3.2 Tools Required

As this project work is based on Java language and flat text format for dataset file, the following tools are required

1. Java Software Development Toolkit
2. JCreator LE

CHAPTER 4
RESULT AND DISCUSSION

4. RESULT AND DISCUSSION

4.1 Software Architectural Design

4.1.1 Data Flow Diagram

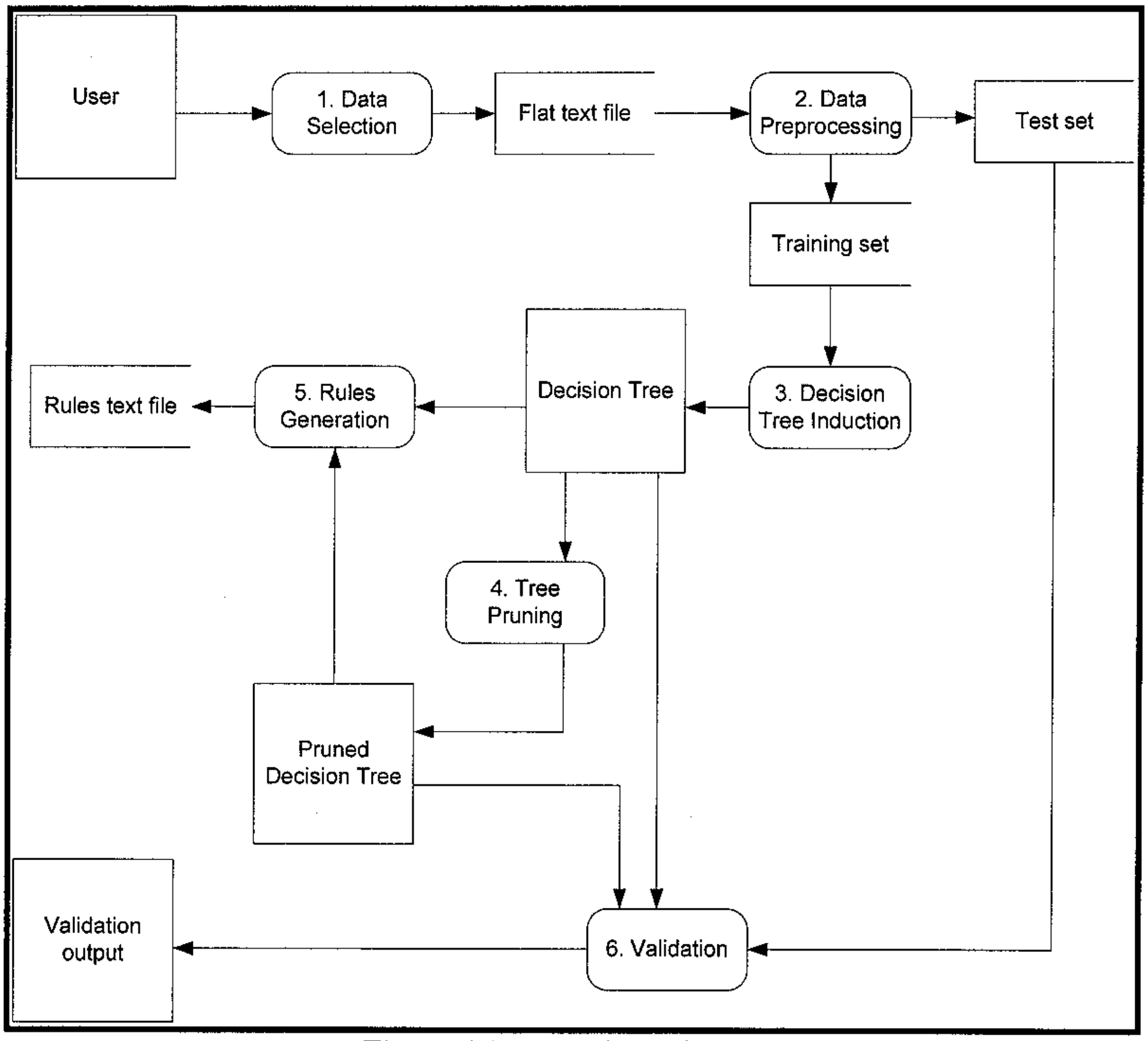


Figure 4.1: Data Flow Diagram

From the data flow diagram shown in Figure 4.1, user starts with selecting the source of the data in the Data Selection process. The selected flat text file is an input to the Data Preprocessing process. This process performs the basic preprocessing tasks on the raw data and partitions the original dataset into the training set and the test set. The test set will remain intact, while the training set is used as input for the Decision Tree Induction process. This process applies the C4.5 algorithm and produces the decision tree. Decision rules can be generated and stored into a text file via the Rules Generation process. Tree Pruning process turns the unpruned tree to the pruned tree to reduce error rate on unseen data. Lastly, the test set is used to validate the accuracy of the unpruned and pruned trees.

Each process is detailed as follows:

- ***Data selection***

In this phase, user specifies the path of the data source. The data can be of any nature, from financial, scientific to medical dataset, provided that it conforms to the format in Appendix A1. The input dataset can contain either discreet or continuous attributes or both. The dataset is loaded into the memory and its contents are displayed.

After the data is loaded, modifications can be made to the original dataset and inconsistent data will be filtered out i.e. if alphabetical characters are filled in place of numeric attribute, they will be marked as missing values for preprocessing. If there is no exception from reading the dataset, a decision tree object is initialized with a set of attribute names and their types (discreet or numeric). The root node of the tree is populated with all the data samples contained in the dataset.

- ***Data preprocessing***

Data preprocessing is one of the most critical phases in any data mining process. Depending on the dataset to be mined, different preprocessing operations will need to be performed on the raw data to obtain good quality data. Among the vast data preprocessing tasks, the data preprocessing phase in this application is involved with (1) cleaning the dataset i.e. handling missing values, removing attributes with noisy data and (2) partitioning the original dataset into training set and test set.

Missing values are handled in two ways: they are either replaced with the most frequent value appeared in the population; OR the records containing the missing values are removed from the population. Attribute with majority of missing values will also be removed from the set of attributes. User can specify the threshold value of the percentage of null values to remove an attribute.

The training set will be used for decision tree induction. Meanwhile, the test set will not be used in the induction process but in the validation process. The percentage of the training set and test set out of the original population samples can be specified by the user at run time. The test set contains records randomly selected from the original population. The rest will be used for training.

- ***Decision Tree Induction***

Decision tree induction is the most important phase whereby the C4.5 decision tree learning algorithm (please refer to section 2.8.2 for details) is used to recursively partition the sample population in the root node into test nodes and leaf nodes to form the decision tree. The input to this phase is the cleaned training set obtained from the Data Preprocessing phase.

Before the induction process can start, numeric attributes need to have their respective samples sorted in ascending order, i.e. there need to be an array for each of the numeric attribute, each stores the index of samples whose values are sorted in ascending order. This was done by modifying the Merge Sort algorithm such that the returned array from the sorting process contains not the sorted values themselves but the indices of the sorted values.

The process starts by finding the best attribute that would best classify the samples contained in the root node. It is worth noting again that the discreet attribute can only be used once in any path from the root node to a leaf node while numeric attribute can be used any number of times to split a node. Unless one of the stopping conditions is met, nodes are recursively split based on entropy-based measures such as information gain, split ratio, gain ratio. Besides the condition of all samples are of the same class, user can specify at run time the minimum number of samples and the percentage of a major class to stop splitting. The output of this process is a fully constructed decision tree.

By using Multithreading, tree building animation speed can also be selected at run time, with the effect of putting the thread to sleep for 1 second if the speed is slow; 0.5 second if the speed is medium and 0 second if the speed is fast; before the next node is split.

- ***Tree Pruning***

The pruning process is performed once the tree induction process is completed. Post-pruning with sub-tree replacement method is employed for the pruning phase. This method works bottom-up, i.e. it would replace the sub-tree at the bottom if a leaf node replaces it has lower expected error rate than that of the sub-tree. Tree pruning will result in an increase of the number of wrongly classified samples in the training set but would decrease the number of wrongly classified samples in the unseen test set. In C4.5 algorithm, pessimistic estimate of error rate (i.e. upper confidence limit of the error probability) is used, with the default confidence level of 25%.

- ***Rules Generation***

Once decision tree induction and tree pruning phases are completed, a set of rules for each of the tree (unpruned and pruned trees) can be derived from the decision tree to express the gained knowledge in simple manner. All paths from the root node to leaf nodes are parsed to find the rules. A rule can be built by ANDing all the internal nodes' names into the IF clause and attaching the leaf node's name into the THEN clause.

Since numeric attribute can be used many times in a path from the root node to the leaf node, the IF clause needs simplifying when these cases happen:

- The rule is "*IF (attribute A1 > a) AND...AND (attribute A1 > b) THEN Decision= Decision1*" and **b> a** , then the condition "*(attribute A1 > a)*" will be removed
- The rule is "*IF (attribute A1 < a) AND...AND (attribute A1 < b) THEN Decision= Decision1*" and **b< a** , then the condition "*(attribute A1 < a)*" will be removed

One advantage of rules in *IF* ... *THEN* clause is easy to read because it is in a form of natural language. However, the slight disadvantage is that the rules in the form of *IF* ... *THEN* clause do not reflect the important hierarchy. The decision tree induction process implies that the closer an attribute is to the root node; the better it is to classify the dataset. In the *IF* ... *THEN* clause, no attribute is more significant than the others.

The set of rules derived can be saved into a text file for future use. With acceptable accuracy, these sets of rules can be used as part of the knowledge base for an expert system.

- ***Validation***

In order to determine the accuracy of the knowledge gained from the decision tree, two validation methods are implemented in this application: holdout validation method and 10-fold cross validation method.

A confusion matrix is a useful way to represent the accuracy of a decision tree classifier. It contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using the data in the matrix.

Assuming that a classifier is used on a population of **N** samples and the target attribute has **m** values, C_i , i=1,..,m. Then a **mXm** matrix is used as follows:

		Predicted		
		C_1	**...**	**C_m**
Actual	**C_1**	C_{11}		C_{1m}
	....	...	...	...
	C_m	C_{m1}	...	C_{mm}

The value of each cell C_{ij}, with i=1,..,m and j=1,..,m represents the number of samples, whose actual class is C_i, classified as belonging to class C_j. This means that correctly classified samples are represented by C_{ij}, where i=j. All other cells C_{ij} where $i \neq j$ represent wrongly classified samples. Therefore, the accuracy of a classifier can be calculated as:

$$\boxed{Accuracy = \frac{\sum_{i=1}^{m} C_{ii}}{N}} \qquad (8)$$

For ***hold-out validation method***, the classification accuracy is calculated for the training set and the test set, based on the confusion matrix of both. The accuracy is calculated as the ratio of number of correctly classified samples over the total number of samples in that set, as in (8).

For ***10-fold validation method***, the original sample population is randomly divided into 10 equal portions, one of them is used as test set, and the rest are used as training set for decision tree induction. The same process is repeated 10 times and the average accuracy of 10 processes is calculated.

4.1.2 Class diagram

Due to the complexity of the whole application, the simplified class diagram for this application is as follows:

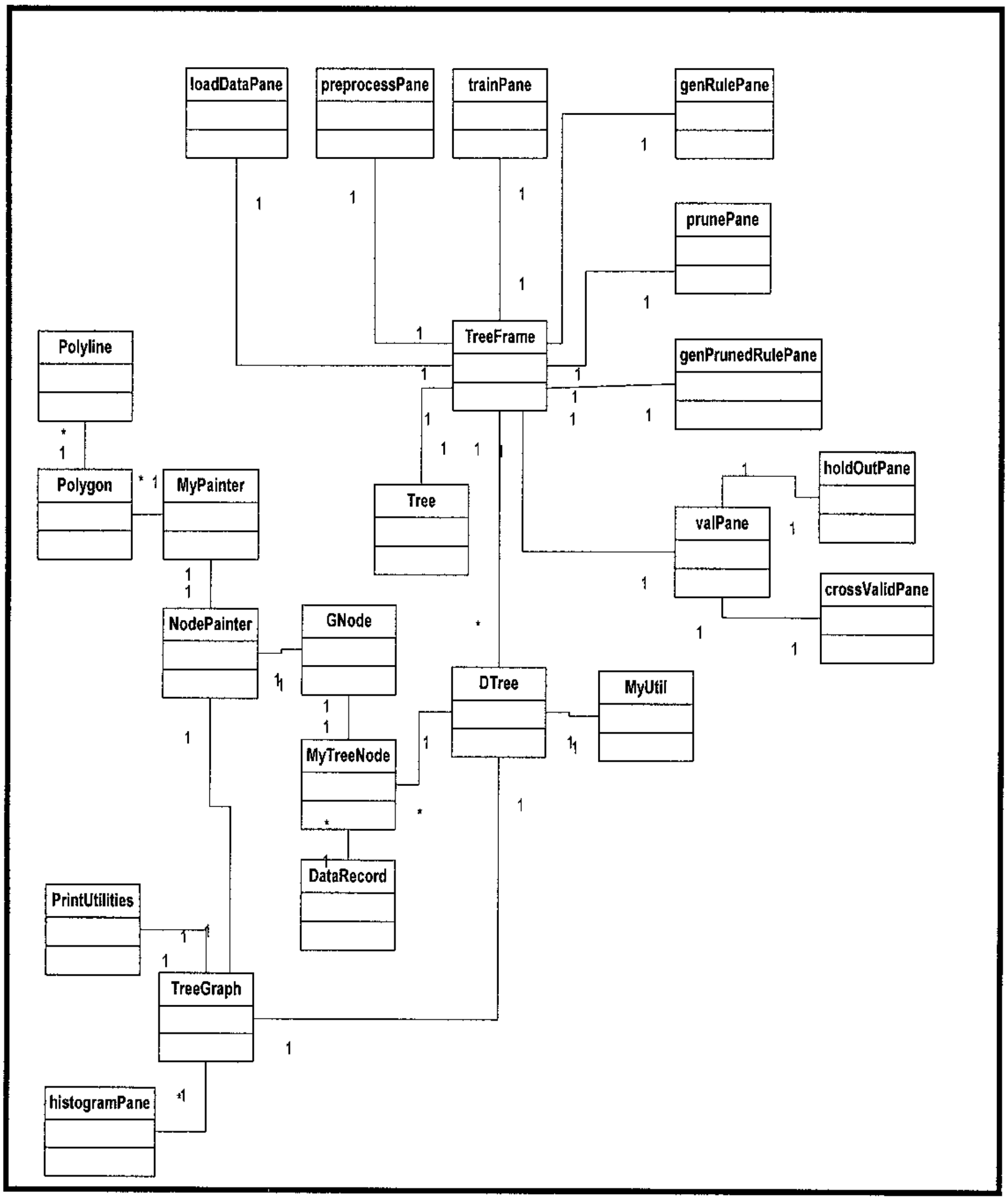


Figure 4.2: Class Diagram

Below are the roles of each class:

- ***Tree***: the main class of the application to hold the interface
- ***TreeFrame***: a JFrame which is the main User Interface of the application
- ***loadDataPane***: a JPanel that displays the tab for "Load Dataset" phase
- ***preprocessPane***: a JPanel that displays the tab for "Preprocess Data" phase
- ***trainPane***: a JPanel that displays the tab for "Decision Tree Induction" phase
- ***genRulePane***: a JPanel that displays the tab for "Rules Generation" phase
- ***prunePane***: a JPanel that displays the tab for "Tree Pruning" phase
- ***genPrunedRulePane***: a JPanel that displays the tab for "Pruned Rules Generation" phase
- ***valPane***, ***holdOutPane***, ***crossValidPane***: JPanels that display the tab for "Validation" phase, the last two for validation using Holdout method and Cross validation method respectively
- ***DTree***: the main class of the application, contains all attributes and methods necessary to build the decision tree. Please refer to Appendix A2 for detail structure of this class.
- ***MyTreeNode***: a class to represents the node in the tree
- ***DataRecord***: a class to represent each record contained in the dataset
- ***MyUtil***: a class contains common utilities e.g. merge sorting, parse rules, etc.
- ***TreeGraph***: a JPanel to display the graphical decision tree
- ***histogramPane***: a JPanel to display the histogram of node's records distribution
- ***PrintUtilities***: a common utility to print a JPanel
- ***GNode***: a class to represent the graphical display of a node
- ***NodePainter***: a class to draw the GNode object
- ***MyPainter***: a class to draw the TreeGraph
- ***Polygon***, ***Polyline***: two classes to draw GNode object

4.2 Implementation

The application was built to demonstrate the capability of decision tree induction algorithm to extract hidden knowledge from raw data. Although it can be used for any dataset conforming to the format proposed, the credit card approval dataset was used to demonstrate the application of decision tree to support decision making when approving credit card applications.

4.2.1 Credit Card Approval Dataset

The credit card approval process is always performed to check if the applicant is qualified for a credit card. A bank will need to ask the applicants certain questions, which will assist the approval department make their decision about whether to grant the applicant a credit line or not. A bank will need to take into consideration many personal details, especially credit history, of the applicant and no simple rule of thumb is in place to assess the risk of granting credit card to any one person. This is where data mining plays an important role to dig into the historical data of approved and rejected cases to find out the patterns of good and bad credit card applicants. The knowledge gained from this data mining process, together with banking experts' judgments would decide whether to approve the credit card application and reduce the risks for the bank.

The credit card screening dataset, originated from J. R. Quinlan, was obtained from the following URL: http://www.csee.usf.edu/~mlast/credit.dat.The dataset contains records from bank credit card applications, including the credit outcomes (accept / reject). There are altogether 14 input attributes and 1 class attribute. The total number of instances is 690. The number of class labels is 2 (accept / reject). This dataset is interesting because there is a good mix of attributes -- continuous, nominal with small numbers of values, and nominal with larger numbers of values. There is no missing value in this dataset.

The short description of the attributes is available in Table 4.1:

Table 4.1: Credit Card Approval Dataset Description

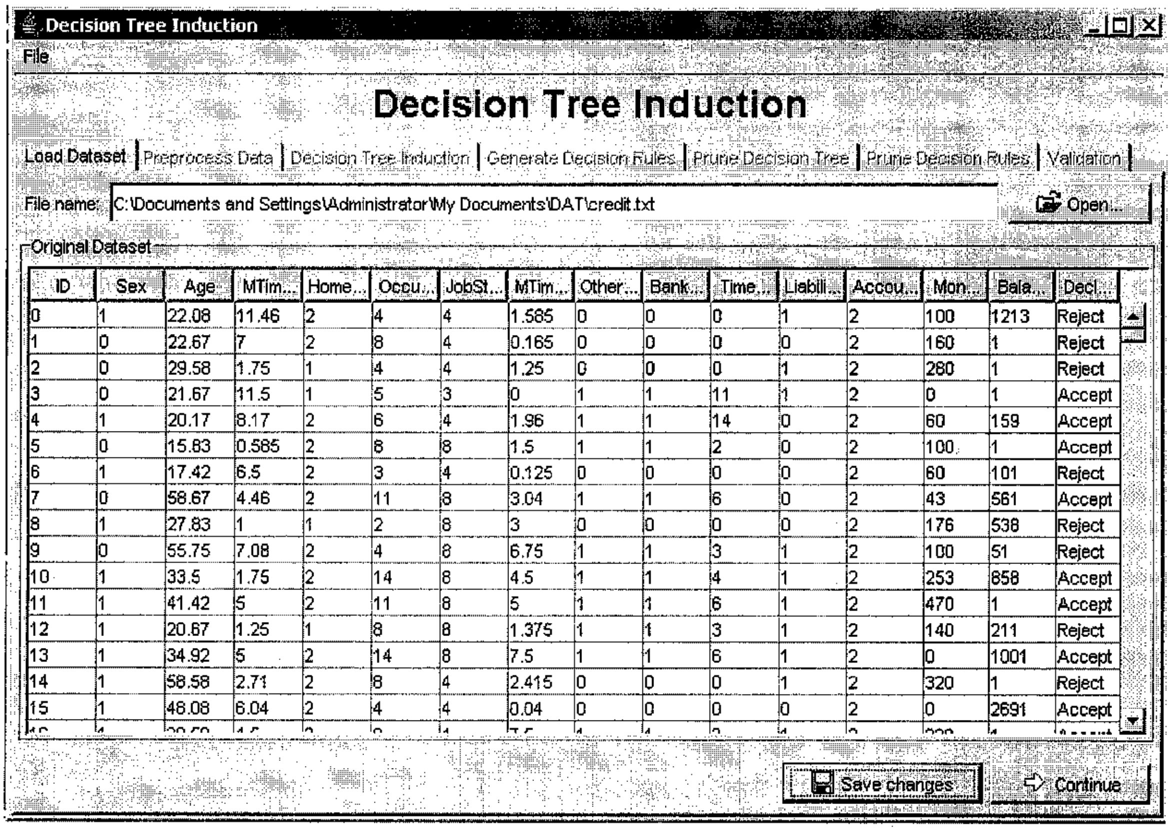
Attribute	**Domain**	**Type**	**Use in Model**
Sex	0,1	Nominal	candidate input
Age	13.75 - 80.25	Continuous	candidate input
Mean time at addresses	0 - 28	Continuous	candidate input
Home status	1,2,3	Nominal	candidate input
Current occupation	1 - 14	Nominal	candidate input
Current job status	1 - 9	Nominal	candidate input
Mean time with employers	0 - 28.5	Continuous	candidate input
Other investments	0,1	Nominal	candidate input
Bank account	0,1	Nominal	candidate input
Time with bank	0 - 67	Continuous	candidate input
Liability reference	0,1	Nominal	candidate input
Account reference	1,2,3	Nominal	candidate input
Monthly housing expense	0 - 2000	Continuous	candidate input
Savings account balance	1 - 100001	Continuous	candidate input
Class (Accept / Reject)	0,1	Nominal	target

4.2.2 Results

The outputs of the decision tree induction tool performed on the credit card approval dataset are as follows:

i. Data Selection

The selected credit card approval dataset is first loaded into the application. Modifications on the original dataset can be made and inconsistent data will be marked as missing values for preprocessing.



File name: C:\Documents and Settings\Administrator\My Documents\DAT\credit.txt

Original Dataset

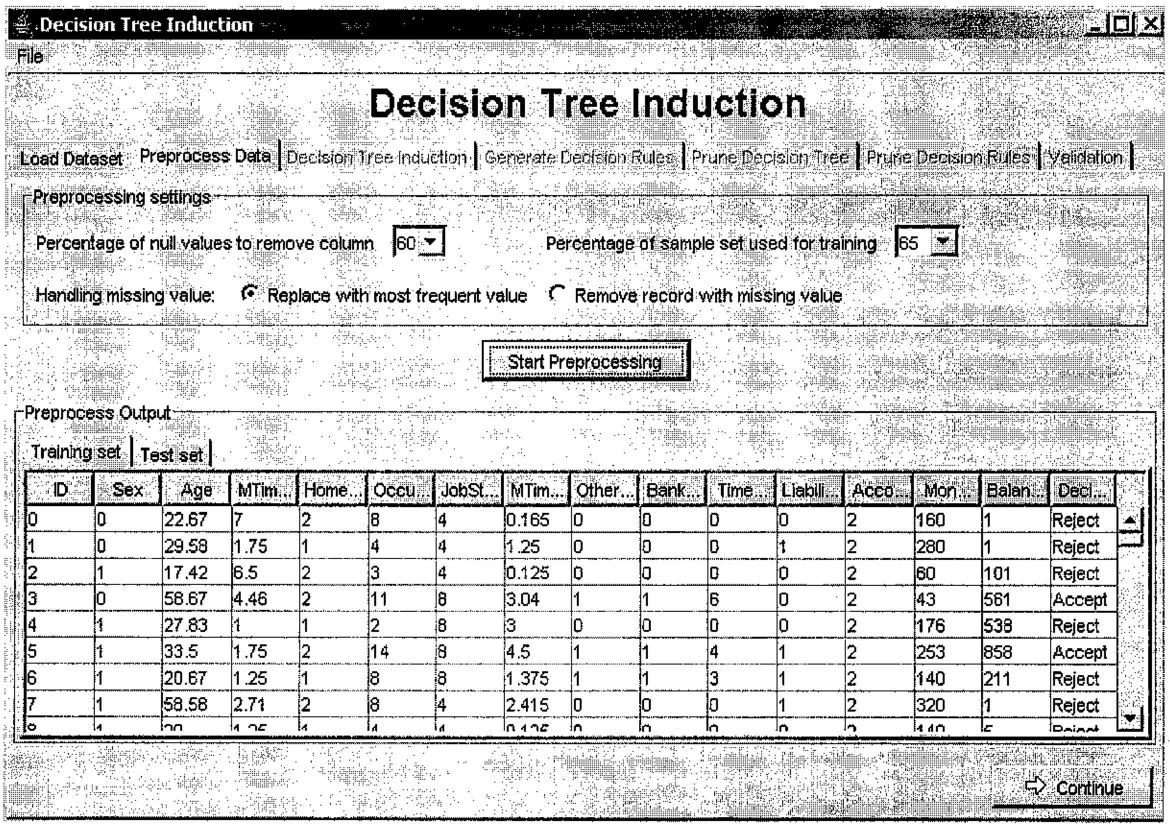
ID	Sex	Age	MTim...	Home...	Occu...	JobSt...	MTim...	Other...	Bank...	Time...	Liabili...	Accou...	Mon...	Bala...	Decl...
0	1	22.08	11.46	2	4	4	1.585	0	0	0	1	2	100	1213	Reject
1	0	22.67	7	2	8	4	0.165	0	0	0	0	2	160	1	Reject
2	0	29.58	1.75	1	4	4	1.25	0	0	0	1	2	280	1	Reject
3	0	21.67	11.5	1	5	3	0	1	1	11	1	2	0	1	Accept
4	1	20.17	8.17	2	6	4	1.96	1	1	14	0	2	60	159	Accept
5	0	15.83	0.585	2	8	8	1.5	1	1	2	0	2	100	1	Accept
6	1	17.42	6.5	2	3	4	0.125	0	0	0	0	2	60	101	Reject
7	0	58.67	4.46	2	11	8	3.04	1	1	6	0	2	43	561	Accept
8	1	27.83	1	1	2	8	3	0	0	0	0	2	176	538	Reject
9	0	55.75	7.08	2	4	8	6.75	1	1	3	1	2	100	51	Reject
10	1	33.5	1.75	2	14	8	4.5	1	1	4	1	2	253	858	Accept
11	1	41.42	5	2	11	8	5	1	1	6	1	2	470	1	Accept
12	1	20.67	1.25	1	8	8	1.375	1	1	3	1	2	140	211	Reject
13	1	34.92	5	2	14	8	7.5	1	1	6	1	2	0	1001	Accept
14	1	58.58	2.71	2	8	4	2.415	0	0	0	1	2	320	1	Reject
15	1	48.08	6.04	2	4	4	0.04	0	0	0	0	2	0	2691	Accept

Figure 4.3: Credit card dataset

ii. Data preprocessing

The next phase is to preprocess the data. If an attribute column contains missing values that are 60% or more of its values, that attribute will be removed from the dataset. The original dataset is divided into training set and test set (randomly extracted from the original dataset) with the ratio of 65%:35% accordingly.

If there are any missing values, they will be replaced by the most frequent values in the attribute they belong to. In this dataset, there is no missing value and no attribute was removed.



Decision Tree Induction

Preprocessing settings

Percentage of null values to remove column: 60

Percentage of sample set used for training: 65

Handling missing value: [✓ Replace with most frequent value] Remove record with missing value

Preprocess Output

ID	Sex	Age	MTim...	Home...	Occu...	JobSt...	MTim...	Other...	Bank...	Time...	Liabili...	Acco...	Mon...	Balan...	Deci...
0	0	22.67	7	2	8	4	0.165	0	0	0	0	2	160	1	Reject
1	0	29.58	1.75	1	4	4	1.25	0	0	0	1	2	280	1	Reject
2	1	17.42	6.5	2	3	4	0.125	0	0	0	0	2	60	101	Reject
3	0	58.67	4.46	2	11	8	3.04	1	1	6	0	2	43	561	Accept
4	1	27.83	1	1	2	8	3	0	0	0	0	2	176	538	Reject
5	1	33.5	1.75	2	14	8	4.5	1	1	4	1	2	253	858	Accept
6	1	20.67	1.25	1	8	8	1.375	1	1	3	1	2	140	211	Reject
7	1	58.58	2.71	2	8	4	2.415	0	0	0	1	2	320	1	Reject

Figure 4.4: Credit card dataset preprocessed.

iii. Decision Tree Induction

The decision tree induction process takes as input the cleaned training set from the data preprocessing process. Users can specify a few parameters for the induction process. The sample parameters chosen to build the decision tree are:

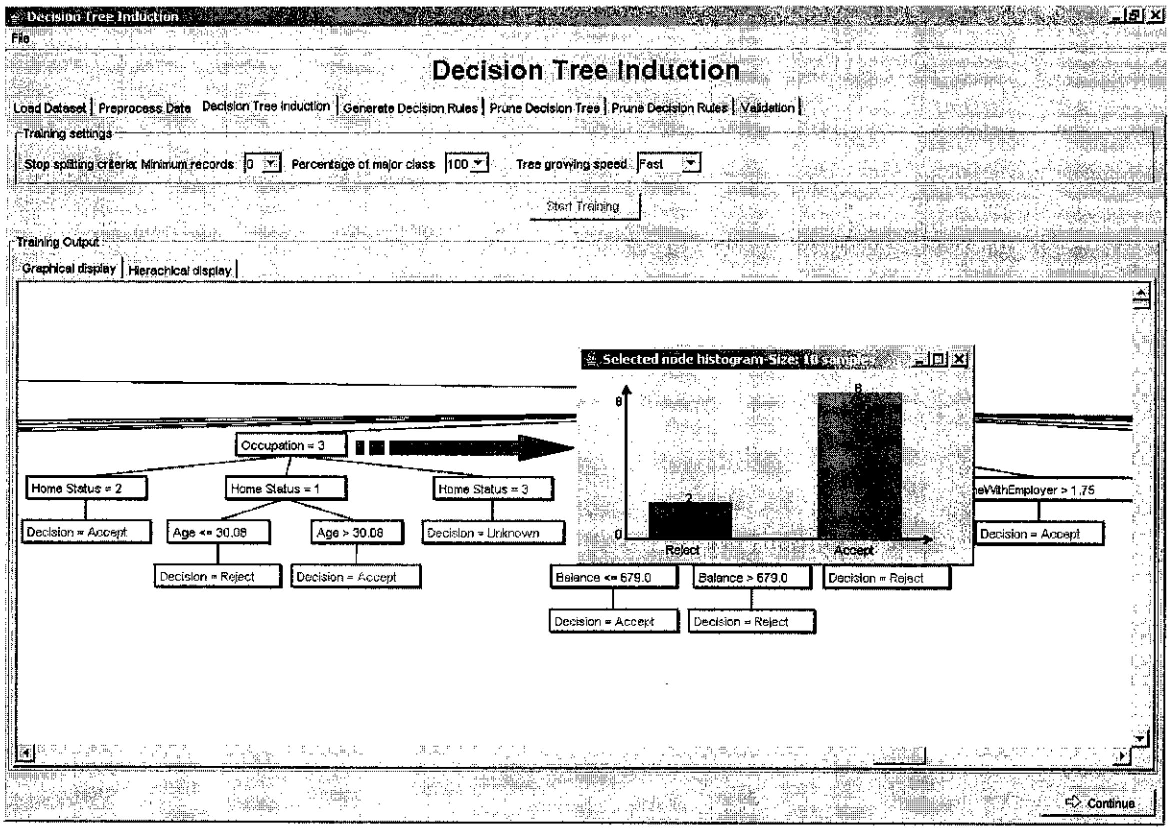
- Minimum number of records to stop splitting: 0
- Percentage of major class to stop splitting: 100
- Animation speed: Fast

The effect of these parameters is that there will be no early stopping of splitting any node. A node will be split until all the records are of the same class. In case a node contains no record, it will be attached with a leaf node classified as "Unknown".

The output of this process is a decision tree, which is represented in two forms: the graphical display and the hierarchy display.

a. Graphical Display

The screenshot in Figure 4.5 is the partial decision tree in its graphical display. For better visualization, the root node is in black color, at the top, the test node is in blue color, and the leaf node is in red color. The histogram is displayed when user clicks on any non-leaf node. It shows the distribution of samples by classes in the corresponding node.


Training settings

Stop splitting criteria: Minimum records: 0 Percentage of major class: 100 Tree growing speed: Fast

Figure 4.5: Decision Tree with Graphical Display

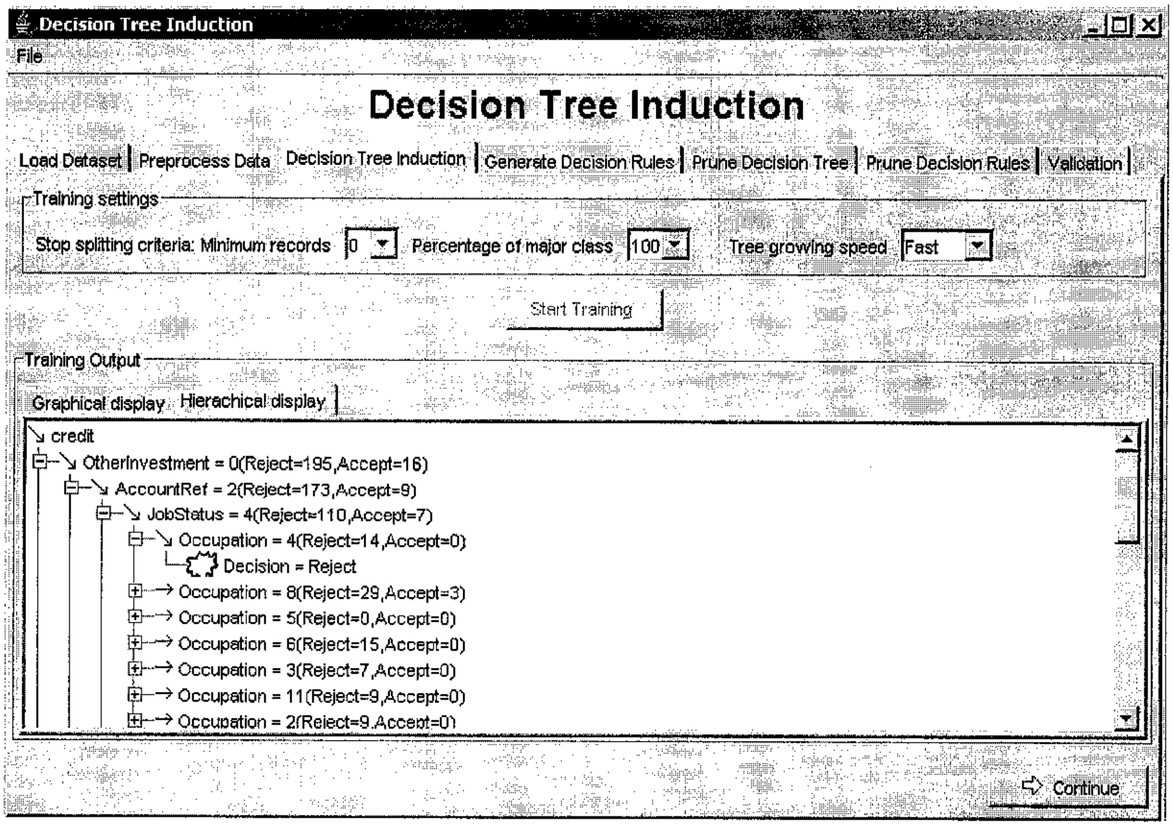
The code portion for drawing this graphical display is based on a reference paper by Sven Moen's "*Drawing Dynamic Trees*" article in IEEE Software, July 1990 and with modification on the source code of the decision tree applet developed by Pierre Geurts, August 1999 available at http://www.montefiore.ulg.ac.be/~geurts/dtapplet/ . The following modifications have been made:

- Use multithreading to refresh the tree every time a new node is added into the tree to create the animation.

- Allow users to specify the speed of the tree building animation, i.e. the sleep time for the thread
- Add a histogram for non-leaf node and an Information Message for a leaf node when user clicks on it.
- Add print feature to print the tree graph directly
- Add save as JPEG feature to save the tree as JPEG image.

b. Hierarchical Display

The decision tree in its graphical display may sometimes be complicated and its size is so large that can be hard to visualize. The hierarchical display of the tree in Figure 4.6 is another way to represent the tree, which is very much similar to folder browsing in Windows OS. Users can focus on any part of the tree by expanding the corresponding node and can defocus by collapsing it. At each test node, the node distribution frequency is displayed beside the node's name.



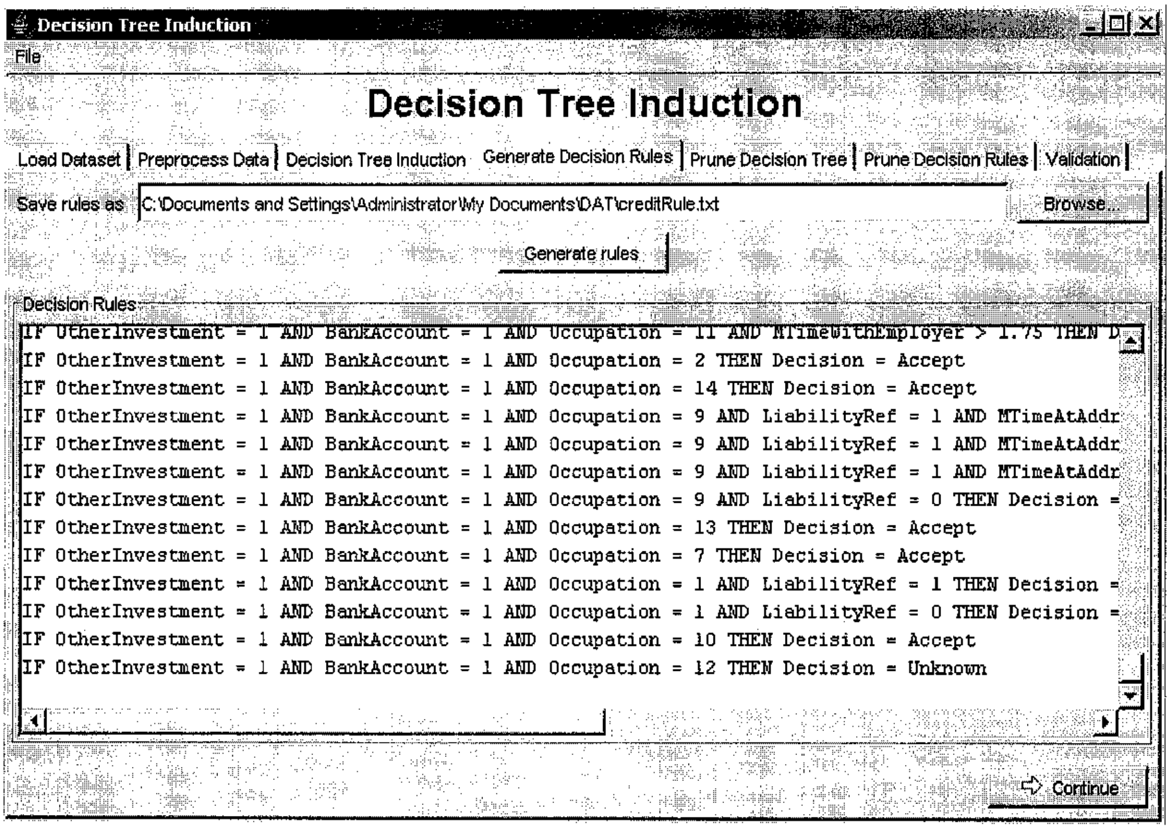
Stop splitting criteria: Minimum records 0
Percentage of major class 100
Tree growing speed Fast

```
credit
 OtherInvestment = 0(Reject=195,Accept=16)
   AccountRef = 2(Reject=173,Accept=9)
     JobStatus = 4(Reject=110,Accept=7)
       Occupation = 4(Reject=14,Accept=0)
         Decision = Reject
       Occupation = 8(Reject=29,Accept=3)
       Occupation = 5(Reject=0,Accept=0)
       Occupation = 6(Reject=15,Accept=0)
       Occupation = 3(Reject=7,Accept=0)
       Occupation = 11(Reject=9,Accept=0)
       Occupation = 2(Reject=9,Accept=0)
```

Figure 4.6: Decision Tree with Hierarchical Display

iv. Rules Generation

Once decision tree has been built, the decision rules can be directly extracted from it as in Figure 4.7. Below are some of the extracted rules in the form of "*IF condition THEN classification*":



Save rules as C:\Documents and Settings\Administrator\My Documents\DAT\creditRule.txt

```
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 11 AND MTimeWithEmployer > 1.75 THEN D
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 2 THEN Decision = Accept
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 14 THEN Decision = Accept
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 9 AND LiabilityRef = 1 AND MTimeAtAddr
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 9 AND LiabilityRef = 1 AND MTimeAtAddr
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 9 AND LiabilityRef = 1 AND MTimeAtAddr
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 9 AND LiabilityRef = 0 THEN Decision =
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 13 THEN Decision = Accept
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 7 THEN Decision = Accept
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 1 AND LiabilityRef = 1 THEN Decision =
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 1 AND LiabilityRef = 0 THEN Decision =
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 10 THEN Decision = Accept
IF OtherInvestment = 1 AND BankAccount = 1 AND Occupation = 12 THEN Decision = Unknown
```

Figure 4.7: Decision Rules

v. Tree Pruning

As mentioned in section 2.9, post-pruning is used to prune the decision tree, i.e. to increase the expected error rate in the training set but to decrease the expected error rate in the test set. The resulted pruned tree is as follows:

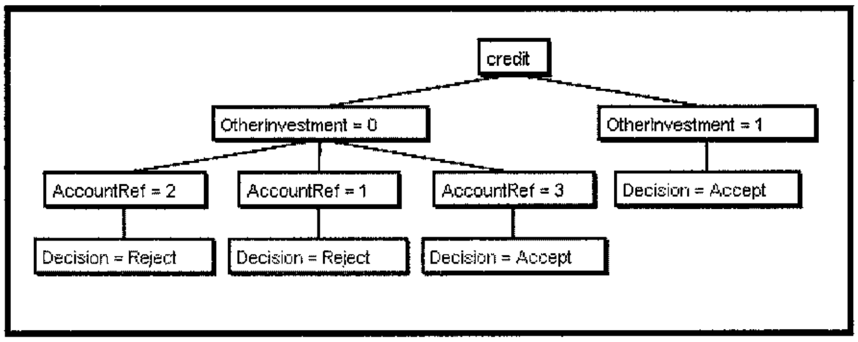


Figure 4.8: Pruned Decision Tree

The pruned decision tree is much simpler than the one presented in Figure 4.5, with the number of nodes and branches reduced substantially. The accuracy comparison will be performed in Validation phase.

This is only one of different pruned trees achieved with this dataset. Different settings for the decision tree classifier would result in different unpruned and pruned trees. Besides, the dataset in use is a simplified real dataset to demonstrate the capability of a decision tree classifier in credit card approval; therefore, the resulting tree would be very much simpler than that achieved from large real dataset.

vi. Pruned Rules Generation

The pruned rules are extracted as follows:

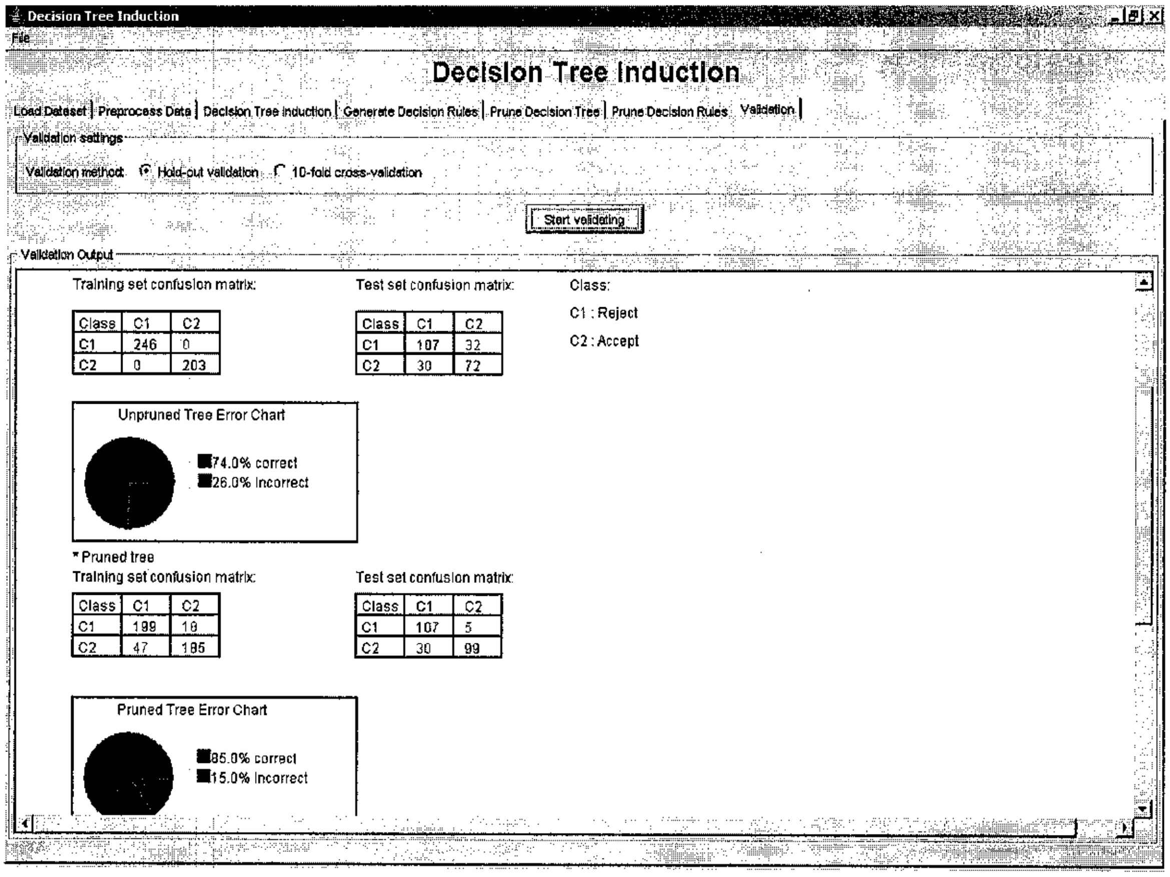
IF OtherInvestment = 0 AND AccountRef = 2 THEN Decision = Reject
IF OtherInvestment = 0 AND AccountRef = 1 THEN Decision = Reject
IF OtherInvestment = 0 AND AccountRef = 3 THEN Decision = Accept
IF OtherInvestment = 1 THEN Decision = Accept

vii. Validation

Validation is the last phase in the data mining process but it plays an important role in determining the overall performance of the classifier. Please refer to Section 2.10 for summary of validation methods.

- ***Validation with Holdout method***

Using Holdout method, the resulting tree is evaluated on how it performs on the training set and the test set. The confusion matrix is used to show how many records were correctly classified and how many were wrongly classified.


Decision Tree Induction
File
Decision Tree Induction
Load Dataset | Preprocess Data | Decision Tree Induction | Generate Decision Rules | Prune Decision Tree | Prune Decision Rules | Validation
Start validating
Unpruned Tree Error Chart
Pruned Tree Error Chart

Validation settings

Validation method: [✓ Hold-out validation] 10-fold cross-validation

Validation Output

Training set confusion matrix:

Class	C1	C2
C1	246	0
C2	0	203

Test set confusion matrix:

Class	C1	C2
C1	107	32
C2	30	72

Class:
C1 : Reject
C2 : Accept

■ 74.0% correct
■ 26.0% incorrect

* Pruned tree

Training set confusion matrix:

Class	C1	C2
C1	199	18
C2	47	185

Test set confusion matrix:

Class	C1	C2
C1	107	5
C2	30	99

■ 85.0% correct
■ 15.0% incorrect

Figure 4.9: Holdout Validation

As shown in Figure 4.9, for the unpruned tree, there is no wrongly classified record in the training set while there are 62 wrongly classified records out of 241 records of the test set. Therefore, the classifying accuracy of the classifier on the test set is 74%.

For the pruned tree, the number of wrongly classified records in the training set increases substantially from 0 to 65 records out of 449 records of the test set, while the number of wrongly classified records in the test set decreases to 35 records out of 241 records of the test set. Therefore, the classifying accuracy of the classifier on the test set is 85%, which shows that the pruned tree has improved performance on the unpruned tree and it also provides better (smaller error rate), shorter tree structure for decision making.

The accuracy of the unpruned and pruned trees can be summarized in Table 4.2:

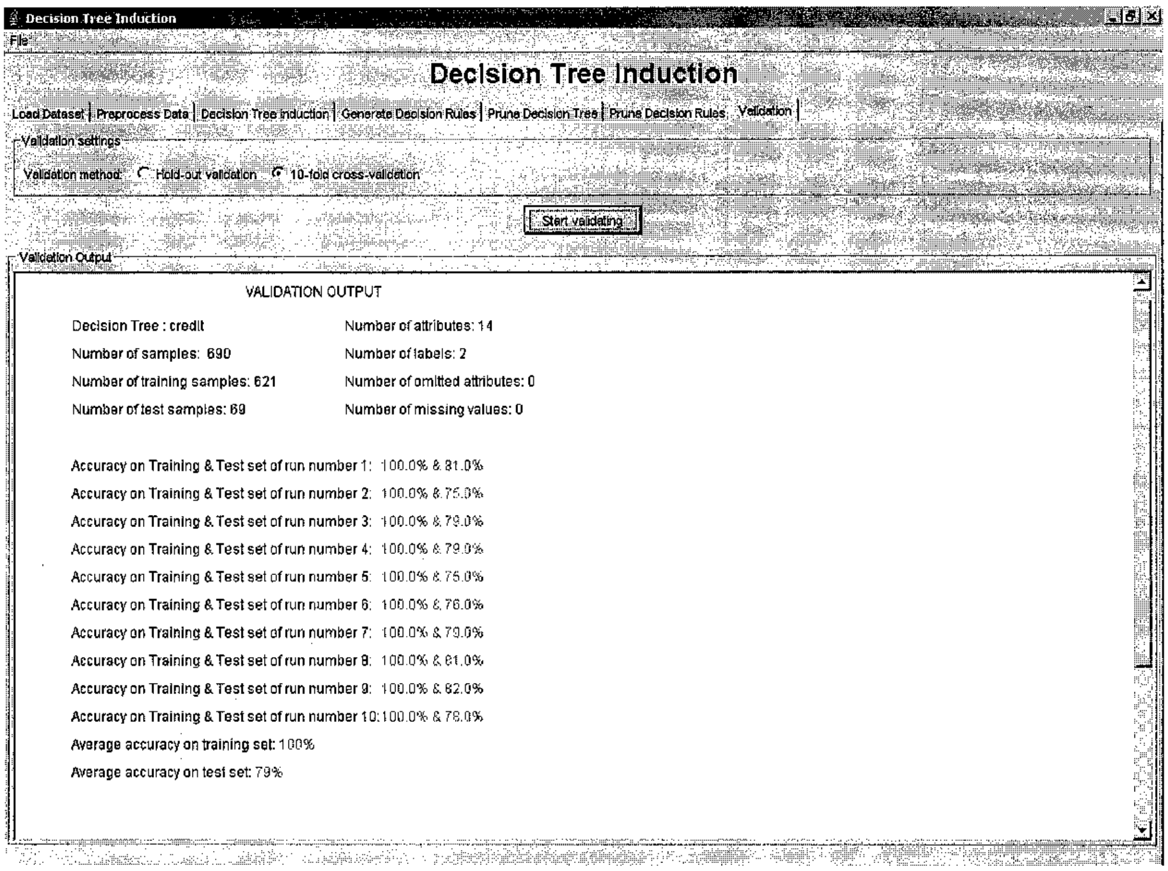
Table 4.2: Unpruned and Pruned Trees' accuracy comparison

Accuracy on	**Training set**	**Test set**
Unpruned tree	100%	74%
Pruned tree	86%	85%

The result shows that the unpruned tree was over-fitted to the training data, which means it performed very well (in this case 100% accuracy) on the given training data, however, on the unseen data (the test set), the tree achieved the accuracy rate of only 74%. Since the accuracy on the unseen test set is more important than the accuracy achieved on the training set, pruning is performed on the decision tree obtained. Pruning the tree helped improved its performance as it removed those branches that have expected error rate higher than that of the substituted leaf node. The pruned tree's accuracy achieved on the unseen data was 85%, which is much better than that of the unpruned tree.

- ***Validation with 10-fold cross validation method***

This validation method can be used to evaluate the overall performance of a decision tree classifier with a particular set of parameters (missing values handling methods, ratio of training set/ test set, minimum records to stop splitting a node, percentage of a major class to stop splitting a node). Each setting would result in a different decision tree with different accuracy. Therefore, to make most use of the available data, 10-fold cross validation is used on each of the settings specified by users to evaluate the overall performance of the classifier with each particular set of parameters. For example, with the current settings (65% of dataset for training set, 35% of dataset for test set, minimum records to stop splitting a node: 0, percentage of a major class to stop splitting a node: 100%), the average accuracy achieved on the test set was 79%. Figure 4.10 shows the accuracy on the training set and the test set for each run.


Decision Tree Induction
File
Decision Tree Induction
Load Dataset | Preprocess Data | Decision Tree Induction | Generate Decision Rules | Prune Decision Tree | Prune Decision Rules | Validation |
Validation settings
Start validating
Validation Output

Validation method: Hold-out validation [✓ 10-fold cross-validation]

```
VALIDATION OUTPUT

Decision Tree : credit                Number of attributes: 14
Number of samples:  690               Number of labels: 2
Number of training samples: 621       Number of omitted attributes: 0
Number of test samples: 69            Number of missing values: 0

Accuracy on Training & Test set of run number 1:  100.0% & 81.0%
Accuracy on Training & Test set of run number 2:  100.0% & 75.0%
Accuracy on Training & Test set of run number 3:  100.0% & 79.0%
Accuracy on Training & Test set of run number 4:  100.0% & 79.0%
Accuracy on Training & Test set of run number 5:  100.0% & 75.0%
Accuracy on Training & Test set of run number 6:  100.0% & 76.0%
Accuracy on Training & Test set of run number 7:  100.0% & 79.0%
Accuracy on Training & Test set of run number 8:  100.0% & 81.0%
Accuracy on Training & Test set of run number 9:  100.0% & 82.0%
Accuracy on Training & Test set of run number 10:100.0% & 78.0%
Average accuracy on training set: 100%
Average accuracy on test set: 79%
```

Figure 4.10: 10-fold Cross Validation

4.2.3 Performance Analysis

The decision tree classifier was tested on 32 set of parameters consisting of percentage of training set, percentage of test set, percentage of major class to stop splitting, and minimum of records to stop splitting on the same credit card approval dataset. The unpruned tree accuracy, pruned tree accuracy, and 10-fold validation accuracy are recorded. The training set is chosen to be at least 60% of the original population to ensure enough data is provided to find information.

The unpruned and pruned tree accuracies reflect the goodness of the outcome from the classifier running on a dataset with some parameters specified by the users. Due to the random partition of the test set from the training set, cross-validation is used to evaluate the overall performance of the classifier with each set of parameters. The following table shows different set of parameters and their corresponding accuracies:

Table 4.3: Experimental results

#	Training set	Test set	Major class per.	Min records	Unpruned tree accuracy	Pruned tree accuracy	10-fold Validation Accuracy
1	60%	40%	90	10	82%	84%	83%
2	60%	40%	100	0	79%	84%	77%
3	60%	40%	70	5	83%	83%	81%
4	60%	40%	80	5	83%	83%	82%
5	65%	35%	85	10	86%	87%	86%
6	65%	35%	100	0	82%	85%	79%
7	65%	35%	70	5	84%	84%	84%
8	65%	35%	90	10	86%	86%	83%
9	70%	30%	80	10	84%	84%	84%
10	70%	30%	100	0	77%	83%	79%
11	70%	30%	70	5	85%	85%	84%
12	70%	30%	90	10	83%	87%	87%
13	75%	25%	90	10	86%	87%	84%
14	75%	25%	100	0	80%	83%	78%
15	75%	25%	70	5	88%	88%	85%
16	75%	25%	80	10	79%	79%	84%
17	80%	20%	75	5	83%	83%	83%
18	80%	20%	100	0	81%	87%	78%
19	80%	20%	90	10	83%	85%	80%
20	80%	20%	80	10	85%	85%	83%
21	85%	15%	70	5	84%	84%	86%
22	85%	15%	90	10	82%	85%	81%
23	85%	15%	100	0	80%	86%	79%
24	85%	15%	80	10	81%	83%	85%
25	90%	10%	90	10	86%	92%	83%
26	90%	10%	100	0	79%	85%	77%
27	90%	10%	70	5	86%	86%	86%
28	90%	10%	80	10	83%	84%	80%
29	95%	5%	90	10	84%	87%	82%
30	95%	5%	100	0	82%	91%	76%
31	95%	5%	70	5	88%	88%	87%
32	95%	5%	80	10	85%	86%	84%

From Table 4.3, we can calculate the mean, min, max of the accuracy of the unpruned tree, pruned tree and 10-fold cross validation as summarized in Table 4.4:

Table 4.4: Summarized accuracies

Accuracy	Unpruned tree	Pruned tree	10-fold cross validation
Mean	**83%**	**85.28%**	**82.2%**
Min	**77%**	**79%**	**76%**
Max	**88%**	**92%**	**87%**

The results from the experiments show that:

- The decision tree classifier performs well on this dataset, with the mean accuracies achieve more than 80%
- Pruned tree generally provides better accuracy than unpruned tree (85.28% compared to 83%)
- The overall accuracy that the classifier can perform on this credit card dataset is 82.2%

The experimental results also show that tree with absolutely no pre-pruning, i.e. it stops splitting only if all records are of the same class or the node is empty, the accuracy is generally less than those with pre-pruning. This proves that the tree tends to overfit to the training data, which results in very good accuracy on the training set but less accurate on the test set containing unseen data than pruned tree. Therefore, pruned tree with its simpler structure provides better classification on this dataset.

From the data in Table 4.3, the standard deviation of 10-fold cross validation accuracy is 3.1%. From this we can calculate the 90% confidence interval of the mean accuracy, which is **[81.3%, 83.1%]**. This means that at 90% confidence level, the mean accuracy achieved by applying on this credit card approval process will fall into the range from 81.3% to 83.1%.

4.3 Social, Ethical Issues of Data Mining

The benefits of data mining have been highlighted in most of the literature about data mining. It is inevitable that data mining techniques have become powerful aids to many organizations, from government agencies and not-for-profit organizations to retail businesses. For a long time, financial and insurance companies have mined their data to detect patterns of fraudulent credit card usage, find hidden correlations between financial indicators, identify behavior patterns of risky customers, and analyze claims. Customers buying details are normally stored in all kinds of membership cards and companies can track the buying history and preference easily.

As companies push for more profits and more market share, data mining is used more frequently to find as much competitive advantage as possible. This leads to the social and ethical concerns on the use of data mining.

It would not be much problematic if the data being mined is related to weather, natural disaster. However, if the data is personal data or production data that are confidential, then it is an issue of how data should be used. Cook and Cook in (John, 2003) stated that technological advances make it possible to track in great detail what a person does in his or her personal life. With this profile of personal details comes a substantial ethical obligation to safeguard this data from disclosure to unauthorized individuals. Otherwise, organizations risk lawsuits and harm individuals in the process of using the data. The arisen situation has led to guidelines for the protection of personal data set by the Organisation of Economic Cooperation and Development (OECD) in 1980. These guidelines state that data cannot be used for any purpose other than that held out when the data were originally obtained from the individual; the reason for collecting personal data should be made clear to the individual prior to collecting it; enable an individual to refrain from providing personal data for any purpose they decide is inappropriate (Wahlstrom and Roddick, 2001). Data mining activities tend to violate these guidelines because organizations tend to maximize the amount of data they can gain, so data are normally utilized for different purposes and without full awareness of those individuals whose data were collected.

Data mining also involves with some other social and legal issue, for example, the extent that the government can intercept into daily communication channels and how accurate, reliable the patterns deduced from the data collected are. Detailed discussion of these areas is out of the scope of this project.

In summary, the use of data mining is attached to some social, ethical and legal issues. Proper handling of data would help to bring benefits to organizations and to avoid problems discussed above.

CHAPTER 5
CONCLUSION AND RECOMMENDATION

5. CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

Data mining has emerged to be a very important research area that helps organizations make good use of the tremendous amount of data they have. In the past, it was almost an impossible task to dig for information from the huge amount of data due to technology, manpower and time constraints. Data mining has unlocked those limitations. With the combination of many other research disciplines, data mining turns raw data into useful information rather than just raw, meaningless data.

Data classification is just a branch of data mining that helps making use of the full potential of the wealth of intelligence buried in data. Data classification, especially decision tree classifier, has proved to be an effective and critical tool for many organizations to conduct their business. Therefore, decision tree classifier has been the topic of research for years with the aim of enhancing its effectiveness when applying to real-life situations.

This project has demonstrated the process of decision tree induction from a sample credit card approval dataset with the typical and necessary phases of data mining process, with focus on the classification model construction. Besides building the decision tree and extracting the rules from the tree, this project also demonstrated the method of evaluating the overall performance of the classifier on that particular dataset so that users can make judgment on which decision tree to make use of. The overall results show that C4.5 algorithm has performed well on this dataset with 90%

confidence interval of the mean accuracy in the range of [81.3%; 83.1%] and provides a reliable set of rules for future use in the credit card approval process.

The application of decision tree induction tool on credit card approval process is just one of many practical fields that this tool can be applied to. It can be applied to many other areas such as insurance fraud detection, medical diagnosis (classification of cancer patients), scientific research work, targeted marketing, etc. With each of the dataset, more robust preprocessing method may be necessary to achieve a good result in classification.

5.2 Recommendations

Since this project serves as a demonstration of the basic features of a typical data mining process, there exist many other extensions to this work to make it become a more and more powerful tool.

Firstly, this tool depends on the hardware, particularly the memory and the CPU, to work successfully. Therefore, when it comes to very large dataset, the capacity of the memory and the computation power of the CPU become critical. This is because the whole dataset currently has to reside in the memory. Large dataset would require bigger memory capacity and it also requires the CPU to perform more computation during its induction process. Therefore, there should be a new design of the application such that it can interface with a database server, for example, Oracle DBMS, or Microsoft SQL Server, etc. This design would offload the whole dataset to the database server, and also the necessary computations can also be performed by the database server.

Secondly, even though this application focuses on the data mining process, it is most likely that the raw data will come from different data sources. This necessitates the process of integrating the data from different sources into one. Again, a database server would best be the tool to perform this task, for example Data Transformation Service (DTS) from Microsoft SQL Server.

Lastly, more preprocessing features, such as scaling data, converting data in date format to other useful format, etc. can be implemented to allow for higher quality data. Robust preprocessing mechanism would improve significantly the performance of the classifier.

REFERENCES

[1] Abbas H. et al: *Data Mining: A heuristic approach*, Idea Group Publishing, 2001

[2] Berghen, F.V: *Classification Trees: C4.5*, http://iridia.ulb.ac.be/~fvandenb/classification/C45.html

[3] Brandel, M.. *Spinning data into gold. ComputerWorld*, March 26, 2001 p. 67.

[4]Dilly,R.:http://www.pcc.qub.ac.uk/tec/courses/datamining/stu_notes/dm_book_1.html, 1995

[5] Freitas, A.A.: *Data mining and Knowledge Discovery with Evolutionary Algorithms*, Natural Computing Series, 2002

[6] Hall, M.: *Printing Swing Components in Java 1.2*, http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/ Swing-Tutorial-Printing.html

[7] Hamilton, H. J.: *C4.5 Tutorial*, http://www2.cs.uregina.ca/~hamilton/courses/831/notes/ml/dtrees/c4.5/tutorial.html

[8] Han, J. and Kamber, M.: *Data mining: Concepts and Techniques*, Morgan Kaufmann, 2001

[9] Holte, R. C.: *Very Simple Classification Rules Perform Well on Most Commonly Used Datasets*, Machine Learning 11:63-91, 1993

[10] Hu, M.: *Demo: Classification Programs C4.5 CBA*, http://www.cs.uic.edu/~piotr/cs594.html, 2003

[11] John W., *Data Mining: Opportunities and Challenges*, Idea Group Publishing, 2003

[12] Joshi, K. P.: http://userpages.umbc.edu/~kjoshi1/data-mine/proj_rpt.htm, 1997

[13] Last, M.: *Credit Card Approval dataset*, http://www.csee.usf.edu/~mlast/credit.dat

[14] Luger, G. L.: *Artificial Intelligence*, 4th Edition, Addison Wesley, 2001

[15] Marks, D.: *Development Methodologies Compared: Why different projects require different development methodologies*, http://www.ncycles.com/e_whi_Methodologies.htm, 2002

[16] Michalski,R.S., Carbonell, J. G., and Mitchell, T. M. (Eds.), *Machine Learning: An Artificial Intelligence Approach,* Pablo Alto: Tioga Publishing Company,1983

[17] Michalski R. et al: *Machine learning and Data Mining: Methods and Application,* John Wiley & Sons Ltd, 1998

[18] Moen, S.: *Drawing Dynamic Trees*, IEEE Software, July 1990

[19] Monash University, *Tutorial: Decision Trees: ID3*, http://www.csse.monash.edu.au/courseware/cse5230/2003/assets/images/decisiontreesTute.ps, 2003

[20]Palace,B.:http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/datamining.htm,1996

[21] Sommerville I., *Software Engineering,* 6th Edition, Addison Wesley, 2001

[22] Stergiou, C.: http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/cs11/article1.html, 1996

[23] Tu & Chung, *A new decision-tree classification algorithm of machine learning*, International conference on Tools with AI, 1992

[24] Wahlstrom, K. and Roddick, J.F: *On the Impact of Knowledge Discovery and Data Mining*, In Proc. Selected papers from the 2nd Australian Institute of Computer Ethics Conference (AICE2000), Canberra, 2001.

[25] Walpole et al: *Probability & Statistics for Engineers & Scientists*, 7th Edition, Prentice Hall, 2002

[26] Witten I.H and Frank E.: *Data Mining: Practical machine learning tools and techniques with Java Implementations*, Morgan Kaufman, 1999

[27] Witten, J. L. et al: *Systems Analysis and Design Methods*, 5th Edition, McGrawHill, 2001

[28] Yang, Q.: *Classification with Decision Tree*, Hong Kong University of science & technology.

APPENDICES

A1. Flat text file format

The input for the decision tree induction process is a flat, comma-separated text file representing the original dataset. The text file has to conform to the following format:

```
ID,Attribute1,.....,Attributen
1,1,.....,0
Value,value,.....,value
```

Whereby:

- The first column contains the id of data records, which are zero-indexed
- The first line contains the attribute names
- The second line contains either value 0 for discreet attribute or value 1 for continuous attribute
- The third line onwards contains the corresponding values of the data
- The last column contains the class labels, which are used to classify the samples from the dataset

A2. DTree class structure

DTree
+attrNames +attrTypes -attrCollection -numOfAttributes -numOfLabelAttrValues -numMissingValues -numAttrOmitted -accuracy -trainConfusionM -testConfusionM -treeGraph -dataSource -minRec -majorPer -mutableRootNode -root -rootTest -done -sleepTime -treeFrame
+DTree() +run() -buildTree() +expandTree() +prune() -recalcConfusionM(in node, in testNode) -refreshTreeGraph() +removeAttr(in nullLimit) -calcExpectedInfoNeeded(in node) -splitSubset(in node, in attrIndex, in numOfSelectedAttrValues) -splitSubsetContinuous(in node, in attrIndex, in barValue, in lowerSub, in upperSub) -calcEntropy(in node, in attrIndex) -calcEntropyContinuous(in node, in attrIndex, in position) -calcGain(in node, in attrIndex) -calcGainContinuous(in node, in attrIndex, in position) -getBestGainContinuousPos(in node, in attrIndex) -getBestGainContinuous(in node, in attrIndex) -calcSplitInfo(in node, in attrIndex) -calcGainRatio(in node, in attrIndex) -calcGainRatioContinuous(in node, in attrIndex) -alreadyUsedToExpand(in node, in attrIndex) -getMajorityLabel(in node) -getMajorityLabel(in node, in MajorAttrIndex) +getDistributionFreq(in node) -sortAttr(in node) -addToTrainConfMtx(in node) -addToTestConfMtx(in testNode, in node) +calcAccuracyHold() -expandNode(in oriNode, in node, in testNode, in animate) +getAttrIndex(in attribute, in attrValue) +loadData() +splitDataset(in percent) +sortAttrValue() -getFreqString(in distFreq) -findMajorLabelAttrIndex(in node) -calcError(in N, in fCount) -calcNodeError(in node) -calcBackedUpError(in node) -shouldPrune(in node) -pruneTree(in oriNode, in node, in testNode) -pruneSubTree(in oriNode, in node, in testNode) -getMostCommonIndex(in attrIndex) +replaceMissingValue() +removeRecWithMissingValue() -readData(in fileName)

A3. Program Code

File name: Tree.java

```
package myprojects.tree;

import javax.swing.UIManager;
import java.awt.*;

public class Tree {
  boolean packFrame = false;

  //Construct the application
  public Tree() {
    TreeFrame frame = new TreeFrame();

    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2);
    frame.setVisible(true);
  }

  //Main method
  public static void main(String[] args) {
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    new Tree();
  }
}
```

File name: TreeFrame.java

```
package myprojects.tree;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.table.*;
import javax.swing.ImageIcon;
import java.net.URL;
//import java.beans.XMLEncoder;
//import java.beans.XMLDecoder;
import java.io.*;
import java.text.*;


public class TreeFrame extends JFrame {

  JPanel contentPane;
  JTabbedPane tabPane= new JTabbedPane();
  loadDataPane loadingPane;
  preprocessPane prePane;
  trainPane trainingPane;
  genRulePane generateRulePane ;
  prunePane pruningPane;
  genPrunedRulePane generatePrunedRulePane;
  valPane validatePane;

  double loadTime;
  double preTime=0;
  double inductionTime=0;
  double ruleGenTime=0;
  double pruneTime=0;
  double prunedRuleGenTime=0;
  double validateTime=0;

  DTree myTree,myPrunedTree ;
  JTree aTree ;
  JTree pTree ;
  JLabel lblTitle ;

  boolean removeRec;
  double percentTrain;
  double percentNull;
  int minRec;
  double majorPer;
  String dataSource;

  //Construct the frame
  public TreeFrame() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      myTree = new DTree();
      myPrunedTree = new DTree();
      aTree = new JTree();
      pTree = new JTree();
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit()   {
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(new BorderLayout(5,5));
    this.setSize(new Dimension(730, 500));
    this.setTitle("Decision Tree Induction");
    loadingPane = new loadDataPane(myTree,tabPane,this);
    prePane = new preprocessPane(myTree,tabPane,this);
    trainingPane = new trainPane(myTree,tabPane,aTree,this);
    generateRulePane = new genRulePane(myTree,tabPane,aTree,this);
    pruningPane = new prunePane(myTree,myPrunedTree,tabPane,pTree,this);
    generatePrunedRulePane = new genPrunedRulePane(myPrunedTree,tabPane,pTree,this);
    validatePane = new valPane(myTree,myPrunedTree,tabPane,aTree,this);

    tabPane.addTab("Load Dataset",loadingPane);
    tabPane.addTab("Preprocess Data",prePane);
    tabPane.addTab("Decision Tree Induction",trainingPane);
    tabPane.addTab("Generate Decision Rules",generateRulePane);
    tabPane.addTab("Prune Decision Tree",pruningPane);
    tabPane.addTab("Prune Decision Rules",generatePrunedRulePane);
    tabPane.addTab("Validation",validatePane);

    myTree.treeFrame=this;
    myPrunedTree.treeFrame=this;
    for (int i=1;i<tabPane.getTabCount();i++)
        tabPane.setEnabledAt(i,false);

    JMenuBar bar= new JMenuBar();
    this.setJMenuBar(bar);

    JMenu fileMenu = new JMenu("File");
    fileMenu.setMnemonic(KeyEvent.VK_F);

    ImageIcon saveIcon = createImageIcon("save.GIF");
    JMenuItem saveItem = new JMenuItem("Save Tree As Image",saveIcon);
    saveItem.setMnemonic(KeyEvent.VK_S);
    saveItem.addActionListener(
        new ActionListener(){
                public void actionPerformed(ActionEvent e)
                {
                    if (myTree.done==false){
                        JOptionPane.showMessageDialog( null, "Cannot print before
building tree" , "Error", JOptionPane.ERROR_MESSAGE );
                    }
                    else {
                        try{
                                JFileChooser fc = new JFileChooser();
                                        int returnVal = fc.showSaveDialog(contentPane);
                                                String fileName="";
                                            if (returnVal == JFileChooser.APPROVE_OPTION)
{
                                                        File file = fc.getSelectedFile();

                                                        String s= file.getName();
                                                        int i = s.lastIndexOf('.');
                                                        if (i > 0 &&  i < s.length() - 1)
                                                                fileName
=fc.getCurrentDirectory() + "\\" + s.substring(0,i) + ".jpg";

                                                }


                                myTree.treeGraph.saveJPEG(fileName);
                                JOptionPane.showMessageDialog( null, "Decision Tree has
been saved successfully" , "Image saved", JOptionPane.INFORMATION_MESSAGE );
                        }
                        catch(Exception ex){
                                JOptionPane.showMessageDialog( null, "Please specify valid
file name" , "Error", JOptionPane.ERROR_MESSAGE );
                        }

                    }
                }
        }
    );
    fileMenu.add(saveItem);

    JMenuItem savePrunedItem = new JMenuItem("Save Pruned Tree As Image",saveIcon);
    savePrunedItem.setMnemonic(KeyEvent.VK_A);
    savePrunedItem.addActionListener(
        new ActionListener(){
                public void actionPerformed(ActionEvent e)
                {
                    if (myPrunedTree.done==false){
                        JOptionPane.showMessageDialog( null, "Cannot print before
building tree" , "Error", JOptionPane.ERROR_MESSAGE );
                    }
                    else {
                        try{
                                JFileChooser fc = new JFileChooser();
                                        int returnVal = fc.showSaveDialog(contentPane);
                                            String fileName="";
                                          if (returnVal == JFileChooser.APPROVE_OPTION)
{
                                                  File file = fc.getSelectedFile();

                                                  String s= file.getName();
                                                  int i = s.lastIndexOf('.');
                                                  if (i > 0 &&  i < s.length() - 1)
                                                      fileName
=fc.getCurrentDirectory() + "\\" + s.substring(0,i) + ".jpg";

                                            }


                                myPrunedTree.treeGraph.saveJPEG(fileName);
                                JOptionPane.showMessageDialog( null, "Pruned Decision
Tree has been saved successfully" , "Image saved", JOptionPane.INFORMATION_MESSAGE );
                        }
                        catch(Exception ex){
                                JOptionPane.showMessageDialog( null, "Please specify valid
file name" , "Error", JOptionPane.ERROR_MESSAGE );
                        }

                    }
                }
        }
    );
    fileMenu.add(savePrunedItem);

    ImageIcon printIcon = createImageIcon("print.GIF");
    JMenuItem printItem = new JMenuItem("Print Tree",printIcon);
    printItem.setMnemonic(KeyEvent.VK_P);
    printItem.addActionListener(
        new ActionListener(){
                public void actionPerformed(ActionEvent e)
                {
                    if (myTree.done==false){
                        JOptionPane.showMessageDialog( null, "Cannot print before
building tree" , "Error", JOptionPane.ERROR_MESSAGE );
                    }
                    else {
                        myTree.treeGraph.printGraph();
                    }
                }
        }
    );
    fileMenu.add(printItem);

    JMenuItem printPrunedItem = new JMenuItem("Print Pruned Tree",printIcon);
    printPrunedItem.setMnemonic(KeyEvent.VK_P);
    printPrunedItem.addActionListener(
        new ActionListener(){
                public void actionPerformed(ActionEvent e)
                {
                    if (myPrunedTree.done==false){
                        JOptionPane.showMessageDialog( null, "Cannot print before
building tree" , "Error", JOptionPane.ERROR_MESSAGE );
                    }
                    else {
                        myPrunedTree.treeGraph.printGraph();
                    }
                }
            }
        );
        fileMenu.add(printPrunedItem);

        fileMenu.addSeparator();
        JMenuItem exitItem = new JMenuItem("Exit");
        exitItem.setMnemonic(KeyEvent.VK_X);
        exitItem.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e)
                {
                        System.exit(0);
                }
            }
        );
        fileMenu.add(exitItem);
        bar.add(fileMenu);


        lblTitle = new JLabel();
            lblTitle.setFont(new java.awt.Font("SansSerif", 1, 24));
        lblTitle.setForeground(new Color(0, 0, 200));
        lblTitle.setHorizontalAlignment(SwingConstants.CENTER);
        lblTitle.setText("Decision Tree Induction");
        contentPane.add(tabPane, BorderLayout.CENTER);
        contentPane.add(lblTitle, BorderLayout.NORTH);
    }

    //Overridden so we can exit when window is closed
    protected void processWindowEvent(WindowEvent e) {
        super.processWindowEvent(e);
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
          System.exit(0);
        }
    }

        protected static ImageIcon createImageIcon(String path) {
         java.net.URL imgURL = TreeFrame.class.getResource(path);
         if (imgURL != null) {
             return new ImageIcon(imgURL);
         } else {
             System.err.println("Couldn't find file: " + path);
             return null;
         }
    }
}
```

File name: DTree.java

```
package myprojects.tree;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
import java.io.*;
import java.util.*;
import java.beans.XMLEncoder;
import java.beans.XMLDecoder;

class DTree implements Runnable {
  String []attrNames;
  int []attrTypes;
  Vector []attrCollection;
  int numOfAttributes;
  int numOfLabelAttrValues;
  int numMissingValues=0;
  int numAttrOmitted=0;
  int [][]trainConfusionM;
  int [][]testConfusionM;
  double trainAccuracy,testAccuracy;
  TreeGraph treeGraph;
  String dataSource;
  int minRec;
  double majorPer;
  DefaultMutableTreeNode mutableRootNode;
  MyTreeNode root ;
  MyTreeNode rootTest;
  volatile boolean done;
  int sleeptime;
  TreeFrame treeFrame;

  public DTree(){
        root = new MyTreeNode(0,numOfAttributes);
        rootTest = new MyTreeNode(0,numOfAttributes);
        done=false;
  }

  public void run(){
        buildTree();
  }

  private void buildTree(){
      DefaultMutableTreeNode oriNode= new DefaultMutableTreeNode(root.name);
      expandNode(oriNode,root,rootTest,true);
      mutableRootNode = oriNode;
      trainAccuracy=calcAccuracyTrain();
      testAccuracy=calcAccuracyTest();

      DefaultTreeModel myTreeModel = new DefaultTreeModel(mutableRootNode);
         treeFrame.trainingPane.aTree.setModel(myTreeModel);
         treeFrame.trainingPane.outTab.setEnabledAt(1,true);
         treeFrame.trainingPane.btnCont.setEnabled(true);
      done=true;
  }

  public void expandTree(){ //separate this out to use in validation process
         DefaultMutableTreeNode oriNode= new DefaultMutableTreeNode(root.name);
      expandNode(oriNode,root,rootTest,false);
      mutableRootNode = oriNode;
  }

  public void prune() {
         try{
             pruneTree(mutableRootNode,root,rootTest);
             DefaultTreeModel myTreeModel = new DefaultTreeModel(mutableRootNode);
             treeFrame.pruningPane.pTree.setModel(myTreeModel);
             refreshTreeGraph();
             //reset confusion matrix
```

```
                trainConfusionM= new int[numOfLabelAttrValues][numOfLabelAttrValues];
                testConfusionM=new int[numOfLabelAttrValues][numOfLabelAttrValues];
            recalcConfusionM(root,rootTest);
            trainAccuracy=calcAccuracyTrain();
        testAccuracy=calcAccuracyTest();
            done=true;
        }
      catch (Exception ex){
                JOptionPane.showMessageDialog( null, ex.getMessage() , "Pruning Error",
JOptionPane.ERROR_MESSAGE );
      }
  }

  private void recalcConfusionM(MyTreeNode node,MyTreeNode testNode){
      if (node.nodeType==2 ) return;
      if (node.nodeType==1 && node.childrenNode[0].nodeType==2) {
              addToTrainConfMtx(node);
              addToTestConfMtx(testNode,node);
              return;
      }
      if (node.childrenNode!= null && node.childrenNode[0].nodeType!=2)
        for (int i=0;i<node.childrenNode.length;i++)
              recalcConfusionM(node.childrenNode[i],testNode.childrenNode[i]);
  }

  private void refreshTreeGraph(){
          treeGraph.resize=true;
      treeGraph.np.calcNodeSize(root.gNode);
      treeGraph.setRootGNode(root.gNode);
      treeGraph.repaint();
  }

  public void removeAttr(double nullLimit){
        int numOfRecords =root.recordCollection.size();
        int []numMissing= new int[numOfAttributes-1];
        for (int i=1;i<numOfAttributes-1;i++){
              for (int j=0;j<numOfRecords;j++)
            {
              DataRecord dr= (DataRecord) root.recordCollection.elementAt(j);
              if (dr.attrIndexValue[i] == -1)
                    numMissing[i]++;
            }
        }

        for (int i=1;i<numOfAttributes-1;i++){
              double ratio= 1.0 * numMissing[i]/numOfRecords;
              if (ratio>=nullLimit && nullLimit>=0 && nullLimit<=1.0){
                    numAttrOmitted++;

                    //begin remove from recordCollection the attribute at position i
                    int count=attrCollection.length;
                    int numMoved = count - i - 1;

                    //training set
                    for (int j=0;j<numOfRecords;j++){
                          DataRecord dr= (DataRecord)
root.recordCollection.elementAt(j);
                          int[] newArray=new int[count-1];
                  System.arraycopy(dr.attrIndexValue,0,newArray,0,i);
                  System.arraycopy(dr.attrIndexValue,i+1,newArray,i,count-i-1);
                  dr.attrIndexValue=new int[count-1];
                  dr.attrIndexValue=newArray;
                    }


                    int[] newArray2=new int[count-1];
              System.arraycopy(attrTypes,0,newArray2,0,i);
              System.arraycopy(attrTypes,i+1,newArray2,i,count-i-1);
              attrTypes=new int[count-1];
              attrTypes=newArray2;
              String[] newArray3=new String[count-1];
              System.arraycopy(attrNames,0,newArray3,0,i);
              System.arraycopy(attrNames,i+1,newArray3,i,count-i-1);
             attrNames=new String[count-1];
              attrNames=newArray3;

                     Vector[] newArray1=new Vector[count-1];
              System.arraycopy(attrCollection,0,newArray1,0,i);
              System.arraycopy(attrCollection,i+1,newArray1,i,count-i-1);
              attrCollection=new Vector[count-1];
              attrCollection=newArray1;
              }
       }
       numOfAttributes=attrCollection.length;
  }

  private double calcExpectedInfoNeeded(MyTreeNode node){
    // get num of records in this collection
    int numOfRecords= node.recordCollection.size();
    if (numOfRecords==0)       return 0;

    double I=0; //variable holding the running sum of expected info needed
    double probability=0;

    for (int j=0;j< numOfLabelAttrValues;j++)
    {
      int intCount=0;
      for (int k=0;k<numOfRecords;k++)
      {
        DataRecord dr= (DataRecord) node.recordCollection.elementAt(k);
        if ( dr.attrIndexValue[numOfAttributes -1] ==  j)
          intCount++;
      }
      probability = 1. * intCount/numOfRecords;
      if (probability==0 || probability==1)
        I +=0;
      else
        I += (-1)* probability * Math.log(probability)/Math.log(2);
    }
    return I;
  }

  private Vector []splitSubset(MyTreeNode node, int attrIndex,int
numOfSelectedAttrValues){
    Vector []subset= new Vector[numOfSelectedAttrValues];
    for (int j=0;j<numOfSelectedAttrValues;j++)
       subset[j]= new Vector();

       int numOfRecords = node.recordCollection.size();
       for (int i=0; i< numOfRecords; i++) {
              DataRecord dr = (DataRecord) node.recordCollection.elementAt(i);
        for (int j=0;j<numOfSelectedAttrValues;j++){
              if (dr.attrIndexValue[attrIndex] == j )
            {
             subset[j].addElement(dr);
             break;
            }
        }
       }
    return subset;
  }

  private void splitSubsetContinuous(MyTreeNode node, int attrIndex,double
barValue,Vector lowerSub,Vector upperSub) {
    int numOfRecords = node.recordCollection.size();
    try {
           for (int i=0; i< numOfRecords; i++) {
                     DataRecord dr = (DataRecord) node.recordCollection.elementAt(i);
                     if
(Double.parseDouble(attrCollection[attrIndex].elementAt(dr.attrIndexValue[attrIndex]).to
String()) > barValue)
```

```
                    upperSub.addElement(dr);
                else
                    lowerSub.addElement(dr);
                }

        }
        catch (Exception ex){
            System.err.println(ex.getLocalizedMessage());
        }

    }

    private double calcEntropy(MyTreeNode node,int attrIndex){
        int numOfRecords= node.recordCollection.size();
        if (numOfRecords==0)      return 0;
        int numOfSelectedAttrValues = attrCollection[attrIndex].size();
        int [][]temp = new int[numOfLabelAttrValues][numOfSelectedAttrValues];

        for (int j=0;j<numOfSelectedAttrValues;j++)
        {
          for (int i=0;i<numOfLabelAttrValues;i++)
          {
            int intCount=0;
            for (int k=0;k<numOfRecords;k++)
            {
              DataRecord dr= (DataRecord) node.recordCollection.elementAt(k);
              if ( dr.attrIndexValue[attrIndex]==j && dr.attrIndexValue[numOfAttributes -1]
==i)
                intCount++;
            }
            temp[i][j]=intCount;
          }
        }

         int []tempSum = new int[numOfSelectedAttrValues];
         for (int i=0;i<numOfSelectedAttrValues;i++)
            for (int j=0;j<numOfLabelAttrValues;j++)
                tempSum[i] += temp[j][i];

         double []temp3= new double[numOfSelectedAttrValues];
         for (int i=0;i<numOfSelectedAttrValues;i++)
         {
            for (int j=0;j<numOfLabelAttrValues;j++)
            {
                double probability ;
                if (tempSum[i]==0 )
                  probability =0;
                else
                  probability = 1.*temp[j][i]/tempSum[i];

                if (probability==0 ||probability==1)
                  temp3[i] +=0;
                else
                  temp3[i] += (-1)* probability * Math.log(probability)/Math.log(2.0);
            }
         }
        double E=0.0;
        for (int i=0;i<numOfSelectedAttrValues;i++)
          E += temp3[i] * tempSum[i] / numOfRecords;

        return E;
    }

    private double calcEntropyContinuous(MyTreeNode node,int attrIndex,int position){
        int numOfRecords= node.recordCollection.size();
        if (numOfRecords==0) return 0;

        int [][]temp = new int[2][numOfLabelAttrValues];
        DataRecord drPos= (DataRecord)
root.recordCollection.elementAt(node.sortedAttr[attrIndex][position]);
```

```
    double
barValue=Double.parseDouble(attrCollection[attrIndex].elementAt(drPos.attrIndexValue[att
rIndex]).toString()) ;

        for (int k=0;k<numOfRecords;k++)
        {
          DataRecord dr= (DataRecord)
root.recordCollection.elementAt(node.sortedAttr[attrIndex][k]);
          double
value=Double.parseDouble(attrCollection[attrIndex].elementAt(dr.attrIndexValue[attrIndex
]).toString()) ;
          for (int j=0;j<numOfLabelAttrValues;j++)
          {
               if (dr.attrIndexValue[numOfAttributes-1] == j )
            {
              if (value<=barValue)
                temp[0][j]++;
              else
                temp[1][j]++;
              break;
            }
          }
        }

    int []tempSum = new int[2];
     for (int i=0;i<2;i++)
     {
        for (int j=0;j<numOfLabelAttrValues;j++)
            tempSum[i] += temp[i][j];
     }

     double []temp3= new double[2];
     for (int i=0;i<2;i++)
     {
        for (int j=0;j<numOfLabelAttrValues;j++)
        {
            double probability =0;
            if (tempSum[i]!=0 ) probability = 1.*temp[i][j]/tempSum[i];
            if (probability==0 ||probability==1)
              temp3[i] +=0;
            else
              temp3[i] += (-1)* probability * Math.log(probability)/Math.log(2.0);
        }
     }

    double E=0.0;
    for (int i=0;i<2;i++)
      E += 1. * temp3[i] * tempSum[i] / numOfRecords;

    return E;
  }

  private double calcGain(MyTreeNode node,int attrIndex){
    return (calcExpectedInfoNeeded(node) - calcEntropy(node, attrIndex));
  }

  private double calcGainContinuous(MyTreeNode node,int attrIndex,int position){
    return (calcExpectedInfoNeeded(node) - calcEntropyContinuous(node,
attrIndex,position));
  }

  private int getBestGainContinuousPos(MyTreeNode node,int attrIndex){
    int bestPos=-1;
    int numOfRecords = node.recordCollection.size();

    double max=0;
    double temp=0;
    for (int i=0;i<numOfRecords;i++)
    {
      temp=calcGainContinuous(node,attrIndex,i);
      if (temp >= max){
```

```
      max= temp;
      bestPos=i;
    }
  }
  return bestPos;
}

private double getBestGainContinuous(MyTreeNode node,int attrIndex){
  int pos=getBestGainContinuousPos(node,attrIndex);
  if (pos==-1) return 0;
  return calcGainContinuous(node,attrIndex,pos);
}

private double calcSplitInfo(MyTreeNode node,int attrIndex){
  int numOfRecords= node.recordCollection.size();
  if (numOfRecords==0)      return 0;
  int numOfSelectedAttrValues = attrCollection[attrIndex].size();
  int []temp= new int[numOfSelectedAttrValues];

  for (int i=0;i<numOfSelectedAttrValues;i++)
  {
    for(int j=0;j<numOfRecords;j++)
    {
      DataRecord dr= (DataRecord) node.recordCollection.elementAt(j);
      if (dr.attrIndexValue[attrIndex]==i)
        temp[i] +=1;
    }
  }
  double SI=0.0;
   for (int i=0;i<numOfSelectedAttrValues;i++)
   {
          double probability ;
          probability = (1.0)*temp[i]/numOfRecords;
          if (probability==0 ||probability==1)
            SI +=0;
          else
            SI += (-1)* probability * Math.log(probability)/Math.log(2);
   }
  return SI;
}

private double calcGainRatio(MyTreeNode node,int attrIndex){
  return (calcGain(node, attrIndex)/calcSplitInfo(node, attrIndex));
}

private double calcGainRatioContinuous(MyTreeNode node,int attrIndex){
  return (getBestGainContinuous(node,attrIndex)/calcSplitInfo(node, attrIndex));
}

private boolean alreadyUsedToExpand(MyTreeNode node, int attrIndex) {
      if (node.childrenNode != null) {
              if (node.expandedAttributeIndex == attrIndex)
                      return true;
      }
      if (node.parentNode == null) return false;
      return alreadyUsedToExpand(node.parentNode, attrIndex);
}


  private int findMajorLabelAttrIndex(MyTreeNode node){
    int numOfRecord = node.recordCollection.size() ;
    int []tempCount = node.freq;
    int MajorAttrIndex=0;
    int tempMax=0;
    for (int i=0;i<numOfLabelAttrValues;i++)
    {
      if (tempCount[i] > tempMax){
        MajorAttrIndex= i;
        tempMax=tempCount[i];
      }
    }
```

```
        return MajorAttrIndex;
      }


    private String getMajorityLabel(MyTreeNode node){
        int MajorAttrIndex = findMajorLabelAttrIndex(node);
        String strTemp= attrNames[numOfAttributes-1] + " = " +
attrCollection[numOfAttributes-1].elementAt(MajorAttrIndex);
        return strTemp;
    }


    private String getMajorityLabel(MyTreeNode node,int MajorAttrIndex){
          String strTemp= attrNames[numOfAttributes-1] + " = " +
attrCollection[numOfAttributes-1].elementAt(MajorAttrIndex);
          return strTemp;
    }

    //get the num of records for each of the label attribute values
    public int[] getDistributionFreq(MyTreeNode node){
      int numOfRecords = node.recordCollection.size();
      int []temp= new int[numOfLabelAttrValues];

      for (int j=0; j < numOfLabelAttrValues;j++)
      {
        for (int i=0;i<numOfRecords;i++)
        {
          DataRecord dr= (DataRecord) node.recordCollection.elementAt(i);
          if (dr.attrIndexValue[numOfAttributes-1]==j)
            temp[j] +=1;
        }
      }
      return temp;
    }

    private void sortAttr(MyTreeNode node){
            int numOfRecords= node.recordCollection.size();
            int parentNumOfRecords =node.parentNode.recordCollection.size();
            int [][]tempArray= new int[numOfAttributes -1][numOfRecords];

            for (int c=1;c<numOfAttributes-1;c++)
            {
            if (attrTypes[c]==0)
              {
                tempArray[c]=null;
                continue;
              }

              int []recID= new int[numOfRecords];
              for (int k=0;k<numOfRecords;k++)
                {
                  DataRecord dr= (DataRecord) node.recordCollection.elementAt(k);

recID[k]=Integer.parseInt(attrCollection[0].elementAt(dr.attrIndexValue[0]).toString())
;
                }

              boolean []found= new boolean[numOfRecords];
              boolean []used= new boolean[parentNumOfRecords];
              for (int l=0;l<parentNumOfRecords;l++)
                used[l]=false;
              for (int k=0;k<numOfRecords;k++)
                found[k]=false;

              int countRecord=0;
              for (int l=0;l<parentNumOfRecords;l++)
              {
                if (countRecord==numOfRecords) break;
                if (used[l]==true) continue;
                for (int k=0;k<numOfRecords;k++)
                {
```

```
                if ((node.parentNode.sortedAttr[c][l]==recID[k]) && (found[k]==false))
                {
                  tempArray[c][countRecord]=recID[k];
                  countRecord++;
                  found[k]=true;
                  used[l]=true;
                  break;
                }
              }
            }
          }
          node.sortedAttr=tempArray;
  }

  private void addToTrainConfMtx(MyTreeNode node){
        int majorIndex=findMajorLabelAttrIndex(node);
        int []freq= getDistributionFreq(node);
        for (int i=0;i<numOfLabelAttrValues;i++)
                trainConfusionM[majorIndex][i]+=freq[i];
  }

  private void addToTestConfMtx(MyTreeNode testNode,MyTreeNode node){
        int majorIndex=findMajorLabelAttrIndex(node);
        int []freq= getDistributionFreq(testNode);
        for (int i=0;i<numOfLabelAttrValues;i++)
                testConfusionM[majorIndex][i]+=freq[i];
  }

  public double calcAccuracyTest(){
        double result=0;
        int numOfCorrect=0;
        int numOfTestRec=rootTest.recordCollection.size();
        if (numOfTestRec==0) return 0;
        for (int i=0;i<numOfLabelAttrValues;i++)
                numOfCorrect += testConfusionM[i][i];

        result = Math.round(10000.0* numOfCorrect/numOfTestRec)/100;
        return result;
  }

  public double calcAccuracyTrain(){
        double result=0;
        int numOfCorrect=0;
        int numOfTrainRec=root.recordCollection.size();
        for (int i=0;i<numOfLabelAttrValues;i++)
                numOfCorrect += trainConfusionM[i][i];

        result = Math.round(10000.0* numOfCorrect/numOfTrainRec)/100;
        return result;
  }

  private void expandNode(DefaultMutableTreeNode oriNode,MyTreeNode node, MyTreeNode
testNode,boolean animate){

    if (animate) {
        try {
            Thread.sleep(sleeptime);
          }
          catch (Exception e) {}
    }

    int numOfRecords = node.recordCollection.size();
    //added on Oct 19
     //modified 30/10/2004 to cater for the case of node with empty population
    //case node splitted does not have any record--> decision is unknown
    if (numOfRecords == 0){
      addToTestConfMtx(testNode,node);    // no need to add to train matrix because node
has no record
      String tempStr=attrNames[numOfAttributes-1] + " = Unknown" ;
      node.addLeafNode(tempStr);
      //add leaf node to draw the tree
      node.childrenNode[0].gNode = new
GNode(tempStr,node.gNode,null,null,0,0,0,node.childrenNode[0]);
      node.childrenNode[0].gNode.color=Color.red;
      node.gNode.child= node.childrenNode[0].gNode;
      if (animate)  refreshTreeGraph();

      testNode.addLeafNode(tempStr);
      DefaultMutableTreeNode tempNode= new DefaultMutableTreeNode(tempStr);
      oriNode.add(tempNode);
      tempNode=null;
      return;
    }
     //check if all records are of the same class by checking of Expected Info Needed is
0
    //check if num of records is less than numRecLimit

    if ((calcExpectedInfoNeeded(node)==0) || (numOfRecords <= minRec)){
      addToTrainConfMtx(node);
      addToTestConfMtx(testNode,node);
      String tempStr=getMajorityLabel(node);
      node.addLeafNode(tempStr);

      //add leaf node to draw the tree
      node.childrenNode[0].gNode = new
GNode(tempStr,node.gNode,null,null,0,0,0,node.childrenNode[0]);
      node.childrenNode[0].gNode.color=Color.red;
      node.gNode.child= node.childrenNode[0].gNode;
      if (animate)  refreshTreeGraph();

      testNode.addLeafNode(tempStr);
      DefaultMutableTreeNode tempNodeLimit= new DefaultMutableTreeNode(tempStr);
      oriNode.add(tempNodeLimit);
      tempNodeLimit=null;
      return;
    }

    //check if one class is the most majority class--> assign the label & stop splitting
    int []nFreq= node.freq;
    double []labelPer = new double [numOfLabelAttrValues];
    for (int i=0;i <numOfLabelAttrValues;i++) {
        labelPer[i]= 1.0 * nFreq[i]/numOfRecords;
        if (labelPer[i]>= majorPer) {
                addToTrainConfMtx(node);
                addToTestConfMtx(testNode,node);
                String tempStr=getMajorityLabel(node,i);
                 node.addLeafNode(tempStr);

                 //add leaf node to draw the tree
                node.childrenNode[0].gNode = new
GNode(tempStr,node.gNode,null,null,0,0,0,node.childrenNode[0]);
                node.childrenNode[0].gNode.color=Color.red;
                node.gNode.child= node.childrenNode[0].gNode;
            if (animate) refreshTreeGraph();

                 testNode.addLeafNode(tempStr);
                 DefaultMutableTreeNode tempNodeLimit= new
DefaultMutableTreeNode(tempStr);
                 oriNode.add(tempNodeLimit);
                 tempNodeLimit=null;
                 return;
        }
    }


    //calculate to find the attribute with maximum gain ratio among those that have gain
above average
    //first get an array of indices of those attributes that are above average

    double sumGain=0;
    int numOfAttr =0;
    double gainAvg=0;
    double  []tempGainArray= new double[numOfAttributes -1];
    //an array to store the index of those gain above average,initialize all index to -1
    int []aboveAvgAttrIndex= new int[numOfAttributes -1] ;
       Arrays.fill(aboveAvgAttrIndex,-1);

    for (int i=1;i<(numOfAttributes -1);i++)
    {
      //check if the attribute is already used and its type is discreet -->skip
      //continuous attribute can be used to split again
      if (attrTypes[i]==1)
          tempGainArray[i]=getBestGainContinuous(node,i);
      else
      {
        if (alreadyUsedToExpand(node,i)) continue;
        else
          tempGainArray[i]=calcGain(node,i);
      }
      sumGain +=tempGainArray[i];
      numOfAttr +=1;
    }
    gainAvg=sumGain/numOfAttr;

    // check if attributes set left is empty
    if (gainAvg==0)
    {
      addToTrainConfMtx(node);
      addToTestConfMtx(testNode,node);
      String tempStr=getMajorityLabel(node);
      node.addLeafNode(tempStr);

       //add leaf node to draw the tree
      node.childrenNode[0].gNode = new
GNode(tempStr,node.gNode,null,null,0,0,0,node.childrenNode[0]);
      node.childrenNode[0].gNode.color=Color.red;
      node.gNode.child= node.childrenNode[0].gNode;
      if (animate) refreshTreeGraph();

      testNode.addLeafNode(getMajorityLabel(node));
      DefaultMutableTreeNode tempNodeLimit= new
DefaultMutableTreeNode(getMajorityLabel(node));
      oriNode.add(tempNodeLimit);
      tempNodeLimit=null;
      return ;
    }

    //if not empty ==> continue
    for (int i=1;i<(numOfAttributes -1);i++)
      if (tempGainArray[i] >= gainAvg) aboveAvgAttrIndex[i]=i; //above avg

    int maxGainRatioAttrIndex=-1;
    double MaxGainRatio=0;
    for (int i=1;i<(numOfAttributes -1);i++)
    {
      if ( aboveAvgAttrIndex[i] > -1)
      {
        double tempGainRatio=0;
        if (attrTypes[i]==0)
          tempGainRatio=calcGainRatio(node,i);
        else
          tempGainRatio=calcGainRatioContinuous(node,i);

        if (tempGainRatio > MaxGainRatio)
          {
            MaxGainRatio=tempGainRatio;
            maxGainRatioAttrIndex=i;
          }
      }
    }

    //if expanded by any attribute
    if (maxGainRatioAttrIndex > -1)
    {
      node.expandedAttributeIndex=maxGainRatioAttrIndex;
      int numOfSelectedAttrValues = attrCollection[maxGainRatioAttrIndex].size();

      //case DISCREET values
      if (attrTypes[maxGainRatioAttrIndex]==0)
      {
        node.childrenNode= new MyTreeNode[numOfSelectedAttrValues];
        testNode.childrenNode= new MyTreeNode[numOfSelectedAttrValues];

        Vector []subset = new Vector[numOfSelectedAttrValues];
        subset = splitSubset(node,maxGainRatioAttrIndex,numOfSelectedAttrValues);

        Vector []subsetTest = new Vector[numOfSelectedAttrValues];
        subsetTest= splitSubset(testNode,maxGainRatioAttrIndex,numOfSelectedAttrValues);

        for (int i=0;i<numOfSelectedAttrValues;i++)
        {
          MyTreeNode newNode= new MyTreeNode(1,numOfAttributes);
          node.childrenNode[i]= newNode;
          newNode.parentNode=node;
          newNode.recordCollection=subset[i];
          newNode.name= attrNames[maxGainRatioAttrIndex] + " = " +
attrCollection[maxGainRatioAttrIndex].elementAt(i) ;

                 newNode.gNode= new
GNode(newNode.name,node.gNode,null,null,0,0,0,newNode);
                 newNode.gNode.color= Color.blue;

                 if (i==0)
                   node.gNode.child=newNode.gNode;
                 else
                      node.childrenNode[i-1].gNode.sibling = newNode.gNode;

                 if (animate) refreshTreeGraph();

          sortAttr(newNode);

          MyTreeNode newTestNode= new MyTreeNode(1,numOfAttributes);
          testNode.childrenNode[i]= newTestNode;
          newTestNode.parentNode=testNode;
          newTestNode.recordCollection=subsetTest[i] ;
          newTestNode.name= attrNames[maxGainRatioAttrIndex] + " = " +
attrCollection[maxGainRatioAttrIndex].elementAt(i) ;

          int []distFreq = getDistributionFreq(newNode);

          newNode.freq= distFreq;

          DefaultMutableTreeNode tempNode= new DefaultMutableTreeNode( newNode.name +
getFreqString(distFreq));
          oriNode.add(tempNode);
          expandNode(tempNode,node.childrenNode[i],testNode.childrenNode[i],animate);

          tempNode=null;
        }
      }

      //case CONTINUOUS values
      if (attrTypes[maxGainRatioAttrIndex]==1)
      {
        //calculate the bar value to split by selecting value with max gain
        int maxGainAttrPos=getBestGainContinuousPos(node,maxGainRatioAttrIndex);
        DataRecord drMax= (DataRecord)
root.recordCollection.elementAt(node.sortedAttr[maxGainRatioAttrIndex][maxGainAttrPos]);
        double
barValue=Double.parseDouble(attrCollection[maxGainRatioAttrIndex].elementAt(drMax.attrIn
dexValue[maxGainRatioAttrIndex]).toString()) ;
               int
id=Integer.parseInt(attrCollection[0].elementAt(drMax.attrIndexValue[0]).toString()) ;
```

```
        //split the node to 2 nodes, one less than or equal, one greater than
        node.childrenNode= new MyTreeNode[2];
        testNode.childrenNode= new MyTreeNode[2];

               Vector lowerSubTest = new Vector();
        Vector upperSubTest = new Vector();
        splitSubsetContinuous(testNode, maxGainRatioAttrIndex,
barValue,lowerSubTest,upperSubTest);

        Vector lowerSub = new Vector();
        Vector upperSub = new Vector();

        for (int c=0;c<=maxGainAttrPos;c++)
        {
          DataRecord dr= (DataRecord)
root.recordCollection.elementAt(node.sortedAttr[maxGainRatioAttrIndex][c]);
          lowerSub.addElement(dr);
        }
        for (int c=maxGainAttrPos +1 ;c<numOfRecords;c++)
        {
          DataRecord dr= (DataRecord)
root.recordCollection.elementAt(node.sortedAttr[maxGainRatioAttrIndex][c]);
          upperSub.addElement(dr);
        }

        //split to node with lower bound value
          MyTreeNode newNode= new MyTreeNode(1,numOfAttributes);
          node.childrenNode[0]= newNode;
          newNode.parentNode=node;
          newNode.recordCollection=lowerSub;
          newNode.name= attrNames[maxGainRatioAttrIndex] + " <= " + barValue ;
          sortAttr(newNode);

                 newNode.gNode= new
GNode(newNode.name,node.gNode,null,null,0,0,0,newNode);
                 newNode.gNode.color= Color.blue;
                 node.gNode.child=newNode.gNode;

                 if (animate) refreshTreeGraph();

          MyTreeNode newTestNode= new MyTreeNode(1,numOfAttributes);
          testNode.childrenNode[0]= newTestNode;
          newTestNode.parentNode=testNode;
          newTestNode.recordCollection=lowerSubTest;
          newTestNode.name= attrNames[maxGainRatioAttrIndex] + " <= " + barValue ;

          int []distFreq = getDistributionFreq(newNode);

          newNode.freq= distFreq;

          DefaultMutableTreeNode tempNode= new DefaultMutableTreeNode( newNode.name
+getFreqString(distFreq));
          oriNode.add(tempNode);

          expandNode(tempNode,newNode,newTestNode,animate);

          //split to node with upper bound value
          MyTreeNode newNode2= new MyTreeNode(1,numOfAttributes);
          node.childrenNode[1]= newNode2;
          newNode2.parentNode=node;
          newNode2.recordCollection=upperSub;
          newNode2.name= attrNames[maxGainRatioAttrIndex] + " > " + barValue ;
          sortAttr(newNode2);

                 newNode2.gNode= new
GNode(newNode2.name,node.gNode,null,null,0,0,0,newNode2);
                 newNode2.gNode.color= Color.blue;
                 newNode.gNode.sibling = newNode2.gNode;
                 if (animate) refreshTreeGraph();
          MyTreeNode newTestNode2= new MyTreeNode(1,numOfAttributes);
          testNode.childrenNode[1]= newTestNode2;
          newTestNode2.parentNode=testNode;
          newTestNode2.recordCollection=upperSubTest;
          newTestNode2.name= attrNames[maxGainRatioAttrIndex] + " > " + barValue ;

          int []distFreq2 = getDistributionFreq(newNode2);

          newNode2.freq= distFreq2;

          DefaultMutableTreeNode tempNode2= new DefaultMutableTreeNode( newNode2.name +
getFreqString(distFreq2));
          oriNode.add(tempNode2);

          expandNode(tempNode2,newNode2,newTestNode2,animate);

          tempNode=null;
          tempNode2=null;
        }
    }
  }

  public int getAttrIndex(int attribute, String attrValue) {
               int index = attrCollection[attribute].indexOf(attrValue);
               if (index < 0) {
                      attrCollection[attribute].addElement(attrValue);
                      return attrCollection[attribute].size() -1;
               }
               return index;
    }

  public void loadData() throws Exception {
      int statusData = readData(dataSource);
      if (statusData <= 0) return ;
      int indStart= dataSource.lastIndexOf('\\');
      int indEnd= dataSource.indexOf('.');
      String strName= dataSource.substring(indStart+1,indEnd);
      root.name=strName;

      root.gNode=new GNode(root.name,null, null, null, 0, 0, 0, root);
      root.gNode.color=Color.black;

      treeGraph = new TreeGraph(this);
      NodePainter nodePainter= new NodePainter(treeGraph);
      treeGraph.setParentDistance(15);
      treeGraph.np=nodePainter;
    }

  public void splitDataset(double percent){
        int numOfRootRecords = root.recordCollection.size();
        int countTrain=(int) Math.round(percent*numOfRootRecords);
        int countTest=numOfRootRecords-countTrain;
        int []ind= new int[countTest];

        Hashtable ht= new Hashtable();
        Random objRandom= new Random();

        for (int i=0;i<countTest;i++){
               int temp=objRandom.nextInt(numOfRootRecords) ;
               while (ht.contains(new Integer(temp))){
                      temp=objRandom.nextInt(numOfRootRecords);

               }
               ind[i]=temp;
               ht.put(new Integer(i),new Integer(temp));

        }
        DataRecord []drDelete= new DataRecord[countTest];

        for (int i=0;i<countTest;i++) {
               DataRecord dr= (DataRecord) root.recordCollection.elementAt(ind[i]);
```

```
                drDelete[i]=dr;
                int []tempAttrIndex= new int[numOfAttributes];
                tempAttrIndex= dr.attrIndexValue;
                DataRecord drNew = new DataRecord(numOfAttributes);
                drNew.attrIndexValue=tempAttrIndex;
                rootTest.recordCollection.addElement(drNew);

        }

        for (int i=0;i<countTest;i++) {
                try{
                        root.recordCollection.remove(drDelete[i]);
                }
                catch (Exception ex) {JOptionPane.showMessageDialog( null, "split error",
"Error", JOptionPane.ERROR_MESSAGE ); }
        }

        for (int i=0;i<countTrain;i++){
                DataRecord dr= (DataRecord) root.recordCollection.elementAt(i);
                dr.attrIndexValue[0]=i;
        }

        for (int i=0;i<countTest;i++){
                DataRecord dr= (DataRecord) rootTest.recordCollection.elementAt(i);
                dr.attrIndexValue[0]=i;
        }

        numOfLabelAttrValues = attrCollection[numOfAttributes -1].size();
        root.freq=getDistributionFreq(root);
        rootTest.freq=getDistributionFreq(rootTest);

        trainConfusionM= new int[numOfLabelAttrValues][numOfLabelAttrValues];
        testConfusionM= new int[numOfLabelAttrValues][numOfLabelAttrValues];

    }

   public void sortAttrValue(){
       int numOfRootRecords = root.recordCollection.size();
       root.sortedAttr = new int[numOfAttributes -1][numOfRootRecords];
       MyUtil mySorter= new MyUtil();

       for (int i=1;i<numOfAttributes -1;i++)
       {
         if (attrTypes[i]==1)
         {
           // first populate the array with the initial value of the current numeric
attribute
           double []oriValue= new double[numOfRootRecords];
           int []ind= new int[numOfRootRecords];
           int []oriIndex= new int[numOfRootRecords];

           for (int j=0;j<numOfRootRecords;j++)
           {
             DataRecord dr= (DataRecord) root.recordCollection.elementAt(j);
             oriValue[j]=
Double.parseDouble(attrCollection[i].elementAt(dr.attrIndexValue[i]).toString()) ;
             ind[j]=j;
             oriIndex[j]=j;
           }
           mySorter.MergeSort (oriValue,0,numOfRootRecords -
1,ind,oriIndex,numOfRootRecords);
           root.sortedAttr[i]=ind;
           oriValue=null;
           ind=null;
           oriIndex=null;
         }
         else
           root.sortedAttr[i]=null;
       }
    }
```

```
     private String getFreqString(int []distFreq){
           String strFreq="(";
           for (int m=0;m<distFreq.length;m++)
             strFreq += attrCollection[numOfAttributes-1].elementAt(m).toString() + "=" +
distFreq[m] + ",";

           strFreq = strFreq.substring(0,strFreq.length()-1);
           strFreq += ")";
           return strFreq;
     }



     private double calcError (int N,int fCount){
       if (N==0) return -1;
       //N : numOfRecords  ,f: numOfError
       double z=0.69; //confidence level=25% --> z=0.69 from normal distribution
       double temp1=0;
       double temp2=0;
       double temp3=0;
       double f=0;

       f = 1.*fCount/N;
       temp2 = 1 + z*z/N;
       temp1 = f/N - f*f/N + z*z/(4*N*N);
       if (temp1 <0) return -1;
       temp3 = f + z*z/(2*N) + z*Math.sqrt(temp1);
       return temp3/temp2;
     }

     private double calcNodeError(MyTreeNode node){
       //find N: no. of records  and f: no. of errors
       int N= node.recordCollection.size();
       int []nodeFreq = node.freq;
       int f=0;

       int MaxAttrIndex = findMajorLabelAttrIndex(node);
       for (int i=0;i<nodeFreq.length;i++)
         if (i != MaxAttrIndex) f += nodeFreq[i];

       return N*calcError(N,f);
     }

     private double calcBackedUpError(MyTreeNode node){
       //return -1 for invalid cases:  leaf node, or test node that has child is leaf
node
       if (node.nodeType==2)   return -1;
       if (node.nodeType==1 && node.childrenNode[0].nodeType==2)   return -1;

       int numOfRecords= node.recordCollection.size();
       int numOfchildrenNodes= node.childrenNode.length;

       double temp=0;
       for (int i=0;i< numOfchildrenNodes; i++)
         temp += calcNodeError(node.childrenNode[i]);
       return temp;
     }

     private boolean shouldPrune(MyTreeNode node){
       if (calcBackedUpError(node)==-1) return false;
       return ( calcBackedUpError( node) > calcNodeError( node) );
     }

     private void pruneTree(DefaultMutableTreeNode oriNode,MyTreeNode node,MyTreeNode
testNode){
       if (node.nodeType==2 || (node.nodeType==1 && node.childrenNode[0].nodeType ==2))
return;
       else
       {
         for ( int i=0;i<node.childrenNode.length;i++)
         {
          DefaultMutableTreeNode tempmutableRootNode= (DefaultMutableTreeNode)
oriNode.getChildAt(i);
          pruneTree(tempmutableRootNode,node.childrenNode[i],testNode.childrenNode[i]);
        }
        pruneSubTree(oriNode,node,testNode);
      }
    }

    private void pruneSubTree(DefaultMutableTreeNode oriNode,MyTreeNode node,MyTreeNode
testNode){
      if (shouldPrune(node))
      {
        //nullify all children nodes and attach a leaf node with majority label
        node.childrenNode =null;
        oriNode.removeAllChildren();
        String tempStr=getMajorityLabel(node);
        node.addLeafNode(tempStr);

        testNode.childrenNode =null;
        testNode.addLeafNode(tempStr);

        //add leaf node to draw the tree
        node.childrenNode[0].gNode = new
GNode(tempStr,node.gNode,null,null,0,0,0,node.childrenNode[0]);
        node.childrenNode[0].gNode.color= Color.red;
        node.gNode.child= node.childrenNode[0].gNode;

        DefaultMutableTreeNode childrenNode= new DefaultMutableTreeNode(tempStr);
        oriNode.add(childrenNode);
      }
    }

        private int getMostCommonIndex(int attrIndex){
                int numOfValues=attrCollection[attrIndex].size();
                int []indexArray=new int[numOfValues] ;
                int numOfRecords= root.recordCollection.size();

                for (int i=0;i<numOfRecords;i++){
                        DataRecord dr=(DataRecord) root.recordCollection.elementAt(i);
                        int index= dr.attrIndexValue[attrIndex];
                        for (int j=0;j<numOfValues;j++){
                                if (j==index){
                                        indexArray[j]++;
                                        break;
                                }
                        }
                }

                int max=0,maxInd=0;
                for (int j=0;j<numOfValues;j++){
                        if (indexArray[j]>=max){
                                max=indexArray[j];
                                maxInd=j;
                        }
                }
                return maxInd;
        }

        public void replaceMissingValue(){
                for (int i=0;i<root.recordCollection.size();i++){
                        DataRecord dr=(DataRecord) root.recordCollection.elementAt(i);
                        for (int j=0;j<numOfAttributes;j++){
                                if (dr.attrIndexValue[j]==-1){
                                        dr.attrIndexValue[j]=getMostCommonIndex(j);
                                        numMissingValues++;
                                }


                        }
                        dr=null;
                }
        }

        public void removeRecWithMissingVal() {
                for (int i=0;i<root.recordCollection.size();i++){
                        DataRecord dr=(DataRecord) root.recordCollection.elementAt(i);
                        for (int j=0;j<numOfAttributes;j++){
                                if (dr.attrIndexValue[j]==-1){
                                        root.recordCollection.removeElementAt(i);
                                        break;
                                }


                        }
                }

                for (int i=0;i<root.recordCollection.size();i++){
                DataRecord dr= (DataRecord) root.recordCollection.elementAt(i);
                dr.attrIndexValue[0]=i;
        }
        }

        private int readData(String filename) throws Exception  {
                FileInputStream in = null;
                try {
                        File inputFile = new File(filename);
                                in = new FileInputStream(inputFile);

                }
                catch ( Exception e) {
                                JOptionPane.showMessageDialog( null, "Invalid File Name",
"Error", JOptionPane.ERROR_MESSAGE );
                                return 0;
                }

                BufferedReader bin = new BufferedReader(new InputStreamReader(in) );

                        String input;
              String input2;
                while(true) {
                        input = bin.readLine();
                                if (input.startsWith("//")) continue;
                                if (input.equals("")) continue;

                                if (input == null) {
                                        System.err.println( "No data found in the data
file: " + filename + "\n");
                                        return 0;
                                }

                     input2 = bin.readLine();
                     if (input2.startsWith("//")) continue;
                                if (input2.equals("")) continue;

                     if (input2 == null) {
                                        System.err.println( "No attribute types found in
the data file: " + filename + "\n");
                                        return 0;
                                }
                                break;
                        }

                StringTokenizer tokenizer = new StringTokenizer(input,",");
                        numOfAttributes = tokenizer.countTokens() ;
                        if (numOfAttributes < 2) {
                                System.err.println( "Read line: " + input);
                                System.err.println( "Could not obtain the names of
attributes in the line");
                                System.err.println( "Expecting at least one input
attribute and one output attribute");
                                return 0;
                        }
                        attrCollection = new Vector[numOfAttributes];
                        for (int i=0; i < numOfAttributes; i++) attrCollection[i] = new
Vector();
                        attrNames = new String[numOfAttributes];

                for (int i=0; i < numOfAttributes; i++)
                        attrNames[i]  = tokenizer.nextToken(",");

                //get attribute types
                int numOfAttrTypes=0;
                StringTokenizer tokenizer2 = new StringTokenizer(input2,",");
                        numOfAttrTypes = tokenizer2.countTokens() ;
                        if (numOfAttributes != numOfAttrTypes) {
                                System.err.println( "Read line: " + input);
                                System.err.println( "Could not obtain the types of
attributes in the line");
                                return 0;
                        }

                        attrTypes = new int[numOfAttrTypes];
                for (int i=0; i < numOfAttrTypes; i++)
                        attrTypes[i] =Integer.parseInt(tokenizer2.nextToken(",")) ;


                while(true) {
                        input = bin.readLine();
                                if (input == null) break;
                                if (input.startsWith("//")) continue;
                                if (input.equals("")) continue;

                                while (input.indexOf(",,")>-1){
                                        int pos=input.indexOf(",,");
                                        StringBuffer sb= new
StringBuffer(input.substring(0,pos+1));
                                        sb.append("?");
                                        sb.append(input.substring(pos+1,input.length()));
                                        input=sb.toString();

                                }

                                tokenizer = new StringTokenizer(input,",");
                                int numtokens = tokenizer.countTokens() ;
                                if (numtokens != numOfAttributes) {
                                        System.err.println( "Read " +
root.recordCollection.size() + " data");
                                        System.err.println( "Last line read: " + input);
                                        System.err.println( "Expecting " + numOfAttributes
+ " attributes");
                                        return 0;
                                }

                                DataRecord dr = new DataRecord(numOfAttributes);
                        for (int i=0; i < numOfAttributes; i++){
                                String tempStr=tokenizer.nextToken(",");

                                if ( tempStr.equals("?"))

                                        dr.attrIndexValue[i]= -1;
                                else
                                        dr.attrIndexValue[i]  = getAttrIndex(i,tempStr );
                        }

                                root.recordCollection.addElement(dr);
                        }
                        bin.close();
                return 1;
        }
}
```

File name: DataRecord.java

```
package myprojects.tree;


class DataRecord {
      public int []attrIndexValue;
      public DataRecord (int numOfAttributes) {
        attrIndexValue= new int[numOfAttributes];
      }
  };
```

File name: MyTreeNode.java

```
package myprojects.tree;

 import java.util.*;

 class MyTreeNode {
    public String name;
    public Vector recordCollection;
    public int expandedAttributeIndex;
    public MyTreeNode []childrenNode;
    public MyTreeNode parentNode;
    public int nodeType; //0 for root node, 1 for test node, 2 for leaf node
    public int []freq;
    int [][]sortedAttr; //contains the sorted record indices for numeric attribute
       GNode gNode; //to draw the tree

    public MyTreeNode(){
      recordCollection= new Vector();
    }

    public MyTreeNode(int type,int freqLength){
      nodeType=type;
      recordCollection= new Vector();
      if (type==0)
      {
        freq= new int[freqLength];
        parentNode=null;
      }

      if (type==1)
        freq= new int[freqLength];

      if (type==2)
      {
        childrenNode = null;
        recordCollection=null;
        freq=null;
      }
    }

    public void addLeafNode(String strName)
    {
      if (this.nodeType!=1) return;
      this.childrenNode= new MyTreeNode[1];

        MyTreeNode tempNode = new MyTreeNode(2,0);
        this.childrenNode[0] = tempNode;
        tempNode.parentNode=this;
        tempNode.name=strName;

    }
  };
```

File name: MyUtil.java

```
package myprojects.tree;

import javax.swing.*;
import javax.swing.tree.*;
import java.util.*;

public class MyUtil {

  public void MergeSort (double []A, int F, int L,int []B,int []AIndex,int MAXSIZE)
  {
    if (F <L)
    {
      int Mid= (F + L)/2;
      MergeSort(A,F,Mid,B,AIndex,MAXSIZE);
      MergeSort(A,Mid +1,L,B,AIndex,MAXSIZE);
      //merge the two halves
      Merge(A,F,Mid,L,B,AIndex,MAXSIZE);
    }
  }

  public void Merge(double []A,int F,int Mid,int L,int []B,int []AIndex,int MAXSIZE)
  {
    double []TempArray= new double[MAXSIZE];
    int First1 = F;
    int Last1= Mid;
    int First2=Mid +1;
    int Last2=L;

    int Index = First1;
    for (;(First1<=Last1) && (First2 <=Last2);++Index)
    {
      if (A[First1] < A[First2])
      {
        TempArray[Index]=A[First1];
        B[Index]=AIndex[First1];
        ++First1;
      }
      else
      {
        TempArray[Index]=A[First2];
        B[Index]=AIndex[First2];
        ++First2;
      }
    }

    for (;First1<=Last1;++First1,++Index)
    {
      TempArray[Index]=A[First1];
      B[Index]=AIndex[First1];
    }

    for (;First2<=Last2;++First2,++Index)
    {
      TempArray[Index]=A[First2];
      B[Index]=AIndex[First2];
    }

    for(Index=F;Index <=L;++Index)
    {
      A[Index]=TempArray[Index];
      AIndex[Index]=B[Index];

    }
  }

  public void parseRule(JTree tree,TreePath parent,Vector v) {
        // Traverse children
        TreeNode node = (TreeNode)parent.getLastPathComponent();
        if (node.getChildCount() > 0) {
            for (Enumeration e=node.children(); e.hasMoreElements(); ) {
                TreeNode n = (TreeNode)e.nextElement();
                TreePath path = parent.pathByAddingChild(n);
                parseRule(tree, path,v);
            }
        }
        if (node.getChildCount() == 0) {
          Object []elements= parent.getPath();
          int count= elements.length;
          String []temp=new String[count];

          for (int i=0;i<count;i++)
               temp[i]=elements[i].toString();
          /**/
          //reduce repeated conditions here
                 int repeat=0;
          for (int i=0;i<count;i++) {
               int pos=Math.max(temp[i].indexOf('>'),temp[i].indexOf('<'));
               if (pos>-1) {
                       for(int j=i+1;j<count;j++){
                               if (temp[i].regionMatches(0,temp[j],0,pos+1)){
                                       temp[i]="";
                                       repeat++;
                                       break;
                               }

                       }
               }
          }



          String []strArray= new String[count-repeat];
          int intTemp=0;
          for(int k=0;k<count;k++) {
               if (temp[k]!=""){
                       strArray[intTemp]=temp[k];
                       intTemp++;
               }
          }
          count=strArray.length;
          String output= new String("IF ");
          for (int i=1;i<count;i++)
          {

            if (i==count-1 )
              output += " THEN " + strArray[i];
            else
              if (i==1)
                output += strArray[i];
              else
                 output += " AND " + strArray[i] ;
          }
          output=simplifyString(output);
          v.addElement(output);
          temp=null;
          strArray=null;

        }
    }

public String []getRules(JTree tree)
{
  TreeNode root = (TreeNode) tree.getModel().getRoot();
  Vector v = new Vector();
  parseRule(tree,new TreePath(root),v);
  int count= v.size();
  String []result = new String[count];
  for (int i=0;i<count;i++)
    result[i]=v.elementAt(i).toString();

  return result;
  }

  public String simplifyString(String s)
  {
    StringBuffer sb= new StringBuffer(s);
    int pos1=s.indexOf('(');
    while (pos1 > -1) {
      int pos2=s.indexOf(')');
      sb.delete(pos1,pos2+1);
      s=sb.toString();
      pos1= s.indexOf('(',pos1);
    }
    return s;
  }
}
```

File name: loadDataPane.java

```
package myprojects.tree;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.util.*;
import javax.swing.table.*;
import javax.swing.ImageIcon;
import java.net.URL;

public class loadDataPane extends JPanel {
        JTable tableLoad;
        JTextField txtFileLoad= new JTextField();
        JScrollPane loadScrollPane= new JScrollPane();
        String fileName;
        JButton btnConLoad,btnSaveChange;
        Cursor hourglassCursor = new Cursor(Cursor.WAIT_CURSOR);
        Cursor normalCursor = new Cursor(Cursor.DEFAULT_CURSOR);
        DTree myTree;
        JTabbedPane tabPane;
        TreeFrame treeFrame;

        public loadDataPane(DTree aTree,JTabbedPane aTabPane,TreeFrame tf){
                Font f = new Font("Verdana",1,26);
        this.setFont(f);

                myTree = aTree;
                tabPane= aTabPane;
                treeFrame= tf;

            this.setLayout(new BorderLayout(5,5));

            JPanel headLoadPanel= new JPanel(new BorderLayout(5,5));
            headLoadPanel.setBorder(BorderFactory.createEmptyBorder(5,5,0,5));

            JPanel middleLoadPanel= new JPanel(new GridLayout(1,1));
            middleLoadPanel.setBorder(BorderFactory.createTitledBorder("Original
Dataset"));
            middleLoadPanel.add(loadScrollPane);

            JPanel bottomLoadPanel= new JPanel(new FlowLayout(FlowLayout.RIGHT));

            JLabel lblFile = new JLabel("File name:");
            lblFile.setHorizontalAlignment(SwingConstants.LEFT);



                ImageIcon openIcon = createImageIcon("open.GIF");
            JButton btnBrowse= new JButton("Open...",openIcon);
            btnBrowse.addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e)   {
                          try {
                                  getFileName();
                                  if (fileName != null){
                                        loadDataset();
                                        loadScrollPane.getViewport().add(tableLoad);
                                        btnConLoad.setEnabled(true);
                                        btnSaveChange.setEnabled(true);
                                  }
                                  else {
                                        JOptionPane.showMessageDialog( null, "Please
select a valid file name" , "Error", JOptionPane.ERROR_MESSAGE );
                                  }
                          }
                          catch (Exception ex) {
                               JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                          }
                    }
                }
            );

        headLoadPanel.add(lblFile, BorderLayout.WEST);
        headLoadPanel.add(btnBrowse, BorderLayout.EAST);
        headLoadPanel.add(txtFileLoad);

        ImageIcon saveIcon = createImageIcon("save.GIF");
            btnSaveChange= new JButton("Save changes", saveIcon);
            btnSaveChange.setEnabled(false);
            btnSaveChange.addActionListener(
                new ActionListener() {
                    public void actionPerformed(ActionEvent e)   {
                      try {
                                  saveChanges(myTree);
                      }
                      catch (Exception ex) {
                           JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                      }
                    }
                }
            );

            ImageIcon conIcon = createImageIcon("cont.GIF");
            btnConLoad= new JButton("Continue",conIcon);
            btnConLoad.setEnabled(false);
        btnConLoad.addActionListener(
                new ActionListener() {
                    public void actionPerformed(ActionEvent e)   {
                      try {
                                  int ind= tabPane.getSelectedIndex()+1;

                                  tabPane.setSelectedIndex(ind);
                                  tabPane.setEnabledAt(ind,true);
                      }
                      catch (Exception ex) {
                           JOptionPane.showMessageDialog( null, ex.getMessage() ,
"Error", JOptionPane.ERROR_MESSAGE );
                      }
                    }
                }
            );

        bottomLoadPanel.add(btnSaveChange);
        bottomLoadPanel.add(btnConLoad);

        this.add(headLoadPanel, BorderLayout.NORTH);
        this.add(bottomLoadPanel, BorderLayout.SOUTH);
        this.add(middleLoadPanel);
    }

    private void getFileName() throws Exception{
            JFileChooser fc = new JFileChooser();
            fc.addChoosableFileFilter(new MyFilter());
     int returnVal = fc.showOpenDialog(this);

            if(returnVal == JFileChooser.APPROVE_OPTION) {
              setCursor(hourglassCursor);

          File dataFile = fc.getSelectedFile();

          String strSource=fc.getCurrentDirectory() + "\\" + dataFile.getName();
          myTree.dataSource= strSource;
          treeFrame.dataSource= strSource;
              txtFileLoad.setText(strSource);
              fileName=strSource;
            }
    }
```

```
        private void loadDataset( ) throws Exception {
                 double start = System.currentTimeMillis();
                 myTree.loadData();

                 double end = System.currentTimeMillis();
          treeFrame.loadTime=(end-start)/1000;
               //prepare data to display on the table
               DefaultTableModel model = new DefaultTableModel(myTree.attrNames, 0);
               tableLoad = new JTable(model);

               //populate the data array
               int numOfRecords =myTree.root.recordCollection.size();
               String [][]data= new String[numOfRecords][myTree.numOfAttributes];
               for (int i=0;i<numOfRecords;i++)
               {
                 DataRecord dr= (DataRecord) myTree.root.recordCollection.elementAt(i);
                 for (int j=0;j<myTree.numOfAttributes;j++)
                 {
                        if (dr.attrIndexValue[j] != -1)

        data[i][j]=myTree.attrCollection[j].elementAt(dr.attrIndexValue[j]).toString();
                        else
                                data[i][j]="?";
                 }
               }
               tableLoad = new JTable(data,myTree.attrNames);
               tableLoad.setPreferredScrollableViewportSize(new Dimension(700, 70));
               setCursor(normalCursor);

        }

        private void saveChanges(DTree myTree) {
                TableModel model= tableLoad.getModel();
                int numOfRecords =myTree.root.recordCollection.size();
                int numOfAttr= myTree.numOfAttributes;

                for (int i=0;i<numOfRecords;i++){
                        DataRecord dr= (DataRecord)
myTree.root.recordCollection.elementAt(i);
                        for (int j=0;j<numOfAttr;j++){
                         String tempStr=model.getValueAt(i,j).toString();
                         if (myTree.attrTypes[j]==1) {
                                int ind=-2;
                                try {
                                      double temp=Double.parseDouble(tempStr);
                                }
                                catch (Exception e){
                                      ind=-1;
                                }
                                finally {
                                      if (ind!=-1) ind =myTree.getAttrIndex(j,tempStr);

                                      dr.attrIndexValue[j]=ind;
                                }
                         }
                         else
                               dr.attrIndexValue[j]=myTree.getAttrIndex(j,tempStr);

                    }
                }
        }

        protected static ImageIcon createImageIcon(String path) {
         java.net.URL imgURL = TreeFrame.class.getResource(path);
         if (imgURL != null) {
             return new ImageIcon(imgURL);
         } else {
             System.err.println("Couldn't find file: " + path);
             return null;
         }
    }
```

```
            class MyFilter extends javax.swing.filechooser.FileFilter {
                  public boolean accept(File file) {
                      String filename = file.getName();
                      return filename.endsWith(".txt");
                  }
                  public String getDescription() {
                      return "*.txt";
                  }
              }
    }
```

File name: preprocessPane.java

```
package myprojects.tree;


import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import javax.swing.table.*;


public class preprocessPane extends JPanel {
        JTable tableTrain,tableTest;
        ButtonGroup preGroup = new ButtonGroup();
        JRadioButton rdFrequent = new JRadioButton("Replace with most frequent
value",true);
        JRadioButton rdRemove = new JRadioButton("Remove record with missing
value",false);
    JScrollPane preScrollPanel= new JScrollPane();
    JScrollPane preScrollPane2= new JScrollPane();
    JTabbedPane outTab= new JTabbedPane();
    JButton btnConPre,btnPre;

    private String numTrain[] = {"60","65","70","75","80","85","90","95","100"};
        private String numNull[] =
{"40","45","50","55","60","65","70","75","80","85","90","95"};
        private JComboBox cboTrain= new JComboBox(numTrain);
        private JComboBox cboNull= new JComboBox(numNull);

        double percentTrain;
        double percentNull;

        Cursor hourglassCursor = new Cursor(Cursor.WAIT_CURSOR);
        Cursor normalCursor = new Cursor(Cursor.DEFAULT_CURSOR);

        DTree myTree;
        JTabbedPane tabPane;
        TreeFrame treeFrame;

        public preprocessPane(DTree aTree,JTabbedPane aTabPane,TreeFrame tf){
                Font f = new Font("Sans Serif",1,26);
        this.setFont(f);

                myTree = aTree;
                tabPane= aTabPane;
            treeFrame= tf;

            this.setLayout(new BorderLayout(5,5));

            JPanel headPanel= new JPanel();
        headPanel.setLayout(new BoxLayout(headPanel, BoxLayout.Y_AXIS));
            headPanel.setBorder(BorderFactory.createEmptyBorder(5,5,0,5));

            JPanel headPanel1= new JPanel();
        headPanel1.setLayout(new BoxLayout(headPanel1, BoxLayout.Y_AXIS));
            headPanel1.setBorder(BorderFactory.createTitledBorder("Preprocessing
settings"));

            JPanel middlePanel= new JPanel(new BorderLayout());
            middlePanel.setBorder(BorderFactory.createTitledBorder("Preprocess
Output"));
            middlePanel.add(outTab,BorderLayout.CENTER);

            JPanel bottomPanel= new JPanel(new FlowLayout(FlowLayout.RIGHT));

            JLabel lblNullPercent= new JLabel("Percentage of null values to remove
column  ");
            JLabel lblMissingVal= new JLabel("Handling missing value:  ");
            JLabel lblSplit= new JLabel("                   Percentage of sample set
used for training  ");
```

```
            preGroup.add(rdFrequent);
            preGroup.add(rdRemove);

                cboNull.setMaximumRowCount(5);
            cboNull.setSelectedIndex(4);
                lblNullPercent.setLabelFor(cboNull);

            cboTrain.setMaximumRowCount(5);
            cboTrain.setSelectedIndex(6);
            lblSplit.setLabelFor(cboTrain);

            JPanel preTemp2= new JPanel(new FlowLayout(FlowLayout.LEFT));
            preTemp2.add(lblNullPercent);
            preTemp2.add(cboNull);
            preTemp2.add(lblSplit);
            preTemp2.add(cboTrain);

            JPanel preTemp3= new JPanel(new FlowLayout(FlowLayout.LEFT));
            preTemp3.add(lblMissingVal);
            preTemp3.add(rdFrequent);
            preTemp3.add(rdRemove);

            btnPre = new JButton("Start Preprocessing");
            btnPre.addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e)  {
                          try {
                                  setCursor(hourglassCursor);
                                  percentNull=1.0*
Integer.parseInt(numNull[cboNull.getSelectedIndex()])  /100;
                                  percentTrain=1.0*
Integer.parseInt(numTrain[cboTrain.getSelectedIndex()])  /100;
                                  treeFrame.percentNull=percentNull;
                                  treeFrame.percentTrain=percentTrain;

                                  if (rdFrequent.isSelected()==true) {
                                       Preprocess(percentNull,percentTrain,false);
                                       treeFrame.removeRec=false;
                                  }

                                  else {
                                       Preprocess(percentNull,percentTrain,true);
                                       treeFrame.removeRec=true;
                                  }

                                 preScrollPane1.getViewport().add(tableTrain);
                                 preScrollPane2.getViewport().add(tableTest);

                                 outTab.add("Training set",preScrollPane1);
                                 outTab.add("Test set",preScrollPane2);

                                  setCursor(normalCursor);
                                  btnConPre.setEnabled(true);

                          }
                          catch (Exception ex) {
                               JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                          }
                        }
                    }
                );

            JPanel preTemp6 = new JPanel();
            preTemp6.add(btnPre);

            headPanel1.add(preTemp2);
            headPanel1.add(Box.createVerticalGlue());
            headPanel1.add(preTemp3);

            headPanel.add(headPanel1);
            headPanel.add(Box.createVerticalGlue());
            headPanel.add(preTemp6);

            ImageIcon conIcon = createImageIcon("cont.GIF");
            btnConPre= new JButton("Continue",conIcon);
            btnConPre.setEnabled(false);
            btnConPre.addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e)  {
                          try {

                                  int ind= tabPane.getSelectedIndex()+1;

                                  tabPane.setSelectedIndex(ind);
                                  tabPane.setEnabledAt(ind,true);
                          }
                          catch (Exception ex) {
                                JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                          }
                        }
                    }
                );
                bottomPanel.add(btnConPre);

            this.add(headPanel, BorderLayout.NORTH);
            this.add(bottomPanel, BorderLayout.SOUTH);
            this.add(middlePanel);
        }

        private void Preprocess(double percentNull,double percentTrain,boolean
removeRec){
                double start = System.currentTimeMillis();

                myTree.removeAttr(percentNull);
                if (removeRec==false)
                myTree.replaceMissingValue();
            else
                myTree.removeRecWithMissingVal();

            myTree.splitDataset(percentTrain);
            myTree.sortAttrValue();
                double end = System.currentTimeMillis();
            treeFrame.preTime=(end-start)/1000;
            ////////////////////////////////////////////////////////////////////
                //prepare Training data to display on the table
                ////////////////////////////////////////////////////////////////////
                DefaultTableModel model = new DefaultTableModel(myTree.attrNames, 0);
                tableTrain = new JTable(model);

                //populate the data array
                int numOfRecords =myTree.root.recordCollection.size();
                Object [][]data= new Object[numOfRecords][myTree.attrNames.length];
                for (int i=0;i<numOfRecords;i++){
                        DataRecord dr= (DataRecord)
myTree.root.recordCollection.elementAt(i);
                     for (int j=0;j<myTree.numOfAttributes;j++)

data[i][j]=myTree.attrCollection[j].elementAt(dr.attrIndexValue[j]);
                }

                tableTrain = new JTable(data,myTree.attrNames);
                tableTrain.setPreferredScrollableViewportSize(new Dimension(700, 70));

                ////////////////////////////////////////////////////////////////////
                //prepare Testing data to display on the table
                ////////////////////////////////////////////////////////////////////
                DefaultTableModel modelTest = new DefaultTableModel(myTree.attrNames, 0);
                tableTest = new JTable(modelTest);

                //populate the data array
                int numOfTestRecords =myTree.rootTest.recordCollection.size();
                Object [][]dataTest= new
Object[numOfTestRecords][myTree.attrNames.length];
                for (int i=0;i<numOfTestRecords;i++){
                        DataRecord drTest= (DataRecord)
myTree.rootTest.recordCollection.elementAt(i);
                    for (int j=0;j<myTree.numOfAttributes;j++)

dataTest[i][j]=myTree.attrCollection[j].elementAt(drTest.attrIndexValue[j]);
                }

                tableTest = new JTable(dataTest,myTree.attrNames);
                tableTest.setPreferredScrollableViewportSize(new Dimension(700, 70));
  }

  protected static ImageIcon createImageIcon(String path) {
        java.net.URL imgURL = TreeFrame.class.getResource(path);
        if (imgURL != null) {
            return new ImageIcon(imgURL);
        } else {
            System.err.println("Couldn't find file: " + path);
            return null;
        }
    }

}
```

File name: trainPane.java

```
package myprojects.tree;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.io.*;
import javax.swing.ImageIcon;
import java.net.URL;
import javax.swing.tree.*;

public class trainPane extends JPanel {
    private JScrollPane trainScrollPane1= new JScrollPane();
    private JScrollPane trainScrollPane2= new JScrollPane();
        private String numRec[] = {"0","1","2","3","4","5","6","7","8","9","10"};
        private String numPer[] = {"60","65","70","75","80","85","90","95","100"};
        private String strSpeed[] = {"Fast","Medium","Slow"};
        private JComboBox cboMinRec= new JComboBox(numRec);
        private JComboBox cboMajorPer= new JComboBox(numPer);
        private JComboBox cboSpeed= new JComboBox(strSpeed);
        JTabbedPane outTab= new JTabbedPane();
        JButton btnCont,btnTrain;

        Cursor hourglassCursor = new Cursor(Cursor.WAIT_CURSOR);
        Cursor normalCursor = new Cursor(Cursor.DEFAULT_CURSOR);

        DTree myTree;
        JTabbedPane tabPane;
        JTree aTree;
        TreeFrame treeFrame;

        public trainPane(DTree theTree,JTabbedPane aTabPane,JTree theJTree,TreeFrame
tf){
                myTree = theTree;
                tabPane= aTabPane;
            aTree= theJTree;
            treeFrame= tf;
            this.setLayout(new BorderLayout(5,5));

             //Set the icon for nodes.
        ImageIcon leafIcon = createImageIcon("middle.GIF");
        ImageIcon expandedIcon = createImageIcon("expanded.GIF");
        ImageIcon collapsedIcon = createImageIcon("collapsed.GIF");
        if (leafIcon != null) {
            DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer();
            renderer.setLeafIcon(leafIcon);
            renderer.setOpenIcon(expandedIcon);
            renderer.setClosedIcon(collapsedIcon);
            aTree.setCellRenderer(renderer);
        }
        else
            System.err.println("Leaf icon missing; using default.");



            cboMinRec.setMaximumRowCount(5);
            cboMinRec.setSelectedIndex(3);

            cboMajorPer.setMaximumRowCount(5);
            cboMajorPer.setSelectedIndex(6);

            cboSpeed.setMaximumRowCount(3);
            cboSpeed.setSelectedIndex(1);

            JPanel headPanel= new JPanel();
        headPanel.setLayout(new BoxLayout(headPanel, BoxLayout.Y_AXIS));
            headPanel.setBorder(BorderFactory.createEmptyBorder(5,5,0,5));

            JPanel middlePanel= new JPanel(new BorderLayout());
            middlePanel.setBorder(BorderFactory.createTitledBorder("Training Output"));

            middlePanel.add(outTab,BorderLayout.CENTER);

            JPanel bottomPanel= new JPanel(new FlowLayout(FlowLayout.RIGHT));

            JLabel lblMinRec= new JLabel("Stop splitting criteria: Minimum records ");
            lblMinRec.setLabelFor(cboMinRec);

            JLabel lblMajorPer= new JLabel(" Percentage of major class ");
            lblMajorPer.setLabelFor(cboMajorPer);

            JLabel lblSpeed= new JLabel("      Tree growing speed ");
            lblSpeed.setLabelFor(cboSpeed);

            JPanel preTempl= new JPanel(new FlowLayout(FlowLayout.LEFT));
            preTempl.setBorder(BorderFactory.createTitledBorder("Training settings"));

            preTempl.add(lblMinRec);
            preTempl.add(cboMinRec);
            preTempl.add(lblMajorPer);
            preTempl.add(cboMajorPer);
            preTempl.add(lblSpeed);
            preTempl.add(cboSpeed);

            btnTrain = new JButton("Start Training");
            btnTrain.addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e)  {
                          try {
myTree.minRec=Integer.parseInt(numRec[cboMinRec.getSelectedIndex()]) ;
                                  myTree.majorPer=1.0*
Integer.parseInt(numPer[cboMajorPer.getSelectedIndex()])  /100;
                                  treeFrame.minRec=myTree.minRec;
                                  treeFrame.majorPer =myTree.majorPer;
                                  if (cboSpeed.getSelectedIndex()==0) myTree.sleeptime=0;
                                  if (cboSpeed.getSelectedIndex()==1)
myTree.sleeptime=500;
                                  if (cboSpeed.getSelectedIndex()==2)
myTree.sleeptime=1000;

                                      setCursor(hourglassCursor);
                                      trainScrollPanel.getViewport().add(aTree);

trainScrollPane2.getViewport().add(myTree.treeGraph);
                                        outTab.add("Graphical
display",trainScrollPane2);
                                        outTab.add("Hierachical
display",trainScrollPanel);
                                        outTab.setEnabledAt(1,false);
                                        double start = System.currentTimeMillis();
                                        Random objRandom= new Random();

                                        Thread myThread= new
Thread(myTree,"training");
                                          myThread.start();
                                          double end = System.currentTimeMillis();
                                        treeFrame.inductionTime=(end-start)/1000;

                                  btnTrain.setEnabled(false);
                                  setCursor(normalCursor);
                                    btnCont.setEnabled(true);
                          }
                          catch (Exception ex) {
                              JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                          }
                        }
                    }
               );
            JPanel preTemp4 = new JPanel();
            preTemp4.add(btnTrain);

            headPanel.add(preTemp1);
            headPanel.add(Box.createVerticalGlue());
            headPanel.add(preTemp4);


            ImageIcon conIcon = createImageIcon("cont.GIF");
            btnCont= new JButton("Continue",conIcon);
            btnCont.setEnabled(false);
            btnCont.addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e)   {
                          try {

                                  int ind= tabPane.getSelectedIndex()+1;

                                  tabPane.setSelectedIndex(ind);
                                  tabPane.setEnabledAt(ind,true);
                          }
                          catch (Exception ex) {
                               JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                          }
                        }
                    }
               );
               bottomPanel.add(btnCont);

            this.add(headPanel, BorderLayout.NORTH);
            this.add(bottomPanel, BorderLayout.SOUTH);
            this.add(middlePanel);
        }

      protected static ImageIcon createImageIcon(String path) {
         java.net.URL imgURL = TreeFrame.class.getResource(path);
         if (imgURL != null) {
             return new ImageIcon(imgURL);
         } else {
             System.err.println("Couldn't find file: " + path);
             return null;
         }
    }
}
```
File name: genRulePane.java
```
package myprojects.tree;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.io.*;
import java.net.URL;


public class genRulePane extends JPanel {
    JScrollPane ruleScrollPane= new JScrollPane();
        JTextArea txtRule = new JTextArea();
        JTextField txtFile= new JTextField();
        String fileName;
        JButton btnCont;

        Cursor hourglassCursor = new Cursor(Cursor.WAIT_CURSOR);
        Cursor normalCursor = new Cursor(Cursor.DEFAULT_CURSOR);

        DTree myTree;
        JTabbedPane tabPane;
        JTree aTree;
        TreeFrame treeFrame;
        public genRulePane(DTree theTree,JTabbedPane aTabPane,JTree theJTree,TreeFrame
tf){
                myTree = theTree;
                tabPane= aTabPane;
            aTree= theJTree;
            treeFrame= tf;

            this.setLayout(new BorderLayout(5,5));

            JPanel headPanel= new JPanel();
        headPanel.setLayout(new BoxLayout(headPanel, BoxLayout.Y_AXIS));
            headPanel.setBorder(BorderFactory.createEmptyBorder(5,5,0,5));

            JPanel middlePanel= new JPanel(new BorderLayout());
            middlePanel.setBorder(BorderFactory.createTitledBorder("Decision Rules"));

            middlePanel.add(ruleScrollPane,BorderLayout.CENTER);

            JPanel bottomPanel= new JPanel(new FlowLayout(FlowLayout.RIGHT));

            JLabel lblFile= new JLabel("Save rules as ");
            lblFile.setLabelFor(txtFile);

            JButton btnBrowse= new JButton("Browse...");
            btnBrowse.addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e)   {
                          try {
                                  getFileName();

                          }
                          catch (Exception ex) {
                                JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                          }
                        }
                    }
                );

            JPanel filePanel= new JPanel(new BorderLayout(5,5));
            filePanel.add(lblFile, BorderLayout.WEST);
            filePanel.add(btnBrowse, BorderLayout.EAST);
            filePanel.add(txtFile);

            JButton btnRule = new JButton("Generate rules");
            btnRule.addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e)   {
                          try {
                                  setCursor(hourglassCursor);
                                  genRule();
                                  ruleScrollPane.getViewport().add(txtRule);
                                  if (fileName!=null)
                                        saveRules();
                                  setCursor(normalCursor);
                                  btnCont.setEnabled(true);

                          }
                          catch (Exception ex) {
                                JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                          }
                        }
                    }
                );

            JPanel preTemp = new JPanel();
            preTemp.add(btnRule);

            headPanel.add(filePanel);
            headPanel.add(Box.createVerticalGlue());
            headPanel.add(preTemp);

            ImageIcon conIcon = createImageIcon("cont.GIF");
            btnCont= new JButton("Continue",conIcon);
            btnCont.setEnabled(false);
            btnCont.addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e)  {
                          try {

                                  int ind= tabPane.getSelectedIndex()+1;

                                  tabPane.setSelectedIndex(ind);
                                  tabPane.setEnabledAt(ind,true);
                          }
                          catch (Exception ex) {
                                JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                          }
                        }
                    }
                );
                bottomPanel.add(btnCont);

            this.add(headPanel, BorderLayout.NORTH);
            this.add(bottomPanel, BorderLayout.SOUTH);
            this.add(middlePanel);
        }

        private void genRule(){
          double start = System.currentTimeMillis();
          MyUtil myUtil = new MyUtil();
       String []rules= myUtil.getRules(aTree);
       double end = System.currentTimeMillis();
       treeFrame.ruleGenTime=(end-start)/1000;
       for (int i=0; i< rules.length;i++)
         txtRule.append(rules[i] + "\n");

       txtRule.setEditable(false);
        }

        private void getFileName() throws Exception {
                JFileChooser fc = new JFileChooser();
         int returnVal = fc.showSaveDialog(this);

            if (returnVal == JFileChooser.APPROVE_OPTION) {
                        File file = fc.getSelectedFile();
                        String s= file.getName();
                        int i = s.lastIndexOf('.');
                        if (i > 0 &&  i < s.length() - 1)
                                fileName =fc.getCurrentDirectory() + "\\" +
s.substring(0,i) + ".txt";
                }

                txtFile.setText(fileName);
        }

        private void saveRules(){

            try {
                    StringReader sr = new StringReader(txtRule.getText());
                        BufferedReader br = new BufferedReader(sr);
                        FileWriter fw = new FileWriter(fileName);
                        PrintWriter pw = new PrintWriter(fw);
                        String s = br.readLine();
                        while(s != null) {
                                pw.println(s);
                                s = br.readLine();
                        }
                        br.close();
                    sr.close();
                    pw.close();
                    fw.close();
            }
            catch(IOException ioe) {
                    System.out.println( ioe.getMessage());
            }

        }

        protected static ImageIcon createImageIcon(String path) {
         java.net.URL imgURL = TreeFrame.class.getResource(path);
         if (imgURL != null) {
             return new ImageIcon(imgURL);
         } else {
             System.err.println("Couldn't find file: " + path);
             return null;
         }
    }
}
```

File name: prunePane.java

```
package myprojects.tree;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.net.URL;
import javax.swing.tree.*;
import javax.swing.ImageIcon;

public class prunePane extends JPanel {
    JScrollPane pruneScrollPane1= new JScrollPane();
    JScrollPane pruneScrollPane2= new JScrollPane();
        JTabbedPane outTab= new JTabbedPane();
        JButton btnCont,btnPrune;

        Cursor hourglassCursor = new Cursor(Cursor.WAIT_CURSOR);
        Cursor normalCursor = new Cursor(Cursor.DEFAULT_CURSOR);

        TreeFrame treeFrame;
        DTree myTree,myPrunedTree;
        JTabbedPane tabPane;
        JTree pTree;

        public prunePane(DTree theTree,DTree thePrunedTree,JTabbedPane aTabPane,JTree
theJTree,TreeFrame tf){
                myTree = theTree;
                myPrunedTree = thePrunedTree;
                tabPane= aTabPane;
            pTree= theJTree;
            treeFrame= tf;

            this.setLayout(new BorderLayout(5,5));

              //Set the icon for  nodes.
        ImageIcon leafIcon = createImageIcon("middle.GIF");
        ImageIcon expandedIcon = createImageIcon("expanded.GIF");
        ImageIcon collapsedIcon = createImageIcon("collapsed.GIF");
        if (leafIcon != null) {
            DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer();
            renderer.setLeafIcon(leafIcon);
            renderer.setOpenIcon(expandedIcon);
            renderer.setClosedIcon(collapsedIcon);
            pTree.setCellRenderer(renderer);
        }
        else
            System.err.println("Leaf icon missing; using default.");

            JPanel headPanel= new JPanel();
        headPanel.setLayout(new BoxLayout(headPanel, BoxLayout.Y_AXIS));
            headPanel.setBorder(BorderFactory.createEmptyBorder(5,5,0,5));

            JPanel middlePanel= new JPanel(new BorderLayout());
            middlePanel.setBorder(BorderFactory.createTitledBorder("Pruning Output"));

            middlePanel.add(outTab,BorderLayout.CENTER);

            JPanel bottomPanel= new JPanel(new FlowLayout(FlowLayout.RIGHT));

            btnPrune = new JButton("Start pruning");
            btnPrune.addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e)   {
                          try {
                                  setCursor(hourglassCursor);

                                 double start = System.currentTimeMillis();
                                 myPrunedTree.dataSource= treeFrame.dataSource;
                                     myPrunedTree.minRec=treeFrame.minRec;
                                         myPrunedTree.majorPer =treeFrame.majorPer;
```

```
                                    //have to copy the training and test set from
original tree
                                    //so have to do partially the job of loadData()
here and copy the two set

                                    myPrunedTree.root.name=myTree.root.name;
                                    myPrunedTree.attrCollection =new
Vector[myTree.numOfAttributes];
                             myPrunedTree.attrCollection = myTree.attrCollection;

                             myPrunedTree.attrNames = new
String[myTree.numOfAttributes];
                             myPrunedTree.attrNames=myTree.attrNames;

                             myPrunedTree.attrTypes = new
int[myTree.numOfAttributes];
                             myPrunedTree.attrTypes=myTree.attrTypes;

                             myPrunedTree.numOfAttributes=myTree.numOfAttributes;
                             myPrunedTree.numAttrOmitted=myTree.numAttrOmitted;
                             myPrunedTree.numMissingValues=myTree.numMissingValues;

myPrunedTree.numOfLabelAttrValues=myTree.numOfLabelAttrValues;

                                 myPrunedTree.root.gNode=new
GNode(myPrunedTree.root.name,null, null, null, 0, 0, 0, myPrunedTree.root);
                                 myPrunedTree.root.gNode.color=Color.black;

                                 myPrunedTree.treeGraph = new
TreeGraph(myPrunedTree);
                                 NodePainter nodePainter= new
NodePainter(myPrunedTree.treeGraph);
                                 myPrunedTree.treeGraph.setParentDistance(15);
                                 myPrunedTree.treeGraph.np=nodePainter;

                                 //copy the two sets here
                                 for (int
i=0;i<myTree.root.recordCollection.size();i++) {
                                         DataRecord dr= (DataRecord)
myTree.root.recordCollection.elementAt(i);
                                         int []tempAttrIndex= new
int[myTree.numOfAttributes];
                                         tempAttrIndex= dr.attrIndexValue;
                                         DataRecord drNew = new
DataRecord(myTree.numOfAttributes);
                                         drNew.attrIndexValue=tempAttrIndex;


        myPrunedTree.root.recordCollection.addElement(drNew);

                                   }

                                 for (int
i=0;i<myTree.rootTest.recordCollection.size();i++) {
                                         DataRecord dr= (DataRecord)
myTree.rootTest.recordCollection.elementAt(i);
                                         int []tempAttrIndex= new
int[myTree.numOfAttributes];
                                         tempAttrIndex= dr.attrIndexValue;
                                         DataRecord drNew = new
DataRecord(myTree.numOfAttributes);
                                         drNew.attrIndexValue=tempAttrIndex;


        myPrunedTree.rootTest.recordCollection.addElement(drNew);

                                   }
                                 myPrunedTree.root.freq=myTree.root.freq;
                                 myPrunedTree.rootTest.freq=myTree.rootTest.freq;
```

```
                                                myPrunedTree.trainConfusionM= new
int[myPrunedTree.numOfLabelAttrValues][myPrunedTree.numOfLabelAttrValues];
                                           myPrunedTree.testConfusionM= new
int[myPrunedTree.numOfLabelAttrValues][myPrunedTree.numOfLabelAttrValues];
                                                myPrunedTree.sortAttrValue();
                                                    myPrunedTree.expandTree();
                                                    myPrunedTree.prune();
                                                    pruneScrollPane1.getViewport().add(pTree);

pruneScrollPane2.getViewport().add(myPrunedTree.treeGraph);
                                           outTab.add("Graphical display",pruneScrollPane2);
                                           outTab.add("Hierachical display",pruneScrollPane1);
                                           double end = System.currentTimeMillis();
                                                    treeFrame.pruneTime=(end-start)/1000;

                                           setCursor(normalCursor);
                                           btnCont.setEnabled(true);
                                           btnPrune.setEnabled(false);

                                  }
                                  catch (Exception ex) {
                                        JOptionPane.showMessageDialog( null, ex.getStackTrace() ,
"Prune Pane Error", JOptionPane.ERROR_MESSAGE );
                                  }
                              }
                         }
                    );

               JPanel preTemp3 = new JPanel();
               preTemp3.add(btnPrune);
               headPanel.add(preTemp3);

               ImageIcon conIcon = createImageIcon("cont.GIF");
               btnCont= new JButton("Continue",conIcon);
               btnCont.setEnabled(false);
               btnCont.addActionListener(
                         new ActionListener() {
                              public void actionPerformed(ActionEvent e)   {
                                try {

                                           int ind= tabPane.getSelectedIndex()+1;

                                           tabPane.setSelectedIndex(ind);
                                           tabPane.setEnabledAt(ind,true);
                                  }
                                  catch (Exception ex) {
                                        JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                                  }
                              }
                         }
                    );
                    bottomPanel.add(btnCont);

               this.add(headPanel,BorderLayout.NORTH);
               this.add(bottomPanel,BorderLayout.SOUTH);
               this.add(middlePanel);
         }

         protected static ImageIcon createImageIcon(String path) {
          java.net.URL imgURL = TreeFrame.class.getResource(path);
          if (imgURL != null) {
              return new ImageIcon(imgURL);
          } else {
              System.err.println("Couldn't find file: " + path);
              return null;
          }
     }
}
```

File name: genPrunedRulePane.java

```
package myprojects.tree;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.io.*;
import java.net.URL;


public class genPrunedRulePane extends JPanel {
    JScrollPane ruleScrollPane= new JScrollPane();
        JTextArea txtRule = new JTextArea();
        JTextField txtFile= new JTextField();
        String fileName;
        JButton btnCont;

        Cursor hourglassCursor = new Cursor(Cursor.WAIT_CURSOR);
        Cursor normalCursor = new Cursor(Cursor.DEFAULT_CURSOR);

        TreeFrame treeFrame;
        DTree myTree;
        JTabbedPane tabPane;
        JTree pTree;

        public genPrunedRulePane(DTree theTree,JTabbedPane aTabPane,JTree
theJTree,TreeFrame tf){
                myTree = theTree;
            pTree= theJTree;
            tabPane= aTabPane;
            treeFrame= tf;

            this.setLayout(new BorderLayout(5,5));

            JPanel headPanel= new JPanel();
        headPanel.setLayout(new BoxLayout(headPanel, BoxLayout.Y_AXIS));
            headPanel.setBorder(BorderFactory.createEmptyBorder(5,5,0,5));

            JPanel middlePanel= new JPanel(new BorderLayout());
            middlePanel.setBorder(BorderFactory.createTitledBorder("Pruned Decision
Rules"));
            middlePanel.add(ruleScrollPane,BorderLayout.CENTER);

            JPanel bottomPanel= new JPanel(new FlowLayout(FlowLayout.RIGHT));

            JLabel lblFile= new JLabel("Save rules as ");
            lblFile.setLabelFor(txtFile);

            JButton btnBrowse= new JButton("Browse..");
            btnBrowse.addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e)   {
                          try {
                                  getFileName();

                          }
                          catch (Exception ex) {
                                JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                          }
                        }
                    }
                );

            JPanel filePanel= new JPanel(new BorderLayout(5,5));
            filePanel.add(lblFile, BorderLayout.WEST);
            filePanel.add(btnBrowse, BorderLayout.EAST);
            filePanel.add(txtFile);

            JButton btnRule = new JButton("Generate rules");
```

```
          btnRule.addActionListener(
                  new ActionListener() {
                      public void actionPerformed(ActionEvent e)   {
                        try {
                                setCursor(hourglassCursor);
                                genRule();
                               ruleScrollPane.getViewport().add(txtRule);
                               if (fileName!=null)
                                     saveRules();
                                setCursor(normalCursor);
                                btnCont.setEnabled(true);

                        }
                        catch (Exception ex) {
                             JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                        }
                      }
                  }
              );


          JPanel preTemp = new JPanel();
          preTemp.add(btnRule);

          headPanel.add(filePanel);
          headPanel.add(Box.createVerticalGlue());
          headPanel.add(preTemp);

          ImageIcon conIcon = createImageIcon("cont.GIF");
          btnCont= new JButton("Continue",conIcon);
          btnCont.setEnabled(false);
          btnCont.addActionListener(
                  new ActionListener() {
                      public void actionPerformed(ActionEvent e)   {
                        try {

                                int ind= tabPane.getSelectedIndex()+1;

                                tabPane.setSelectedIndex(ind);
                                tabPane.setEnabledAt(ind,true);
                        }
                        catch (Exception ex) {
                             JOptionPane.showMessageDialog( null, ex.toString() ,
"Error", JOptionPane.ERROR_MESSAGE );
                        }
                      }
                  }
              );
              bottomPanel.add(btnCont);

          this.add(headPanel, BorderLayout.NORTH);
          this.add(bottomPanel, BorderLayout.SOUTH);
          this.add(middlePanel);
      }

      private void genRule(){
        double start = System.currentTimeMillis();
        MyUtil myUtil = new MyUtil();
     String []rules= myUtil.getRules(pTree);
     double end = System.currentTimeMillis();
     treeFrame.prunedRuleGenTime=(end-start)/1000;
     for (int i=0; i< rules.length;i++)
       txtRule.append(rules[i] + "\n");

     txtRule.setEditable(false);
      }

      private void getFileName() throws Exception {
              JFileChooser fc = new JFileChooser();
       int returnVal = fc.showSaveDialog(this);
```

```
            if (returnVal == JFileChooser.APPROVE_OPTION) {
                    File file = fc.getSelectedFile();
                    String s= file.getName();
                    int i = s.lastIndexOf('.');
                    if (i > 0 &&  i < s.length() - 1)
                            fileName =fc.getCurrentDirectory() + "\\" +
s.substring(0,i) + ".txt";
              }

              txtFile.setText(fileName);
       }

       private void saveRules(){

           try {
                 StringReader sr = new StringReader(txtRule.getText());
                    BufferedReader br = new BufferedReader(sr);
                    FileWriter fw = new FileWriter(fileName);
                    PrintWriter pw = new PrintWriter(fw);
                    String s = br.readLine();
                    while(s != null) {
                            pw.println(s);
                            s = br.readLine();
                    }
                    br.close();
                    sr.close();
                    pw.close();
                    fw.close();
              }
              catch(IOException ioe) {
                    System.out.println( ioe.getMessage());
              }

       }

       protected static ImageIcon createImageIcon(String path) {
        java.net.URL imgURL = TreeFrame.class.getResource(path);
        if (imgURL != null) {
            return new ImageIcon(imgURL);
        } else {
            System.err.println("Couldn't find file: " + path);
            return null;
        }
    }

}
```

File name: valPane.java

```
package myprojects.tree;


import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.text.*;
import java.net.URL;
import javax.swing.tree.*;

public class valPane extends JPanel {
    JScrollPane valScrollPane= new JScrollPane();
        JFormattedTextField txtK;
        ButtonGroup rdGroup = new ButtonGroup();
        JRadioButton rdHold = new JRadioButton("Hold-out validation",true);
        JRadioButton rdCross = new JRadioButton("10-fold cross-validation ",false);

        JTextArea txtVal= new JTextArea();
        JPanel display = new JPanel();
        //errorChart eChart;
        Cursor hourglassCursor = new Cursor(Cursor.WAIT_CURSOR);
        Cursor normalCursor = new Cursor(Cursor.DEFAULT_CURSOR);

        DTree myTree,myPrunedTree;
        JTabbedPane tabPane;
        TreeFrame treeFrame;
        JTree aTree;
        //int k;
        double validateTime=0;
        holdOutPane hoPane;

        public valPane(DTree theTree,DTree thePrunedTree,JTabbedPane aTabPane,JTree
theJTree,TreeFrame tf){
                myTree = theTree;
                myPrunedTree=thePrunedTree;
                tabPane= aTabPane;
            aTree= theJTree;
            treeFrame= tf;
            this.setLayout(new BorderLayout(5,5));

            rdGroup.add(rdHold);
            rdGroup.add(rdCross);

            NumberFormat numFormat = NumberFormat.getIntegerInstance();
            numFormat.setMaximumIntegerDigits(2);

            txtK= new JFormattedTextField(numFormat);
            txtK.setValue(new Integer(10));
            txtK.setColumns(4);

            JPanel headPanel= new JPanel();
        headPanel.setLayout(new BoxLayout(headPanel, BoxLayout.Y_AXIS));
            headPanel.setBorder(BorderFactory.createEmptyBorder(5,5,0,5));

            JPanel middlePanel= new JPanel(new BorderLayout());
            middlePanel.setBorder(BorderFactory.createTitledBorder(" Validation
Output"));
            middlePanel.add(valScrollPane,BorderLayout.CENTER);

            JPanel bottomPanel= new JPanel(new FlowLayout(FlowLayout.RIGHT));

            JLabel lblValMethod= new JLabel("Validation method: ");

            JPanel preTemp= new JPanel(new FlowLayout(FlowLayout.LEFT));
            preTemp.setBorder(BorderFactory.createTitledBorder("Validation settings"));

            preTemp.add(lblValMethod);
            preTemp.add(rdHold);
            preTemp.add(rdCross);
        //preTemp2a.add(txtK,BorderLayout.WEST);

        JButton btnval = new JButton("Start validating");
        btnval.addActionListener(
               new ActionListener() {
                   public void actionPerformed(ActionEvent e)  {
                     try {
                            setCursor(hourglassCursor);
                            if (rdHold.isSelected()==true) {
                                 //validateHold();
                                 hoPane= new holdOutPane(myTree,myPrunedTree);
                                 valScrollPane.getViewport().add(hoPane);
                            }
                            else {
                                 //Integer intK = (Integer)txtK.getValue();

                                 //k=intK.intValue() ;
                                 //validateCross(10);
                                 crossValidPane crossPane= new
crossValidPane(myTree,treeFrame,10);
                                 valScrollPane.getViewport().add(crossPane);
                                 //Thread myThread= new Thread (crossPane,"Cross
validation");
                                 //myThread.start();

                            }


                            setCursor(normalCursor);

                     }
                     catch (Exception ex) {
                            JOptionPane.showMessageDialog( null, ex.getMessage() ,
"Error", JOptionPane.ERROR_MESSAGE );
                     }
                   }
               }
           );

        JPanel preTemp3 = new JPanel();
        preTemp3.add(btnval);

        headPanel.add(preTemp);
        headPanel.add(Box.createVerticalGlue());
        headPanel.add(preTemp3);

        this.add(headPanel, BorderLayout.NORTH);
        this.add(bottomPanel, BorderLayout.SOUTH);
        this.add(middlePanel);
    }


}
```

File name: crossValidPane.java

```
package myprojects.tree;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class crossValidPane extends JPanel {
        private DTree myTree;
        private TreeFrame treeFrame;
        private int topOffset,leftOffset;
        private double unitVal;
        FontMetrics fontMetrics;
        int fontHeight;
        Font font;
        double []accuracyTrain;
        double []accuracyTest;
        double avgAccuracyTrain=0,avgAccuracyTest=0;
        boolean done=false;
        int k;

        public crossValidPane( DTree theTree,TreeFrame tf,int K) {
                super();
                myTree=theTree;
                treeFrame=tf;
                k=K;
                topOffset=20;
                leftOffset=50;
                fontHeight=12;
                this.setPreferredSize(new Dimension(768,700));
                this.setBackground(Color.white);
                font=new Font("Arial",Font.PLAIN,fontHeight);
                this.setFont(font);
        fontMetrics=this.getFontMetrics(font);
        validateCross();
        }


        private void validateCross() {

        //load from source again -->preprocess again
        //loop for k times --> split again by the percentage of (1-1/k)

        accuracyTrain= new double[k];
        accuracyTest= new double[k];
            //loop for k times here
            try {
                double start = System.currentTimeMillis();
                for (int i=0;i<k;i++){
                        myTree = null;
                        myTree = new DTree();
                        myTree.dataSource= treeFrame.dataSource;
                        myTree.minRec=treeFrame.minRec;
                            myTree.majorPer =treeFrame.majorPer;
                        myTree.loadData();
                        myTree.removeAttr(treeFrame.percentNull);
                        if (treeFrame.removeRec ==false)
                                myTree.replaceMissingValue();
                            else
                                myTree.removeRecWithMissingVal();
                        myTree.splitDataset(1-1.0/k);
                        myTree.sortAttrValue();
                        myTree.expandTree();
                                accuracyTrain[i]=myTree.calcAccuracyTrain();
                                accuracyTest[i]=myTree.calcAccuracyTest();
                                //System.err.println(i + " " + accuracy[i]);
                                avgAccuracyTrain +=accuracyTrain[i];
                                avgAccuracyTest +=accuracyTest[i];
                                if (i==k-1) {
                                        avgAccuracyTrain /=k;
                                        avgAccuracyTest /=k;
                                        //done=true;
                                }
                        }
                double end = System.currentTimeMillis();
                        treeFrame.validateTime = (end-start)/1000;
                }
                catch (Exception e){
                        JOptionPane.showMessageDialog( null, e.toString() , "Error",
JOptionPane.ERROR_MESSAGE );
                }
        }


        public void paint(Graphics g){
                super.paint(g);

                Dimension d = getSize();
                int pWidth = d.width;
                int pHeight = d.height;

                Graphics2D g2d = (Graphics2D) g;
        g2d.setPaint(Color.black);
        g2d.setStroke(new BasicStroke(2.0f));

        String str1= " VALIDATION OUTPUT";
        g2d.drawString(str1,(float) leftOffset+ 150,(float) topOffset+ fontHeight);

        g2d.drawString("Decision Tree : "+ myTree.root.name,leftOffset,(float)
topOffset+ 3*fontHeight +5);
        g2d.drawString("Number of samples:  "+
(myTree.root.recordCollection.size()+myTree.rootTest.recordCollection.size()),leftOffset
,(float) topOffset+ 5*fontHeight +5);
        g2d.drawString("Number of training samples: "+
myTree.root.recordCollection.size(),leftOffset,(float) topOffset+ 7*fontHeight +5);
        g2d.drawString("Number of test samples: "+
myTree.rootTest.recordCollection.size(),leftOffset,(float) topOffset+ 9*fontHeight +5);

        g2d.drawString("Number of attributes: "+ (myTree.numOfAttributes-2),leftOffset +
240,(float) topOffset+ 3*fontHeight +5);
        g2d.drawString("Number of labels: "+ myTree.numOfLabelAttrValues,leftOffset +
240,(float) topOffset+ 5*fontHeight +5);
        g2d.drawString("Number of omitted attributes: "+
myTree.numAttrOmitted,leftOffset + 240,(float) topOffset+ 7*fontHeight +5);
        g2d.drawString("Number of missing values: "+ myTree.numMissingValues,leftOffset+
240,(float) topOffset+ 9*fontHeight +5);

        //unpruned tree here

        //g2d.drawString("* Unpruned tree ",leftOffset,(float) topOffset+ 11*fontHeight
+5);

        for (int i=1;i<=k;i++){
                //if (accuracy[i-1]!=0){
                        g2d.drawString("Accuracy on Training & Test set of run number "
+i +": ",leftOffset ,(float) topOffset+ (11+ 2*i)*fontHeight +5);
                        g2d.setPaint(Color.red);
                        g2d.drawString(accuracyTrain[i-1] + "% & " + accuracyTest[i-1] +
"% ",leftOffset + fontMetrics.stringWidth("Accuracy on Training & Test set of run number
i :  ") ,(float) topOffset+ (11+ 2*i)*fontHeight +5);
                        g2d.setPaint(Color.black);
                //}
        }

        /*for (int i=k/2+1;i<=k;i++){
                //if (accuracy[i-1]!=0){
                        g2d.drawString("Accuracy on Training & Test set of run number "
+i +": ",leftOffset +300 ,(float) topOffset+ (11+ 2*(i-k/2))*fontHeight +5);
                        g2d.setPaint(Color.red);
                        g2d.drawString(accuracyTrain[i-1] + "% & " + accuracyTest[i-1] +
"% ",leftOffset + fontMetrics.stringWidth("Accuracy on Training & Test set of run number
i :  ") +240 ,(float) topOffset+ (11+ 2*(i-k/2))*fontHeight +5);
                        g2d.setPaint(Color.black);
                //}
        }*/

        //if (done==true){
                g2d.drawString("Average accuracy on training set: ",leftOffset ,(float)
topOffset+ (13+2*k)*fontHeight +5);
                g2d.setPaint(Color.red);
                g2d.drawString(Math.round(avgAccuracyTrain) + "%",leftOffset +
fontMetrics.stringWidth("Average accuracy on training set: ") ,(float) topOffset+
(13+2*k)*fontHeight +5);
                g2d.setPaint(Color.black);

                g2d.drawString("Average accuracy on test set: ",leftOffset ,(float)
topOffset+ (15+2*k)*fontHeight +5);
                g2d.setPaint(Color.red);
                g2d.drawString(Math.round(avgAccuracyTest) + "%",leftOffset +
fontMetrics.stringWidth("Average accuracy on test set: ") ,(float) topOffset+
(15+2*k)*fontHeight +5);
                g2d.setPaint(Color.black);

        /*
                int X= leftOffset ;
                int Y= topOffset+ (15+k)*fontHeight ;

                ////////////////////draw pie chart here//////////////////////
                int goodAngle=(int) Math.round(avgAccuracy * 360.0/100) ;
                int diameter=80;

                //draw frame for pie chart first

                Y += 10;
                g2d.setColor(Color.black);
                g2d.drawRect(X,Y,250,120);
                g2d.drawString("Error Chart ",X+70,Y+15);

                X += 10;
                Y += 30;
                //draw the pie chart

                g2d.setColor(Color.blue);
                g2d.fillArc(X,Y,diameter,diameter,0,goodAngle);
                g2d.setColor(Color.red);
                g2d.fillArc(X,Y,diameter,diameter,goodAngle,360-goodAngle);

                X += diameter + 20;
                        Y += 15;

                        g2d.setColor(Color.blue);
                        g2d.fillRect(X,Y,fontHeight,fontHeight);
                        g2d.setColor(Color.black);
                        g2d.drawString(Math.round(avgAccuracy)+ "% correct",X +
fontHeight,Y + fontHeight);

                        g2d.setColor(Color.red);
                        g2d.fillRect(X,Y+ fontHeight +5,fontHeight,fontHeight);
                        g2d.setColor(Color.black);
                        int temp=(int) Math.round(100.0-avgAccuracy);
                        g2d.drawString(temp + "% incorrect",X + fontHeight ,Y+
2*fontHeight +5);


                        //pruned tree here
            X -= diameter + 20;
            Y += 150;
                g2d.drawString("* Pruned tree ",X,Y);
```

```
		for (int i=1;i<=k/2;i++){
			//if (accuracy[i-1]!=0){
				g2d.drawString("Accuracy of run number " +i +":
",leftOffset ,(float) topOffset+ (11+ 2*i)*fontHeight +5);
				g2d.setPaint(Color.red);
				g2d.drawString(accuracy[i-1] + "%",leftOffset +
fontMetrics.stringWidth("Accuracy of run number i :  ") ,(float) topOffset+ (11+
2*i)*fontHeight +5);
				g2d.setPaint(Color.black);
			//}
		}

		for (int i=k/2+1;i<=k;i++){
			//if (accuracy[i-1]!=0){
				g2d.drawString("Accuracy of run number " +i +":
",leftOffset +240 ,(float) topOffset+ (11+ 2*(i-k/2))*fontHeight +5);
				g2d.setPaint(Color.red);
				g2d.drawString(accuracy[i-1]+ "%",leftOffset +
fontMetrics.stringWidth("Accuracy of run number i :  ") +240 ,(float) topOffset+ (11+
2*(i-k/2))*fontHeight +5);
				g2d.setPaint(Color.black);
			//}
		}

		//if (done==true){
			g2d.drawString("Average accuracy: ",leftOffset ,(float)
topOffset+ (13+k)*fontHeight +5);
			g2d.setPaint(Color.red);
			g2d.drawString(Math.round(avgAccuracy) + "%",leftOffset +
fontMetrics.stringWidth("Average accuracy: ") ,(float) topOffset+ (13+k)*fontHeight +5);
			g2d.setPaint(Color.black);

		    X= leftOffset ;
			Y= topOffset+ (15+k)*fontHeight ;

			////////////////////draw pie chart here//////////////////////
			goodAngle=(int) Math.round(avgAccuracy * 360.0/100) ;

			//draw frame for pie chart first

			Y += 10;
			g2d.setColor(Color.black);
			g2d.drawRect(X,Y,250,120);
			g2d.drawString("Error Chart ",X+ 70,Y +15);

			X += 10;
			Y += 30;
			//draw the pie chart

			g2d.setColor(Color.blue);
			g2d.fillArc(X,Y,diameter,diameter,0,goodAngle);
			g2d.setColor(Color.red);
			g2d.fillArc(X,Y,diameter,diameter,goodAngle,360-goodAngle);

			X += diameter + 20;
				Y += 15;

				g2d.setColor(Color.blue);
				g2d.fillRect(X,Y,fontHeight,fontHeight);
				g2d.setColor(Color.black);
				g2d.drawString(Math.round(avgAccuracy)+ "% correct",X +
fontHeight,Y + fontHeight);

				g2d.setColor(Color.red);
				g2d.fillRect(X,Y+ fontHeight +5,fontHeight,fontHeight);
				g2d.setColor(Color.black);
				int temp=(int) Math.round(100.0-avgAccuracy);
				g2d.drawString(temp + "% incorrect",X + fontHeight ,Y+
2*fontHeight +5);
				*/
	//}
```

```
          }
}
```

File name: holdOutPane.java

```
package myprojects.tree;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class holdOutPane extends JPanel {
	private DTree myTree,myPrunedTree;
	private int topOffset,leftOffset;
	private double unitVal;
	FontMetrics fontMetrics;
	int fontHeight;
	Font font;
	public holdOutPane(DTree theTree,DTree thePrunedTree) {
		super();
		myTree=theTree;
		myPrunedTree=thePrunedTree;
		topOffset=20;
		leftOffset=50;
		fontHeight=12;
		this.setPreferredSize(new Dimension(1024,1000));
		this.setBackground(Color.white);
		font=new Font("Arial",Font.PLAIN,fontHeight);
		this.setFont(font);
	fontMetrics=this.getFontMetrics(font);
	}

	public void paint(Graphics g){
		super.paint(g);

		Dimension d = getSize();
		int pWidth = d.width;
		int pHeight = d.height;

		Graphics2D g2d = (Graphics2D) g;
	 g2d.setPaint(Color.black);
	 g2d.setStroke(new BasicStroke(3.0f));

	 int size=myTree.numOfLabelAttrValues +1;
	 int cellWidth=fontMetrics.stringWidth("Class")+10;
	 int cellHeight= fontHeight + 6;

	 String str1= " VALIDATION OUTPUT";
	 g2d.drawString(str1,(float) leftOffset+ size*cellWidth,(float) topOffset+
fontHeight);

	 g2d.drawString("Decision Tree : "+ myTree.root.name,leftOffset,(float)
topOffset+ 3*fontHeight +5);
	 g2d.drawString("Number of samples:  "+
(myTree.root.recordCollection.size()+myTree.rootTest.recordCollection.size()),leftOffset
,(float) topOffset+ 5*fontHeight +5);
	 g2d.drawString("Number of training samples: "+
myTree.root.recordCollection.size(),leftOffset,(float) topOffset+ 7*fontHeight +5);
	 g2d.drawString("Number of test samples: "+
myTree.rootTest.recordCollection.size(),leftOffset,(float) topOffset+ 9*fontHeight +5);

	 g2d.drawString("Number of attributes: "+ (myTree.numOfAttributes-2),leftOffset +
size*cellWidth + 120,(float) topOffset+ 3*fontHeight +5);
	 g2d.drawString("Number of labels: "+ myTree.numOfLabelAttrValues,leftOffset +
size*cellWidth + 120,(float) topOffset+ 5*fontHeight +5);
	 g2d.drawString("Number of omitted attributes: "+
myTree.numAttrOmitted,leftOffset + size*cellWidth + 120,(float) topOffset+ 7*fontHeight
+5);
	 g2d.drawString("Number of missing values: "+ myTree.numMissingValues,leftOffset+
size*cellWidth + 120,(float) topOffset+ 9*fontHeight +5);

	 g2d.drawString("Accuracy on training set: ",leftOffset ,(float) topOffset+
11*fontHeight +5);
	 g2d.setPaint(Color.red);
```

```
        g2d.drawString(myTree.trainAccuracy + "%",leftOffset +
fontMetrics.stringWidth("Accuracy on training set: ") ,(float) topOffset+ 11*fontHeight
+5);
        g2d.setPaint(Color.black);


        g2d.drawString("Accuracy on test set: ",leftOffset + size*cellWidth +
120,(float) topOffset+ 11*fontHeight +5);
        g2d.setPaint(Color.red);
        g2d.drawString(myTree.testAccuracy + "%",leftOffset + size*cellWidth + 120 +
fontMetrics.stringWidth("Accuracy on test set: ") ,(float) topOffset+ 11*fontHeight +5);
        g2d.setPaint(Color.black);

        int X= leftOffset ;
        int Y= topOffset+ 17*fontHeight ;

        //Training set confusion matrix
        g2d.drawString("* Unpruned tree ",leftOffset,(float) topOffset+ 13*fontHeight
+5);
        g2d.drawString("Training set confusion matrix: ",leftOffset,(float) topOffset+
15*fontHeight +5);
        //draw train-matrix frame here
        g2d.setPaint(Color.blue);
        g2d.setStroke(new BasicStroke(2.0f));
        for (int i=0;i<=size;i++){
              g2d.draw(new Line2D.Double(X,Y + i*cellHeight,X+ size*cellWidth,Y +
i*cellHeight));
              g2d.draw(new Line2D.Double(X + i*cellWidth ,Y,X + i*cellWidth,Y+
size*cellHeight));
        }

        //draw the content of the matrix
        g2d.setPaint(Color.black);
        g2d.drawString("Class",X+5,Y +3 + fontHeight);
        for (int i=1;i<size;i++){
              g2d.drawString("C"+i,X+10+i*cellWidth,Y +3 + fontHeight);
        }

        for (int i=1;i<size;i++){
              g2d.drawString("C" +i,X+5,Y +3 + fontHeight+ i*cellHeight);
              for (int j=0;j<size-1;j++){
                    if (j!=i-1) g2d.setColor(Color.red);
                    else g2d.setColor(Color.black);
                    g2d.drawString(String.valueOf(myTree.trainConfusionM[i-
1][j]),X+10+(j+1)*cellWidth,Y +3 + fontHeight+ i*cellHeight);
              }
              g2d.setColor(Color.black);
        }

        X += size*cellWidth + 120;
        g2d.drawString("Test set confusion matrix: ",X,(float) topOffset+ 15*fontHeight
+5);

        //draw test-matrix frame here
        g2d.setPaint(Color.blue);
        g2d.setStroke(new BasicStroke(2.0f));
        for (int i=0;i<=size;i++){
              g2d.draw(new Line2D.Double(X,Y + i*cellHeight,X+ size*cellWidth,Y +
i*cellHeight));
              g2d.draw(new Line2D.Double(X + i*cellWidth ,Y,X + i*cellWidth,Y+
size*cellHeight));
        }

        //draw the content of the matrix
        g2d.setPaint(Color.black);
        g2d.drawString("Class",X+5,Y +3 + fontHeight);
        for (int i=1;i<size;i++){
              g2d.drawString("C"+i,X+10+i*cellWidth,Y +3 + fontHeight);
        }

        for (int i=1;i<size;i++){
                g2d.drawString("C" +i,X+5,Y +3 + fontHeight+ i*cellHeight);
                for (int j=0;j<size-1;j++){
                        if (j!=i-1) g2d.setColor(Color.red);
                        else g2d.setColor(Color.black);
                        g2d.drawString(String.valueOf(myTree.testConfusionM[i-
1][j]),X+10+(j+1)*cellWidth,Y +3 + fontHeight+ i*cellHeight);
                }
                g2d.setColor(Color.black);
        }

        X += size*cellWidth + 60;

        //draw the legend for matrix
                String strLegend= "Class: ";
                g2d.drawString(strLegend,X,(float) topOffset+ 15*fontHeight +5);
                for (int j=0;j<size-1;j++){
                        g2d.drawString("C" + (j+1) + " : " +
myTree.attrCollection[myTree.numOfAttributes-1].elementAt(j).toString(),X ,(float)
topOffset+ 15*fontHeight + 2*(j+1)*fontHeight +5);
                }

        ////////////////////draw pie chart here/////////////////////
        X -= 2*size*cellWidth + 180;
        Y += size*cellHeight + 25;

        int goodAngle=(int) Math.round(myTree.testAccuracy * 360.0/100) ;
        int diameter=80;


        //draw frame for pie chart first
        g2d.setColor(Color.black);
        g2d.drawRect(X,Y,250,120);
        g2d.drawString("Unpruned Tree Error Chart ",X + 40, Y +15);

        X += 10;
        Y += 30;
        //draw the pie chart

        g2d.setColor(Color.blue);
        g2d.fillArc(X,Y,diameter,diameter,0,goodAngle);
        g2d.setColor(Color.red);
        g2d.fillArc(X,Y,diameter,diameter,goodAngle,360-goodAngle);

        X += diameter + 20;
                Y += 15;

                g2d.setColor(Color.blue);
                g2d.fillRect(X,Y,fontHeight,fontHeight);
                g2d.setColor(Color.black);
                g2d.drawString(myTree.testAccuracy+ "% correct",X + fontHeight,Y +
fontHeight);

                g2d.setColor(Color.red);
                g2d.fillRect(X,Y+ fontHeight +5,fontHeight,fontHeight);
                g2d.setColor(Color.black);
                g2d.drawString((100-myTree.testAccuracy) + "% incorrect",X + fontHeight
,Y+ 2*fontHeight +5);



                //pruned tree here
                X -= diameter + 30;
                Y += 110;
                //Training set confusion matrix
                g2d.drawString("* Pruned tree ",leftOffset,Y-fontHeight - 5);
        g2d.drawString("Training set confusion matrix: ",leftOffset,Y);
        Y += 10;
        //draw train-matrix frame here
        g2d.setPaint(Color.blue);
        g2d.setStroke(new BasicStroke(2.0f));
        for (int i=0;i<=size;i++){
                g2d.draw(new Line2D.Double(X,Y + i*cellHeight,X+ size*cellWidth,Y +
i*cellHeight));
                g2d.draw(new Line2D.Double(X + i*cellWidth ,Y,X + i*cellWidth,Y+
size*cellHeight));
        }

        //draw the content of the matrix
        g2d.setPaint(Color.black);
        g2d.drawString("Class",X+5,Y +3 + fontHeight);
        for (int i=1;i<size;i++){
                g2d.drawString("C"+i,X+10+i*cellWidth,Y +3 + fontHeight);
        }

        for (int i=1;i<size;i++){
                g2d.drawString("C" +i,X+5,Y +3 + fontHeight+ i*cellHeight);
                for (int j=0;j<size-1;j++){
                        if (j!=i-1) g2d.setColor(Color.red);
                        else g2d.setColor(Color.black);
                        g2d.drawString(String.valueOf(myPrunedTree.trainConfusionM[i-
1][j]),X+10+(j+1)*cellWidth,Y +3 + fontHeight+ i*cellHeight);
                }
                g2d.setColor(Color.black);
        }

        X += size*cellWidth + 120;
        Y -= 10;
        g2d.drawString("Test set confusion matrix: ",X,Y);

        Y += 10;

        //draw test-matrix frame here
        g2d.setPaint(Color.blue);
        g2d.setStroke(new BasicStroke(2.0f));
        for (int i=0;i<=size;i++){
                g2d.draw(new Line2D.Double(X,Y + i*cellHeight,X+ size*cellWidth,Y +
i*cellHeight));
                g2d.draw(new Line2D.Double(X + i*cellWidth ,Y,X + i*cellWidth,Y+
size*cellHeight));
        }

        //draw the content of the matrix
        g2d.setPaint(Color.black);
        g2d.drawString("Class",X+5,Y +3 + fontHeight);
        for (int i=1;i<size;i++){
                g2d.drawString("C"+i,X+10+i*cellWidth,Y +3 + fontHeight);
        }

        for (int i=1;i<size;i++){
                g2d.drawString("C" +i,X+5,Y +3 + fontHeight+ i*cellHeight);
                for (int j=0;j<size-1;j++){
                        if (j!=i-1) g2d.setColor(Color.red);
                        else g2d.setColor(Color.black);
                        g2d.drawString(String.valueOf(myPrunedTree.testConfusionM[i-
1][j]),X+10+(j+1)*cellWidth,Y +3 + fontHeight+ i*cellHeight);
                }
                g2d.setColor(Color.black);
        }

        X += size*cellWidth + 60;

        //draw the legend for matrix --> no need for pruned tree


        ////////////////////draw pie chart here/////////////////////
        X -= 2*size*cellWidth + 180;
        Y += size*cellHeight + 35;

        goodAngle=(int) Math.round(myPrunedTree.testAccuracy * 360.0/100) ;

        //draw frame for pie chart first
        g2d.setColor(Color.black);
        g2d.drawRect(X,Y,250,120);
        g2d.drawString("Pruned Tree Error Chart ",X + 40, Y +15);

        X += 10;
        Y += 30;
        //draw the pie chart

        g2d.setColor(Color.blue);
        g2d.fillArc(X,Y,diameter,diameter,0,goodAngle);
        g2d.setColor(Color.red);
        g2d.fillArc(X,Y,diameter,diameter,goodAngle,360-goodAngle);

        X += diameter + 20;
               Y += 15;

               g2d.setColor(Color.blue);
               g2d.fillRect(X,Y,fontHeight,fontHeight);
               g2d.setColor(Color.black);
               g2d.drawString(myPrunedTree.testAccuracy+ "% correct",X + fontHeight,Y +
fontHeight);

               g2d.setColor(Color.red);
               g2d.fillRect(X,Y+ fontHeight +5,fontHeight,fontHeight);
               g2d.setColor(Color.black);
               g2d.drawString((100-myPrunedTree.testAccuracy) + "% incorrect",X +
fontHeight ,Y+ 2*fontHeight +5);
        }
}
```

File name: histogramPane.java

```
package myprojects.tree;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class histogramPane extends JPanel {
        private MyTreeNode targetNode;
        private DTree myTree;
        private int topOffset,bottomOffset,leftOffset,rightOffset,minVal,maxVal;
        private double unitVal;
        FontMetrics fontMetrics;
        int fontHeight;

        public histogramPane(MyTreeNode theNode, DTree theTree) {
                super();
                setPreferredSize(new Dimension(200,150));
         setMinimumSize(new Dimension(60, 60));
                targetNode= theNode;
                myTree=theTree;
                topOffset=20;
                leftOffset=40;
                bottomOffset=20;
                rightOffset=40;
                minVal=0;
                maxVal=0;
                fontHeight=12;
        fontMetrics=this.getFontMetrics(new Font("Verdana",Font.BOLD,fontHeight));
        }

        public void paint(Graphics g){
                super.paint(g);

                Color []color = new Color[10];
                color[0]=Color.red;
                color[1]=Color.green;
                color[2]=Color.blue;
                color[3]=Color.yellow;
                color[4]=Color.cyan;
                color[5]=Color.orange;
                color[6]=Color.magenta;
                color[7]=Color.pink;
                color[8]=Color.white;

                Dimension d = getSize();
                int pWidth = d.width;
                int pHeight = d.height;

                int numOfCol = myTree.numOfLabelAttrValues;
                int colWidth=(int) (pWidth - leftOffset - rightOffset - 40)/(2*numOfCol -
1);
                int []val = targetNode.freq;
                int numOfRecords = targetNode.recordCollection.size();

            for (int i=0;i<numOfCol;i++)
                if (val[i] > maxVal) maxVal=val[i];

            unitVal=1.0 * (pHeight - topOffset-bottomOffset)/(maxVal - minVal);

                Graphics2D g2d = (Graphics2D) g;
         g2d.setPaint(Color.black);
         g2d.setStroke(new BasicStroke(2.0f));

         //draw x,y-axis here
         g2d.draw(new Line2D.Double(leftOffset,topOffset-5,leftOffset,pHeight-
bottomOffset));
         g2d.draw(new Line2D.Double(leftOffset,pHeight-bottomOffset,pWidth-
rightOffset+5,pHeight-bottomOffset));
```

```
        //draw x,y-arrow here
        g2d.draw(new Line2D.Double(leftOffset,topOffset-5,leftOffset-3,topOffset));
        g2d.draw(new Line2D.Double(leftOffset,topOffset-5,leftOffset+3,topOffset));
        g2d.draw(new Line2D.Double(pWidth-rightOffset+5,pHeight-bottomOffset,pWidth-
rightOffset,pHeight-bottomOffset-3));
        g2d.draw(new Line2D.Double(pWidth-rightOffset+5,pHeight-bottomOffset,pWidth-
rightOffset,pHeight-bottomOffset+3));


        //draw value on y-axis here
                g.drawString(String.valueOf(maxVal),leftOffset-
fontMetrics.stringWidth(String.valueOf(maxVal)),topOffset +fontHeight );
                g.drawString(String.valueOf(minVal),leftOffset-
fontMetrics.stringWidth(String.valueOf(minVal)),pHeight-bottomOffset );
        g2d.draw(new Line2D.Double(leftOffset-1,topOffset,leftOffset+1,topOffset));

        //draw histogram here
        for (int i=0;i<numOfCol;i++){
                if (i<color.length) g.setColor(color[i]);
                else g.setColor(color[i%color.length]);
                        g.fillRect(leftOffset+20 + 2*i*colWidth,(int) Math.round(pHeight-
bottomOffset-val[i]*unitVal) ,colWidth,(int) Math.round(val[i]*unitVal) );
                }

                //draw class on x-axis here
                g2d.setPaint(Color.black);
                g2d.setStroke(new BasicStroke(3.0f));
                for (int i=0;i<numOfCol;i++){
                        String temp=myTree.attrCollection[myTree.numOfAttributes-
1].elementAt(i).toString();
                        g2d.drawString(temp, leftOffset+20 + 2*i*colWidth +
(float)colWidth/2 - (float) fontMetrics.stringWidth(temp)/2,(float) (1.0*pHeight -
bottomOffset +fontHeight +2));
                        if (val[i]>0) g2d.drawString(String.valueOf(val[i]),
leftOffset+20 + 2*i*colWidth + (float)colWidth/2 - (float)
fontMetrics.stringWidth(String.valueOf(val[i]))/2,(int) Math.round(pHeight-bottomOffset-
val[i]*unitVal  ));
                }
        }
}
```

File name: PrintUtilities.java

```
/* Reference from http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/Swing-Tutorial-
Printing.html*/

package myprojects.tree;

import java.awt.*;
import javax.swing.*;
import java.awt.print.*;


public class PrintUtilities implements Printable {
  private Component componentToBePrinted;

  public static void printComponent(Component c) {
    new PrintUtilities(c).print();
  }

  public PrintUtilities(Component componentToBePrinted) {
    this.componentToBePrinted = componentToBePrinted;
  }

  public void print() {
    PrinterJob printJob = PrinterJob.getPrinterJob();
    printJob.setPrintable(this);
    if (printJob.printDialog())
      try {
        printJob.print();
      } catch(PrinterException pe) {
        System.out.println("Error printing: " + pe);
      }
  }

  public int print(Graphics g, PageFormat pageFormat, int pageIndex) {
    if (pageIndex > 0) {
      return(NO_SUCH_PAGE);
    } else {
      Graphics2D g2d = (Graphics2D)g;
      g2d.translate(pageFormat.getImageableX(), pageFormat.getImageableY());
      disableDoubleBuffering(componentToBePrinted);
      componentToBePrinted.paint(g2d);
      enableDoubleBuffering(componentToBePrinted);
      return(PAGE_EXISTS);
    }
  }

  /** The speed and quality of printing suffers dramatically if
   *  any of the containers have double buffering turned on.
   *  So this turns if off globally.
   *  @see enableDoubleBuffering
   */
  public static void disableDoubleBuffering(Component c) {
    RepaintManager currentManager = RepaintManager.currentManager(c);
    currentManager.setDoubleBufferingEnabled(false);
  }

  /** Re-enables double buffering globally. */

  public static void enableDoubleBuffering(Component c) {
    RepaintManager currentManager = RepaintManager.currentManager(c);
    currentManager.setDoubleBufferingEnabled(true);
  }
}
```

File name: Polyline.java

```
/*Sven Moen's "Drawing Dynamic Trees" article in IEEE Software, July 1990.  */
/*Reference with modification from Decision Tree program by Pierre Geurts, August 1999
at http://www.montefiore.ulg.ac.be/~geurts/dtapplet/ */

package myprojects.tree;

public class Polyline {
    int      dx, dy;
    Polyline link;

    public Polyline(int dx, int dy, Polyline link) {
               this.dx = dx;
               this.dy = dy;
               this.link = link;
    }
};
```

File name: Polygon.java

```
/*Sven Moen's "Drawing Dynamic Trees" article in IEEE Software, July 1990.  */
/*Reference with modification from Decision Tree program by Pierre Geurts, August 1999
at http://www.montefiore.ulg.ac.be/~geurts/dtapplet/ */

package myprojects.tree;

public class Polygon {
    Polyline lowerHead, lowerTail,upperHead, upperTail;
};
```

File name: TreeGraph.java

```
/*Sven Moen's "Drawing Dynamic Trees" article in IEEE Software, July 1990.  */
/*Reference with modification from Decision Tree program by Pierre Geurts, August 1999
at http://www.montefiore.ulg.ac.be/~geurts/dtapplet/ */


package myprojects.tree;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.awt.image.BufferedImage;
import com.sun.image.codec.jpeg.*;
import javax.swing.ImageIcon;

public class TreeGraph extends JPanel {
  int defaultBorder;
  MyPainter myPainter;
  NodePainter np;
  GNode rootGNode;
  boolean resize;
  int rootX;
  MyTreeNode activeNode;
  DTree myTree;

  public TreeGraph(DTree theTree) {
    super();
    this.myTree =theTree;
    myPainter = new MyPainter();
    rootGNode = null;
    defaultBorder = 5;
    rootX = 20;
    resize = false;
    this.setPreferredSize(new Dimension(20000,20000)); //modified 30/10/04 to accomodate
bigger tree
    this.setMinimumSize(new Dimension(60,60));
    this.addMouseListener(
        new MouseAdapter() {
               public void mousePressed(MouseEvent e) {
                     int x=e.getX();
                     int y=e.getY();
                     if (rootGNode!=null) {
                              MyTreeNode activeNode= rootGNode.getActiveNode(x,y);
                              if (activeNode!=null) {
                                if (activeNode.nodeType==2)
                                     JOptionPane.showMessageDialog(
null,activeNode.name + " is a leaf node" , "Node information",
JOptionPane.INFORMATION_MESSAGE );
                                else {
                                       histogramPane histPane= new
histogramPane(activeNode,myTree);
                                       JFrame histFrame= new JFrame();
                                       histFrame.setSize(350,200);
                                       Image image =
Toolkit.getDefaultToolkit().getImage("graph.gif");
                                       //histFrame.setIconImage(image);
                                       histFrame.setTitle("Selected node histogram-
Size: " + activeNode.recordCollection.size() + " samples");
                                       Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
                                Dimension frameSize = histFrame.getSize();
                                       histFrame.setLocation((screenSize.width -
frameSize.width) / 2, (screenSize.height - frameSize.height) / 2);
                                       histFrame.getContentPane().add(histPane);
                                histFrame.show();
                                }
                              }
                     }
                   }
        }
    );
  }

  public void printGraph(){
       PrintUtilities.printComponent(this);
  }

  public void saveJPEG(String filename) {
       Dimension size = this.getSize();
       BufferedImage myImage = new BufferedImage(size.width,
size.height,BufferedImage.TYPE_3BYTE_BGR);
       Graphics2D g2 = myImage.createGraphics();
       this.paint(g2);
       try {
         OutputStream out = new FileOutputStream(filename);
         JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
         encoder.encode(myImage);
         out.close();
       }
       catch (Exception e) {
         System.out.println(e.getMessage());
       }
  }

  public void setRootGNode(GNode gNode) {
    this.rootGNode = gNode;
    resize = true;
  }

  public void setDefaultBorder(int b) {
    defaultBorder = b;
  }

  public void setParentDistance(int distance) {
    myPainter.setParentDistance(distance);
  }

  public void paint(Graphics g) {
    super.paint(g);
    Dimension d = getSize();
    g.setColor(Color.white);
    g.fillRect(0, 0, d.width, d.height);

    if(rootGNode == null) return;

    if(resize == true) {
      myPainter.layout(rootGNode);
      myPainter.setTree(rootGNode, rootX, rootX-myPainter.calcMaxOffset(rootGNode,0));
      resize = false;
    }
    myPainter.drawTree(g, rootGNode, np);
  }
};
```

File name: GNode.java

```
/*Sven Moen's "Drawing Dynamic Trees" article in IEEE Software, July 1990.  */
/*Reference with modification from Decision Tree program by Pierre Geurts, August 1999
at http://www.montefiore.ulg.ac.be/~geurts/dtapplet/ */

package myprojects.tree;

import java.awt.*;

public class GNode {
  GNode   parent, child, sibling;
  int     width, height, border;
  String  label;
  Point   pos, offset;
  Polygon contour;
  Color color;
  MyTreeNode node;
  boolean active=false;

  public GNode(String nodeLabel,GNode parentNode, GNode childNode, GNode siblingNode,
int w, int h, int b, MyTreeNode node) {
    label = nodeLabel;
    parent = parentNode;
    child = childNode;
    sibling = siblingNode;
    width = w;
    height = h;
    border = b;
    pos = new Point(0, 0);
    offset = new Point(0, 0);
    contour = new Polygon();
    this.node=node;
  }

  public boolean containsPoint(int x, int y) {
    return (((y>=pos.x) && (y<=pos.x+width)) && ((x>=pos.y) && (x<=pos.y+height)));
  }

  public MyTreeNode getActiveNode(int x,int y) {
    if (containsPoint(x,y))
      return node;
    if (child!=null && y>pos.x) {
      MyTreeNode childNode=child.getActiveNode(x,y);
      if (childNode!=null)
              return childNode;
    }
    if (sibling!=null)
      return sibling.getActiveNode(x,y);
    return null;
  }
}
```

File name: NodePainter.java

```
/*Sven Moen's "Drawing Dynamic Trees" article in IEEE Software, July 1990.  */
/*Reference with modification from Decision Tree program by Pierre Geurts, August 1999
at http://www.montefiore.ulg.ac.be/~geurts/dtapplet/ */


package myprojects.tree;

import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.geom.*;

public class NodePainter {
  JPanel panel;
  FontMetrics fontMetrics;
  int fontHeight;
  int defaultBorder=5;


  public NodePainter(JPanel panel) {
    this.panel=panel;
        fontHeight=10;
    fontMetrics=panel.getFontMetrics(new Font("Verdana",Font.BOLD,fontHeight));
  }

  public void calcNodeSize(GNode gNode) {
    gNode.width=2 * fontHeight;
    gNode.height=fontMetrics.stringWidth(gNode.label) + 10;
    gNode.border=defaultBorder;

    if (gNode.sibling !=null) {
      calcNodeSize(gNode.sibling);
    }

    if (gNode.child != null) {
      calcNodeSize(gNode.child);
    }
  }

  public void drawEdge(Graphics g, int x1, int y1, int x2,int y2) {
    Graphics2D g2d = (Graphics2D) g;
    //g2d.setPaint(Color.green);
    g2d.setStroke(new BasicStroke(1.5f));
    g2d.draw(new Line2D.Double(x1, y1, x2,y2));
  }

  public void drawNode(Graphics g, GNode gNode) {
    int nodePosX=(gNode.pos.x);
    int nodePosY=(gNode.pos.y);
    int nodeWidth=(gNode.width);
    int nodeHeight=(gNode.height);

    Graphics2D g2d = (Graphics2D) g;
    g2d.setPaint(gNode.color);

    g2d.setStroke(new BasicStroke(2.0f));
    g2d.draw(new Line2D.Double(nodePosY + 2, nodePosX + nodeWidth + 1, nodePosY +
nodeHeight,
              nodePosX + nodeWidth + 1));
        g2d.draw(new Line2D.Double(nodePosY + nodeHeight + 1, nodePosX + nodeWidth + 1,
              nodePosY + nodeHeight + 1, nodePosX + 2));
        g2d.draw(new Rectangle2D.Double(nodePosY, nodePosX, nodeHeight, nodeWidth));
    g.drawString(gNode.label,
                nodePosY + (nodeHeight - fontMetrics.stringWidth(gNode.label)) / 2,
                nodePosX + nodeWidth - (nodeWidth - fontHeight) / 2);
  }


}
```

File name: MyPainter.java

```
/*Sven Moen's "Drawing Dynamic Trees" article in IEEE Software, July 1990.  */
/*Reference with modification from Decision Tree program by Pierre Geurts, August 1999
at http://www.montefiore.ulg.ac.be/~geurts/dtapplet/ */


package myprojects.tree;

import java.awt.*;
import java.util.*;


public class MyPainter {

  int parentDistance = 20;

  public void setParentDistance(int distance) {
    parentDistance = distance;
  }

  public void layout(GNode gNode) {
    GNode c;

    if(gNode == null) {
      return;
    }

    c = gNode.child;
    while(c != null) {
      layout(c);
      c = c.sibling;
    }

    if(gNode.child != null) {
      attachParent(gNode, join(gNode));
    } else {
      layoutLeaf(gNode);
    }
  }


  public void attachParent(GNode gNode, int h) {
    int x, y1, y2;

    x = gNode.border + parentDistance;
    y2 = (h - gNode.height) / 2 - gNode.border;
    y1 = y2 + gNode.height + 2 * gNode.border - h;
    gNode.child.offset.x = x + gNode.width;
    gNode.child.offset.y = y1;
    gNode.contour.upperHead = new Polyline(gNode.width, 0, new Polyline(x, y1,
                                                         gNode.contour.upperHead));
    gNode.contour.lowerHead = new Polyline(gNode.width, 0, new Polyline(x, y2,
                                                         gNode.contour.lowerHead));
  }


  public void layoutLeaf(GNode gNode) {
    gNode.contour.upperTail = new Polyline(gNode.width + 2 * gNode.border, 0, null);
    gNode.contour.upperHead = gNode.contour.upperTail;
    gNode.contour.lowerTail = new Polyline(0, -gNode.height - 2 * gNode.border, null);
    gNode.contour.lowerHead = new Polyline(gNode.width + 2 * gNode.border, 0,
                                 gNode.contour.lowerTail);
  }


  public int join(GNode gNode) {
    GNode c;
    int d, h, sum;

    c = gNode.child;
    gNode.contour = c.contour;
    sum = h = c.height + 2 * c.border;
    c = c.sibling;
    while(c != null) {
      d = merge(gNode.contour, c.contour);
      c.offset.y = d + h;
      c.offset.x = 0;
      h = c.height + 2 * c.border;
      sum += d + h;
      c = c.sibling;
    }

    return sum;
  }


  public int merge(Polygon c1, Polygon c2) {
    int x, y, total, d;
    Polyline lower, upper, b;

    x = y = total = 0;
    upper = c1.lowerHead;
    lower = c2.upperHead;

    while(lower != null && upper != null) {

      d = offset(x, y, lower.dx, lower.dy, upper.dx, upper.dy);
      y += d;
      total += d;

      if(x + lower.dx <= upper.dx) {
        y += lower.dy;
        x += lower.dx;
        lower = lower.link;
      } else {
        y -= upper.dy;
        x -= upper.dx;
        upper = upper.link;
      }
    }


    if(lower != null) {
      b = bridge(c1.upperTail, 0, 0, lower, x, y);
      c1.upperTail = (b.link != null) ? c2.upperTail : b;
      c1.lowerTail = c2.lowerTail;
    } else {
      b = bridge(c2.lowerTail, x, y, upper, 0, 0);
      if(b.link == null) {
        c1.lowerTail = b;
      }
    }

    c1.lowerHead = c2.lowerHead;

    return total;
  }


  public int offset(int p1, int p2, int a1, int a2, int b1, int b2) {
    int d, s, t;

    if(b1 <= p1 || p1 + a1 <= 0) {
      return 0;
    }

    t = b1 * a2 - a1 * b2;
    if(t > 0) {
      if(p1 < 0) {
        s = p1 * a2;
        d = s / a1 - p2;
      } else if(p1 > 0) {
        s = p1 * b2;
        d = s / b1 - p2;
      } else {
        d = -p2;
      }
    } else if(b1 < p1 + a1) {
      s = (b1 - p1) * a2;
      d = b2 - (p2 + s / a1);
    } else if(b1 > p1 + a1) {
      s = (a1 + p1) * b2;
      d = s / b1 - (p2 + a2);
    } else {
      d = b2 - (p2 + a2);
    }

    if(d > 0) {
      return d;
    } else {
      return 0;
    }
  }


  public Polyline bridge(Polyline line1, int x1, int y1, Polyline line2, int x2,int y2)
{
    int dy, dx, s;
    Polyline r;

    dx = x2 + line2.dx - x1;
    if(line2.dx == 0) {
      dy = line2.dy;
    } else {
      s = dx * line2.dy;
      dy = s / line2.dx;
    }

    r = new Polyline(dx, dy, line2.link);
    line1.link = new Polyline(0, y2 + line2.dy - dy - y1, r);

    return r;
  }

  public void setTree(GNode gNode, int off_x, int off_y) {
    GNode c, s;
    int cur_y;

    gNode.pos.x = off_x + gNode.offset.x;
    gNode.pos.y = off_y + gNode.offset.y;


    c = gNode.child;
    if(c != null) {
      setTree(c, gNode.pos.x, gNode.pos.y);

          s = c.sibling;
          cur_y = gNode.pos.y + c.offset.y;
          while(s != null) {
                setTree(s, gNode.pos.x + c.offset.x, cur_y);
                cur_y += s.offset.y;
                s = s.sibling;
      }
    }
  }

  public int calcMaxOffset(GNode gNode, int y) {
    GNode c, s;
    int cur_y;
    int current_pos_y;

    current_pos_y = y + gNode.offset.y;
    c = gNode.child;
    if(c != null) {
      int min=calcMaxOffset(c, current_pos_y), newmin=min;


      s = c.sibling;
      cur_y = current_pos_y + c.offset.y;
      while(s != null) {
               newmin=calcMaxOffset(s, cur_y);
               if (newmin<min)
                 min=newmin;
               cur_y += s.offset.y;
               s = s.sibling;
      }
      return min;
    } else
      return current_pos_y;
  }


  public void drawTree(Graphics g, GNode gNode, NodePainter nodePainter) {

    nodePainter.drawNode(g,gNode);

    if(gNode.parent != null) {
      int x1 = gNode.pos.y + gNode.height / 2;
      int y1 = gNode.pos.x;
      int x2 = gNode.parent.pos.y + gNode.parent.height / 2;
      int y2 = gNode.parent.pos.x + gNode.parent.width;
      nodePainter.drawEdge(g,x1,y1,x2,y2);
    }

    if(gNode.sibling != null)
      drawTree(g, gNode.sibling, nodePainter);

    if(gNode.child != null)
      drawTree(g, gNode.child, nodePainter);

  }
}
```