Scalable Algorithms for Learning Large Decision Trees

by

Trong Dung NGUYEN

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Associate Professor Hiroshi Shimodaira

School of Information Science
Japan Advanced Institute of Science and Technology

January 12, 2001

Abstract

Decision tree learning is one of the most widely used and practical methods in machine learning. Among early and basic works on decision tree learning are Hunt's Concept Learning System (Hunt et al. 1966), Friedman and Breiman's CART system (Friedman 1977; Breiman et al. 1984), and Quinlan's ID3 system (Quinlan 1986). Following them, numerous researches continue searching for alternative approaches and algorithms to improve the effectiveness (predictive accuracy) and the efficiency of the method. The development of decision tree learning leads to and is encouraged by a growing number of commercial systems such as CART (Salford Systems), MineSet (SGI), and Intelligent Miner (IBM).

Traditional research issues in decision tree learning include attribute selection, pruning, discretization, handling missing values, and nonstandard forms of decision trees (e.g., oblique decision trees). When decision tree learning becomes one of the most applicable techniques in data mining—a rapidly growing area of research and application—a new challenge is how it can deal with very large and complex databases. For that, several new research issues arise such as handling of relational and complex types of data, handling noisy or incomplete data, efficiency and scalability of learning algorithms, parallel and distributed learning algorithms, presentation and visualization of data mining results, etc.

The ultimate purpose of our research is to develop an integrated decision tree learning system that can be applied effectively and efficiently to data mining applications. To that end, we try to solve several problems of decision tree learning on large and complex datasets. These problems include attribute selection when data are incomplete or uncertain, the scalability of rule post-pruning algorithms, and visualization of large decision trees. Based on the results of our research on those problems, we develop a prototype of a decision tree learning system to demonstrate the effectiveness and efficiency of our

solutions.

Attribute selection is one of basic issues in decision tree learning. Most measures for selecting attributes are either information theory-based such as information gain (Quinlan 1986), gain-ratio (Quinlan 1993), normalized information gain (Mantaras 1991), or statistics-based such as gini-index (Breiman et al. 1984), χ^2 (Liu et al. 1994), etc. Using an approach based on the theory of rough sets (Pawlak, 1991), a mathematical tool to deal with uncertain and incomplete information, we have proposed a new attribute selection measure (R-measure). In our experimental comparative evaluation of R-measure and other there well-known measures, R-measure outperformed or was comparable to the others in many cases. Especially, the experiment showed that R-measure dealt with noisy data more effectively in comparing to the others.

As a rule set has advantages over a decision tree in many cases, some decision tree learning systems use a rule post-pruning algorithm to generate rules from a decision tree. For example, C4.5 system (Quinlan 1993) provides C4.5rules to generate rules. However, due to the algorithm complexity (C4.5rules has a time complexity of $O(n^3)$ on the number of input data records), the algorithm fails to deal with large databases. On the other hand, some other rule learning algorithms such as IREP (Furnkranz, 1994) or RIPPER (Cohen, 1995) are scalable, but have a problem called over-pruning or hasty generalization that can effect the accuracy. By applying the separate-and-conquer strategy and taking the advantage of post-pruning (in contrast to pre-pruning) approach, we have proposed a new post-pruning algorithm (CABROrule) that is scalable (the time complexity is $O(nlogn)$) and can avoid the problem of over-pruning. In our experiments, CABROrule produced smaller rule sets with higher accuracy in comparing to C4.5rules on a major number of applied datasets while running time was reduced substantially.

Data and knowledge visualization is an active research issue in data mining as it is crucial for data mining systems. Especially, visualization probably is the best way to understand a decision tree. Most decision tree learning systems provide a tree visualizer such as a visualizer with a tree map in CART or 3D visualizer in MineSet. However, these visualizers work well with average-size decision trees but have several problems to visualize large ones. On other hand, the fields of information visualization and graph drawing have

many researches on methods to visualize large hierarchical structures, such as cone trees (Robertson, 1991), hyperbolic trees (Lamping et al. 1997), tree-map (Johnson et al., 1991), etc. By applying methodology of information visualization we have proposed a new algorithm (T2.5D) for decision tree visualization. T2.5D overcomes many difficulties of current methods in viewing and navigating large devision trees.

Based on the results of those researches we have developed a prototype of a decision tree learning system as a first step toward our ultimate purpose—an integrated decision tree learning system for data mining. The R-measure is used for attribute selection, CABROrule is used for generating rules, and decision trees are visualized with T2.5D. The program has been tested successfully with several large and complex datasets.

Acknowledgments

First of all, I would like to express my gratitude and thanks to my advisor Associate Professor Hiroshi Shimodaira who has guided me through my study and research. Someday I may be able to find words to express my gratefulness for his patience and forgiveness.

I would like to express my gratitude and thanks to my co-advisor Professor Ho Tu Bao who has taught me how to do research. During my years as a graduate student, I appreciate very much his patience and effort in checking and revising methods, experiments, referred papers, and presentations related to my research.

I would like to express my gratitude and thanks to Professor Shigeki Sagayama for advice on my research.

I owe a debt of gratitude to Professor Masayuki Kimura, who gave me all his kindness and forgiveness from the first day I came to the institute.

I would like to thank Professor Hiroshi Motoda and Professor Satoshi Tojo for many detail comments on this thesis. I would like to thank Professor Milan Vlach for many fruitful discussion on rough set theory.

Members of Artificial Intelligence Laboratory gave me many helps throughout my study. Particularly, Associate Mitsuru Nakai and Mr. Hisao Koba patiently explained me many questions about computer and network systems.

Contents

Chapter 1

Introduction

1.1 Motivation and Reseach Context

Decision tree learning (DTL) is one of the most widely used and practical methods for inductive learning. It is a method for approximating discrete-valued functions that is robust to noisy data and capable of learning disjunctive expressions. Among early and basic works on decision tree learning are Hunt's Concept Learning System (CLS) [40] and Friedman and Breiman's work resulting in the CART system [26, 9]. Quinlan's ID3 system [77, 78] is the ancestor of one of the most well-known system C4.5 [83]. Other early work on decision tree learning includes ASSISTANT [50, 14].

There are numerous researches on decision tree learning that mostly aim at improving the effectiveness (e.g., the predictive accuracy of decision trees) and the efficiency (e.g., the running time) of the method. Two basic issues in decision tree learning are attribute selection [63] and pruning [64]. As decision tree learning algorithms recursively split examples belonging to a node in order to build children nodes, the first issues concerns how to make that split effectively. The second issues concerns a problem in machine learning called over-fitting, that means a decision tree is too specific for training data and has low predictive accuracy on new unseen data.

Other issues in decision tree learning include incorporating continuous-valued attributes, handling training examples with missing attribute values, and handling attribute with differing costs. Initially, most of decision tree learning algorithms only deal with

nominal attributes. There are many researches on discretization [22, 96, 5] or adapting learning algorithms to handle numeric data [84]. In certain cases, the available data may be missing values for some attributes. There are several strategies to deal with them, such as those described in [65, 83, 9].

When decision tree learning becomes one of the most applicable techniques in data mining, a rapidly growing area of research and application, a new challenge is how it can deal with very large and complex databases. For that, several new research issues arise such as handling of relational and complex types of data, handling noisy or incomplete data, efficiency and scalability of learning algorithms, parallel and distributed learning algorithms, and presentation and visualization of data mining results [37].

The ultimate purpose of our research is to develop an integrated decision tree learning system that can be applied effectively and efficiently to data mining applications. To that end, we try to solve several problems of decision tree learning on large and complex datasets. These problems include attribute selection when data are incomplete or uncertain, the scalability of rule post-pruning algorithms, and visualization of large decision trees.

Attribute selection is one of basic issues in decision tree learning. Most measures for selecting attributes are either information theory-based such as information gain [79], gain-ratio [83], normalized information gain [58], or statistics-based such as gini-index [9], χ^2 [56]. Using an approach based on the theory of rough sets [73], a mathematical tool to deal with uncertain and incomplete information, we try to proposed a new attribute selection measure that is expected to deal better with incomplete and uncertain data.

As a rule set has advantages over a decision tree in many cases, some decision tree learning systems such as C4.5 [83] uses a rule post-pruning algorithm (C4.5rules) to generate rules from a decision tree. However, due to the algorithm complexity (C4.5rules has a time complexity of $O(n^3)$ on the number of input data records [29]), it fails to deal with large databases. In the other hand, some other rule learning algorithms such as IREP [27] or RIPPER [16] are scalable, but have a problem called over-pruning or hasty generalization that can effect the accuracy [25]. By applying the separate-and-conquer strategy and taking the advantage of post-pruning (in contrast to pre-pruning) approach,

we try to proposed a scalable post-pruning algorithm that is expected to be as accurate as C4.5rules and can avoid the problem over-pruning.

Visualization is very helpful to understand decision trees. Although visualizing large trees attracts many researches in the field of information visualization [38, 53, 41, 31], there is still no available tool or algorithm in decision tree learning that allows the user viewing and navigating large decision trees comfortably. By analyzing characteristics of decision trees together with techniques from the field of information visualization, we try to propose a new algorithm that allows the user viewing and navigating large decision trees comfortably.

1.1.1 The Issue of Attribute Selection

To learn decision trees, learning algorithms have to provide methods for: (1) *attribute selection*, i.e. choosing the "best" attribute to split a decision node in terms of a measure for "goodness of split", (2) *pruning*, i.e. cutting off unstable leaves to avoid overfitting and obtain statistical reliability, and (3) *discretization*, i.e. transforming continuous attributes into discrete ones to deal with mixed data. As attribute selection is of key importance to the decision tree generation, it has attracted many DTL work until recently, e.g. [50], [43]. Most measures for selecting attributes are either information theory-based such as information gain [79], gain-ratio [83], normalized information gain [58], or statistics-based such as gini-index [9], χ^2 [56], etc. In this work we introduce alternatively a rough set-based measure for attribute selection called R-measure. The theory of rough sets introduced by Pawlak in early 1980s is a mathematical tool to deal with imprecise and incomplete information [74], [68]. The limitation of the deterministic model of rough set theory when dealing with uncertain information has been recognized and there have been several attempts to overcome this restriction such as probabilistic model [73] and the variable precision model [103]. However, the former cannot inherit all useful properties of the original rough set model, and the latter raises a new problem of specifying an appropriate threshold. R-measure, inspired by the attribute dependency measure in rough set theory, aims at dealing with uncertain information while preserving properties of the rough set model without requiring thresholds, and it can be used as a solution for attribute

selection in DTL.

1.1.2 Rule Post-Pruning of Large Decision Trees

Data mining algorithms have usually to deal with very large databases. For the prediction data mining task, in addition to the requirements of high accurate and understandability of discovered knowledge, the mining algorithms must be scalable, i.e., given a fixed amount of main memory, their runtime increases linearly with the number of records in the input database.

Decision tree learning has become a popular and practical method in data mining because of its significant advantages: the generated decision trees usually have acceptable predictive accuracy; the hierarchical structure of generated trees makes them are quite easy to understand if trees are not large; and especially the learning algorithms, which employ the *divide-and-conquer* (or simultaneous covering) strategy to generate decision trees, do not require complex processes of computation. However, it happens that in certain domains the comprehensibility and predictive accuracy of decision trees decrease considerably because of the problem known as *subtree replication* [72] (when the subtree replication occurs, identical subtrees can be found at several different places in the same tree structure).

The solution to the problem of subtree replication in the most well-known decision tree learning system C4.5 [83] is to convert a generated decision tree into a set of rules using a *post-pruning* strategy [62]. The conversion of trees into rules is not only an effective way to avoid the subtree replication problem but also offers other significant advantages: while large trees generated from large datasets are difficult to understand, discovered knowledge in form of rules is much easier to understand. Also, in our practical experience domain experts often feel more comfortable to analyze and validate rules than trees if trees become large. Moreover, it appears that the generated rule sets usually have equal or higher predictive accuracy than the original decision tree. However, the C4.5rules algorithm is not scalable to large databases as the simulated annealing, which is employed to achieve an optimal generalization, requires $O(n^3)$ time complexity where n is the number of records in the input database [15].

The *separate-and-conquer* (or simultaneous covering) strategy is an alternative approach to learn rules directly from databases. The most well-known separate-and-conquer algorithms include CN2 [17], REP [11], IREP [27], RIPPER [16], PART [25]. Among them, CN2 and REP also require a computation with high complexity, and therefore cannot be applicable to large data bases. IREP and RIPPER solve the problem of complexity by using a scheme called *incremental pruning*. The result is that they can run very fast and generate small rule sets with acceptable predictive accuracy. However, incremental pruning may lead to the problem of *overpruning* (or hasty generalization) that reduces the accuracy of the algorithms in many cases. PART [25] is an attempt to combine divide-and-conquer and separate-and-conquer strategies, and was claimed to be effective and efficient.

Our research concerns with scalable algorithms for rule-post pruning from large decision trees. In particular it proposes a solution to the problem of high complexity in C4.5rules by using a scheme similar to incremental pruning. The essence of the proposed algorithm is to avoid the problem of overpruning by appropriate improvements in incremental pruning. Experiments show that the proposed algorithm produces rule sets that as accurate as those generated by C4.5 and is scalable for very large data sets.

1.1.3 Visualizing Large Decision Trees

Though decision trees are a simple notion, we can understand their content and hierarchical structure easily if they are small but cannot understand or understand difficultly if they are large. Research on visualization of decision trees has recently received a great attention from the KDD (knowledge discovery and data mining) community because of its practical importance. Many works have been done, e.g., the 3D Tree Visualizer in system MineSet [12], CAT scan (classification aggregation tablet) for inducing bagged decision trees [87], the interactive visualization in decision tree construction [4], the tree visualizer with a tree map in system CART [9] of Salford Systems, etc. However, it is still difficult to view and navigate large trees with these systems. On the other hand, new approaches in information visualization field for representing large hierarchical structures, e.g., cone trees [91], hyperbolic trees [53], have not been well considered in AI, machine learning

and data mining.

By applying methodology of information visualization we have proposed a new algorithm (T2.5D) for decision tree visualization. T2.5D overcomes many difficulties of current methods in viewing and navigating large devision trees.

1.2 Main Results

In the first research issue, to develop the a new criterion for attribute selection, we have proposed a variant of attribute dependency measure of the probabilistic model of rough sets [73] in order (1) to overcome the limitations of the original model in case of noisy data, (2) to make the model more coherent, and (3) to preserve the convenience of non-parameter. Based on this model, *R-measure* is developed to measure how much the class attribute depends on a predictive attribute. Using *R-measure* as an attribute selection criterion, an experimental comparative evaluation on 32 datasets shows that it can be considered as a good alternative criterion for attribute selection. Especially, the experiment showed that R-measure dealt with noisy data more effectively in comparing to the others.

In the second research issue, we have proposed a new algorithm for rule post-pruning of decision trees. It can be considered an alternative algorithm for C4.5rules when the input data become very large. The problem of high complexity in C4.5 is solved by adopting an incremental pruning scheme. However the algorithm does not suffer the problem of hasty generalization such as in the original incremental pruning approach. Experiments have shown that the new algorithm generates rule sets as accuracy as those of C4.5 but with far less time of computation.

In the third research issue, we have developed a new technique for visualizing large decision trees (T2.5D). The technique has several advantages comparing to other techniques: (1) it easily handles decision trees with more than 20000 nodes, and more than 1000 nodes can be displayed together on the screen, (2) it gives the user a clear view of an active path and an image of the overall structure of the tree at the same time, (3) it facilitates the tree navigation as only a minimum number of operations (e.g., click, scroll,

etc.) is needed.

Based on the results of those researches we have developed a prototype of a decision tree learning system as a first step toward our ultimate purpose—an integrated decision tree learning system for data mining. The R-measure is used for attribute selection, CABROrule is used for generating rules, and decision trees are visualized with T2.5D. The program has been tested successfully with several large and complex datasets.

1.3 Thesis Structure

This thesis proposes several methods and techniques related to learning large decision trees. It is organized as following:

- Chapter 2 introduces several basic concepts that will be used in the whole thesis.
- Chapter 3 presents our research on the problem of attribute selection for decision tree learning. It includes the introduction to the problem and rough set theory, our solution to the problem using rough sets, the experimental results, and some remarks about the research.
- Chapter 4 presents our research on rule post-pruning. It describes several related works in the field together with their advantages and drawbacks. We will introduce our new algorithm and explain how it can overcome the problems of current methods. The experimental results on predictive accuracy and running time will be shown together with some analysis.
- Chapter 5 presents our new technique for visualizing large decision trees. Several current methods will be presented and we will explain why they are not suitable for large decision trees. We will propose a new technique that may have advantages in comparing with current methods in many cases.
- Chapter 6 gives an overview about the data mining system CABRO. We will explain why model selction is important in data mining and our solution for it.

Chapter 2

Preliminaries

2.1 Basic Concepts

Inductive learning algorithms take some data collected from a domain as input and produce a model of the domain's structure as output. In other words, they induce a model of the domain from a given set of observations. The individual observations are called instances, and a set of observations is called a dataset. Each instance contains a set of values that measure certain properties of the instance. These properties are called attributes. Each instance is described by the same set of attributes. Most implementations of learning algorithms assume that the attributes are either nominal or numeric. Nominal attributes consist of a set of unordered values, for example, a set of colors. Numeric attributes can be either integers or real numbers. There are several other possible attribute types [102], but it is generally straightforward to adapt existing learning algorithms to deal with them. The space of all possible combinations of attribute values is called the instance space.

Decision trees and rule sets belong to a group of models called classifiers. Classifiers divide the instance space into disjoint regions and assign one of a fixed set of unordered values to each region. In other words, they assume that each instance in the instance space is labeled with an additional nominal attribute value, called the class of the instance. Each region of instance space is assigned to exactly one class, but more than one region can be assigned to the same class. In other words, classifiers define a function that maps the instance space onto a set of unordered values. They differ from methods for numeric

prediction—for example, linear regression [101] because the target value is nominal rather than numeric. Thus they can be used to model domains that pose prediction problems that have a nominal target value. These problems are called classification problems.

Learning algorithms for classification problems have many practical applications. Consider, for example, one of the first applications of classification learning: the diagnosis of soybean diseases [60]. In this application, the individual instances are soybean plants that are described by a set of attributes. Most of the attributes correspond to symptoms of various soybean diseases and their values indicate whether a particular symptom is present or absent. The class values are the different soybean diseases that can occur. A classifier for this problem defines a function that maps a particular combination of attribute values to a corresponding disease. This means that the classifier can be used to automatically obtain a diagnosis for a particular soybean plant, given a set of observed attribute values.

The task of a learning algorithm is to induce a classifier automatically from a set of training instances. These instances have been randomly collected from the domain and have class labels assigned to them by some other process, for example, by a human expert for soybean diseases. The learning algorithm constructs a classifier by partitioning the instance space according to the class labels of the training instances. Ideally the induced classifier will maximize the number of correctly assigned class values for all possible instances—even instances that have not occurred in the training data. In that case the learning algorithm has correctly identified the structure of the domain. If an instance is assigned to the wrong class, we say that it is misclassified. The predictive performance of a classifier is measured by its error rate: the expected percentage of misclassifications on independent instances randomly sampled from the domain. Given that the structure of the domain is unknown, an infinite number of instances is required to obtain the true value of the error rate. In practice, infinite amounts of labeled data are not available, and the error rate must be estimated using an independent set of labeled instances that are unavailable to the learning algorithm when it generates the classifier. This set of instances is called the test data. Unlike the error rate on the training data, the observed error on the test data is an unbiased estimate of the classifier's error rate on future instances.

In many practical classification problems no learning algorithm can achieve an error rate of zero, even given an infinite amount of training data. This is because most domains contain a certain amount of noise. If noise is present, there is a non-zero probability that different class labels will be observed if the same instance is sampled multiple times from the domain. There are several possible reasons for the occurrence of noise, for example, errors in measuring the attribute and class values of an instance. There can also be a degree of uncertainty inherent in the domain—for example, uncertainty due to the fact that not all relevant properties of an instance are known.

Apart from affecting the minimum error rate that can be achieved a consequence that cannot be avoided by improving the learning algorithm, noise also has a detrimental effect because it potentially misleads the learning algorithm when the classifier is induced. This phenomenon can further decrease the classifier's performance. The learning algorithm can be misled by noise because training instances may have an incorrect class label assigned to them a class label different from the the one that is most likely to be assigned to the same instance in the test data. This is a problem because the learning algorithm constructs a classifier according to the class labels from the training data. Consequently it is important to detect instances that are labeled incorrectly and prevent them from affecting the structure of the classifier. If the classifier fits the training instances too closely, it may fit noisy instances, and that reduces its usefulness. This phenomenon is called overfitting, and various heuristics have been developed to deal with it. In decision trees and lists, a common strategy is to eliminate those parts of a classifier that are likely to overfit the training data. This process is called pruning, and can increase both the accuracy and the comprehensibility of the resulting classifier. The success of a pruning mechanism depends on its ability to distinguish noisy instances from predictive patterns in the training data.

2.2 Decision Tree and Rule Learning

1.2 Decision Trees and Lists Decision trees [79] and rule sets [90] are two closely related types of classifier. In contrast to most other classification paradigms, for example, instance-based learning [1], neural networks [92], Bayesian networks [42], and logistic re-

gression [6], they embody an explicit representation of all the knowledge that has been induced from the training data. Given a standard decision tree or a rule set, a user can determine manually how a particular prediction is derived, and which attributes are relevant in the derivation without performing any numeric operations (other than comparison). This makes it very easy to explain how these classifiers are to be interpreted, and how they generate a prediction. To derive a prediction, a test instance is filtered down the tree, starting from the root node, until it reaches a leaf. At each node one of the instance's attributes is tested, and the instance is propagated to the branch that corresponds to the outcome of the test. The prediction is the class label that is attached to the leaf. Multivariate decision trees can test for higher-order relationships that involve more than one attribute, for example, linear combinations of attribute values [10]. This makes them potentially more powerful predictors. However, they are also harder to interpret and computationally more expensive to generate.

S1. Select the "best" attribute at the node being considered by a selection measure.

S2. Extend the tree by adding a new branch for each value of the selected attribute.

S3. Sort instances of the node to new leaf nodes.

S4. If instances unambiguously classified then Stop else repeat steps 1-4 for leaf nodes.

S5. Prune the induced tree to obtain a more reliable tree.

Figure 2.1: General scheme of decision tree induction

Standard learning algorithms for decision trees, for example, C4.5 [83] and CART [9], generate a tree structure by splitting the training data into smaller and smaller subsets in a recursive top-down fashion. Starting with all the training data at the root node, at each node they choose a split and divide the training data into subsets accordingly. They proceed recursively by partitioning each of the subsets further. Splitting continues until all subsets are pure, or until their purity cannot be increased any further. A subset is pure if it contains instances of only one class. The aim is to achieve this using as few splits as possible so that the resulting decision tree is small and the number of instances supporting each subset is large. To this end, various split selection criteria have been designed, for example the information gain [79], the Gini index [9], and the gain ratio [83]. They all provide ways of measuring the purity of a split. Some also consider the

resulting support in each subset. At each node, the learning algorithm selects the split that corresponds to the best value for the splitting criterion.

Straightforward purity-based tree induction cannot deal successfully with noisy data because the strategy of creating pure subsets will isolate incorrectly labeled instances and use them for prediction. Consequently most decision tree inducers incorporate a pruning facility to deal with noisy data by eliminating unreliable branches or subtrees. Two different regimes for pruning are used: post-pruning and pre-pruning. Post-pruning is invoked after the full tree has been created, and deletes those parts of the classifier that do not improve its predictive performance. Algorithms for performing post-pruning will be discussed in more detail in Chapter 3. Pre-pruning, on the other hand, attempts to avoid the problem of noise by terminating the splitting process if further splits are likely to overfit the training data. This class of pruning methods will be investigated in Chapter 4.

Rule sets are similar to decision trees in that a sequence of decisions is required to derive a prediction. The difference is that all decisions have a binary outcome, true or false, and further tests on an instance are only required if the outcome of all previous decisions was negative. Individual decisions are made according to a rule that consists of combinations of attribute-value tests and a class label. A rule set is a list of rules that are evaluated in sequence. A rule fires if a test instance passes each attribute-value test that the rule contains. In that case the classification process stops and the rule's class label is assigned to the test instance. Otherwise, the next rule in the list is evaluated.

The last rule in the list is called the default rule and fires for every instance that reaches it. The default rule is required so that no instances are left unclassified. Compared to a decision tree, this rule-based classifier has the advantage that it can impose an ordering on the knowledge that is acquired: rules that are frequently used and reliable can be presented at the start of the list, and that are less important and not very accurate deferred until the end. In some domains rule sets can represent the underlying structure of the domain much more succinctly than decision trees. The reason for this phenomenon is discussed in Chapter 4.

The way in which rule sets are generally created is quite similar to the standard

learning procedure for decision trees. The top-down induction of decision trees proceeds according to a divide-and-conquer strategy where the training data is partitioned into subsets and the algorithm is applied recursively to each subset. Similarly, inducers for rule sets employ a separate-and-conquer procedure where one rule is built for a subset of the instances and further rules are generated recursively for the remaining data. Rule generation is guided by a criterion similar to the split selection measure for decision trees. Tests are added to a rule by optimizing this criterion. The aim is to find rules that cover large, pure subsets of instances in order to maximize the empirical support for each rule.

Like the basic divide-and-conquer algorithm, the standard separate-and-conquer procedure cannot deal successfully with noisy data because it aims at identifying pure subsets of instances. To make rule learning useful in real-world domains, some kind of pruning mechanism is essential. There are two main pruning strategies for separate-and-conquer rule learners. The first builds a full unpruned rule set and then simplifies the classifier by eliminating tests from rules or by deleting individual rules. This is done by globally optimizing the rule set according to a pruning criterion. Global pruning of rule sets is related to post-pruning methods for decision trees because a full, unpruned classifier is generated before pruning begins. However, there are some important differences and they will be discussed in Chapter 5.

The second strategy adopts a simpler approach where each rule is simplified immediately after it has been generated. This simplification strategy is called incremental pruning, and it turns out that it has several conceptual advantages over the global optimization approach. Considering the rule set as a whole, incremental pruning is similar to pre-pruning in decision trees because rules are pruned before the structure of the full, unpruned rule set is known. However, at the rule level, incremental pruning more closely resembles post-pruning because rules are pruned after they have been fully expanded. The differences between global and incremental pruning will be discussed in more detail in Chapter 5.

2.3 Experimental Methodology

All methods investigated in this thesis are evaluated empirically on benchmark problems from the UCI repository of machine learning datasets [67]. They contain a wide range of practical problems. About half of the datasets have binary class labels and the other half represent multi-class domains. Most problems contain a mix of nominal and numeric attributes, some are purely numeric, and a few are purely nominal. A significant fraction also contain missing values.

As mentioned above, accuracy on the training data is not a good indicator of a classifier's future performance. Instead, an independent sample of test instances must be used to obtain an unbiased estimate. One way of evaluating a learning algorithm is to split the original dataset randomly into two portions and use one portion for training and the other for testing. However, the resulting estimate depends on the exact split that is used and can vary significantly for different splits, especially if the original dataset is small. A more reliable procedure is to repeat the process several times with different random number seeds and average the results. Cross-validation is a slightly more sophisticated version of this basic method for performance evaluation. In a k-fold cross-validation, the training data is split into k approximately equal parts. The first of these k subsets is used for testing and the remainder for training. Then the second subset is used for testing and all other k - 1 subsets are used for training. This is repeated for all k subsets and the results are averaged to obtain the final estimate. Compared to the naive procedure, cross-validation has the advantage that each instance is used for testing exactly once.

Usually the parameter k is set to ten. It has been found empirically that this choice produces the most reliable estimates of the classifier's true performance on average [48], and there is also a theoretical result that supports this finding [45]. The variance of the estimate can be further reduced by taking the average of a repeated number of cross-validation runs, each time randomizing the original dataset with a different random number seed before it is split into k parts [47]. Ideally the cross-validation is performed for all possible permutations of the original dataset. However, this kind of complete cross-validation is computationally infeasible for all but very small datasets [47], and must be approximated by a limited number of cross-validation runs. All performance estimates

presented in this thesis are derived by repeating ten-fold cross-validation ten times and averaging the results.

When comparing two learning algorithms, the difference in performance is important. The same ten cross-validation runs, using the same ten randomizations of the dataset, can be used to obtain estimates for both schemes being compared. However, to make maximum use of the data we would like to use estimates from a complete cross-validation for the comparison. Fortunately the ten given estimates for the two schemes can be used to get some information on the outcome that would be obtained if they were compared using complete cross-validation, because the mean of a limited number of cross-validation estimates is approximately normally distributed around the true mean the result of a complete cross-validation. Consequently a two-tailed paired t-test [101] on the outcome of the ten cross-validation runs can be used to test whether the result of a complete cross-validation would be likely to show a difference between the two schemes. In this thesis a difference in performance is called significant according to a t-test at the 5% significance level applied in this fashion.

Chapter 3

A Measure for Attribute Selection Based on Rough Sets

3.1 Introduction

There are three problems of decision tree learning which have been intensively investigated in the machine learning: (1) attribute selection, i.e. choosing the "best" attribute to split a decision node in terms of a measure for "goodness of split", (2) pruning, i.e. cutting off unstable leaves to avoid overfitting and obtain statistical reliability, and (3) discretization, i.e. transforming continuous attributes into discrete ones to deal with mixed data. The performance of a DTL system principally depends on methods to solve these problems. As attribute selection is of key importance to the decision tree generation, it has attracted many DTL work until recently, e.g. [50, 43]. Most measures for selecting attributes are either information theory-based such as information gain [79], gain-ratio [83], normalized information gain [58], or statistics-based such as gini-index [9], χ^2 [56], etc. In this work we introduce alternatively a rough set-based measure for attribute selection called R-measure. The theory of rough sets introduced by Pawlak in early 1980s is a mathematical tool to deal with imprecise and incomplete information [74, 68]. The limitation of the deterministic model of rough set theory when dealing with uncertain information has been recognized and there have been several attempts to overcome this restriction such as probabilistic model [73] and the variable precision model [103]. However, the former cannot inherit all useful properties of the original rough set model, and the latter raises a new problem of specifying an appropriate threshold. R-measure, inspired by the attribute

dependency measure in rough set theory, aims at dealing with uncertain information while preserving properties of the rough set model without requiring thresholds, and it can be used as a solution for attribute selection in DTL.

3.2 Attribute Selection Measures

In order to facilitate a common understanding of different attribute selection measures, we use the statistic notations presented in [56, 50]. Suppose that we are dealing with a problem of learning a classifier with k classes C_i $(i = 1, k)$ from a set of training instances described by a set of attributes. We assume that all attributes are discrete each of which is with a finite number of possible values. Let $n_{..}$ denotes the total number of training instances, $n_{i.}$ the number of instances from class C_i, $n_{.j}$ the number of instances with the j-th value of the given attribute A, and n_{ij} the number of instances from class C_i and with the j-th value of A. Let further

$$p_{ij} = \frac{n_{ij}}{n_{..}}, \quad p_{i.} = \frac{n_{i.}}{n_{..}}, \quad p_{.j} = \frac{n_{.j}}{n_{..}}, \quad p_{i|j} = \frac{n_{ij}}{n_{.j}} \tag{3.1}$$

denote the approximation of the probabilities from the training set. Let

$$H_C = -\sum_i p_{i.} log p_{i.}, \quad H_A = -\sum_j p_{.j} log p_{.j}, \tag{3.2}$$

$$H_{CA} = -\sum_i \sum_j p_{ij} log p_{ij}, \quad H_{C|A} = H_{CA} - H_A \tag{3.3}$$

be the entropy of the classes, of the values of the given attribute, of the joint example class–attribute value, and of the class given the value of the attribute, respectively (all logarithms introduced here are of the base two).

The well-known decision tree algorithm C4.5 use the gain-ratio [83]

$$GainR = \frac{H_C + H_A - H_{CA}}{H_A} \tag{3.4}$$

Gini-index used in decision tree learning algorithm CART [9] can be rewritten as

$$Gini = \sum_j p_{.j} \sum_i p_{i|j}^2 - \sum_i p_{i.}^2 \tag{3.5}$$

Another statistics-based measure of interest is χ^2 and it has been tested with high performance [63]

$$\chi^2 = \sum_i \sum_j \frac{(e_{ij} - n_{ij})^2}{e_{ij}}, \qquad e_{ij} = \frac{n_{.j} n_{i.}}{n_{..}} \tag{3.6}$$

3.3 Rough Set Theory and Extended Models

Rough set theory, introduced by Zdzislaw Pawlak in the early 1980s, is a mathematical tool to deal with vagueness and uncertainty.

3.3.1 Basic Concepts of Rough Sets

The theory of rough sets was recognized as a fruitful theory for discovering relationship in data. Though closely related to statistics, its approach is entirely different: rough sets are based on equivalence relations describing partitions made of classes of indiscernible objects instead of employing probability to express data vagueness. In the rough set theory a subset of a universe is approximated by a pair of ordinary sets called lower and upper approximations. The starting point of the rough set theory is the assumption that our "view" on elements of the object set O depends on indiscernibility relations among them, that mean equivalence relations $E \subseteq O \times O$. Two objects $o_1, o_2 \in O$ are called to be *indiscernible* regarding E if $o_1 E o_2$. The *lower* and *upper* approximations of any $X \subseteq O$, regarding an equivalence relation E, are defined as

$$E_*(X) = \{o \in O : [o]_E \subseteq X\} \tag{3.7}$$

$$E^*(X) = \{o \in O : [o]_E \cap X \neq \emptyset\} \tag{3.8}$$

where $[o]_E$ denotes the equivalence class of objects which are indiscernible with o with respect to the equivalence relation E. Thus, these approximations consist of all objects which surely and possibly belong to X regarding E, respectively. A subset P of the set of attributes used to describe objects of O determines an equivalence relation that divides O into equivalence classes each contains objects with the same values on all attributes of P.

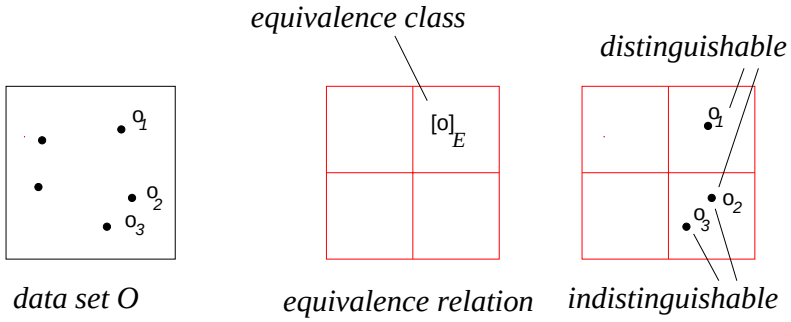

Figure 3.1: Indiscernibility relation

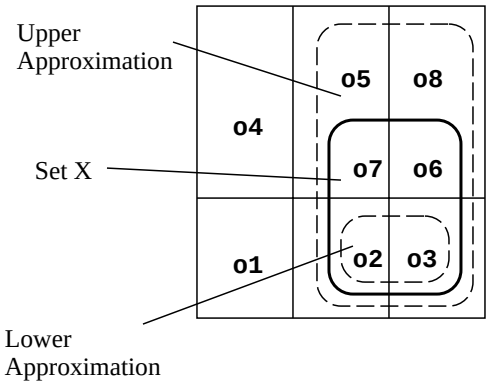

Figure 3.2: Basis concepts of rough sets

3.3.2 Measure of Attribute Dependency

A key concept in the rough set theory is the *degree of dependency* of a set of attributes Q on a set of attributes P, denoted by $\mu_P(Q)$ $(0 \leq \mu_P(Q) \leq 1)$, defined as

$$\mu_P(Q) = \frac{card(\bigcup_{[o]_Q} P_*([o]_Q))}{card(O)} = \frac{card(\{o \in O : [o]_P \subseteq [o]_Q\})}{card(O)} \tag{3.9}$$

If $\mu_P(Q) = 1$ then Q totally depends on P; if $0 < \mu_P(Q) < 1$ then Q partially depends on P; if $\mu_P(Q) = 0$ then Q is independent of P. The measure of dependency is fundamental in the rough set theory as based on it many other basic notions are defined, such as reducts and minimal sets of attributes, significance of attributes, etc.

We give an illustration and analyze of the formula (3.9) through a Pawlak's small information table (Table 3.1) consisting of eight objects described by two descriptive attributes *Temperature, Headache*, and the class attribute *Flu*. From the formula (3.9) we can cal-

culate $\mu_{\{Temperature,Headache\}}(Flu) = 1$, $\mu_{Temperature}(Flu) = 5/8$ and $\mu_{Headache}(Flu) = 0$. That means, according to this measure, Flu totally depends on $\{Temperature, Headache\}$, partially depends on $Temperature$ and is independent of $Headache$.

Table 3.1: Information table

	Temperature (T)	*Headache (H)*	*Flu (F)*
e_1	*normal*	*yes*	*no*
e_2	*high*	*yes*	*yes*
e_3	*very_high*	*yes*	*yes*
e_4	*normal*	*no*	*no*
e_5	*high*	*no*	*no*
e_6	*very_high*	*no*	*yes*
e_7	*high*	*no*	*no*
e_8	*very_high*	*yes*	*yes*

An interpretation of (3.9) can be obtained by expressing the causal relation between attributes in the form of usual rules. For example, consider how the attribute Flu depends on the attribute $Temperature$. It can be easily verified that

If $Temperature = normal$ then $Flu = no$

If $Temperature = very\_high$ then $Flu = yes$

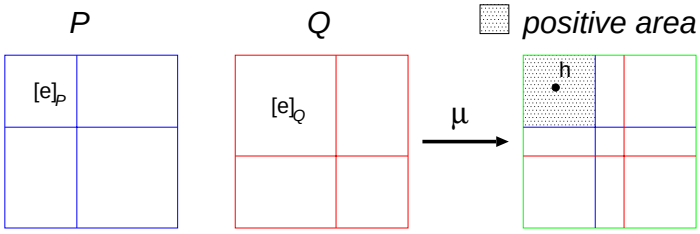

Figure 3.3: The measure of attribute dependency

The number of objects that satisfy these rules is 5 out of 8. In the other words, the proportion of objects whose values on Flu are correctly predicted by values of $Temperature$ is 5/8. This argument is analogous with the definition of the degree of dependency, where each rule corresponds to an equivalent class with respect to P which is included in an equivalent class w.r.t Q.

3.4 R-measure for Attribute Selection Problem

In this chapter we propose a new measure for attribute dependency which overcomes some drawbacks of the one used in rough sets. An application of the new measure to the attribute selection problem in DTL is also introduced.

3.4.1 A New Measure for Attribute Dependency

The attribute dependency measure (3.9) in the deterministic model of rough sets deliberately ignores the available probabilistic information in it formalism, and deals poorly with noisy data. Among approaches to overcome this restriction, the variable precision model [103] extends rough sets by employing relations named *majority inclusion relations*. A majority inclusion relation considers that a set A is included in a set B if the intersection is a majority of set A w.r.t a threshold. Based on such a relation the model redefines all the notions of rough sets. Although those redefinitions aimed at better handing uncertain and noisy data, they also raised a new problem of specifying appropriate thresholds in a particular application.

Differently from the variable precision model, in [73] the authors approached to this problem by the probabilistic model without requiring any threshold. However this approach is somehow mixed. On the one hand, the definitions of the basic notions (i.e. upper, lower approximations and boundary) are totally consistent with Bayes' decision procedure. As a matter of fact, they are special cases of the ones of the variable precision model when 0.5 is taken as the threshold. On the other hand, the definitions of the derived notions (e.g. attribute dependency, reduct, core) are based exclusively on the information theory. This mixed phenomenon makes the model incoherent and do not directly inherit all useful properties of the original model.

We propose alternative definitions of the derived notions for the probabilistic model that are consistent with Bayes' decision procedure. In short, our proposal aims at (1) overcoming the limitations of the original model for noisy data, (2) making the probabilistic model be more coherent, and (3) preserving the convenience of requiring no threshold. We describe here only our modifications to the definition of the attribute dependency

measure. Other derived notions of rough sets (attribute significance, reduct, core, superfluous) are definitely based on this key notion, and can be defined accordingly. Returning to Table 3.1, we can obtain the following probabilistic rules about the relation between *Flu* and *Headache*

$$\textit{If } \textit{Headache} = \textit{yes} \text{ then } \textit{Flu} = \textit{yes} \ (3/4)$$
$$\textit{If } \textit{Headache} = \textit{no} \text{ then } \textit{Flu} = \textit{no} \ (3/4)$$

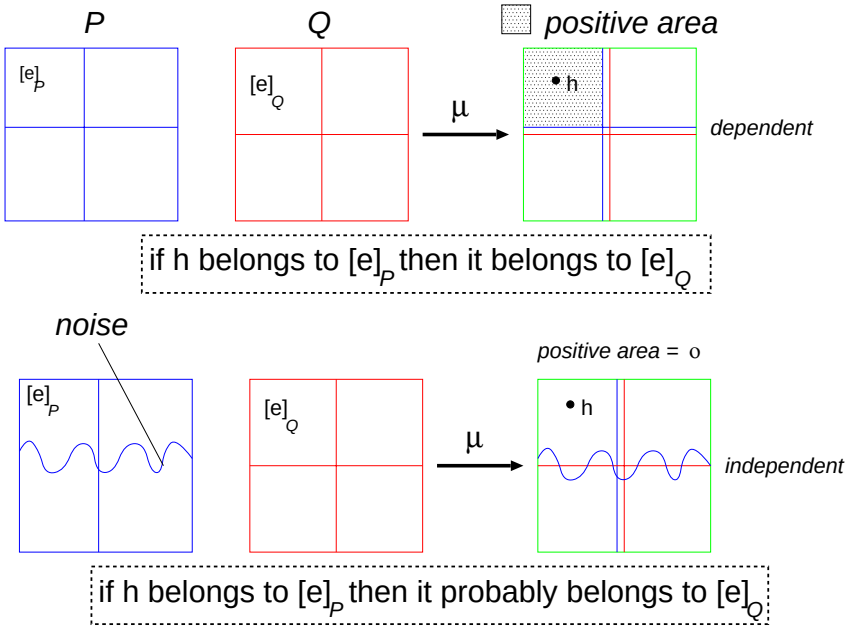

Figure 3.4: The drawback of the measure of attribute dependency

These rules show that *Flu* somehow depends on *Headache*, but the formula (3.9), by its value 0 in this case, says that *Flu* is independent of *Headache*. Consider further probabilistic rules. Suppose that the value on *Headache* of a new object is known, and an agent wants to predict the value on *Flu* of this object. For example, if $\textit{Headache} = \textit{yes}$, then there are two possibilities: $\textit{Flu} = \textit{yes}$ $(3/4)$, or $\textit{Flu} = \textit{no}$ $(1/4)$. To minimize the probability of error, $\textit{Flu} = \textit{yes}$ is certainly chosen as it is the value with the maximum likelihood of occurrence among all possibilities. Due to the risk of $\textit{Flu} = \textit{no}$, this prediction is uncertain and has an estimated accuracy of $3/4$. Similarly, the value $\textit{Flu} = \textit{no}$ will be predicted if $\textit{Headache} = \textit{no}$ with the estimated accuracy is also $3/4$. Denote by X the event that the prediction of the agent is true, we have

$$\begin{aligned} \mathrm{P(X)} &= \mathrm{P}(H = yes) \times \mathrm{P(X} \mid H = yes) + \mathrm{P}(H = no) \times \mathrm{P(X} \mid H = no) \\ &= 1/2 \times 3/4 + 1/2 \times 3/4 = 3/4 \end{aligned}$$

This value can be interpreted as the degree of dependency of *Flu* on *Headache* established by the above argument. This argument can be generalized and formulated for a measure of degree of dependency of an attribute set Q on an attribute set P

$$\mu'_P(Q) = \frac{1}{card(O)} \sum_{[o]_P} max_{[o]_Q} card([o]_Q \bigcap [o]_P) \tag{3.10}$$

The degree of dependency *Flu* on *Temperature* calculated by (3.9) is 3/4. The main difference between $\mu_P(Q)$ and $\mu'_P(Q)$ is that the latter measures the dependency of Q on P in maximizing the predicted membership of an instance in the family of equivalence classes generated by Q given its membership in the family of equivalence classes generated by P.

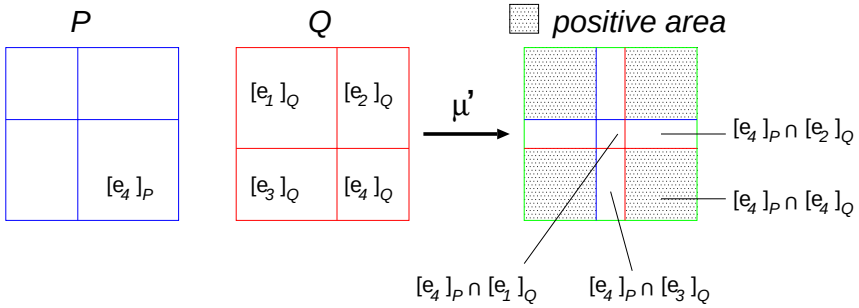

Figure 3.5: R-measure for attribute dependency

Proposition *For every set P and Q we have*

$$\frac{max_{[o]_Q} card([o]_Q)}{card(O)} \leq \mu'_P(Q) \leq 1 \tag{3.11}$$

Proof. Denote by M and N the numbers of equivalence classes regarding Q and P. Let $n_{..}$ denotes the total number of instances in O, $n_{i.}$ the number of instances from the i-th equivalence class regarding Q, $n_{.j}$ the number of instances with the equivalence class j-th equivalence class regarding P, and n_{ij} the number of instances in the intersection of these two classes, $i = 1, ..., M; j = 1, ..., N$. We rewrite (3.11) in following form

$$\frac{max_i n_{i.}}{n_{..}} \leq \frac{\sum_{1 \leq j \leq N} max_i n_{ij}}{n_{..}} \leq 1 \tag{3.12}$$

The right part of (3.12) is clear. We need to prove the left one, which is equivalent to

$$max_i n_{i.} \leq \sum_{1 \leq j \leq N} max_i n_{ij} \tag{3.13}$$

We prove it by induction on N. For $N = 1$ the inequality is clearly true. Assume that the inequality is true for N, we prove that it also to be true for $N + 1$. We have

$$\sum_{1 \leq j \leq N+1} max_i n_{ij} = \sum_{1 \leq j \leq N-1} max_i n_{ij} + max_i n_{iN} + max_i n_{iN+1}$$

Now we consider an equivalence relation P' corresponding to N equivalence classes which are the same as those of P but only its N-th equivalence class is the union of N-th and $N + 1$-th classes of P. For the sake of distinction, number this equivalence class N'-th. Suppose that $max_i n_{iN'} = n_{i^\star N'}$, we can see that

$$n_{i^\star N'} = n_{i^\star N} + n_{i^\star N+1} \leq max_i n_{iN} + max_i n_{iN+1}$$

From the induction assumption we obtain

$$\begin{aligned} max_i n_{i.} &\leq \sum_{1 \leq j \leq N-1} max_i n_{ij} + max_i n_{iN'} \\ &= \sum_{1 \leq j \leq N-1} max_i n_{ij} + n_{i^\star N'} \\ &= \sum_{1 \leq j \leq N-1} max_i n_{ij} + n_{ij} n_{i^\star N} + n_{i^\star N+1} \\ &\leq \sum_{1 \leq j \leq N-1} max_i n_{ij} + max_i n_{iN} + max_i n_{iN+1} \\ &= \sum_{1 \leq j \leq N+1} max_i n_{ij} \end{aligned}$$

□

From the theorem we have the following standardized version of $\mu^{'}$

$$\mu^{\star}{}_P(Q) = \frac{\sum_{[o]_P} max_{[o]_Q} card([o]_Q \bigcap [o]_P) - max_{[o]_Q} card([o]_Q)}{card(O) - max_{[o]_Q}} \tag{3.14}$$

and the inequality becomes

$$0 \leq \mu^{\star}_P(Q) \leq 1 \tag{3.15}$$

We can define that Q totally depends on P iff $\mu^{\star}_P(Q) = 1$; Q partially depends on P iff $0 < \mu^{\star}_P(Q) < 1$; Q is independent of P iff $\mu^{'}_P(Q) = 0$.

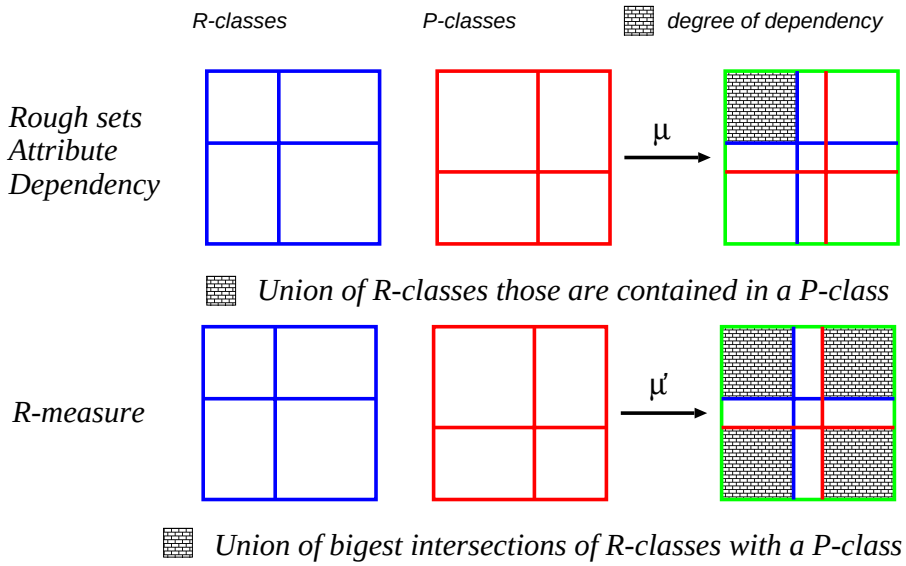

Figure 3.6: Comparison between the measure used in rough sets and R-measure

3.4.2 Application to Attribute Selection Problem

In (3.10), if we consider P a descriptive attribute and Q the class attribute, we can rewrite $\mu'_P(Q)$ in the form

$$\mu' = \sum_j p_{.j} max_i p_{i|j} \tag{3.16}$$

As this formula describes how much the class attribute depends on a descriptive attribute, we can naturally consider it as an candidate for a new attribute selection measure. However, despite the fact that it shows good results in some datasets, the results become unstable when both the vertical and horizontal sizes of data increase. The fact is that the measure is too greedy in finding "best" attributes for the front step while tree growing is a multistep procedure. An analysis based on the notion of impurity function gives us a clearer view of this phenomenon, and provides a basis to go from μ' to $\tilde{\mu}$ (R-measure) for DTL.

Let O be a set of instances with each object $o \in O$ belonging to one of the classes $C_1, C_2, ..., C_I$. Vector $PC =< p_1, p_2, ..., p_I >$ is the *class probability vector* of O, where each component p_i is the proportion of i-class objects.

Definition *Let O be a set of objects having a class probability of $PC =< p_1, p_2, ..., p_I >$. An impurity function is a function ϕ defined on PC with the properties*

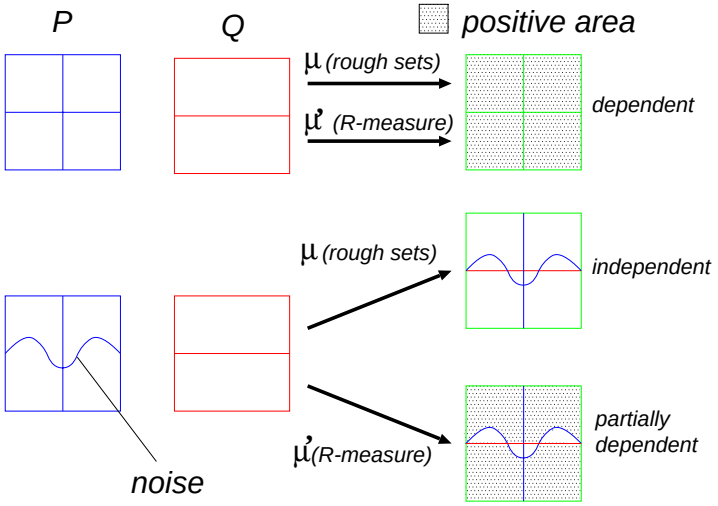

Figure 3.7: R-measure is more stable with noisy data

(i) ϕ is a maximum only at the point (1/I , 1/I , ..., 1/I),

(ii) ϕ attains its minimum only at the points (1, 0, ..., 0), (0, 1, 0, ..., 0), ..., (0, 0, ..., 0, 1),

(iii) ϕ is a symmetric function of $p_1, ..., p_I$.

Given an impurity function ϕ we can define an attribute selection measure as

$$\psi = \sum_j p_{.j}(1 - \phi(p_{1|j}, ..., p_{I|j})) \tag{3.17}$$

where I is the number of classes. In the case of $\mu^{'}$ the corresponding impurity function can be defined as

$$\phi(p_1, ..., p_J) = 1 - max_j p_j \tag{3.18}$$

As show in [9], decision tree learning needs another requirement for impurity function, otherwise the corresponding attribute selection measure will have the defects of degeneracy and not good for the overall multistep tree growing procedure

(iv) $\frac{\partial^2 \phi}{\partial p_j^2} < 0$.

Furthermore, we prefer that $\frac{\partial^2 \phi}{\partial p_j^2}$ not only is negative but also is a constant. It makes ϕ not only downwards with respect to any of the components of PC but also downwards in a constant degree. The most obvious modification of our impurity function is

$$\phi(p_1, ..., p_J) = 1 - (max_j p_j)^2 \tag{3.19}$$

and the corresponding attribute selection measure will be

$$\tilde{\mu} = \sum_j p_{.j}(max_i p_{i|j})^2 \tag{3.20}$$

We call $\tilde{\mu}$ in (3.20) R-measure and for arbitrary attribute sets P and Q it becomes

$$\tilde{\mu}_P(Q) = \frac{1}{card(O)} \sum_{[o]_P} max_{[o]_Q} \frac{card([o]_Q \bigcap [o]_P)^2}{card([o]_P)} \tag{3.21}$$

In the next chapter we carry an experimental comparative evaluation among $\tilde{\mu}$ and three other attribute selection measures.

3.5 Experimental Results

To evaluate R-measure in terms of attribute selection measures, we carried out an experimental comparative evaluation on 32 datasets for four models. These models are formed by combining fixed methods of error-complexity pruning and entropy-based discretization with four attribute selection measures: gain-ratio (c), gini-index (g), χ^2 (χ) and R-measure (R).

In order to study the effect of noise attributes on each measure we used the same datasets from the work of Lim et al. [55]. Besides 16 original datasets, most of them are from UCI repository, there are 16 datasets created by adding noise attributes. The numbers and types of noise attributes added are given in right panel of Table 3.1. This table also contains the name, number of classes, and number of attributes of the original datasets.

Table 3.2 presents experimental results of size and error rates (both unpruned and pruned trees) of these four measures on each datasets.

Some observations and conclusions can be drawn from these results.

- On the original datasets, gain-ratio, R-measure, gini-index, and χ^2 attained the lowest error rates 9, 7, 6, and 4, respectively. We notice that while on a majority of datasets the error rates of different measures are significantly different, on some datasets all or almost the measures attained the same error rates. If we consider the fact that χ^2 attained no unique lowest value and that it had comparatively high values in general, we can say that this measure showed a poor performance in our

Table 3.1: Datasets used in experiment

Name	Examples	Class	Original attributes NumAtt	Original attributes NomAtt	Noise attributes NumAtt	Noise attributes NomAtt
Wisconsin breast cancer (bcw)	683	2	9		9	
Contraceptive method choice (cmc)	1473	3	2	7	6	
StatLog DNA (dna)	3186	3		60		20
StatLog heart disease (hea)	270	2	7	6	7	
Boston housing (bos)	506	3	12	1	12	
LED display (led)	6000	10		7		17
BUPA liver disorders (bld)	345	2	6		9	
PIMA Indian diabetes (pid)	532	2	7		8	
StatLog satellite image (sat)	6435	6	36		24	
Image segmentation (seg)	2310	7	19		9	
Attitude towards smoking (smo)	2855	3	3	5	7	
Thyroid disease (thy)	7200	3	6	15	4	10
StaLog vehicle (veh)	846	4	18		12	
Congressional voting (vot)	435	2		16		14
Waveform (wav)	3600	3	21		19	
TA evaluation (tae)	151	3	1	4	5	

evaluation. The gini-index is showed to be better as it attained the lowest error rates 6 times and the middle values on almost other datasets. Our evaluation confirm again the fact that there are significant differences between the attribute selection measures and also there is no absolute superior measure. In this evaluation the gain-ratio and R-measure overall attained lowest error rates. The gini-index and χ^2 showed average and high error rates accordingly.

- On the noisy datasets, R-measure is the most accurate measure, as it attained the lowest error rates on 10 datasets in comparing to 7, 5, and 2 datasets of gain-ration, gini-index, and χ^2. This experiment showed that R-measure is better than the other when dealing with noise.

- For the tree size, the gain-ratio is the only measure that showed a significant advantage due to the fact that the measure was designed with this bias in mind. However, in practice the differences between tree sizes are not very important when the trees are not very large. For the datasets (Spice, Waveform, Segmentation) on those we had rather big trees, the gain-ratio did not show any significant advantage. Only one thing worth to note is that in these datasets χ^2 always has biggest trees. The gain-ratio showed its advantage of smaller tree when trees are small or middle-size, but it did not when trees become big. Additionally, on the datasets every measure attained big trees χ^2 has noticeable bigger ones.

- R-measure showed a very promising result in the application to the problem of attribute selection in DTL, especially when dealing with noise. This make us believe that R-measure is an appropriate solution when we applying DTL to datamining problems where noisy data are very common.

Table 3.2: Experimental comparative evaluation results

		original data				noisy data			
		unpruned		pruned		unpruned		pruned	
dataset	measure	size	errors	size	errors	size	errors	size	errors
bwc	c	21.3	5.5	12.5	4.6	21.3	5.5	12.5	4.6
	g	23.3	5.7	12.1	5.1	41.3	5.7	30.1	5.1
	χ	21.3	5.7	10.1	5.1	21.3	5.7	10.1	5.1
	R	22.9	5.7	9.9	5.1	30.1	5.7	9.9	5.1
cmc	c	232.2	51.4	22.6	46.1	258.2	51.6	21.1	45.6
	g	319.2	53.2	26.8	48.1	367.2	53.0	57.8	48.0
	χ	321.8	53.1	28.2	47.5	360.6	52.7	61.2	48.9
	R	315.2	54.3	22.0	45.5	555.7	52.7	47.2	48.1
dna	c	267.8	8.5	125.8	7.1	271.0	8.4	121.0	7.1
	g	281.8	10.1	112.6	8.0	280.6	10.1	107.8	8.0
	χ	289.0	9.6	119.0	7.8	287.0	9.7	119.4	8.0
	R	310.6	11.5	103.0	9.3	311.4	12.0	96.2	9.2
hea	c	32.1	25.2	15.7	23.3	27.9	23.3	12.6	23.7
	g	51.5	25.9	23.8	23.0	63.9	24.4	40.7	23.0
	χ	48.2	25.9	14.6	23.3	50.4	24.4	17.7	23.0
	R	68.7	25.2	16.3	23.0	109.3	24.8	26.6	22.6
bos	c	77.2	24.9	17.1	22.3	77.2	24.9	17.1	22.7
	g	103.4	25.9	13.2	24.9	127.4	25.9	38.5	24.5
	χ	103.5	25.5	18.0	22.4	113.1	25.5	18.0	22.4
	R	107.8	25.9	15.1	25.1	184.6	25.9	22.5	25.1
led	c	97.8	27.0	78.4	26.9	950.8	38.2	83.0	26.8
	g	115.2	27.0	82.4	27.0	1162.8	40.5	90.4	27.3
	χ	113.4	26.9	83.0	26.9	1183.8	40.6	83.0	26.9
	R	129.4	26.9	105.4	26.8	1211.2	40.1	93.6	27.5
bld	c	3.0	37.1	3.0	37.1	3.0	36.6	3.0	36.6
	g	13.0	37.1	13.0	37.1	31.0	36.6	31.0	36.6
	χ	3.0	37.1	3.0	37.1	3.0	36.6	3.0	36.6
	R	3.0	37.1	3.0	37.1	3.0	36.6	3.0	36.6
pid	c	25.2	21.1	14.8	21.7	23.2	20.6	12.4	23.0
	g	37.0	23.0	13.0	21.3	45.0	21.3	32.8	20.4
	χ	36.8	23.0	10.6	21.3	44.2	21.3	11.8	23.3
	R	42.2	23.0	11.8	21.5	75.2	21.3	17.8	23.0
sat	c	1732.1	21.1	338.5	19.4	1856.7	19.3	371.4	17.6
	g	1626.9	19.0	484.7	17.3	1689.1	18.9	420.3	18.4
	χ	1757.5	19.9	602.0	18.9	1844.0	19.7	579.6	17.5
	R	1672.3	19.6	357.9	17.1	1722.2	18.7	403.6	17.0
seg	c	360.2	7.3	324.2	7.1	383.9	7.1	335.7	7.2
	g	308.5	6.8	258.1	6.9	367.0	6.5	321.3	6.6
	χ	346.4	7.3	268.2	7.8	420.0	7.5	384.0	7.7
	R	313.2	7.3	240.1	7.9	368.0	6.5	307.1	6.6
smo	c	77.1	32.4	1.0	30.5	91.2	34.0	1.0	30.5
	g	102.1	33.1	1.0	30.5	125.7	34.2	1.0	30.5
	χ	101.3	33.0	1.0	30.5	110.5	32.7	1.0	30.5
	R	118.7	33.3	1.0	30.5	115.3	34.9	1.0	30.5
thy	c	55.1	1.0	47.3	1.0	66.4	1.0	50.2	1.0
	g	68.4	0.9	62.4	0.9	70.7	1.0	67.5	1.0
	χ	70.8	1.1	56.2	1.1	80.3	1.2	57.4	1.1
	R	67.7	0.9	56.0	0.9	71.9	0.9	60.8	0.9
veh	c	169.3	33.2	93.7	33.4	169.3	33.2	93.7	33.4
	g	229.6	33.6	66.0	30.8	253.6	33.6	90.0	30.7
	χ	238.5	32.6	92.2	31.9	250.5	32.6	92.2	31.9
	R	234.8	33.4	73.7	30.7	417.2	33.4	94.2	31.1
vot	c	18.1	5.7	5.8	4.5	23.2	7.5	4.0	4.3
	g	21.1	6.4	8.5	5.2	25.3	7.0	7.9	4.3
	χ	20.8	6.6	8.5	5.5	25.3	5.7	7.9	4.3
	R	21.4	6.4	8.2	5.2	27.1	7.5	7.9	4.1
wav	c	851.9	29.8	162.3	26.8	837.5	27.9	159.5	25.4
	g	1254.2	27.6	193.2	28.1	1677.8	28.4	198.3	27.9
	χ	1137.1	30.2	201.4	29.6	1558.4	29.5	243.3	27.2
	R	1884.6	26.2	184.7	25.9	2007.3	27.4	195.8	24.8
tae	c	50.8	63.8	20.8	63.1	39.8	58.1	28.9	60.0
	g	141.4	57.5	87.5	58.8	122.6	53.8	68.3	56.2
	χ	139.4	58.1	85.9	59.4	117.0	55.6	70.4	56.9
	R	139.6	60.6	70.1	62.5	131.3	55.0	59.8	56.2

3.6 Summary

In this research issue, to develop the a new criterion for attribute selection, we have proposed a variant of attribute dependency measure of the probabilistic model of rough sets [73] in order (1) to overcome the limitations of the original model in case of noisy data, (2) to make the model more coherent, and (3) to preserve the convenience of non-parameter. Based on this model, *R-measure* is developed to measure how much the class attribute depends on a predictive attribute. Using *R-measure* as an attribute selection criterion, an experimental comparative evaluation on 32 datasets—half of them are noisy—showed that it can be considered as a good alternative criterion for attribute selection, especially for noisy data.

Chapter 4

A Scalable Algorithm for Rule Post-Pruning of Large Decision Trees

4.1 Introduction

Data mining algorithms have usually to deal with very large databases. For the prediction data mining task, in addition to the requirements of high accurate and understandability of discovered knowledge, the mining algorithms must be scalable, i.e., given a fixed amount of main memory, their runtime increases linearly with the number of records in the input database.

Decision tree learning has become a popular and practical method in data mining because of its significant advantages: the generated decision trees usually have acceptable predictive accuracy; the hierarchical structure of generated trees makes them quite easy to understand if trees are not large; and especially the learning algorithms, which employ the *divide-and-conquer* (or simultaneous covering) strategy to generate decision trees, do not require complex processes of computation. However, it happens that in certain domains the comprehensibility and predictive accuracy of decision trees decrease considerably because of the problem known as *subtree replication* [72] (when the subtree replication occurs, identical subtrees can be found at several different places in the same tree structure).

Figure 4.1 shows an example domain where subtree replications is inevitable. The target concept, described by the simple rules in Figure 4.1a, leads to the rather complex

decision tree in Figure 4.1b because information must be replicated in the subtrees attached to nodes three and five. The complexity of the decision tree cannot be reduced by changing the order in which the attributes are used for splitting: every correct decision tree for this problem has the same size.

Figure 4.1 illustrates that subtree replication occurs whenever an accurate description of the target concept consists of a set of rules, the rules include tests on disjoint sets of attributes, and they cannot be decoupled by different tests on the same attribute. Unfortunately there are many situations where this scenario is likely to hold in practice. For example, consider a disease that can be diagnosed using either one of two groups of symptoms, and the two groups are not mutually exclusive. In this application a decision tree is not a perspicuous representation of the underlying concept. Fortunately there is an alternative to decision trees that avoids the problem of subtree replication: the target concept can be modeled as a set of rules, where each rule is a conjunction of attribute-value tests together with a corresponding class assignment.

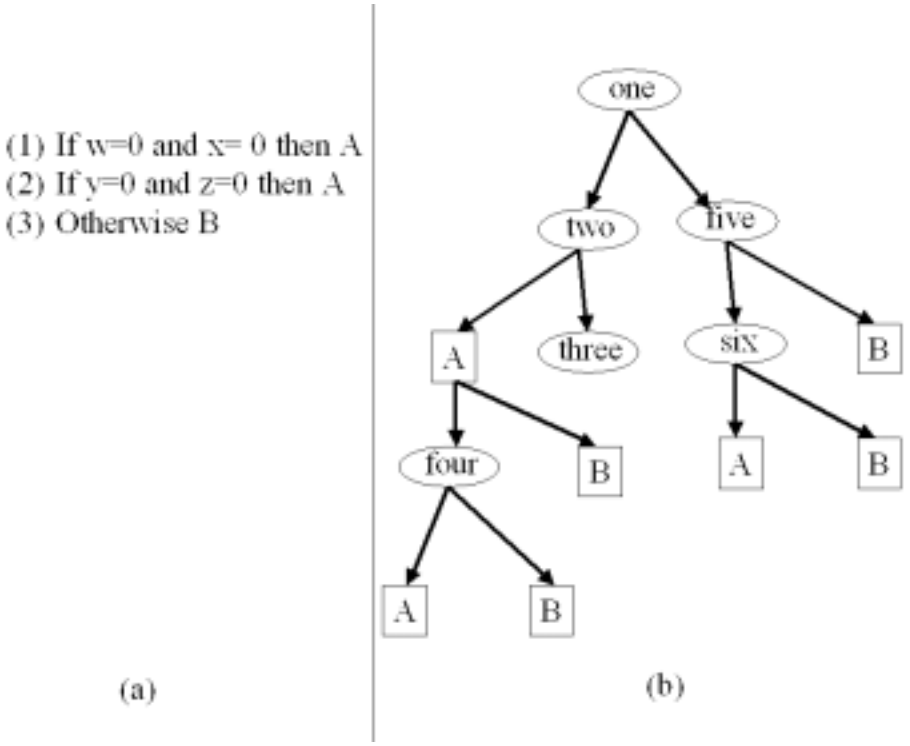

Figure 4.1: An example of subtree replication

The solution to the problem of subtree replication in the most well-known decision tree learning system C4.5 [83] is to convert a generated decision tree into a set of rules using a *post-pruning* strategy [62]. The conversion of trees into rules is not only an effective way to avoid the subtree replication problem but also offers other significant advantages: while large trees generated from large datasets are difficult to understand, discovered knowledge in form of rules is much easier to understand. Also, in our practical experience domain experts often feel more comfortable to analyze and validate rules than trees if trees become large. Moreover, it appears that the generated rule sets usually have equal or higher predictive accuracy than the original decision tree. However, the C4.5rules algorithm is not scalable to large databases as the simulated annealing, which is employed to achieve an optimal generalization, requires $O(n^3)$ time complexity where n is the number of records in the input database [15].

The *separate-and-conquer* (or simultaneous covering) strategy is an alternative approach to learn rules directly from databases. The most well-known separate-and-conquer algorithms include CN2 [17], REP [11], IREP [27], RIPPER [16], PART [25]. Among them, CN2 and REP also require a computation with high complexity, and therefore cannot be applicable to large data bases. IREP and RIPPER solve the problem of complexity by using a scheme called *incremental pruning*. The result is that they can run very fast and generate small rule sets with acceptable predictive accuracy. However, incremental pruning may lead to the problem of *overpruning* (or hasty generalization) that reduces the accuracy of the algorithms in many cases. PART [25] is an attempt to combine divide-and-conquer and separate-and-conquer strategies, and was claimed to be effective and efficient.

Our research concerns with scalable algorithms for rule-post pruning from large decision trees. In particular it proposes a solution to the problem of high complexity in C4.5rules by using a scheme similar to incremental pruning. The essence of the proposed algorithm is to avoid the problem of overpruning by appropriate improvements in incremental pruning. Experiments show that the proposed algorithm produces rule sets that as accurate as those generated by C4.5 and is scalable for very large data sets.

4.2 General Concepts of Pruning

4.2.1 Why Pruning is Required?

A learning algorithm can generate a too complex classifier that "overfits the data" by inferrring more structure than is justified by the training cases. We reintroduce an example taken in the book C4.5 [83] to illustrate this effect. It is also applicable to other rule learning algorithms.

The effect is seen in the extreme example of random data in which the class of each case is quite unrelated to its attribute values. We constructed an artificial dataset of this kind with ten attributes, each of which took the value 0 or 1 with equal probability. The class was also binary, *yes* with probability 0.25 and *no* with probability 0.75. One thousand randomly generated cases were split into a training set of 500 and a test set of 500. From this data, C4.5's initial tree-building routine produces a a nonsensical tree of 119 nodes that has an error rate of more than 35% on the test cases.

This small example illustrates the twin perils that can come from too gullible acceptance of the initial tree: It is often extremely complex, and can actually have a higher error rate than a simpler tree. For the random data above, a tree consisting of just the leaf no would have an expected error rate of 25% on unseen cases, yet the elaborate tree is noticeably less accurate. While the complexity comes as no surprise, the increased error attributable to overfitting is not intuitively obvious. To explain this, suppose we have a two-class task in which a case's class is inherently indeterminate, with proportion $p \geq 0.5$ of the cases belonging to the majority class (here *no*). If a classifier assigns all such cases to this majority class, its expected error rate is clearly $1 - p$. If, on the other hand, the classifier assigns a case to the majority class with probability p and to the other class with probability $1 - p$, its expected error rate is the sum of

- the probability that a case belonging to the majority class is assigned to the other class, $p(1 - p)$, and
- the probability that a case belonging to the other class is assigned to the majority class, $(1 - p)p$

which comes to $2p(1-p)$. Since p is at least 0.5, this is generally greater than $1-p$, so the second classifier will have a higher error rate. Now, the complex decision tree bears a close resemblance to this second type of classfiers. The tests are unrelated to class so, like a symbolic pachinko machine, the tree sends each case randomly to one of the leaves. We would expect the leaves themselves to be distributed in proportion to the class frequencies in the training set. Consequently, the tree's expected error rate for the random data above is 2.25.75 or 37.5% quite close to the observed value.

The above example and analysis is not only applicable for artificial datasets. Noise in real world data also lead to that effect. Noisy data are a problem for many learning algorithms, because it is hard to distinguish between rare exceptions and erroneous examples. The algorithm forms a complete and consistent theory, i.e. it tries to cover all of the positive and none of the negative examples. In the presence of noise it will therefore attempt to add literals to rules in order to exclude positive examples that have a negative classification in the training st and add rules in order to cover negative examples that have erroneously been classified as positive. Thus complete and consistent theories generated from nosy examples are typically very complicated and exhibit low predictive accuracy on classifying unseen examples. This problem is known as *overfitting the noise*.

One remedy for the overfitting problem is to try to increase the predictive accuracy by considering not only complete and consistent theories, but also approximate, but simpler theories. A simple theory that covers most positive examples and excludes most negative examples of the training set will often be more predictive than a complete and consistent, but very complex theory. Such a bias towards simpler theories has been termed *overfitting avoidance bias* [95].

Several algorithms, such as CLASS [99], rely on the noise handling capabilities of search heuristics like the Laplace-estimate, which can prefer rules that cover only a few negative examples over clauses that cover no negative examples if the former cover more positive examples. Other algorithms such as PROGOL [66] can also reply on a severely constrained hypothesis language which is unlikely to contain overfitting hypotheses. On the other hand, a wide variety of algorithms employs techniques that are specifically designed for overfitting avoidance. The remainder of this section is devoted to a discussion of two most

common approaches to pruning in rule learning algorithms. Work that directly relate to our research will be described in the next section.

4.2.2 Pre-pruning

Pre-pruning methods deal with noise during concept generation. They are implemented into a stop criterion of learning algorithms. Their basic idea is to stop the refinement of rules although they may still be over-general. Thus, rules are allowed to cover a few negative examples if excluding the negative examples is esteemed to be too costly by the stopping criterion.

The most commonly used stopping criteria are

- *Minimum Purity Criterion*: This simple criterion requires that a certain percentage of the examples covered by the learned rules is positive. It is for example used in the SFOIL algorithm [76] as a termination criterion for the stochastic search. In FOIL [82] this criterion is used as a stop condition: When the best rule is below a certain purity threshold (usually 80%) it is rejected and the learned theory is considered to be complete.

- *Encoding Length Restriction*: This heuristic used in the ILP algorithm FOIL is based on the Minimum Description Length Principle [89]. It tries to avoid learning complicated rules that cover only a few examples by making sure that the number of bits that are needed to encode a clause is less than the number of bits needed to encode the instances covered by it.

- *Significance Testing* was first used as rule stopping criterion in the propositional CN2 induction algorithm [17] and later on in the relational learner mFOIL [18]. It tests for significant differences between the distribution of positive and negative examples covered by a rule and the overall distribution of positive and negative examples. For this test it exploits the fact that the likelihood ratio statistic that can be derived from the J-measure as $LRS(r) = 2(P + N)J(r)$ is approximately distributed χ^2 with 1 degree of freedom. Insignificant rules can thus be rejected. In BEXA [97] this test is also used for comparing the distribution of instances covered

by a rule to that of its direct predecessor. If the difference is insignificant, the rule is discarded.

- *The Cutoff Stopping Criterion* compares the heuristic evaluation of a literal to a user set threshold and only admits literals that have an evaluation above this *cutoff*. It has been used in the relational separate-and-conquer learning system FOSSIL [28]. Under the assumption that the search heuristic returns values between 0 and 1, FOSSIL will fit all of the data at $cutoff = 0$ (no pre-pruning). On the other hand, $cutoff = 1$ means that FOSSIL will learn an empty theory (maximum prepruning). Values between 0 and 1 trade off the two extremes. For the correlation heuristic, a value of 0.3 has been shown to yield good results at different training set sizes and at differing levels of noise as well as across a variety of test domains.

4.2.3 Post-pruning

While pre-pruning techniques try to account for the noise in the data while constructing the final theory, post-pruning methods attempt to improve the learned theory in a post-processing phase. A commonly used post-processing technique aims at removing redundant conditions from the body of a rule and removing unnecessary rules from the concept. The latter technique has already been used in various version of the AQ algorithm [61]. The basic idea is to test whether the removal of a single condition or even of an entire rule would lead to a decrease in the quality of the concept description, usually measured in terms of classification accuracy on the training set. If this is not the case, the condition or rule will be removed.

This framework has later been generalized in the POSEIDON system [7]. POSEIDON can simplify a complete and consistent concept description, which has been induced by AQ15 [61], by removing conditions and rules and by contracting and extending interval and internal disjunctions. POSEIDON successively applies the operator that results in the highest coverage gain as long as the resulting theory increases some quality criterion.

This method can be easily adopted for avoiding overfitting of noisy data. A frequently used approach is to maximize the predictive accuracy measured on a separate set of data

that has not been available to the learner during theory construction. This method has been suggested in [72] based on similar algorithms for pruning decision trees [80]. Before learning a complete and consistent concept description the training set is split into two subsets: a *growing set* (usually 2/3) and a *pruning set* (1/3). The concept description that has been learned from the growing set is subsequently simplified by greedily deleting conditions and rules from the theory until any further deletion would result in a decrease of predictive accuracy measured on the pruning set.

Simplifications that are usually tried are deleting an entire rule, or deleting the last condition of a rule as in *reduced error pruning* (REP) [11]. Other algorithms employ additional simplification operators like deleting each condition of a rule [27], deleting a final sequence of conditions [15], finding the best replacement for a condition [100], and extending and contracting internal disjunctions and intervals [7]. If the accuracy of the best simplification is not below the accuracy of the unpruned theory, REP will continue to prune the new theory. This is repeated until the accuracy of the best pruned theory is below that of its predecessor.

Prunk and Pazzani (1991) have empirically shown that REP can learn more accurate theories than FOIL, which uses pre-pruning. However, post-pruning has also several disadvantages, most notably efficiency. Cohen (1993) has shown that REP has a time complexity of $O(n^4)$ on purely random data. Therefore he proposed GROW a new pruning algorithm based on a technique used in the GROVE learning system [72]. Like REP, GROW first finds a theory that overfits the data. But instead of pruning the intermediate theory until any further deletion results in a decrease of accuracy on the pruning set, generalizations of rules from this theory are successively selected to form the final concept description until no more rules will improve predictive accuracy on the pruning set. Thus GROW performs a top-down search instead of REP's bottom-up search. For noisy data the asymptotic costs of this pruning algorithm have been shown to be below the costs of the initial phase of overfitting.

4.3 Related Work

A variety of approaches to learning rules have been investigated. One is to begin by generating a decision tree, then to transform it into a rule set, and finally to simplify the rules (the *divide-and-conquer* strategy as used in the system C4.5 [83]). Another is to use the *separate-and-conquer* strategy [72] to generate and an initial rule set, then applying a rule pruning algorithm.

4.3.1 Rule Post-Pruning in C4.5

The rule learner in C4.5 does not employ a separate-and-conquer method to generate a set of rules—it achieves this by simplifying an unpruned decision tree using the decision tree inducer included in the C4.5 software. Then it transforms each leaf of the decision tree into a rule. This initial rule set will usually be very large because no pruning has been done. Therefore C4.5 proceeds to prune it using various heuristics.

First, each rule is simplified separately by greedily deleting conditions in order to minimize the rule's estimated error rate. Following that, the rules for each class in turn are considered and a "good" subset is sought, guided by a criterion based on the minimum description length principle. The next step ranks the subsets for the different classes with respect to each other to avoid conflicts, and determines a default class. Finally, rules are greedily deleted from the whole rule set one by one, so long as this decreases the rule set's error on the training data.

Unfortunately, the global optimization process is rather lengthy and time-consuming. Cohen [15] shows that C4.5 can scale with the cube of the number of examples on noisy datasets.

4.3.2 Other Related Rule Pruning Algorithms

The earliest approaches to pruning rule sets are based on *global optimization.* These approaches build a full, unpruned rule set using a separate-and-conquer strategy. Then they simplify these rules by deleting conditions from some of them, or by discarding entire

rules. The simplification procedure is guided by a pruning criterion that the algorithm seeks to optimize [11]. The optimized solution can only be found via exhaustive search. In practice, some heuristic searches are applied, but they are still quite time consuming.

There is a faster approach to rule pruning called incremental pruning that is introduced first in IREP [27], and also used in RIPPER and RIPPERk [16]. The key idea is to prune a rule immmediately after it has been built, before any new rules are generated in subsequent steps of the separate-and-conquer algorithm. By integrating pruning into each step of the separate-and-conquer algorithm, this approach can avoid the high complexity of a global optimization process.

4.3.3 The Problem of Overpruning

Table 4.1: Potential rules in a hypothetical dataset

Rule	Coverage			
	Training Set		Pruning Set	
	yes	no	yes	no
1: $A = true \rightarrow yes$	600	60	200	20
2: $A = false \wedge B = true \rightarrow yes$	1200	60	400	20
3: $A = false \wedge B = false \rightarrow no$	0	30	0	10

Although the incremental pruning used in IREP (and its variants) avoids the high complexity process of global optimization of C4.5rules, it may suffer the problem of overpruning or hasty generalization [25]. By using pre-pruning approach, the algorithm does not know about potential new rules when it consider a rule to prune; the pruning decisions are based on the accuracy estimation of the current rule only. In other words, the algorithm cannot estimate how the pruning decisions on a single rule will effect the accuracy of the whole final rule set. Therefore, it may happen that the pruning decisions increase the accuracy of the current rule but may in fact decrease the accuracy of the potential final rule set on the same estimation.

Table 4.1 shows a simple example of overpruning. The example is taken from [25] with a modification to make it easier to calculate. Consider a binary dataset with two attributes A and B; examples can belong to either class yes or no. There are three potential rules

on the dataset as shown on the table. Assume that the algorithm generates first rule

$$A = true \rightarrow yes. \tag{4.1}$$

Now consider whether the rule should be further pruned. Its error rate on the pruning set is 1/10, and the pruned rule

$$\rightarrow yes \tag{4.2}$$

has an error rate of 1/13, which is smaller, thus the rule will be pruned to that null rule. As the null rule covers all the data the algorithm stops and satisfies with a final rule set consisting only of that trivial rule. But the found rule set actually has a greater error rate compare comparing to 4/65 which is the error rate of the set three rules showed in the table. Note that this happens because the algorithm concentrates on the accuracy of rule 1 when pruning—it does not make any guesses about the benefits of including further rules in the classier.

4.4 A Scalable Algorithm for Rule Post-Pruning

As an attempt to solve the high complexity problem of C4.5rules we have developed a new algorithm that adopts a scheme similar to incremental pruning used in IREP, we have named it CABROrule as it is integrated in our decision tree learning CABRO [70]. Similar to C4.5rules, CABROrule uses a bottom-up search instead of IREP's top-down approach: The final rule set is found by repeatedly removing conditions and rules from an input unpruned rules rather than adding new rules to an initial empty set. In other words, CABROrule uses post-pruning approach in contrast to pre-pruning one used in IREP. By taking the advantage of working with a full grown rule set throughout the pruning process, we can improve the incremental pruning scheme in CABROrule to avoid the problem of hasty generalization.

Table 4.2: The Main Procedure of CABROrule

procedure $CABROrule(UnprunedSet, Data)$

$PrunedSet \leftarrow \emptyset$
while $(Data \neq \emptyset)$
  $Rule \leftarrow SelectRule(UnprunedSet, Data)$
  $UnprunedSet \leftarrow UnprunedSet \setminus \{Rule\}$
  $PrunedRule \leftarrow PruneRule(Rule, UnprunedSet, Data)$
  $PrunedSet \leftarrow PrunedSet \cup \{PrunedRule\}$
  $Data \leftarrow Data \setminus Match(PrunedRule, Data)$
return $PrunedSet$

4.4.1 Description of the Algorithm

Similar to C4.5rules, CABROrule begins with a set of unpruned rules. The rule set is taken directly from an unpruned decision tree where each rule corresponds to a path from the tree root and a leave node. To prune the rule set, CABROrule follows a separate-and-conquer strategy: first choosing one rule to prune at a time, then removing the covered examples and repeating the process on the remaining examples. Table 4.2 shows the main procedure of CABROrule, the procedure to prune a single rule is in Table 4.3.

The efficiency of CABROrule comes from the avoidance of the process of searching for an "optimized" subset of rules such as the one in C4.5rule. We will analyze the reason why C4.5rules requires a global optimization but CABROrule does not. In C4.5rules, each individual rule is pruned with respect to the *all* training data. Deleting conditions from a rule—and thereby increasing its coverage—ultimately may result in a rule set with many overlaps. The optimized exclusive subset of rules can only be found via exhaustive search. In practise, exhaustive search is infeasible and C4.5rules apply some heuristic approximations (two alternatives in C4.5rules are greedy search and simulated annealing), but even these approximate algorithms are quite time consuming. In contrast, each single rule in CABROrule is pruned with respect to the *remaining* training data after removing all examples covered by previous pruned rules. The exhaustion of training data serves as a stop condition. It is noticeable that while there is no natural order for rules generated by

Table 4.3: Pruning a Single Rule

procedure $PruneRule(Rule, UnprunedSet, Data)$

```
repeat
  Accuracy ← EstimateAccuracy({Rule} ∪ UnprunedSet, Data)
  DeltaAccuracyA ← 0
  for each (Condition ∈ Rule) do
    NewRule ← Rule \ Condition
    NewAccuracy ← EstimateAccuracy({NewRule} ∪ UnprunedSet, Data)
    NewDeltaAccuracy ← NewAccuracy − Accuracy
    if (NewDeltaAccuracy > DeltaAccuracy)
      DeltaAccuracy ← NewDeltaAccuracy
      BestCondition ← Condition
  if (DeltaAccuracy > 0)
    Rule ← Rule \ Condition
until (DeltaAccuracy < 0)
return Rule
```

C4.5rules, CABROrule generates ordered rule sets those sometimes are known as decision lists [90].

To calculate the complexity of CABROrule, we assume that the data set consists of n examples described by a attributes. To choose a condition to prune the procedure $PruneRule$ needs to examine all the conditions of the considered rule each require n tests on the training examples. Therefore complexity of pruning a condition is $O(an)$. Suppose the length of an unpruned rule is a, then pruning a rule requires $O(a^2n)$. If we assume that the size of the final theory is constant [27], the complexity will be linear to n. Because a decision tree can be built in time $O(nlogn)$, the overall cost to build a rule set from data is $O(nlogn)$. The complexity is the same as that of PART [25] and better than $O(nlog^2n)$ of IREP or RIPPER, or $O(n^3)$ of C4.5rules.

Before going to the next subsection which addresses the problem of overpruning we discuss briefly about the procedure *SelectRule* in CABROrule. There is only a number of input rules that have their chance to be considered to prune and add to the final rule set. Certainly, we want as many "significant" rules having that chance as possible. A measure

is necessary to judge the "significance" of a rule. In general, there are several existing measures that may be suitable for that purpose such as *relative frequency*, *m-estimate of accuracy*, or *entropy* [62]. However, when we apply CABROrule on an unpruned decision tree resulting from C4.5 we use the coverage of a rule as a criterion to select which rule will be prune first. That because when growing a decision tree, C4.5 already optimized each path (corresponding to an unpruned rule) by information gain. Therefore, it is reasonable that rules with larger coverage may be more important and need to be considered first in the pruning process. If CABROrule is applied on rule sets that grown by other algorithms, other criteria for selecting rules can be better choices.

4.4.2 Avoiding Overpruning

CABROrule uses a greedy search algorithm for pruning a single rule. At a time, the algorithm searches for a condition to prune. The pruning continues until the accuracy estimation cannot be improved anymore. A description of the algorithm for pruning a single rule is in Table 4.3.

To overcome the problem of overpruning in the original algorithm of incremental pruning used in IREP and its variants, CABROrule takes a different approach to estimate the accuracy when making pruning decisions. Instead of estimating the accuracy only on the rule under consideration, the procedure $EstimateAccuracy$ does estimation on that rule together with all remaining unpruned rules. As we have stated in the previous section, the overpruning occurs when pruning decisions on a rule improving the accuracy estimation of that rule, but in fact potentially reducing the accuracy of the final rule set. By taking into account of remaining unpruned rules when pruning a single rule, we can make sure that a condition will be pruned if that potentially improves the accuracy on the whole final rule set not only on that single rule locally.

We return to the example of overpruning in section 2 to illustrate the new approach. Assume that the CABROrule considers pruning rule 1 back to a null rule. Instead of estimating the accuracy of only rule 1 before and after pruning it, the algorithm does estimation on all three rules to make that pruning decision. From Table 4.2, we can see that after pruning rule 1, the accuracy estimation is 1/13 which less than 4/65 before the

pruning decision. Therefore the algorithm cancels that pruning decision and avoids a case of overpruning.

The problem of overpruning or hasty generation is not restricted to a particular method of accuracy estimation [25], and our solution to the problem does not depend on estimation methods. The estimation of reduce error pruning is used in the example only to make calculations easier. In CABROrule we uses pessimistic estimation [83] similar to the one used in C4.5 to estimate the accuracy of a rule set. The estimation is done by calculating the rule accuracy over the training data, then calculating the standard deviation in this estimated accuracy assuming a binomial distribution. For a given confidence level (we used 95% in our experiments), the lower-bound estimate is then taken as the measure of rule performance. The accuracy estimate of a rule set is the average of the estimates over its members with respect to their coverage on the data set.

4.5 Experimental Results

In order to evaluate the performance of CABROrule we designed two experiments. The first experiment evaluates the predictive accuracy of CABROrule comparing to C4.5 and C4.5rules, the second evaluates the run-time of CABROrule comparing to C4.5rules.

For the first experiment we used 31 standard datasets from UCI collection. The datasets and their characteristics, together with experimental results are listed in Table 4.4. We performed 10-fold cross-validation on these datasets with C4.5, C4.5rules and CABROrule. The same folds were used for each program. A numbers in the result columns is the average of error rates or size of rule sets over ten times of running. A symbol "•" in the last column indicates that CABROrule has an error rate lower than both C4.5 and C4.5rules on that dataset, while a "◦" indicates that CABROrule has an error rate lower than C4.5rules, and a "×" indicates that C4.5rules has an error rate lower than that of CABROrule.

We can observe from Table 4.4 that CABROrule outperforms C4.5rules on 15 over 31 datasets, among them there are 9 datasets CABROrule outperforms both C4.5rules and C4.5, whereas C4.5rules has a lower error rate comparing to CABROrule on 6 datasets

Table 4.4: Experimental Results

					C4.5		C4.5rules		CABROrule		
Dataset	#Exam	NumAtt	NomAt	Class	size	error	size	error	size	error	
anneal	898	6	32	5	60.0	4.3	11.3	4.4	12.0	3.1	•
audiology	226	0	69	24	49.0	9.3	21.0	8.8	22.0	8.8	
australian	690	6	9	2	34.5	15.2	13.2	16.2	9.6	14.5	•
auto	205	15	10	6	68.7	19.5	22.2	18.4	20.8	20.4	×
balance-scale	625	4	0	3	82.0	22.1	37.0	21.1	28.7	22.2	×
breast	699	9	0	2	27.4	5.1	9.0	4.6	8.2	4.9	×
breast-cancer	286	0	9	2	12.1	25.9	7.8	29.7	3.0	26.2	◦
german	100	7	13	2	86.0	29.7	19.7	29.6	14.6	29.1	•
glass	214	9	0	6	44.0	32.7	13.8	30.8	13.6	30.8	
glass2	163	9	0	2	23.4	21.9	8.1	20.2	8.0	20.2	
heart	303	6	7	2	24.0	12.2	8.8	14.4	7.2	11.5	•
hepatitis	155	6	13	2	18.6	25.2	8.4	20.1	6.7	21.4	×
horse-colic	168	7	15	2	8.2	15.2	5.9	16.0	4.1	14.7	•
hypothyroid	3772	7	22	4	12.2	0.6	6.0	0.6	5.1	0.6	
ionosphere	351	34	0	2	25.0	9.4	9.1	8.8	9.2	8.8	
iris	150	4	0	3	8.8	5.3	4.1	4.6	4.0	4.6	
labor-neg	57	8	8	2	5.7	19.3	4.0	21.0	2.6	19.3	◦
lymphography	148	3	15	4	26.9	22.8	9.6	22.8	9.2	22.9	×
mushroom	8124	0	22	2	29.7	0.0	17.0	0.0	16.9	0.0	
pima	768	8	0	2	45.2	25.7	10.7	26.3	9.9	25.7	◦
primary-tumor	339	0	17	21	77.8	59.3	17.1	60.2	13.9	59.9	◦
segment	2310	19	0	7	87.0	2.8	28.2	3.7	27.6	3.7	
sick-euthyroid	372	7	22	2	24.6	2.2	12.0	2.4	9.2	2.2	◦
sonar	208	60	0	2	27.2	28.9	8.7	30.3	8.7	30.3	
soybean-large	683	0	25	19	94.9	7.8	35.8	7.0	3.1	6.7	•
splice	3190	0	61	3	220.2	5.9	74.1	6.5	60.0	6.0	◦
vehicle	840	18	0	4	135.8	28.7	26.6	27.1	25.9	26.8	•
vote	435	0	16	2	13.0	6.0	6.4	5.3	5.1	6.2	×
waveform-21	301	21	0	3	542.2	23.2	68.1	22.4	67.6	22.3	•
waveform-40	5002	34	0	3	584.6	24.9	66.1	23.4	68.0	23.0	•
zoo	101	1	15	7	17.4	7.6	7.8	7.6	7.8	7.6	

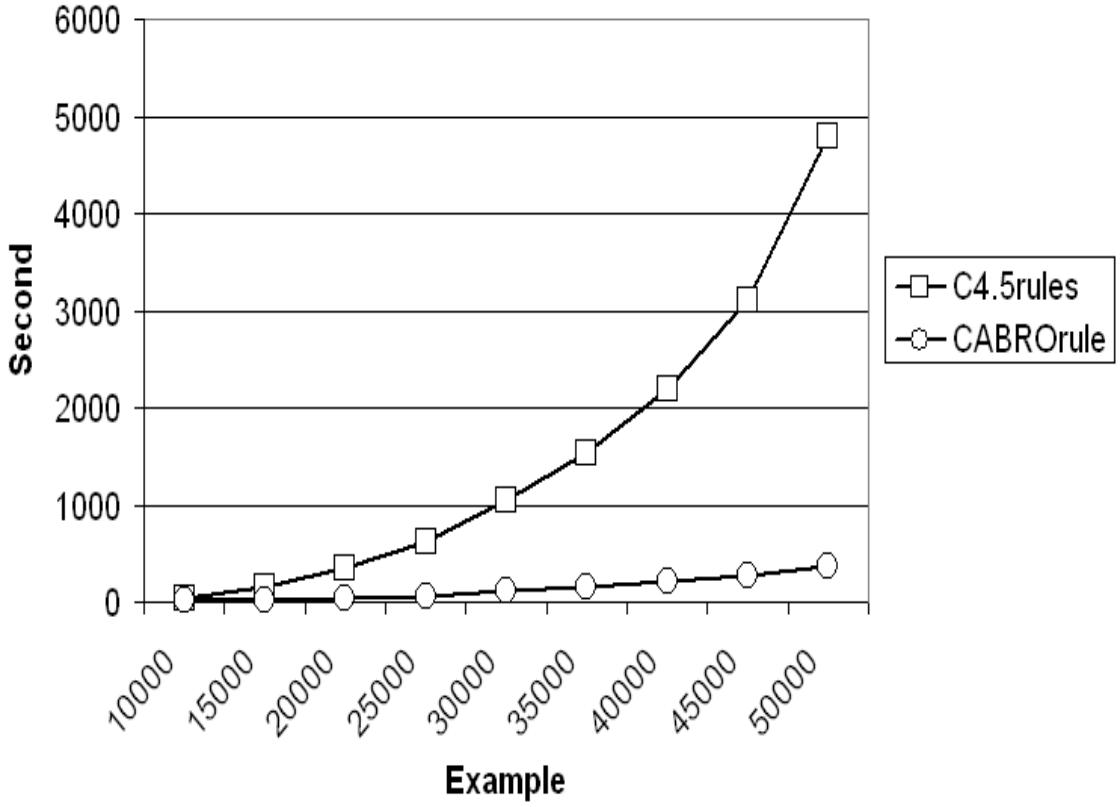

Figure 4.2: Comparison of Running Time

(totally C4.5rules has lower error than C4.5rules on 15 datasets, while is with higher error rate on 4 datasets). In some datasets the differences between error rates are too small to say that they are significant, but this experiment showed that CABROrule at least as good as C4.5rules if not better in the predictive accuracy.

About the size of rule sets, CABROrule generated smaller rule sets on a major number of datasets comparing to C4.5rules, and both reduce the number of rules comparing to C4.5 substantially. That reduction, in many case, will increase the understandability of result models, and this experiment reconfirms the advantage of transforming decision trees to rules.

In order to evaluate the efficiency of CABROrule, the second experiment is done with the census-income dataset. We began with 10000 examples and repeatedly ran CABROrule and C4.5rules, each time with a bigger number of examples, to learn in what order the run-time increases according to the size of data. Figure 1 is the graph drawn

from the experiment results. This graph confirms and illustartes that the run-time of C4.5rules is higher than $O(n^2)$, while the run-time of CABROrule is about $O(nlogn)$ that confirms our calculation about the algorithm complexity in the previous section.

Some significant conclusions can be drawn from these two experiments:

- By using incremental pruning approach to post-pruning problem, CABROrule can reduce the run-time substantially in comparison to C4.5rules. It allows us to apply the algorithm to large datasets that are very common in data mining.
- There is no lost in criteria of predictive accuracy and model size. In fact, there is some gain in accuracy, and CABROrule usually generates smaller rule sets in comparison to C4.5rules.
- Transferring decision trees to rules may increase both understandability and predictive accuracy of models.

4.6 Summary

This chapter has presented a new algorithm for rule post-pruning of decision trees. It can be considered an alternative algorithm for C4.5rules when the input data become very large. The problem of high complexity in C4.5 is solved by adopting an incremental pruning scheme. However the algorithm does not suffer the problem of hasty generalization such as in the original incremental pruning approach. Experiments have shown that the new algorithm generates rule sets as accuracy as those of C4.5 but with far less time of computation.

Chapter 5

Visualizing Large Decision Trees

5.1 Introduction

Learning decision trees from large datasets is quite different from small or moderately sized datasets. We have to face with new problems that we may never see previously such as converting trees into rules or understanding, accessing, and manipulating large trees, etc. For example, on a workstation Alpha 21264 (OS: Digital UNIX V4.0E; Clock frequency: 500MHz; RAM: 2GB), in our experiments the well-known system C4.5 [83] – from the U.S. census bureau dataset consisting of 199,523 instances with 32 symbolic attributes and 8 numeric attributes, and size of 103 MB [67] – produces a pruned tree after ten minutes but cannot convert the tree into a set of rules after two days.

```
leaves = norm:
— seed-discolor = absent:
— — temp = lt-norm: rhizoctonia-root-rot (19.0/1.3)
— — temp = norm: anthracnose (24.0/1.3)
— — temp = gt-norm: anthracnose (0.0)
— seed-discolor = present:
— — canker-lesion = dna: diaporthe-pod-stem-blight (5.5/1.2)
— — canker-lesion = brown: purple-seed-stain (0.0)
— — canker-lesion = dk-brown-blk: purple-seed-stain (0.0)
— — canker-lesion = tan: purple-seed-stain (9.0/1.3)
...
```

Figure 5.1: Part of decision tree displayed by C4.5

Though decision trees are a simple notion, we can understand their content and hierarchical structure easily if they are small but cannot understand or understand difficultly

if they are large. Research on visualization of decision trees has recently received a great attention from the KDD (knowledge discovery and data mining) community because of its practical importance. Many works have been done, e.g., the 3D Tree Visualizer in system MineSet [12], CAT scan (classification aggregation tablet) for inducing bagged decision trees [87], the interactive visualization in decision tree construction [4], the tree visualizer with a tree map in system CART [9] of Salford Systems, etc. However, it is still difficult to view and navigate large trees with these systems. On the other hand, new approaches in information visualization field for representing large hierarchical structures, e.g., cone trees [91], hyperbolic trees [53], have not been well considered in AI, machine learning and data mining.

Our research concerns with interactive visualization in learning decision trees, in particular large decision trees. The work was motivated by the need of an efficient tree visualizer when we apply our system CABRO [70] to large datasets. Section 5.2 of this chapter addresses the requirements for tree visualizers and related works. Section 5.3 describes some related work in the field of information visualization. Section 5.4 presents the tree visualizer of our system CABRO. Section 5.5 describes how this tree visualizer is used in the learning process.

sectionVisualization Requirements

The decision tree structure contains two kind of information: structural information associated with the tree, and content information associated with each node. Based on our DTL task analyses, we specify the primary requirements for tree visualizers, which share many common points with [41], as follows:

- *Embedded in the DTL process*: Structural and content information should be available not only after inducing decision trees but also during the learning process. The tree visualizer must provide the ability to quickly access and manipulate a large tree according to operations in decision tree learning.
- *Comprehension*: The tree visualizer must facilitate the understanding of tree structure and its relation to nodes in focus.
- *Efficiency*: Efficient use of space and visual tools is essential for the tree visualizer

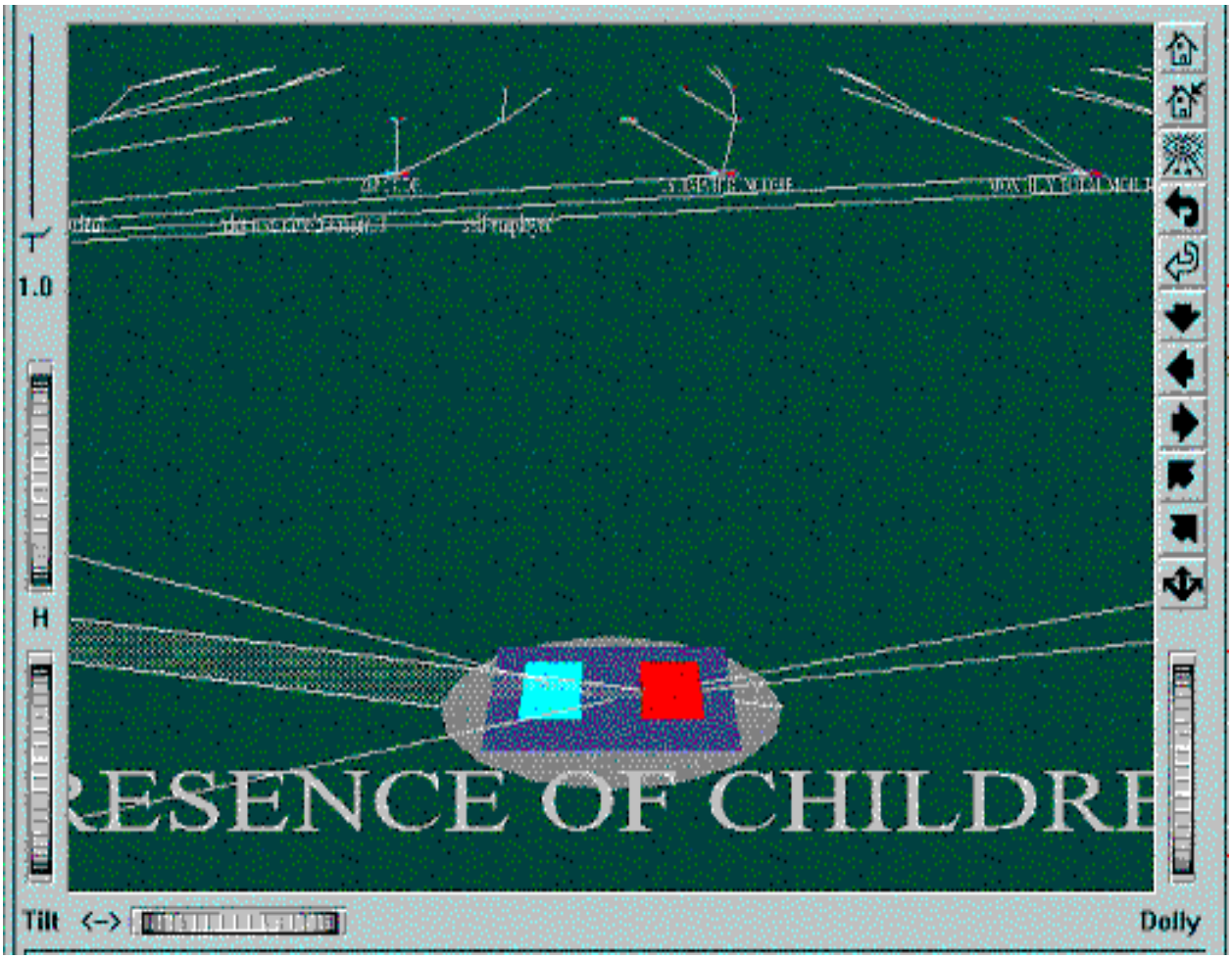

Figure 5.2: Difficult to navigate large trees in Mineset

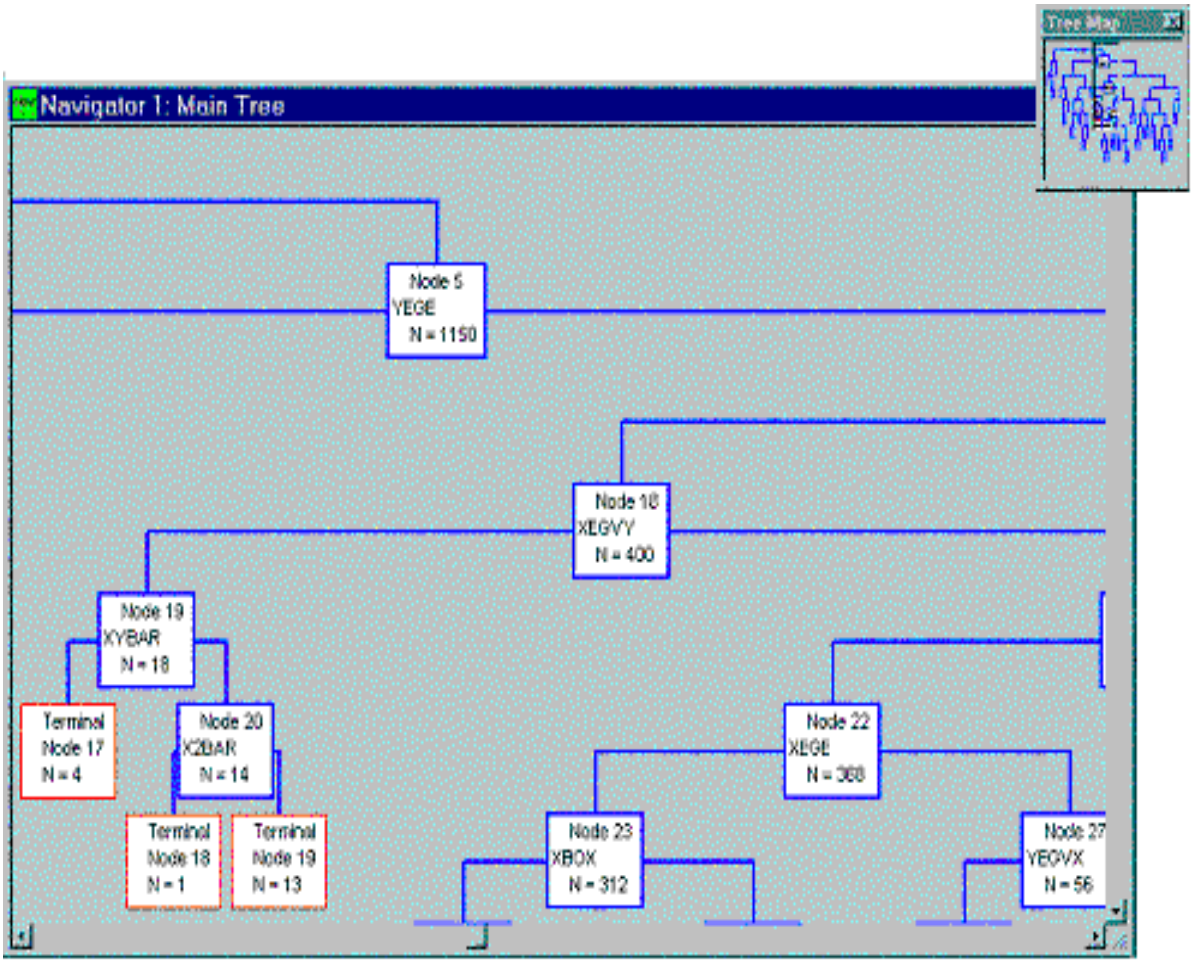

Figure 5.3: Difficult to navigate large trees in CART

to deal with large trees.

- *Interactivity*: Interactive control over the structure and the ability of users to customize the layout to meet their current needs and interests are essential.

- *Esthetics*: Drawing and feedback must be esthetically pleasing.

Most decision tree learning systems employ visualization methods that belong to one of two categories: outlines or node-link diagrams.

The *outline* methods to represent trees are similar to that of PC Shell under DOS or Microsoft Windows. Basically, the number of display lines required in the outline method is equal to the number of nodes in the hierarchy, but with the zooming functions the tree can be reduced dynamically to a smaller one with size is linearly proportional to the number of nodes. However, this method is somehow poor to associate with graphic operators in order to deal with large trees. System C4.5 and its successor C5.0 are in this category. Figure 5.1 shows a portion of a decision tree displayed by C4.5 from the small soybean dataset [67]. From the large U.S. census bureau dataset, C4.5 produces a decision tree of nearly 18,500 nodes with 2624 leaf nodes (about 1,850 times larger than this portion). It is extremely difficult, even impossible, to understand such a big tree in this outline representation.

The *node-link diagram* 5.11 has a main advantage of being easy to understand as it is a natural mapping of structured relationships. However, the original node-link diagram does not use efficiently the available display space and therefore is not adequate for large trees. Different attempts have been done to overcome this limitation which can be roughly classified into two groups of 2D and 3D visualizers.

Many tree learning systems employ 2D tree visualizers. CART (see [9] – the widely used system of Salford Systems – has a 2D tree visualizer associated with a *tree map* that provides an overview of the tree. Another 2D tree browser having good features of multi-level dynamic queries and pruning is developed in [51].

Two special 2D tree visualizers are that of hyperbolic trees and treemap. The *hyperbolic tree* method at Xerox is excellent for visualizing and manipulating large trees. It

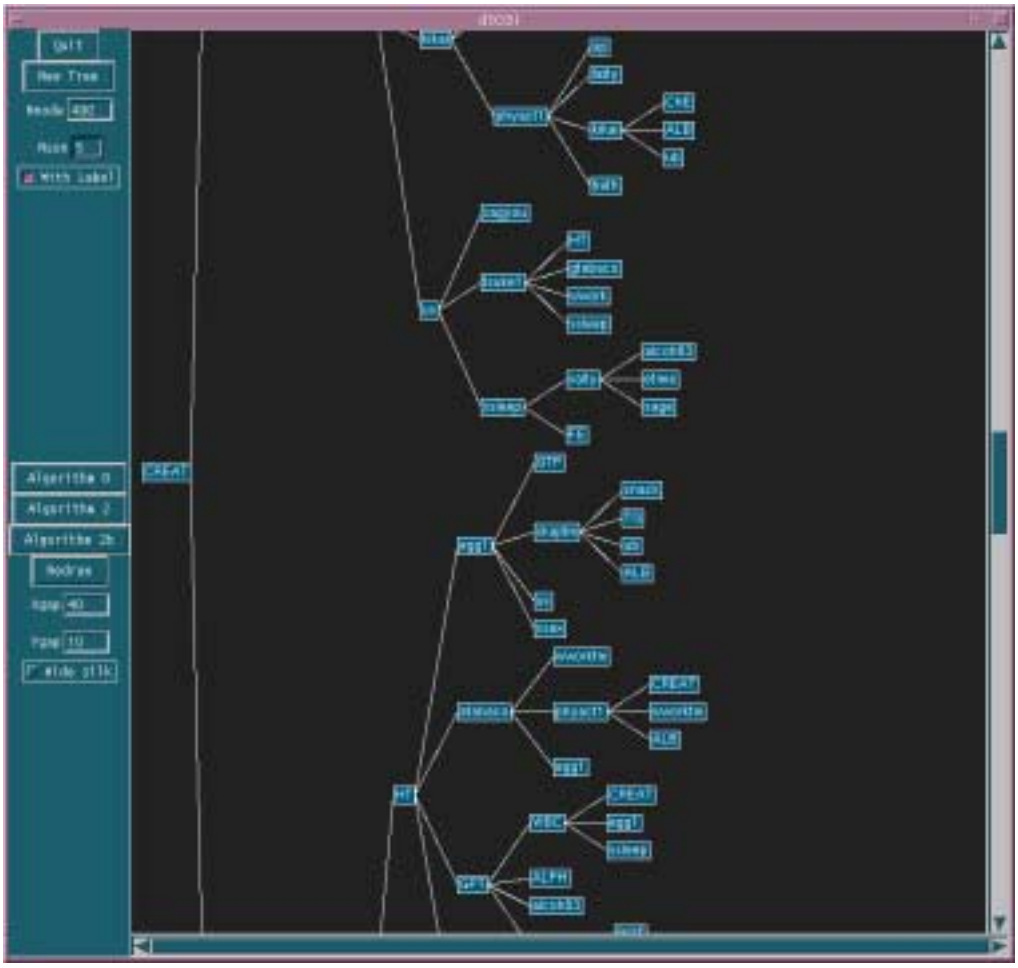

Figure 5.4: A Node-link Diagram Tree Representation

lays out the hierarchy in a uniform way on a hyperbolic plane and map this plane onto a display region. The hyperbolic browser can display more than 1000 nodes of which about 50 nearest the focus can show from three to dozens of characters of text [53]. However, it also meets the problem of presenting partially trees for large tree of thousands nodes, and it is somehow not a natural way in learning decision trees. The *treemap* visualization method is a special 2D visualizer that makes 100% use of the available display space, mapping the full hierarchy onto a rectangular region in a space-filling manner [41]. This efficient use of space allows very large hierarchical data to be displayed in their entirety. However it tends to obscure the hierarchical structure of the values and provides no way of focusing on one part of a hierarchy without losing the context.

Among typical 3D tree visualizers are that of MineSet (SGI) [12] and the *cone tree* method at Xerox [88]. Mineset 3D Tree Visualizer can display the tree hierarchy and map attributes to histogram at each node. The *cone tree* method of Xerox PARC embeds trees in a three dimensional space with the rotating function that allows bringing different parts of the tree into focus [88]. However, this technique requires expensive 3D animation support and manipulates difficultly trees with more than approximately 1000 nodes.

One common problem to all tree visualizers is if nodes on the tree, even only leaf nodes, are to be given adequate spacing, the nodes near the root must be placed very far apart, leaving no nice way to display the context of the entire tree [53]. In fact, the user does not go from node to node randomly. He/she usually wants to move from a node to one of its relatives, and often get lost (unable to find the way) in a huge hierarchy. Figure 5.2 shows that it is hard to navigate large trees in CART (left) and Mineset (right) where relatives of the node in focus cannot be seen from these screens.

Our work was motivated by the lack of adequate tools for the visualization of large trees. It contributes a new alternative to tree visualization with the node-link diagram. In particular, it can be more natural to human perception and cognition than the hyperbolic trees in learning decision trees, and it use more efficiently the display space with multi-level dynamic browsers and the fish-eye view [31].

5.2 Related Work in Visualizing Large Trees

A large quantity of the world's information is hierarchically structured: manuals, outlines, corporate organizations, family trees, directory structures, interned addressing, library cataloging, computer programs, etc. Most people come to understand the content and organization of these structures easily if they are small, but have great difficulty if the structures are large. That is why visualizing large hierarchical structures is one of active research problem in the field of information visualization. In this section, we will describe two well-known methods for this problem.

5.2.1 Tree-Maps

Tree-Maps [41] visualization technique makes 100% use of the available display space by mapping the full hierarchy onto a rectangular region in a space-filling manner. This efficient use of space allows very large hierarchies to be displayed in their entirety and facilitates the presentation of semantic information.

Tree-Maps displays look similar to the partition diagrams of quad-tree and k-D trees. The key difference is the direction of the transformation. Quad-trees create hierarchical structures to store 2D images efficiently while Tree-Maps present hierarchical information structures efficiently on 2D display surfaces.

Tree-Maps require that a weight be assigned to each node, this weight is used to determine the size of a nodes bounding box. The weight may represent a single domain property (such as disk usage or file age for a directory tree), or a combination of domain properties. A nodes weight (bounding box) determines its display size and can be thought of as a measure of importance of degree of interest.

Structure information in Tree-Maps is implicitly presented, although it may also be explicitly indicated by nesting child nodes within their parent. Nesting provides for the direct selection of all nodes, both internal and leaf. Although the space required for nesting reduces the number of nodes which can be drawn in a given display space, and hence reduces the size of the trees that can be adequately displayed compared to non-

nested drawings.

A non-nested display explicitly provides direct selection only for leaf nodes, but a pop-up display can provide path information as well as further selection facilities. Non-nested presentations cannot depict internal nodes in degenerate linear sub-paths, as the bounding boxes of the internal nodes in the sub-path may be exactly equal. Such paths seldom occur and tasks dependent on long chains of single child nodes will require special treatments.

5.2.2 Hyperbolic Browser

Hyperbolic browser [53] is a tree visualization technique based on hyperbolic geometry. The technique assigns more display space to a portion of the hierarchy while still embedding it in the context of the entire hierarchy. The drawing algorithm lays out the hierarchy in a uniform way on a hyperbolic plane and map this plane onto a display region. The chosen mapping provides a fisheye distortion that supports a smooth blending of focus and context.

The hyperbolic browser initially displays a tree with its root at the center, but the display can be smoothly transformed to bring other nodes into focus. In all cases, the amount of space available to a node falls off as a continuous function of its distance in the tree from the node in focus. Thus the context always includes several generations of parents, siblings, and children, making it easier for the user to explore the hierarchy without getting lost.

While the hyperbolic plane is a mathematical abstraction, it can be mapped in a natural way onto the Euclidean unit disk, which provides a basis for display on conventional screens. The mapping focuses on one point on the hyperbolic plane by using more of the disk for potions of the plane near that point than on other potions of the plane; remote parts of the hyperbolic plane get miniscule amounts of space near the edge of the disk. Moving the focus point over the hyperbolic plane—equivalent to translating the hierarchy on the hyperbolic plane—provides a mechanism for controlling which portion of the structure receives the most space without compromising the illusion of viewing the entire

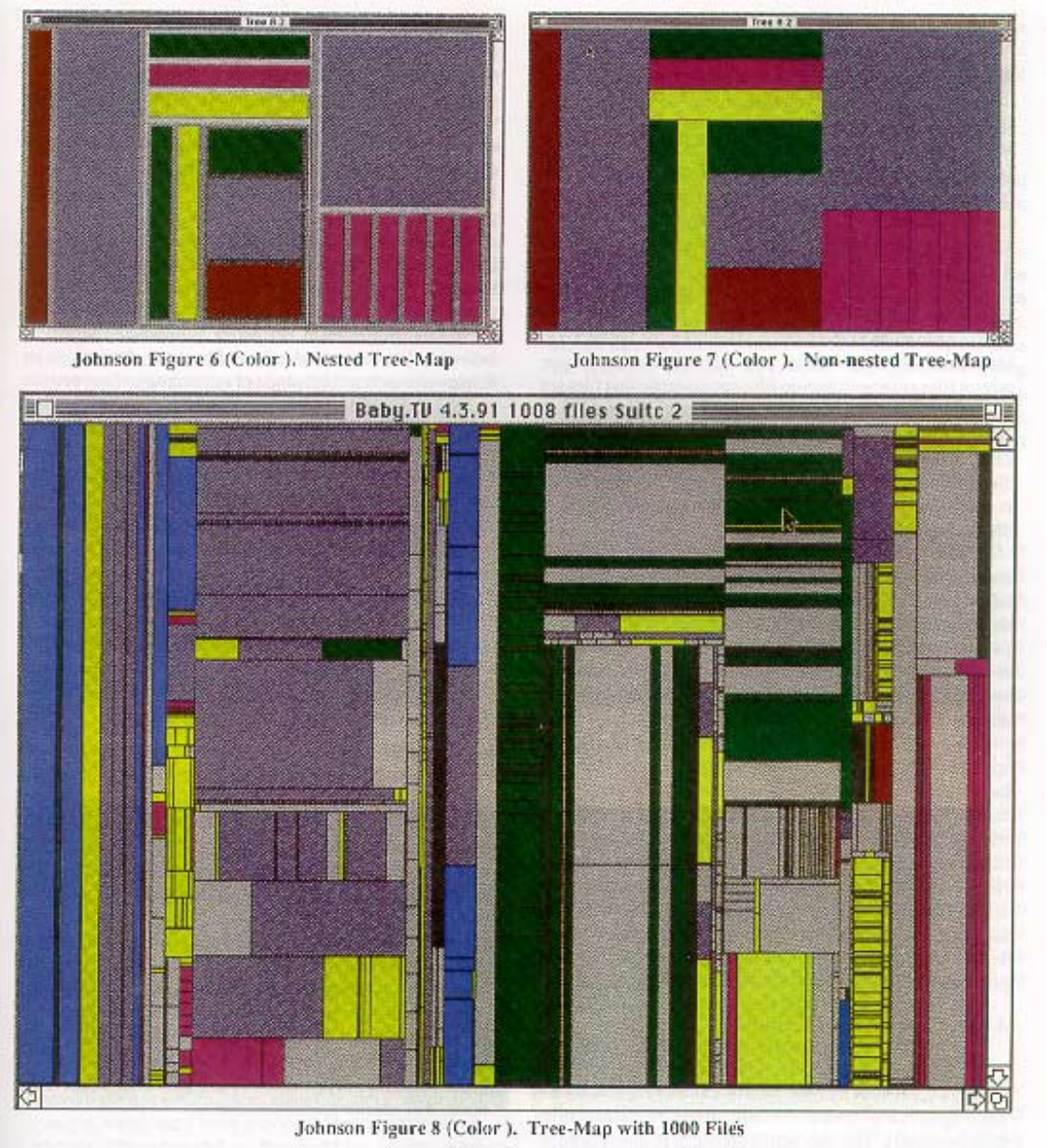

Figure 5.5: Tree-Maps Representation of Directory Hierarchies

hyperbolic plane. Other transformations of the mapping from the hyperbolic plane to the display can yield other effects including changing the relative amount of the display dedicated to the focus nodes and providing multiple foci.

5.3 The Tree Visualizer in CABRO

In CABRO, a mining process concerns with *model selection* in which the user try different settings of decision tree learning to attain most appropriate decision trees. To help the user to understand the effects of settings on result trees, the tree visualizer is capable of handling multiple views of different trees at any stage in the learning.

CABRO with its tree visualizer has been implemented in UNIX workstations under the X Window, and recently in MS Windows. Our attempt aims at improving the node-link diagram of CABRO tree visualizer to deal with the above problems for large decision trees. The main features of the CABRO Tree Visualizer are:

1. It provides several modes of view: zoomed, tiny, tightly-coupled and fish-eyed, in each mode the user can interactively change the layout of the structure to fit the current interests.

2. It provides a new and efficient technique, called T2.5D, to make navigating large tree easier.

5.3.1 Different Modes of View

CABRO tree visualizer is a graphical interface that displays the same tree in two tightly-coupled views, one a *global view* and the other a *detailed view*. Each of these views can be expanded or collapsed and is with one of the following modes (Figure 3):

- *Tightly-coupled views*: The global view (on the left) shows the tree structure with nodes in same small size without labels and therefore it can display a tree fully or a

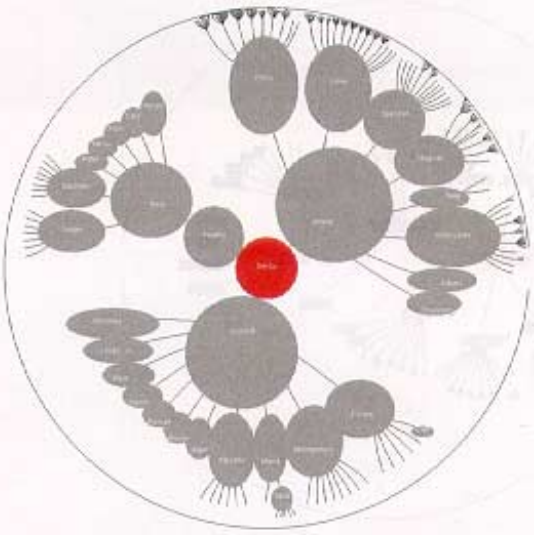

FIGURE 8

The regions available to nodes for displaying node information.

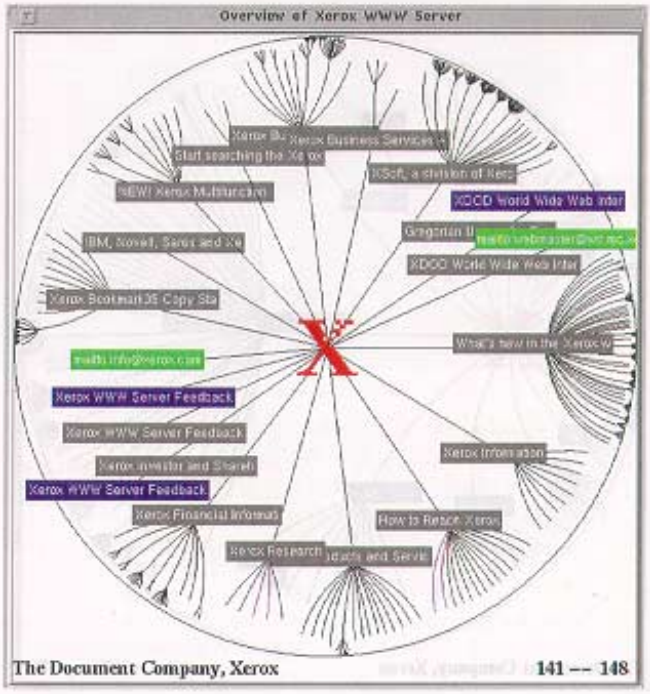

FIGURE 9

Long text mode with up to 25 characters displayed for each node that would normally display at least 2 characters.

Figure 5.6: Screenshots of Hyperbolic Browser

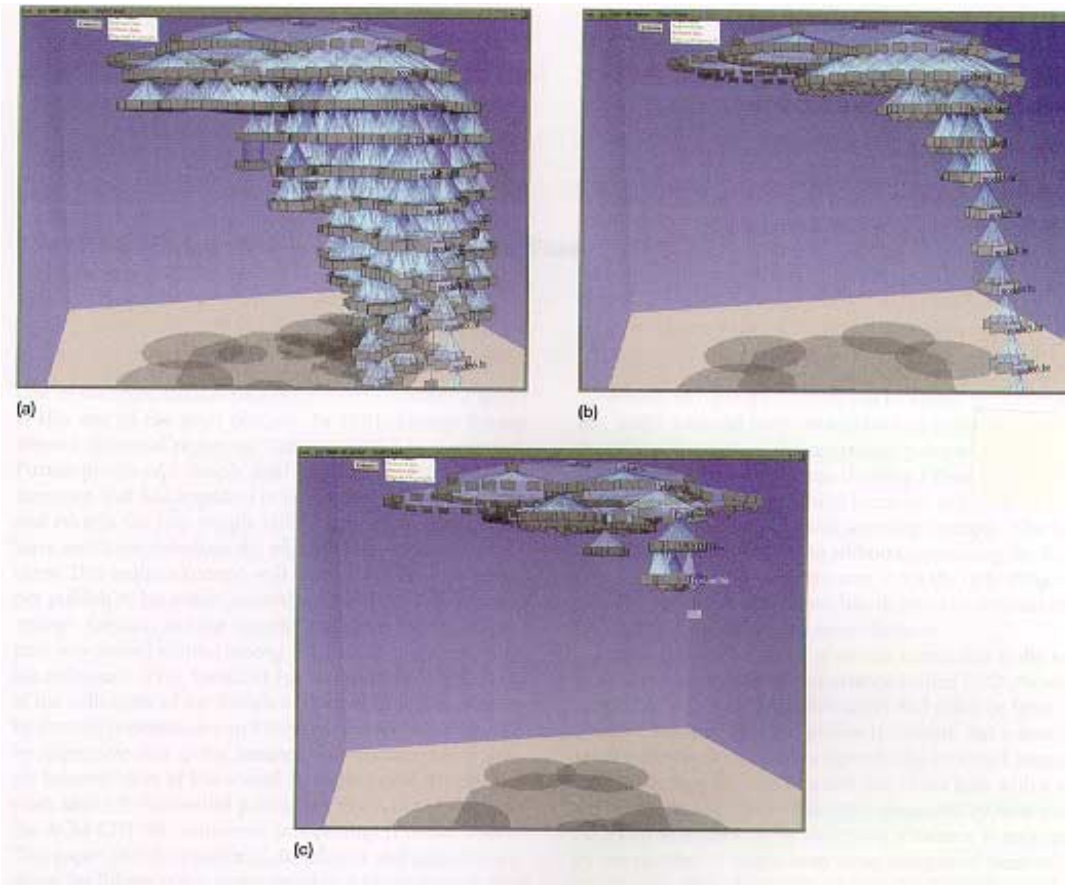

Figure 5.7: Screenshots of Cone Tree

large part of it, depending on the tree size. The detailed view (on the right) shows the tree structure and nodes with their labels associated with operations to display node information. The global view is associated with a *field-of-view* or *panner* (a wire-frame box) that corresponds to the detailed view [75]. These two views are tightly-coupled as the field-of-view can be moved around in the global view in order to pan the detailed view. Also, when the detailed view is scrolled the position of the field-of-view will be updated accordingly. The windows for these two views can be resized by the user, and the field-of-view shape and size will be automatically changed. Figure 5.8 shows a screen of CABRO tree visualizer with tightly-coupled views for the well-known soybean dataset.

- *Customizing views*: Initially, according to the user's choice, the tree is either displayed fully or with only the root node and its direct sub-nodes. The tree then can be *collapsed* or *expanded* partially or fully from the root or from any intermediate node. Any subtree with the root at an uncollapsed node (node with +) can be collapsed into one node (at its root with −). Thus, the user is able to interactively customize views of the tree to meet his/her need and interests. Also, the user is provided the focus view on one class and its relation to other classes in the whole hierarchical structure with different colors.

- *View of decision/leaf nodes*: The user can click a node to see its information: branching attribute and branched attribute-value, number of covered cases, the major class, the percentage of major class, the leading path from the root, etc.). When focusing on one class, only its leaf nodes are highlighted and proportions of cases bearing this class label are indicated approximately.

- *Tiny mode with fish-eye view*

  Note that no current visualization technique allows us to display efficiently the entire tree when it has, says, ten thousands nodes. If we note that, "the foundational period of information visualization is now ending", and "in the next period, information visualization will pass out of the realm of an exotic research specialty and into the mainstream of the user interface and an application design" [88], we might have to think about efficient ways to manipulate large trees without expecting to

see it entirely.

The tightly-coupled views are extended with three viewing modes according to the user's choice: normal size, small size and tiny size. The tiny mode uses much more efficiently the space to visualize the tree structure, on which the user can determine quickly the field-of-view and pan to the region of interest. It allows the user to be able to see the tree structure while focusing on any particular part so that the relationship of parts to the whole can be seen and the focus can be moved to other parts in a smooth and continuous way.

Fish-eye is an interesting variant of the classic overview-detail browser, proposed in [31]. This view distorts the magnified image so that the center of interest is displayed at high magnification, and the rest of the image is progressively compressed. In CABRO tree visualizer, we define three fish-eye components as follows:

1. focal point f: some node of current interest in the tree;
2. distance from focal point f to a node x: $D(f, x) = d(f, x)$ where $d(x, y)$ between two points x and y on the tree is the number of links intervening on the path connecting them in the tree;
3. detail level, importance, resolution: $LOD(x) = -d(r, x)$ where r is the root of the tree.

CABRO tree visualizer permits the user to change dynamically the point of interest on the tree, and a dynamic threshold k for the distance from focus. By default, $k = 2$ for the distance from the left and from the right of focal point. A variant of visualization is to use the global browser in tiny mode with fish-view for the field of interest.

5.3.2 Visualization with T2.5D Technique

With very large tree, the user might still find it is so difficult to navigate, even with tiny mode and fish-eye view. To address the problem, we have been developing a new technique called T2.5D (stands for 2.5 Dimensions Tree). The starting points of T2.5D's design are the following:

- The screen space is limited, therefore only a number of nodes of a tree can be displayed simultaneously. Other nodes may be out of screen, pruned or displayed in the background.

- How to display together a large subset of related nodes in a tree within a limited space is essential. For a decision tree, as paths (we will define them latter) correspond to rules, tracing nodes belonging or relating to a path is an common task. T2.5D aims to support doing this task as conveniently as possible.

- Maintaining the structural information of a tree when the user navigates the tree is indispensable. It helps the user easily to identity where she/he is and where she/he wants to go.

- The more nodes are displayed in the screen, the less scrolling operations the user need to do. Therefore, arranging nodes such that a large number of them can be displayed in a compact screen space is important.

To describe the technique we need to clarify some terms that will be used. We define a *wide path* to a node in the tree as the set of all nodes in the path from the root to this node and their siblings. Therefore, the wide path to a node contains all of its direct relatives. When visualizing a tree, at each moment, the wide path to the *node in focus* is called the *active wide path*. We summary below the main ideas and characteristics of T2.5D:

- The active wide path is displayed in the front of the screen with highlighted colors. Other nodes of the tree are displayed in the background with dim colors. They are drawn in a 3D form to save space while the active wide path is drawn in a 2D form in order to give a clear view.

- The user can change the node in focus by clicking to another node of the tree, and its wide path then becomes the active wide path and is highlighted. The user does not have to make a lot of navigation in order to learn about the relatives of the node in focus.

	3D in Mineset	2D in CART	Hyperbolic Tree	T2.5D
Embedded in DTL	no	no	n/a	yes
Comprehension	average	average	high	high
Efficiency	average	average	high	high
Interactivity	average	high	high	average
Esthetics	high	average	very high	high

Table 5.1: Comparison of several tree visualizers

- The labels of some nodes in the background may be overlapped by other nodes. For that case, a floating balloon is provided. The balloon will display clearly the label of the node which is currently pointed by the mouse pointer.

- The mix between 2D and 3D drawing has two main merits. Firstly, the node in focus and its relatives have a clear view, at the same time, many other nodes also can be seen to maintain the tree structural information. Secondly, many nodes can be displayed in a compact screen space.

We have been tried T2.5D with several datasets both real and artificial. In our experiment, the new technique have several advantages:

- T2.5D easily handles decision trees with more than 20000 nodes, and more than 1000 nodes can be displayed together on the screen.

- It is useful to have a very clear view of a wide path to a node in a tree and an image of the overall structure of the tree at the same time.

- Navigation even on huge decision trees is easy and fast. We do not have to perform many operations (scrolling, expanding, collapsing, or using the map) to navigate trees.

Figure 5.9 shows a screenshot of the CABRO tree visualizer with T2.5D technique. The tree generated from a dataset of stomach cancer data and has more than 2000 nodes. Most of nodes in the tree appear in the window.

5.4 Interactive Learning of Large Decision Trees

5.4.1 Support for Model Selection

There exist various methods for solving three main DTL problems of attribute selection, pruning and discretization, and it is well known that none of them is universally superior than others. It raises in practice the problem of *model selection*, that is how to choose the most appropriate DTL methods/models for a given application task. This problem requires meta-knowledge and/or empirical comparative evaluations of methods/models.

Three main criteria for selecting DTL models are size, accuracy and understandability of trees. The tree size and accuracy can be quantitatively evaluated, among them the accuracy is widely considered to be of great importance. The understandability of trees is difficult to be quantified or measured, and the idea here is to use tree visualizer to support the understanding of users.

Though the starting point of CABRO is R-measure, a new proposed measure for attribute selection [70], CABRO associates its tree visualizer with other modules to provide a number of well-known DTL available techniques and to combine them to form DTL models and evaluate them (accuracy, size and understandability).

In the current version of CABRO, the user can generate new models each is composed by an attribute selection measure chosen from the gain-ratio [83], the gini-index [9], χ^2 and R-measure; a pruning technique from error-complexity, reduced-error and pessimistic error [9], [83]; and a discretization technique from the entropy-based and error-based techniques [21].

Information of each trial on a model chosen by the user is registered in a form called *plan* of that model. The realization of that plan yields a model that may or may not be accepted by the user, and the user usually has to try a number of different plans. This process is repeated automatically for different model candidates by a k-fold stratified cross validation (by default, $k = 10$).

For each model candidate, CABRO tree visualizer displays graphically the corresponding *pruned tree, its size, its prediction error rate*. It offers the user a multiple view of

these trials and facilitates the user to compare results of trials in order to make his/her final selection of techniques/models of interest.

5.4.2 Support for Matching of Unknown Objects

CABRO tree visualizer is used not only in inducing decision trees but also in classifying unknown objects. It plays the role of the interface for visual explanation of the matching process, in a way similar to the explanation in knowledge-based systems. CABRO tree visualizer supports three modes of matching an unknown object according to the way that the unknown object is declared.

- The whole record of the unknown object is read from a database: CABRO directly show the leaf node that matches the object. The path from the tree root until that leaf node will be highlighted. Information accumulated along the path can be viewed at any node.
- Values of attributes are given by the user when answering the system questions: Questions about attributes will be asked according to the hierarchical structure in a top-down manner from the root. From menu the user will choose one value in the list of discrete values of the attribute or enter a numerical value in case of continuous attribute. Questions are asked dynamically according to the stepwise refinement of the matching process.
- The user declares values of attributes he/she knows: The user is able to select attributes that he/she wishes to query on. These attributes can be selected from the attribute list with corresponding values. Once the attribute-values pairs are entered, the tree visualizer will limit the regions on the tree that partially satisfy the data. The system will then ask additional questions to fulfill the match.

Table 5.1 summarizes our evaluation on the properties of some tree visualizers according to the requirements introduced in section 2.

5.5 Summary

In chapter we addressed problems of visualizing large decision trees, related works on tree visualizers and information visualization field. We presented the interactive tree visualizer of system CABRO that employs recent visualization techniques in mining decision trees. We described our attempt to deal with large tree by introducing multiple views with a new tiny mode, a variant of fish-eye view, and the new technique T2.5D for tree visualization. Though there are still a lot of work for improving this interactive tree visualizer, we believe that it contributes an efficient solution to the state-of-the-art of visualization in decision tree learning, especially T2.5D is very promising as a new display and navigation technique for large trees.

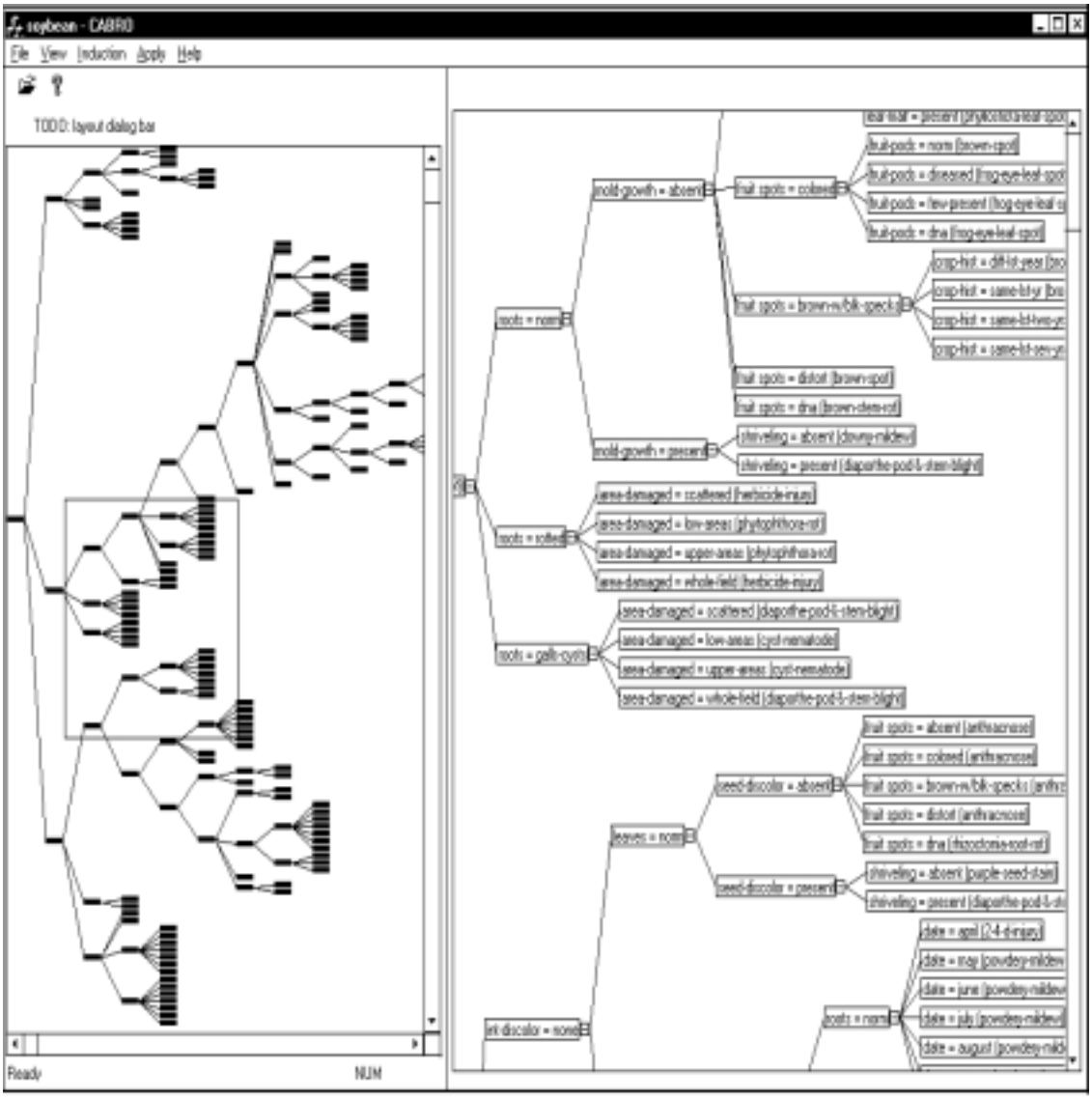

Figure 5.8: CABRO Tree Visualizer: tightly-coupled views

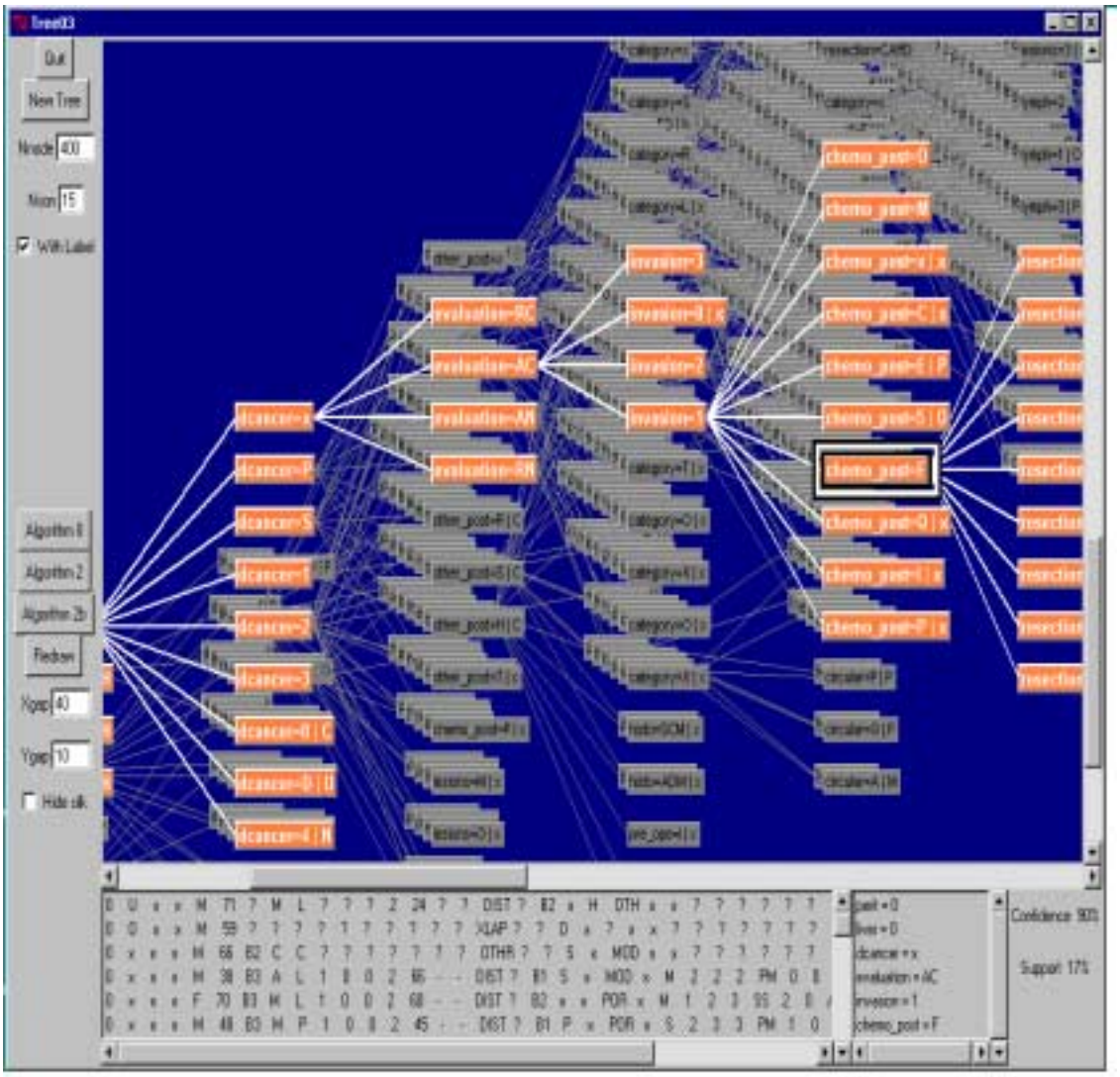

Figure 5.9: T2.5D visualization

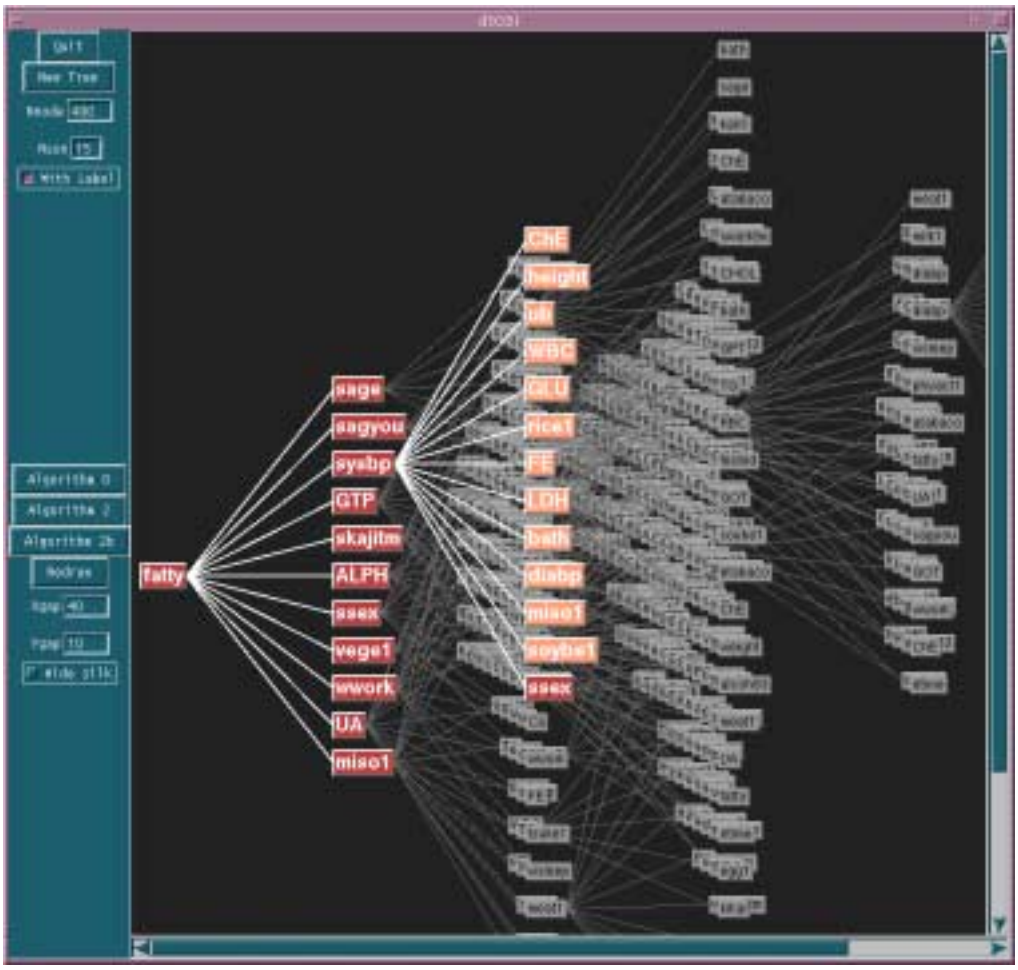

Figure 5.10: Changing for focus node in T2.5D (a)

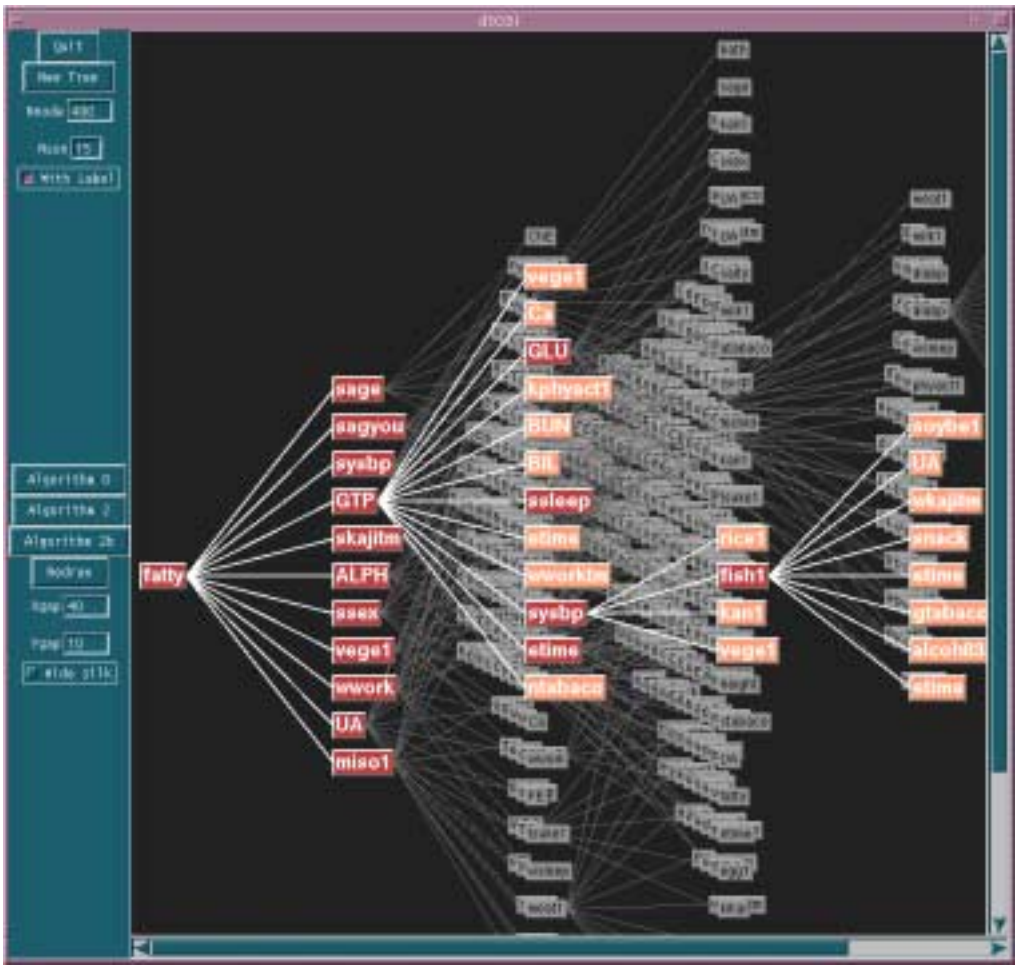

Figure 5.11: Changing for focus node in T2.5D (b)

Chapter 6

A Data Mining System That Supports Model Selection

6.1 Introduction

Knowledge discovery in databases (KDD) - the rapidly growing interdisciplinary field of computing that involves from its roots in artificial intelligence (AI), statistics, and algorithmics - aims at finding useful knowledge from large databases. The following steps are generally common to the process of knowledge discovery: (1) understanding the application domain and formulating the problem; (2) collecting and preprocessing the data; (3) mining to extract useful knowledge as patterns or models hidden in data (data mining); (4) interpreting and evaluating discovered knowledge (postprocessing); and (5) putting discovered knowledge in practical use. The main step (3) in this process consists of "particular data mining algorithms that, under some acceptable computational efficiency limitations, find patterns or models in data" [23]. It is crucial that these steps are *iterative* and *interactive*, i.e., one cannot expect to extract useful knowledge by just pushing one time a large amount of data into a black box.

In the KDD process, different knowledge discovery methods usually share many common processing tasks such as those for feature selection, discretization of continuous attributes, evaluation of discovered knowledge, visualization, etc. The final result of the knowledge discovery process depends on interoperations between algorithms in different steps. This makes the KDD process difficult for the user and very time-consuming. Also, many knowledge discovery methods compose of different algorithms or variants to deal

with influential factors in KDD such as the very large size of databases; non-incremental or incremental mining tasks; mixed categorical and numerical data; irrelevant, noise and missing data; constraints on the structure of discovered knowledge (e.g., disjoint or overlapping hierarchical structure). It is commonly agreed that there is no universally superior algorithm for all applications and the result of each algorithm may depend largely on its parameter settings. The problem of *model selection* - choosing the appropriate algorithms/settings for a given application - is difficult for the user because it requires various empirical comparative evaluations and/or meta-knowledge about the algorithms. Visualization of data and discovered knowledge has recently received a great attention in KDD because of its practical importance. From large datasets, data mining algorithms often yield large structures of discovered knowledge. However, visualization of large discovered knowledge structures is still inefficient and remains a research challenge.

The aim of our research is to develop a knowledge discovery system that emphasizes integration of KDD common tasks and with a focus on two issues of model selection and visualization. The system facilitates the interaction and decision of the user in the complicated tasks of model selection. It is associated with a new proposed visualization technique that allows displaying efficiently large hierarchical structures. Within the system framework, we have implemented and used decision tree induction methods with different algorithms including the new R-measure and method CABRO [70], and conceptual clustering methods including the method OSHAM [33] and its variants [34], [35]. The system is extendible in the sense that data mining methods generating hierarchical models can be added.

There are several works related to ours. MLC++ [49] is a library of classification algorithms with guide to compare and select appropriate algorithms for the classification task. It also provides a visualization and interfaces between programs for both end-users and software developers. Intelligent Miner [36] is IBM's data mining tool kit. It provides a broad selection of mining algorithms which are scalable for large data volumes on IBM platforms with a client/server structure. FlexiMine [20] is a KDD system designed as a testbed for data mining research and a generic knowledge discovery tool. It is currently a prototype requiring a lot of further implementation and evaluation. Our system shares with their systems many features but differs from them in the interoperability between

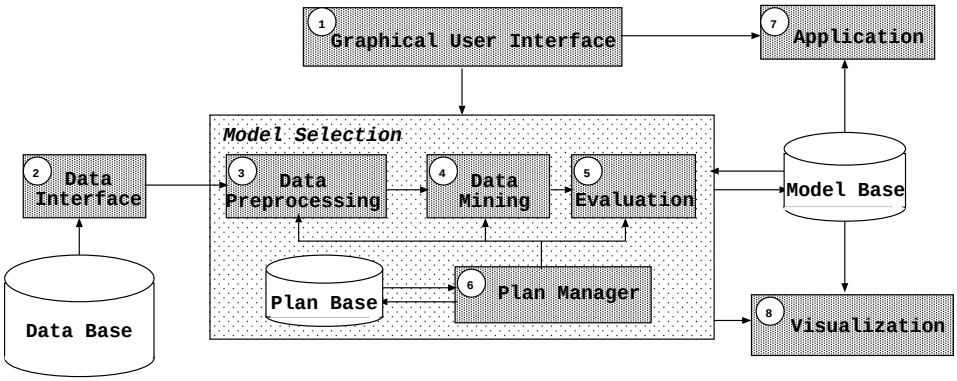

Figure 6.1: System architecture

algorithms, and the focus on model selection and efficient knowledge visualization.

Section 2 of this paper presents an overview of the system and our solutions to model selection and visualization. Section 3 presents how decision tree induction and conceptual clustering methods have been implemented and used in the system. Section 4 summarizes the work and outlines further research.

6.2 Overview of the System and Solutions

This section presents an overview of the system and our solution to the problems of model selection and knowledge visualization. Figure 6.1 shows the system architecture.

6.2.1 Overview of the System

The system consists of eight modules: Graphical user interface, Data interface, Data processing, Data mining, Evaluation, Plan management, Visualization, and Application. Additionally, it has a Plan base and a Model base.

Graphical User Interface

A flexible graphical user interface is designed to facilitate the complicated interaction between the user and the system in the knowledge discovery process. Especially, the user interface provides facilities to allow the user to control multi interactive mining, to view concurrently multiple models, and to compare evaluation results.

Data Interface

The native input format of the system is a flat form of data files. The system supports other common database formats by using standard interfaces (e.g., ODBC).

Data Preprocessing

This module provides different techniques of data preprocessing such as feature selection, elimination of corrupt data records, encoding missing values, discretization of continuous attributes, etc. For example, available discretization algorithms include the k-means algorithm for unsupervised data, and the entropy-based and error-based algorithms for supervised data [21].

Data Mining

The system is designed to support several types of classification and clustering algorithms. At present, available algorithms include those of decision tree induction with a number of related techniques, and a conceptual clustering method with several variants. Most of these algorithms originated in our earlier development of two methods CABRO [70] and OSHAM [33].

Evaluation

Discovered models require a careful evaluation. The module carries out that evaluation based on evaluation methods such as the k-fold cross validation and the t test. It also generates automatically tables containing the evaluation results of different models on different testing datasets to help the user making decision in model selection.

Plan Management

The main function of this module is to manage a plan base containing *plans* declared by the user. It also coordinates other modules such as data mining, evaluation, and visu-

alization in model selection.

Visualization

This module is a set of algorithms and techniques to visualize data and knowledge. The mining algorithms can frequently invoke this module during their running in order to help the user taking part in it effectively. The module provides techniques to visualize data and a browser to visualize and navigate hierarchical structures, especially large structures can be handled efficiently by using our developing technique called T2.5D.

Application

This module contains utilities which help the user to use discovered models. A model can be used to match an unknown instance, or to generate an interactive dialogue where the system conducts a series of questions/answers in order to predict the outcome.

6.2.2 Model Selection

Given input data, mining algorithms try to find useful patterns/models. A *pattern* "is a local structure, perhaps relating to just a handful of variables and a few cases", whereas a *model* can be seen as "a global representation of a structure that summarizes the systematic component underlying the data or that describes how the data may have arisen" [32]. We use the term 'model' to refer to hierarchical models generated by mining algorithms of the system.

To extract a model the user has to carry out different operations in steps (2) and (3) and invoke a mining algorithm. While doing that, the user chooses a series of settings. These settings concern algorithms for discretization and noise cleaning, feature selection and data transformation, data mining algorithms and related techniques (e.g., attribute selection and pruning techniques in decision tree induction), as well as parameters in these algorithms. For example, to generate a decision tree, the user needs to choose: which attribute selection and pruning algorithms to be used; the minimum number of instances at leaf nodes; the lowest accepted error rate; which attributes to be grouped;

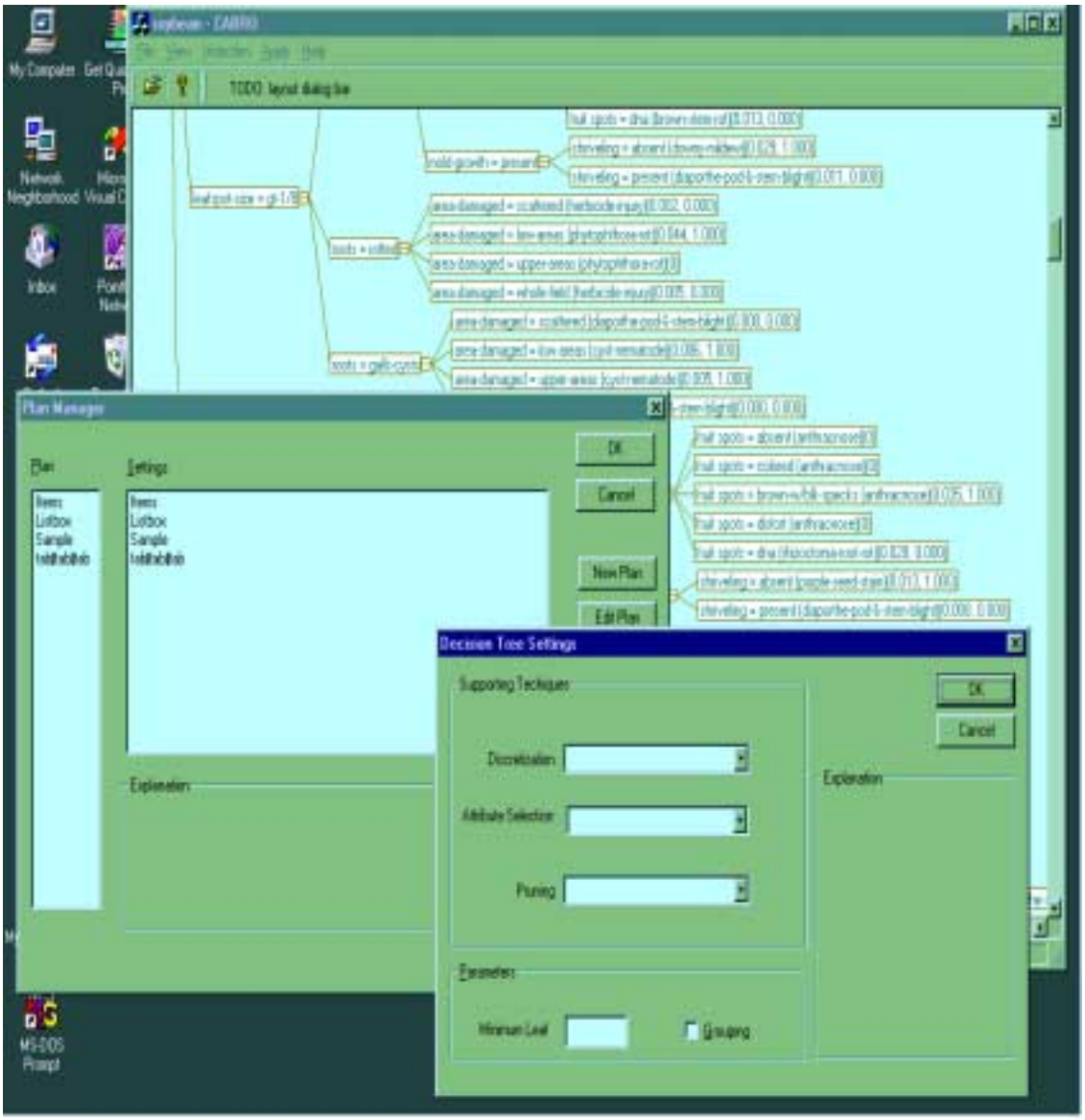

Figure 6.2: Support for model selection

encoding missing values, etc. Furthermore, if there are some continuous attributes the user may have to choose which discretization algorithms should be applied before mining.

The chosen settings will be registered in a form called a plan. The realization of that plan will yield a model that may or may not be accepted by the user. As there are many possible combinations of settings, the user usually has to try a number of plans to achieve satisfied models. We call *model selection* the operation of running different plans to achieve the most appropriate model.

In model selection, the plan management module has a coordination role. It maintains a list of *profiles* about available data preprocessing and mining algorithms. A profile contains information about types and effect of parameters, as well as related techniques that a knowledge discovery method may require. Based on information from profiles, the system asks the user to choose settings required by the method when it is applied. The user can either realize a plan just after it has been filled up, or prepare a number of plans then realize them altogether (Figure 6.2). The plan management module manages registered plans and their links to corresponding generated models stored in a *model base*. It provides functions to make, delete, modify, read and write plans. During generating models are, the user can call other modules to visualize, evaluate and compare them. This iterative cycle may be repeated until reaching appropriate models. Usually, the model selection is a daunting and very time consuming task, but thanks to the plan management module and supporting tools of the system, the task becomes easier and more effective.

6.2.3 Data and Knowledge Visualization

Data and knowledge visualization plays an increasing important role in KDD. The system provides several specialized visualizers for the display of data to be mined and discovered knowledge in hierarchical structures.

In data visualization, the system provides the user graphical views on the relations between attributes and statistics of the input data. These include mode, mean, standard deviation for numeric data, and cross-tabulation [54]. It supports the user to have an intuitive understanding about data in order to choose correct attributes, methods, as well

as the links between the visualization of discovered knowledge and the visualization of data.

In knowledge visualization, the system provides the user specialized tools integrated in a visualizer. The visualizer serves for three tasks: (1) graphical views of hierarchical structure of discovered knowledge; (2) support for model selection; (3) support for the use of discovered knowledge.

The system distinguishes two kinds of information in the hierarchical structure: structural information associated with the hierarchy, and content information associated with each node. While the latter can be defined freely according to each method, the former is common to all methods and is described by a system's common data structure. The visualizer is capable of visualizing both disjoint hierarchical models like decision trees generated by CABRO (Figure 6.3), or non-disjoint hierarchical models like those generated by OSHAM. The visualizer is designed to achieve the following features:

- *Embedded in the knowledge discovery process*: Structural and content information can be displayed not only after completing the discovery process but also during this process. This is particularly significant for the interactive mining mode.
- *Comprehension*: The hierarchical structures are drawn esthetically and they can be easily browsed and understood even for the user with low perceptual and cognitive loads.
- *Efficiency*: The hierarchy visualizer uses efficiently the space and visual tools to deal with large hierarchies. In particular, the new technique T2.5D introduced in the system offers many advantages in comparison to many current techniques in terms of visualization efficiency.
- *Interactivity*: The system provides users the interactive control over the knowledge structure and the ability to customize the layout to meet their needs and interests.

Models discovered from the KDD process usually are very large in size and complicated in structure. With traditional visualization methods the user gets lost in these huge structures and navigation become almost impossible. In order to handle them, the system

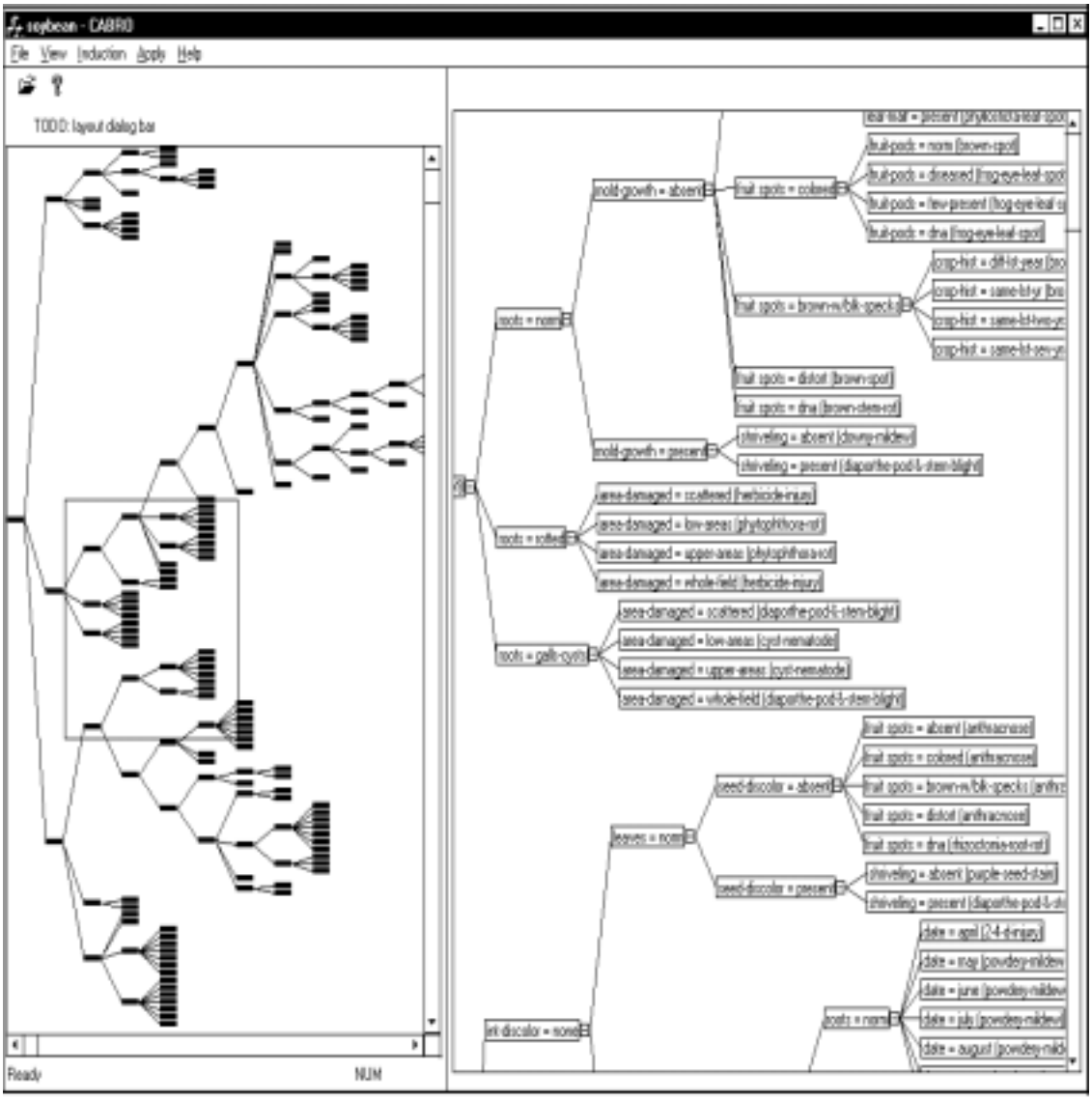

Figure 6.3: Hierarchy visualizer: tightly-coupled views

adopts three special visualization techniques: tightly-coupled views, fish-eye view, and our developing method named T2.5D.

Tightly-coupled and Fish-eye Views

The tightly-coupled views are extended with three viewing modes according to the user's choice: normal size, small size and tiny size. The tiny mode uses much more efficiently the space to visualize the hierarchical structure, on which the user can determine quickly the field-of-view and pan to the region of interest. It allows the user to be able to see the hierarchical structure while focusing on any particular part so that the relationship of parts to the whole can be seen and the focus can be moved to other parts in a smooth and continuous way. Fish-eye is an interesting variant of the classical overview-detail browser, proposed in [31]. This view distorts the magnified image so that the center of interest is displayed at high magnification, and the rest of the image is progressively compressed.

T2.5D

Very large hierarchical structures are still difficult to be navigated and viewed even with tightly-coupled and fish-eye techniques. To solve the problem, we have been developing a special technique called T2.5D (stands for Tree 2.5 Dimensions).

The 3D browsers usually can display more nodes in a compact area of the screen but require currently expensive 3D animation support and the structure somehow not easy to navigate, while the 2D browsers have a limitation in displaying many nodes in one view. The T2.5D technique combines the advantages of both the 2D and 3D drawing techniques to provide the user an efficient display with lower processing cost. The T2.5D browser can display more than 1000 nodes in one view where the most of nodes may be partially overlapped but they all are in full size. In T2.5D, a node can be highlighted or dim. The highlighted nodes are those the user currently pays most attention on, and they are displayed in 2D for ease of view and navigation. The dim nodes are displayed in 3D to save the space, they allow the user to get an idea about overall structure of the hierarchy (Figure 6.4). The main features of T2.5D are:

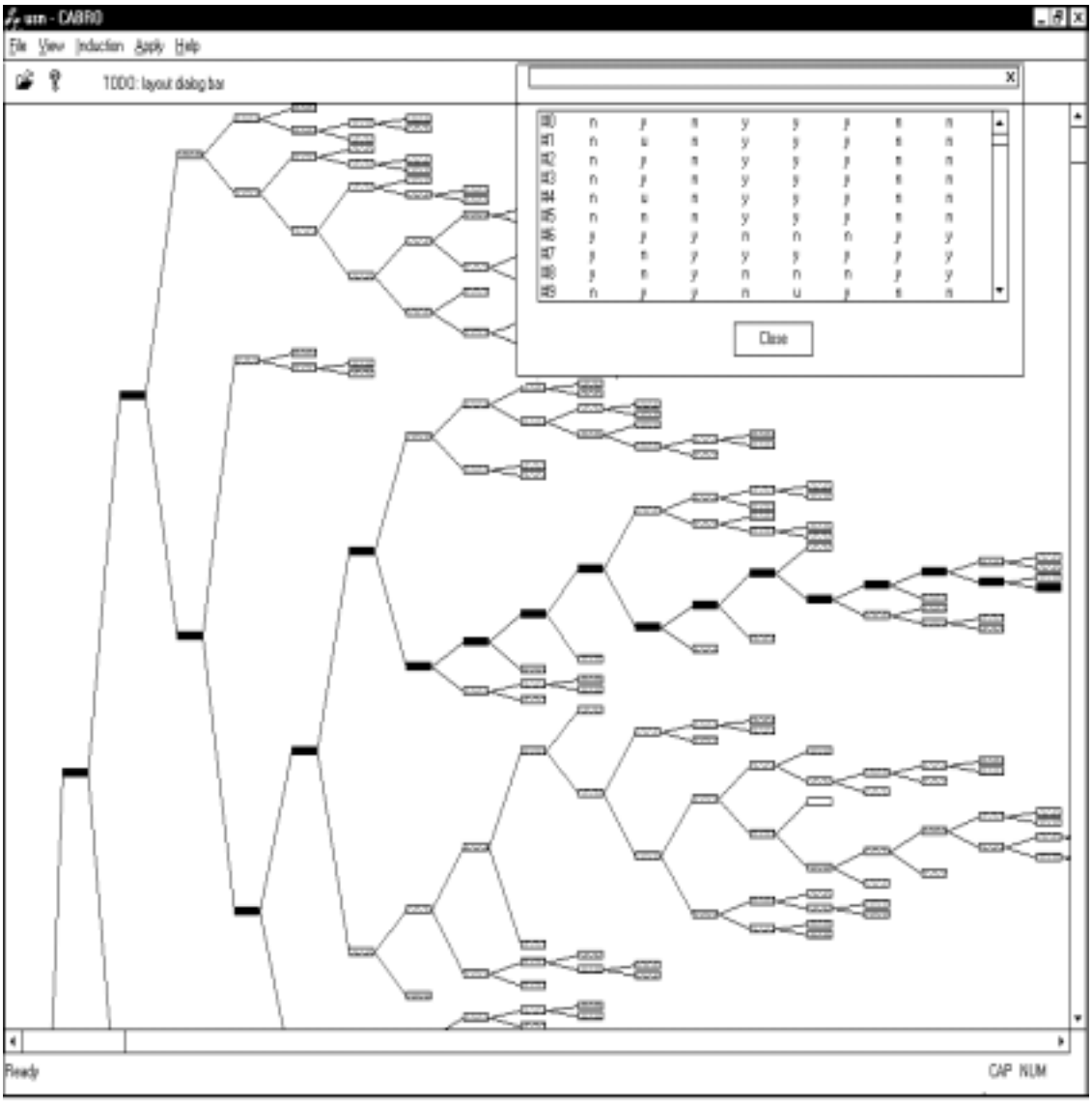

Figure 6.4: Matching a decision tree with data

- All highlighted nodes are displayed "on-top" of the screen, drawn by bright colors, and have a bigger size. Other nodes might be overlapped, and are drawn by dim colors. That allows the user to view easily and get all information about highlighted nodes even in a complicated hierarchy of nodes and edges.

- Whenever a node is chosen to be focused, its siblings, its ancestors, and the siblings of the ancestors are brought into highlight automatically. In other words, the system opens a broad path of highlighted nodes from the root to the focused node. That means the most related nodes under the focus always get a clear view altogether.

- Nodes in background are positioned such that at least a part of each node is displayed, hence the user can click to bring any of them back to highlight. A floating balloon dynamically displays the information of the node under the mouse pointer to allow the user to know more about it.

- The algorithm follows some common esthetics rules. Siblings are located at the same horizontal coordinate and vertical distances between them are even.

6.3 Knowledge Discovery Methods in the System

There are two knowledge discovery methods that are provided in the system, a decision tree and rule learning method (CABRO), and a conceptual clustering method (OSHAM). We have described CABRO in Chapter 3 and Chapter 4, in this section we will describe OSHAM briefly.

6.3.1 A Conceptual Clustering Method

Conceptual clustering is a typical knowledge discovery method for unsupervised data. Its basic task is from a given set of unlabelled instances to find simultaneously a hierarchical model that determines useful object subsets and intensional definitions for these subsets of objects. There are two main problems in conceptual clustering: representation of concepts and constraints of categorization. Among concept representations, the classical, prototype and exemplar ones are widely known and used [69]. Among categorization

constraints, the similarity, feature correlation, and structure of the concept hierarchy are widely known and used. These two problems relate to another crucial problem of interpreting hierarchical models discovered by unsupervised methods.

The method OSHAM (Making Automatically Hierarchies of Structured Objects) [33] employs a proposed hybrid representation of concepts that combines in a reasonable way some advantages of three main representation schemes mentioned above, and depending on settings of parameters it can extract non-disjoint or disjoint hierarchical models of concepts. OSHAM is a non-incremental divisive algorithm that works recursively and at each step it seeks for an acceptable solution according to a quality function defined on its hybrid representation. OSHAM is associated with an interpretation procedure to use discovered models. There are several variants of OSHAM: incremental I-OSHAM [34] that can learn when databases are regularly updated, and approximate A-OSHAM [35] that can learn approximate concepts when data are uncertain and imprecise.

6.3.2 Implementation and Experiments with Model Selection

The conceptual clustering method OSHAM and its variants have been implemented and used in the system with common tools on data preparation, discretization, visualization, evaluation by k-fold cross validation, etc. Different from supervised discovery methods, unsupervised discovery methods as OSHAM cannot provide feedback about the appropriateness of their results. In such cases, to obtain an appropriate model, the user needs to interact with the system and try different plans with various parameters. The plan management module facilitates doing this task by its support for comparing models, data and knowledge visualization.

Linked with the visualizer, disjoint or non-disjoint models generated by OSHAM can be displayed. The form and the size of OSHAM's hierarchical models depend on plans that concern a number of parameters: (1) method is OSHAM or its variants; (2) the discovered hierarchy is disjoint or non-disjoint; (3) the minimum size of each node; (4) the threshold about the concept dispersion; and (5) the number of competitors for beam search. The user can visualize the hierarchical model gradually in the discovery process, observe node's content information and the quality estimation of model. The user may also modify the

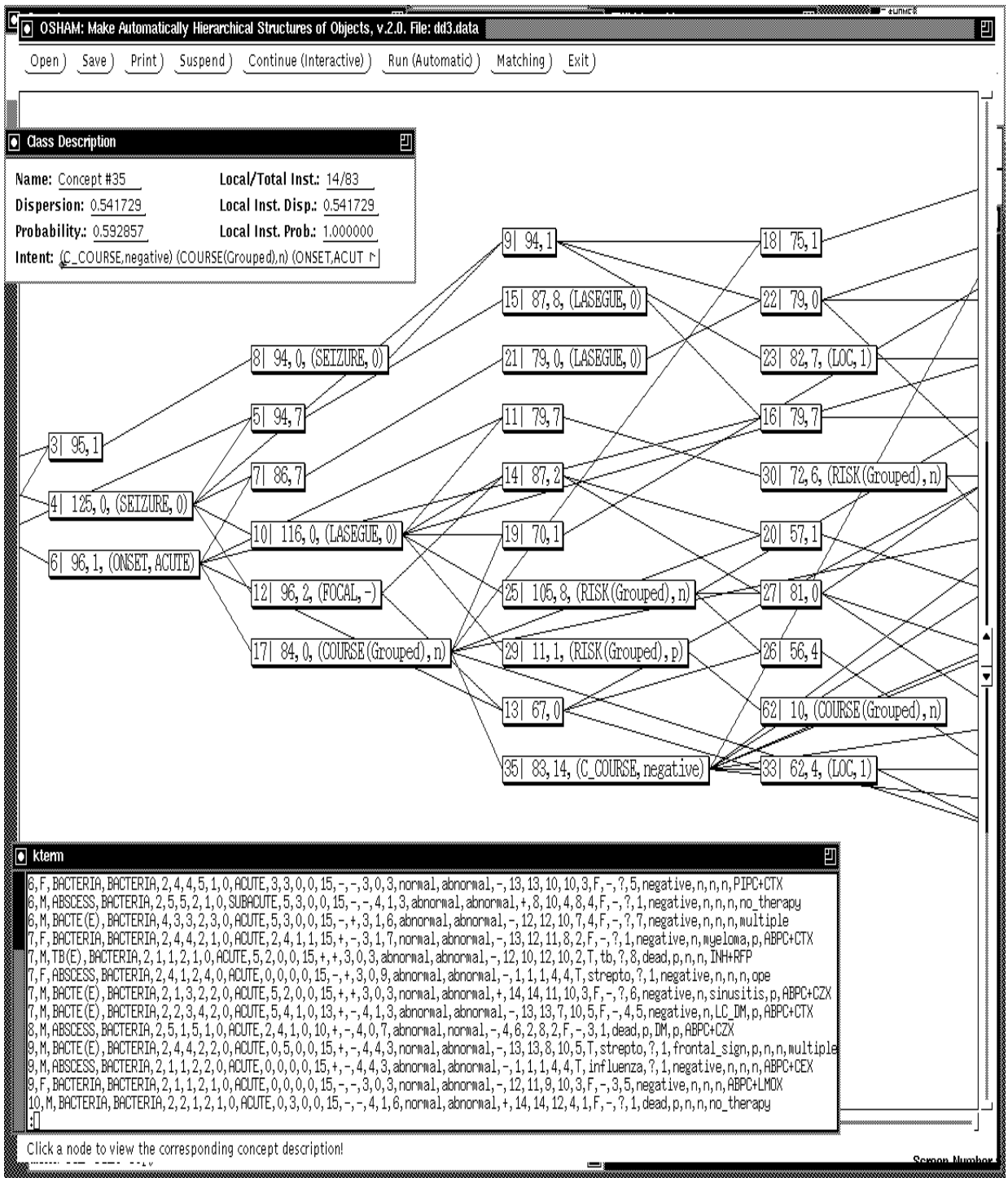

Figure 6.5: Using generated models to interpret unknown cases

parameters when necessary before continuing to go further to cluster subsequent data, or backtrack to regrow branches of the model with respect to the categorization scheme. This function can be done in the interactive mining mode: as the hierarchy is generated level by level, the user can point out a node generated previously from which he/she wants to regrow the hierarchy with changed parameters. Moreover, the visualizer is used also in the predictive task in which it links OSHAM's interpretation procedure with data and hierarchy visualization (Figure 6.5).

We illustrate an application of the system to the clinical database on meningoencephalitis collected at the Medical Research Institute, Tokyo Medical and Dental University [39]. This database is recently experimented by different data mining groups in Japan [98]. Each patient record in this database contains 38 attributes, where 7 continuous and discrete attributes describe the clinical history; 8 continuous and discrete attributes describe the physical examination; 11 continuous attributes describe the laboratory examination; 1, 1, 4, 1 and 1 discrete attributes describe the diagnosis, therapy, clinical course, final status and risk factor, respectively. The third attribute DIAG presents the diagnosis results into 6 classes (ABSCESS, BACTERIA, BACTE(E), TB(E), VIRUS(E), VIRUS), which are summarized into 2 groups 'VIRUS' and 'BACTERIA' in the fourth attribute DIAG2.

The first task is to find predicting prognosis relating to three pairs of influential attributes, considered as the class attributes: DIAG and DIAG2, CULT-FIND and CULTURE, C_COURSE and COURSE. The continuous attributes are all discretized by discretization tools of the system. Using OSHAM we investigate the "natural" clusters of the database in order to answer a number of questions, for example, which attributes are the most significant in discovered clusters? if we know *a priori* that some attributes are important then which hierarchical models could be extracted? The relationship between the size of models and their comprehension?

It is worth noting that without the supervision in data, there is not always a sharp boundary between groups in conceptual clustering. Figure 6.5 illustrates the interpretation of OSHAM in this application. It concludes that the class number 195 matches the unknown case number 5 with the degree "very strong match by the nearest neighbor and

logical multiple match". Also in this application thanks to the plan management module and visualization tools, different trials can be done and compared easily. For example, with the plan (1, 1, 8, 0.4, 8) OSHAM generated a model of 93 nodes, and with the plan (1, 1, 4, 0.2, 4) OSHAM generated a model of 2929 nodes which can be all observed and evaluated with the system support.

6.4 Summary

We have presented our research and development on an interactive-graphic system for knowledge discovery. The system emphasizes integration of KDD common tasks and algorithms according to the knowledge discovery process. The key idea behind the system is to support doing model selection with different trials on algorithms with different settings, and to support visualizing large hierarchical structures. Two methods of decision tree induction and conceptual clustering have been implemented and used within the system framework. These methods share different tools of the system in preprocessing data, data and knowledge visualization, testing procedure and using discovered knowledge. With the features of the systems and the benefit in implementing two methods, we hope that the system contributes a solution to difficult system design problems and that may be of interest to the KDD developer and user.

Chapter 7

Conclusion

The ultimate purpose of our research is to develop an integrated decision tree learning system that can be applied effectively and efficiently to data mining applications. To that end, we try to solve several problems of decision tree learning on large and complex datasets. These problems include attribute selection when data are incomplete or uncertain, the scalability of rule post-pruning algorithms, and visualization of large decision trees. Based on the results of our research on those problems, we have developed a prototype of a decision tree learning system as a first phase toward our ultimate purpose.

In the first research issue, to develop the a new criterion for attribute selection, we have proposed a variant of attribute dependency measure of the probabilistic model of rough sets [73] in order (1) to overcome the limitations of the original model in case of noisy data, (2) to make the model more coherent, and (3) to preserve the convenience of non-parameter. Based on this model, *R-measure* is developed to measure how much the class attribute depends on a predictive attribute. Using *R-measure* as an attribute selection criterion, an experimental comparative evaluation on 32 datasets shows that it can be considered as a good alternative criterion for attribute selection. Especially, the experiment showed that R-measure dealt with noisy data more effectively in comparing to the others.

In the second research issue, we have proposed a new algorithm for rule post-pruning of decision trees. It can be considered an alternative algorithm for C4.5rules when the input data become very large. The problem of high complexity in C4.5 is solved by adopting

an incremental pruning scheme. However the algorithm does not suffer the problem of hasty generalization such as in the original incremental pruning approach. Experiments have shown that the new algorithm generates rule sets as accuracy as those of C4.5 but with far less time of computation.

In the third research issue, we have developed a new technique for visualizing large decision trees (T2.5D). The technique has several advantages comparing to other techniques: (1) it easily handles decision trees with more than 20000 nodes, and more than 1000 nodes can be displayed together on the screen, (2) it gives the user a clear view of an active path and an image of the overall structure of the tree at the same time, (3) it facilitates the tree navigation as only a minimum number of operations (e.g., click, scroll, etc.) is needed.

Based on the results of those researches we have developed a prototype of a decision tree learning system as a first step toward our ultimate purpose—an integrated decision tree learning system for data mining. The R-measure is used for attribute selection, CABROrule is used for generating rules, and decision trees are visualized with T2.5D. The program has been tested successfully with several large and complex datasets.

For the future work, we would like to continue our research on DTL for datamining. How DTL can deal effectively with data with more complex structures, such as preference, time, and image data is still an open problem. The predictive accuracy of CABROrule can be further improved by applying more sophisticated approach to post-processing of rule sets. A rule learning algorithm with linear time complexity may be the next interesting target. Ultimately, we will try to build an integrated DTL system that can be apply to real problems in datamining.

Bibliography

[1] Aha, D., Kibler, D. & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning, 6(1)*, 37-66.

[2] Almuallim, H. (1996). An Efficient Algorithm for Optimal Pruning of Decision Trees. *Artificial Intelligence, 83(2)*, 347–362.

[3] Alonso, F., Mate, L., Juristo, N., Munoz, P.L., & Pazos, J. (1994). Applying Metrics to Machine-Learning Tools, *AI Magazine, Fall 1994*, 63–75.

[4] Ankerst, M., Elsen, C., Ester, M., & Kriegel, H. P. (1999). Visual Classification: An Interactive Approach to Decision Tree Construction. *Proceedings of Fifth Inter. Conf. on Knowledge Discovery and Data Mining*, 392–397.

[5] Araki, D. & Kojima, S. (1992). Inductive Decision Tree Learning From Numerical Data. *Journal of Japanese Society for Artificial Intelligence, 7(6)*, 992–1000.

[6] Agresti, A. (1990). *Categorical Data Analysis*. New York: John Wiley & Sons.

[7] Bergadano, F. & Gunetti, D. (1993). An interactive system to learn functional logic programs. In Bajcsy, R. (Ed.), *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1044-1049.

[8] Bohanec, M. & Bratko, I. (1994). Trading accuracy for simplicity in decision trees. *Machine Learning, 15(3)*, 223–250.

[9] Breiman, L., Friedman, Jb. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Belmont, California: Wadsworth.

[10] Brodley, C. E. & Utgoff, P. E. (1995). Multivariate Decision Trees. *Machine Learning, 19(1)*, 45–77.

[11] Brunk, C. A. & Pazzani, M. J. (1991). An Investigation of Noise-Tolerant Relational Concept Learning Algorithms. *Proceedings of the 8th International Workshop on Machine Learning*, 389–393.

[12] Brunk, C., Kelly, J., & Kohavi, R. (1997). MineSet: An Integrated System for Data Mining. *Proceedings of Third Inter. Conf. on Knowledge Discovery and Data Mining* 135–138.

[13] Buntine, W. & Niblett, T. (1992). A Further Comparison of Splitting Rules for Decision Tree Induction. *Machine Learning, 8(1)*, 75–86.

[14] Cestnik, B., Kononenko, I., & Bratko, I. (1987). ASSISTANT 86: A Knowledge-Elicitation Tool for Sophisticated Users. In Bratko, I. & Lavrac, N. (Eds.), *Progress in Machine Learning* (pp. 31–45). Bled, Slovenia: Sigma Press, Wilmslow, UK.

[15] Cohen, W. W. (1993). Efficient Pruning Methods for Separate-and-Conquer Rule Learning Systems. *Proceeding of the 13th International Joint Conference on Artificial Intelligence*, 988–995.

[16] Cohen, W. W. (1995). Fast Effective Rule Induction. In Prieditis, A. & Russell, S. (Eds.), *Proceedings of the 12th International Conference on Machine Learning* (pp. 115–123). Tahoe City, California: Morgan Kaufmann, San Francisco, CA.

[17] Clark, P. & Niblett, T. (1989). The CN2 Induction Algorithm. *Machine Learning, 3* 261–283.

[18] Dzeroski, S. & Bratko, I. (1992). Handling Noise in Inductive Logic Programming. In *Proceedings of the International Workshop on Inductive Logic Programming*, Tokyo, Japan.

[19] Dietterich, T.G. (1996). Statistical Tests for Comparing Supervised Classification Learning Algorithms, *Technical Report*, Dept. of Computer Science, Oregon State University.

[20] Domslak, C., Gershkovich, D., Gudes, E., Liusternik, N., Meisels, A., Rosen, T., & Shimony, S.E., (1998). FlexiMine - A Flexible Platform for KDD Research and

Application Construction, *Proceedings 4th Inter. Conf. on Knowledge Discovery and Data Mining KDD'98*, 184–188.

[21] Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and Unsupervised Discretization of Continuous Features, in *Proceedings 12th International Conference on Machine Learning*, 194–202.

[22] Fayyad, U. M. & Irani K .B. (1992). On the Handling of Continuous-Valued Attributes in Decision Tree Generation, *Machine Learning, 8(1)*, 87–102.

[23] Fayyad, U.M., Piatetsky-Shapiro G., Smyth P. & Uthurusamy R. (1996). From Data Mining to Knowledge Discovery: An Overview, in U.M. Fayyad et al. (Eds.) *Advances in Knowledge Discovery and Data Mining*, 1–36.

[24] Forsyth, R. (1994). Overfitting revisited: an information-theoretic approach to simplifying discrimination trees. *Journal of Experimental & Theoretical Artificial Intelligence, 6(3)*, 289–302.

[25] Frank, E. & Witten, I. H. (1998). Generating Accurate Rule Sets without Global Optimization. In Shavlik, J. (Ed.), *Proceedings of the 15th International Conference on Machine Learning*, 144–151.

[26] Friedman, J. H. (1997). A Recursive Partioning Decision Rule for Non-Parametric Classification. *IEEE Transactions on Computers*,404–408.

[27] Furnkranz, J. & Widmer, G. (1994). Incremental Reduced Error Pruning. In Cohen, W. W. & Hirsh, H. (Eds.), *Proceedings of the 11th International Conference on Machine Learning*, 70–77.

[28] . Furnkranz, J. (1994). FOSSIL: A Robust Relational Learner. In Bergadano, F. & De Raedt, L. (Eds.), *Proceedings of the 7th European Conference on Machine Learning (ECML-94)*, Vol. 784 of *Lecture Notes in Artivicial Intelligence*, 122–137

[29] Furnkranz, J. (1997). Pruning Algorithms For Rule Learning. *Machine Learning, 27(2)*, 139–171.

[30] Furnkranz, J. (1999). Separate-and-Conquer Rule Learning. *Artificial Intelligence Review, 13(1)*, 3–54.

[31] Furnas, G. W. (1981). The FISHEYE View: A New Look at Structured Files. *Bell Laboratories Technical Memorandum* #81-11221-9.

[32] Hand, D. J. (1998). Data Mining: Statistics or More?, *The American Statistician, 52(2)*, 112–118.

[33] Ho, T. B. (1997). "Discovering and Using Knowledge From Unsupervised Data", *Decision Support Systems, 21(1)*, Elsevier Science, 27–41.

[34] Ho, T. B., (1997). Incremental Conceptual Clustering in the Framework of Galois Lattices, in Lu, H., Motoda, H., & Luu, H. (Eds.), *KDD: Techniques and Applications*, World Scientific, 49–64.

[35] Ho, T. B., (1997). Unsupervised Concept Learning Using Rough Concept Analysis, in C. Hayashi et al. (Eds.) *Data Science, Classification and Related Methods*, Springer, 404–411.,

[36] Cabena, P., Hadjinian, P., Stadler, R., Verhees, J., & Zanasi, A., (1998). *Discovering Data Mining. From Concept to Implementation*, Prentice Hall.

[37] Han, J., & Kamber, M. (2001). *Data Mining: Concepts and Techniques*. Academic Press.

[38] Herman, I., Delest, M., & Melancon, G. (1998). Tree Visualization and Navigation Clues for Information Visualization. *Centrum voor Wiskunde en Informatica (CWI)*, 202–216.

[39] Ho, T. B., Nguyen, T. D., & Nguyen, N. B. (1999). Comparative Experimental Evaluation of Two Learning Systems CABRO and OSHAM Using the Common Medical Data, *Proceedings 42th SIG-KBS Symposium on Comparison and Evaluation of Knowledge Discovery Methods Using A Common Dataset*, Japanese Society for Artificial Intelligence, 83–88.

[40] Hunt, E. B., Marin, J., & Stone, P. J. (1966). *Experiments in Induction*, Academic Press.

[41] Johnson, B. & Shneiderman, B. (1991). Treemaps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures. *Proceedings of IEEE Information Visualization*, 275–282.

[42] Jordan, M. I. (Ed.). (1999). *Learning in Graphical Models*. MIT Press.

[43] Jun, B.H., Kim, C.S., & Kim, J. (1997). A New Criterionin Selection and Discretization of Attributes for the Generation of Decision Trees. *IEEE Transaction on Pattern Analysis and MAchine Intelligence, 19(12)*, 1371–1375.

[44] Kalkanis, G. (1993). The Application of Confidence Interval Error Analysis to the Design of Decision Tree Classifiers. *Pattern Recognition Letters, 14(5)*, 355–361.

[45] Kearns, M. (1996). A Bound on the Error of Cross-Validation Using the Approximation and Estimation Rates, with Consequences for the Training-test Split. In Touretzky, D. S., Mozer, M. C. & Hasselmo, M. E. (Eds.), *Advances in Neural Information Processing Systems 8*, MIT Press, 183–189.

[46] Kervahut, T. & Potvin, J.Y. (1996). An Interactive-Graphic Environment for Automatic Generation of Decision Trees, *Decision Support Systems, 18*, 117–134.

[47] Kohavi, R.A. (1995). Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection, *Proceedings International Joint Conference on Artificial Intelligence IJCAI'95*, 1137–1143.

[48] Kohavi, R. (1995b). *Wrappers for Performance Enhancements and Oblivious Decision Graphs*. PhD thesis, Stanford University, Department of Computer Science.

[49] Kohavi, R., Sommerfield D., & Dougherty J., (1997). Data Mining using MLC++, a Machine Learning Library in C++. *International Journal of Artificial Intelligence Tools, 6(4)*, 537–566.

[50] Kononenko, I., Bratko, I., & Roskar, E. (1984). *Experiments in Automatic Learning of Medical Diagnostic Rules* (Technical report). Jozef Stefan Institute, Ljubljana, Yugoslavia.

[51] Kumar, H. P., Plaisant, C., & Shneiderman, B. (1997). Browsing Hierarchical Data with Multi-level Dynamic Queries and Pruning. *International Journal of Human-Computer Studies, 46(1)*, 103–124.

[52] Langley, P. & Simon, H.A. (1995). Applications of Machine Learning and Rule Induction, *Communications of the ACM, 38(11)*, 55–64.

[53] Lamping, J. & Rao, R. (1997). The Hyperbolic Browser: A Focus + Context Techniques for Visualizing Large Hierarchies. *Journal of Visual Languages and Computing, 7(1)*, 33–55.

[54] Lee, H. Y., Ong, H. L., & Quek, L. H. (1995). Exploiting Visualization in Knowledge Discovery. *Proceedings of First International Conference on Knowledge Discovery and Data Mining*, 198–203.

[55] Lim, T., Loh, W., & Shih, Y. (2000). An Empirical Comparison of Prediction Accuracy, Complexity, and Traning TIme of Thirty-Three Old and New Classification Algorithms. *Machine Learning, 40(3)*, 203–230.

[56] Liu, W. Z. & White, A. P. (1994). The Importance of Attribute Selection Measures in Decision Tree Induction. *Machine Learning, 15*, 25–41.

[57] Malerba, D., Floriana, E., & Semeraro, G. (1995). A Further Comparison of Simplification Methods for Decision Tree Induction. In D. Fisher & H. Lenz (Eds.), *Learning from data: AI and statistics*. Springer-Verlag.

[58] López de Mantaras, R. (1991). A Distance-Based Attribute Selection Measure for Decision Tree Induction, *Machine Learning, 6(1)*,81–92.

[59] Mehta, M., Rissanen, J. & Agrawal, R. (1995). MDL-Based Decision Tree Pruning. In Fayyad, U. M. & Uthurusamy, R. (Eds.), *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 216–221.

[60] Michalski, R. & Chilausky, R. (1980). Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis. *International Journal of Policy Analysis and Information Systems, 4(2)*.

[61] Michalski, R. S., Mozetic, I., Hong, J., & Lavrac, N. (1986). The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains. In *Proceedings of the 5th National Conference on Artificial Intelligence*, 1041-1045.

[62] Mitchell, T.M. (1997) *Machine Learning*, McGraw-Hill.

[63] Mingers,, J. (1987). Expert systems—rule induction with statistical data. *Journal of the Operational Research Society, 38*, 39–47.

[64] Mingers, J. (1988). An Empirical Comparison of Selection Measures for Decision-Tree Induction. *Machine Learning, 3(4)*, 319–342.

[65] Mingers, J. (1989). An Empirical Comparison of Pruning Methods for Decision Tree Induction. *Machine Learning, 4(2)*, 227–243.

[66] Muggleton, S., H., (1995). Inverse Entailment and Progol. *New generation Computing, 13(3,4)*, 245-286.

[67] Murphy, P. M., & Aha, D. W. (1994). *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California, Department of Information and Computer Science.

[68] Nakamura, A., Tsumoto, S., Tanaka, H., & Kobayashi, S. (1996). Rough Set Theory and Its Applications, *Journal of Japanese Society for Artificial Intelligence, 11(2)*, 35–41.

[69] Van Mechelen, I., Hampton, J., Michalski, R. S., Theuns, P., (1993). *Categories and Concepts. Theoretical Views and Inductive Data Analysis*, Academic Press.

[70] Nguyen, D. T., Ho, T. B. (1999). An Interactive-Graphic System for Decision Tree Induction. *Journal of Japanese Society for Artificial Intelligence, 14*, 131–138.

[71] Niblett, T. (1987). Constructing Decision Trees in Noisy Domains. In Bratko, I. & Lavrac, N. (Eds.), *Progress in Machine Learning*,Sigma Press, 67–78.

[72] Pagallo, G. & Haussler, D. (1990). Boolean Feature Discovery in Empirical Learning. *Machine Learning, 5*, 71–99.

[73] Pawlak, Z., Wong, S. K. M., & Ziarko, W. (1988). Rough Sets, Probabilistic versus Deterministic Approach. *International Journal of Man-Machine Studies, 29*, 81–95.

[74] Pawlak, Z. (1991). *Rough sets: Theoretical Aspects of Reasoning About Data*, Kluwer Academic Publishers.

[75] Plaisant, C., Carr, D., & Shneiderman, B, (1995). Image Browser Taxonomy and Guidelines for Designers. *IEEE Software, 12*, 21–32.

[76] Pompe, U., Kovacic, M., & Knononenko, I. (1993). SFOIL: Stochastic Approach to Inductive Logic Programming. In *Proceedings of the 2nd Slovenian Conference on Electrical Engineering and Computer Science (ERK-93)*, Vol. B, 189-192.

[77] Quinlan, J. R. (1979). Discovering Rules by Induction from Large Collections of Examples. In D. Michie (Eds.), *Expert Systems in the Micro Electronic Age*. Edinburgh: Edinburgh University Press.

[78] Quinlan, J. R. (1983). Learning Efficient Classification Procedures and Their Application to Chess End Games. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann.

[79] Quinlan, J. R. (1986). Induction of Decision Trees, *Machine Learning, 1*, 81–96.

[80] Quinlan, J. R. (1987a). Simplifying decision trees. *International Journal of Man-Machine Studies, 27(3)*, 221–234.

[81] Quinlan, J. R. & Rivest, R. (1989). Inferring Decision Trees Using the Minimum Description Length Principle. *Information and Computation, 80(3)*, 227–248.

[82] Quinlan, J. R. (1990). Learning Logical Definitions from Relations. *Machine Learning, 5*, 239–266.

[83] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Los Altos, Morgan Kaufmann.

[84] Quinlan, J. R. (1996). Improved Use of Continuous Attributes in C4.5. *Journal of Artificial Intelligence, 4*, 77–90.

[85] Oates, T. & Jensen, D. (1998). Large Datasets lead to Overly Complex Models: an Explanation and a Solution. In Agrawal, R. & Stolorz, P. (Eds.), *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 294–298.

[86] Oates, T. & Jensen, D. (1998). Large Datasets Lead to Overly Complex Models: An Explanation and a Solution. In Agrawal, R. & Stolorz, P. (Eds.), *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 294–298.

[87] Rao, J. S., & Potts, W. J. E. (1997). Visualizing Bagged Decision Trees. *Proceedings of Third International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 243–246.

[88] Card, S. K., Mackinlay, J. D., & Shneiderman, B. (1999). *Readings in Information Visualization*, Morgan Kaufmann.

[89] Rissanen, J. (1978). Modeling by Shortest Data Description. *Automatica, 14*, 465–471.

[90] Rivest, R. L. (1987). Learning Decision Lists. *Machine Learning, 2(3)*, 229–246.

[91] Robertson, G. G., Mackinlay, J. D., & Card, S. K. (1991). Cone Trees: Animated 3D Visualization of Hierarchical Information. *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 189–194.

[92] Rumelhart, D. E. & McClelland, J. L. (1986). *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. 1: Foundations*, MIT Press.

[93] Salzberg, S. L. (1995). On Comparing Classifiers: A Critique of Current Research and Methods. *Technical Report JHU-95/06*, Department of Computer Science, Johns Hopkins Univ.

[94] Schaffer, C. (1993). Selecting a Classification ethod by Cross-Validation, *Machine Learning 13*, 135–143.

[95] Schaffer, C. (1993). Overfitting Avoidance as Bias, *Machine Learning 10*, 153–178.

[96] Utgoff, P. E., & Brodley, C. E. (1991). *Linear Machine Decision Trees*, (COINS Technical Report 91-10). University of Massachusetts, Amherst, MA.

[97] Theron, H., Cloete, I. (1996). BEXA: A Covering Algorithm for Learning Propositional Concept Descriptions. *Machine Learning, 24*, 5-40.

[98] Tsumoto, S. (1999). Information About Clinical Databases on Meningoencephalitis, *Proceedings 42th SIG-KBS Symposium on Comparison and Evaluation of Knowledge Discovery Methods Using A Common Dataset*, Japanese Society for Artificial Intelligence, 1–5.

[99] Webb, G., I. (1993). Learning Disjunctive Class Descriptions by Least Generalization. (TR C92/9 Technical Report). Deakin University, School of Computing and Mathematics, Geelong, Australia.

[100] Weiss, S. M. & Indurkhya, N. (1991). Reduced Complexity Rule Induction. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, 678–684.

[101] Wild, C. & Weber, G. (1995). *Introduction to Probability and Statistics*. University of Auckland.

[102] Witten, I. H. & Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques With Java Implementations*, Morgan Kaufmann.

[103] Ziarko, W. (1993). Variable Precision Rough Set Model. *Journal of Computer and System Sciences, 46*, 39–59.

Publications

[1] T.D. Nguyen & T. B. Ho: "An Interactive-Graphic System for Decision Tree Induction", *Journal of Japanese Society for Artificial Intelligence*, Vol. 14, N. 1, 1999, 131-138.

[2] T.D. Nguyen, T.B. Ho, & H. Shimodaira: "A Visualization Tool for Interactive Learning of Large Decision Trees", *Proceedings of 12th IEEE International Conference on Tools With Artificial Intelligence, ICTAI 2000*, IEEE, November 2000.

[3] T.D. Nguyen, T.B. Ho, & H. Shimodaira: "A Scalable Algorithm for Rule Post-Pruning of Large Decision Trees", *Proceedings of 5th Pacific-Asaia Conference, PAKDD 20001*.

[4] T.D. Nguyen, T.B. Ho, & H. Shimodaira: "Interactive Visualization in Mining Large Decisions", *4th Pacific-Asaia Conference, PAKDD 2000*, Kyoto, Appril 2000,*Lecture Notes in Artificial Intelligence 1805*, Springer, April 2000, 345-348.

[5] T.B. Ho, T.D. Nguyen, H. Shimodaira, M. Kimura, & N.B., Nguyen: "A knowledge discovery system with focus on model selection and visualization", (submitted to *Applied Intelligence*).

[6] T.B. Ho, T.D. Nguyen, & N.B. Nguyen: "An Agent-based Architecture in Knowledge Discovery and Data Mining", *1st Pacific-Asia Conference Intelligent Agent Technology*, Hongkong, December 1999, World Scientific, 259-263.

[7] T.B. Ho & T.D. Nguyen: "Integrating Human Factors With An Concept Formation Process", *Proceedings 6th International Conference on Human-Computer Interaction*, Yokohama, July 1995, 74.

[8] T.B. Ho, T.D. Nguyen, & M. Kimura: "Induction of Decision Trees Using Rough Sets", *Proceedings Fifth Conference of the International Federation of Classification Societies IFCS'96*, Kobe, March 1996, 148-151. Revised version "Induction of Decision Trees Based on the Rough Set Theory" published in the book *Data Science, Classification and Related Methods*, C. Hayashi et al. (Eds.) Springer-Verlag Tokyo, June 1997.

[9] T.B. Ho & T.D. Nguyen: "Evaluation of Attribute Selection Measures in Decision Tree Induction", *Proceedings 9th International Conference on Industrial Engineering Applications of Artificial Intelligence Expert Systems*, Fukuoka, June 1996, Gordon and Breach Publisher, 413-418.

[10] T.B. Ho, T.D. Nguyen, H. Shimodaira, & M. Kimura: "An Interactive-Graphic Environment for Discovering and Using Conceptual Knowledge", *Proceedings 7th European-Japanese Conference on Information Modelling and Knowledge Bases*, Toulouse, May 1997, 327-343. *Information Modelling and Knowledge Bases IX*, IOS Press.

[11] T.B. Ho & T.D. Nguyen: "Interactive Visualisation for Predictive Modelling with Decision Tree Induction", *Lecture Notes in Artificial Intelligence: Principle of Data Mining and Knowledge Discovery, Second European Symposium, PKDD'98*, Sep 1998, Springer, 158-166.