Algorithms for Finding Multivariate Discriminant Rules for Classification and Regression Trees

Yasuhiko Morimoto

March 2002

Abstract

Progress in technologies for data input, such as POS (Point Of Sales) systems, and technologies for data storage, such as high density magnetic or optical recording devices, have made it easier for enterprises to collect massive amounts of data and to store them on hard disk at a very low cost. From the early 90's, many enterprises have been interested in extracting previously unnoticed information that inspires new marketing strategies from these huge databases. Technologies for extracting such information, or knowledge, from huge databases are called "data mining."

Data mining covers technologies for association analysis, classification and regression, cluster analysis, and evolution analysis. Most of these have been widely studied in the field of databases, statistics, and machine learning. Data mining, in general, is focusing on efficiency so that we can handle emerging huge databases whose size is too large to be processed by the conventional techniques.

Among these technologies, the author focused on the association analysis and the classification and regression in this dissertation. The author considered association rules on numerical attributes while conventional data mining can only effective for categorical attributes. The accomplishment significantly expanded applications of the association analysis. Then, the author explored classification and regression problems. By utilizing techniques developed for the numerical association rules, the author proposed accurate and comprehensive classification and regression trees. Among these accomplishments, primary contributions of the author are the works on the classification and regression problems.

In general, huge databases often contain many attributes and there are many correlations among attributes. However, conventional data mining techniques cannot handle correlations well. In the statistics literature, multivariate analysis methods have been used to handle correlations in numerical databases. Many statistical methods, such as "principal component analysis," "factor analysis," and so forth, are categorized as multivariate analysis. Most of the methods in the multivariate analysis assume a linear correlation. Such conventional techniques are effective for data that have linear correlations. However, data contain various types of correlations that cannot be handled by the conventional methods.

In order to handle various types of correlations, the author proposed multivariate optimized discriminant rules that can be defined on more than one attribute and presented efficient algorithms for finding the rules. The algorithms efficiently find multivariate discriminant rules such as optimized region rules that can be defined on two numeric attributes, and optimized conjunctive rules that can be defined on arbitrary number of categorical attributes.

Such multivariate discriminant rules are capable of representing various types of correlations among attributes. These achievements have been applied to classification and regression tree learning, which is one of the most widely used methods for inductive inference. Diverse experiments show that the proposed multivariate rule approach can create compact trees whose accuracy is better than that of conventional trees.

In Chapter 1, the author summarizes the background and motivation of the study of data mining.

In Chapter 2, the author discussed association rules, which is the most fundamental functionality of data mining. For numerical attributes, conventional data mining techniques are not applicable. The author considered optimized numerical association rules and proposed efficient algorithms for finding optimal rules, in particular, optimized numerical association rules (range rules) and optimized two dimensional association rules (region rules).

Next, the author explored classification problems in Chapter 3. How to find high quality discriminant rules is the key for modeling and predicting unknown value, which we are focusing on. The author proposed efficient algorithms for finding rules on categorical attributes by using techniques of computational geometry. For numerical attributes, the author modified the efficient algorithms for optimized numerical association rules (range rules) and optimized two dimensional association rules (region rules) to find discriminant rules.

Then, in Chapter 4, the author consider effective ways of utilizing discriminant rules in decision trees. Conventional decision trees that use univariate discriminant rules used to suffer from correlations. The author proposed to use multivariate rules, such as region rules, for handling correlation in a database.

In Chapter 5, regression problems are discussed. The author utilized the algorithms of multivariate rules for regression problems and proposed regression trees that use such rules.

Finally, Chapter 6 summarizes the achievements of these works and gives some open problems towards future direction of this study.

Acknowledgements

The author would like to take this opportunity to express his deep appreciation to those who supported him to achieve the study.

The author thank researchers who were or have been in the data mining project at IBM Tokyo Research Laboratory and supported him to complete this study. Professor Shinichi Morishita, currently with the University of Tokyo, Professor Takeshi Tokuyama, currently with Tohoku University, Dr. Takeshi Fukuda, Mr. Hirofumi Matsuzawa, and Mr. Kunikazu Yoda, provided invaluable assistance throughout the study. Mr. Yoshihiro Ohta and Mr. Hiromu Ishii helped him for implementing the system and provided various data mining functions that are indispensable for the system.

Thanks are also due to the past and present members of the data mining project at IBM Almaden Research Center. In particular, Dr. Rakesh Agrawal, Dr. Manish Mehta, Dr. John Shafer, Dr. Ramakrishnan Srikant, and Dr. Kyuseok Shim gave him invaluable advice in the early stage of the study and gave him data mining tools such as visualization tools.

The author owe it to Professor Tadao Ichikawa of Hijiyama University and Professor Hasahito Hirakawa of Hiroshima University that the author was able to complete the dissertation. They have always been giving invaluable guidance since the author was in Hiroshima University. The author would not be able to pursue the study without their continuous encouragement after the author finished his master's thesis by their supervision at Hiroshima University.

The author would like to thank Professor Tadashi Ae, Professor Kenichi Morita, Professor Ryuei Nishii, Professor Toshimasa Watanabe, and Professor Masahito Hirawaka who served as examination committee of the dissertation in Graduate School of Information Engineering, Hiroshima University for their valuable comments and suggestions for improving this dissertation.

This study was partly supported by the Advanced Software Enrichment Project of the Information-Technology Promotion Agency (IPA), Japan. Thanks to the support, the author, jointly with members of Stanford University, the University of Wisconsin, Madison, IBM Almaden Research Center, and IBM Tokyo Research Laboratory, published an advanced data mining system that contains the results of this study.

Some results of the study have been productized and have been utilized as powerful data mining tools in Japanese market. In 1998, IBM Japan released a product called "Default Meter," which estimates the credit risk of companies. In 1999, Daiwa Securities and IBM jointly developed a product called "Rating Meter," which evaluates the current rating of a company by using financial statements and various kinds of data on macro economy's conditions and predicts future movements of the ratings. Those products used the data mining techniques of this study as an essential part of their functions. Clearly, these products would not exist without the contributions of a great many talented people of the project members of these products in Daiwa Securities and IBM Japan.

Finally, the author would like to express his deep appreciation to many editors and editorial assistants for helping related publication works of this study and to anonymous reviewers for valuable comments for improving quality of this study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Progress in technologies for data input, such as POS (Point Of Sales) systems, and technologies for data storage, such as high density magnetic or optical recording devices, have made it easier for enterprises to collect massive amounts of data and to store them on hard disk at a very low cost. As a result, a lot of enterprises have been eager to utilize their stored information property in more intelligent ways.

Many enterprises have been interested in extracting previously unnoticed information that inspires new marketing strategies from these huge databases. Technologies for extracting such information, or knowledge, from huge databases are called "data mining."

Data mining covers technologies for finding frequent or rare patterns, modeling and predicting value of an attribute, grouping or clustering data, classifying data, and so forth. Most of these have been widely studied in the field of databases, statistics, and machine learning. Data mining, in general, is focusing on efficiency so that we can handle emerging huge databases whose size is too large to be processed by the conventional techniques [PSF91, PS91, AIS93a, FPSSU96, Han98].

1.1 Towards Strategic Data Analysis

Many enterprises have been interested in utilizing their information property in more intelligent ways. Consequently, such enterprises have been investing in *decision support* applications in which current and stored data are comprehensively analyzed and explored.

Databases that manage daily operations, which we call *operational databases*, are updated frequently. Most of the update operations, say *transactions*, make small changes into the databases. However, operational databases must be reliable and must be able to handle a large number of transactions efficiently. We call technologies to realize the requirement of such operational databases **OLTP (OnLine Transaction Processing)**. Many enterprises have several kinds of operational databases for each daily operation such as sales management, reservation management, stock management, and so forth.

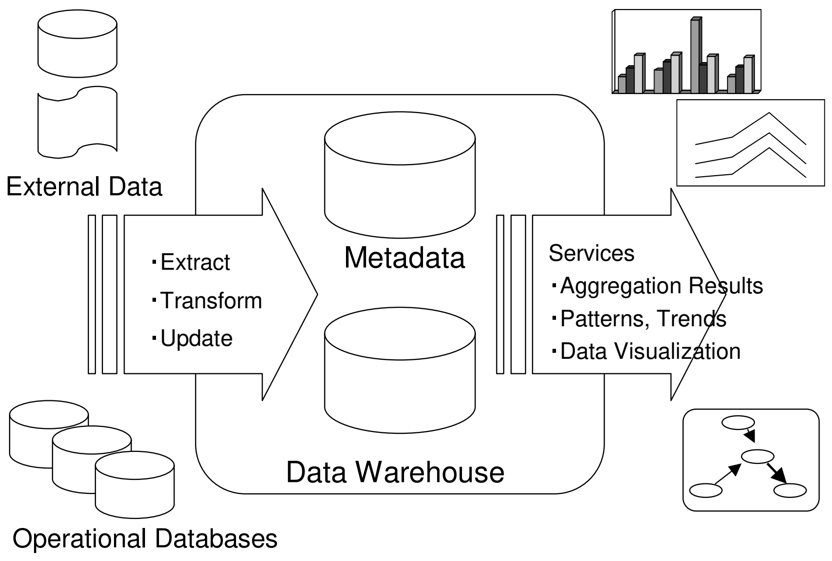


Figure 1.1: Data Warehouse

1.1.1 Data Warehousing

From the early 1990's, many enterprises have constructed *data warehouses* to manage their information resources that are stored in each operational database and conventional database (for example a customer database and an employee database), which is not frequently updated like operational databases [Inm92]. Figure 1.1 illustrates conceptual data warehouse architecture.

Data warehouses extract information from several data sources for example data from external web sites in addition to their operational databases. Sometimes, data needs to be transformed. One of the typical cases in which we need to transform data is to adjust the unit of numbers, such as currency (prices in US dollar into Japanese Yen), length (inches into centimeters), weight (pounds into grams), volume (ounces into cubic centimeters), and temperature (Fahrenheit into Celsius), to be consistent in the data warehouse. Tables and records from many operational databases are copied into the data warehouse. In general, some projected tables and / or selected records are redundantly copied into several tables in the data warehouse in order to realize good scalability and high availability. Data in a data warehouse is periodically refreshed to reflect updates in operational databases. Data warehouses manage meta information that is needed to manage data extraction, transformation, and refreshment.

In addition to the mentioned data management tasks, data warehouses have to provide a variety of services for data analysis such as **OLAP (OnLine Analytical Processing)** queries, data mining, visualizations, statistical calculations, and reportings.

Assume a sales database in Table 1.1. The database is a collection of facts that "the quantity of the product was sold by the customer at the shop with the price." The OLAP query functions are the most fundamental service of data warehouses. Typical OLAP queries on the table are kinds of data aggregation, i.e., computing average (`AVG`), count (`COUNT`), maximal and minimal value (`MIN, MAX`), sum (`SUM`), and so on. Such queries can be described with `GROUP BY` predicate

Table 1.1: Sales Database

Date	Shop	Product	Customer	Price	Quantity
1999/07/16	Shop-1	Chips-A	Customer-X	20,000	2
1999/07/16	Shop-2	Beer-B	Customer-Y	8,000	4
1999/07/19	Shop-3	Aspirin-A	Customer-Z	12,0000	1
⋮	⋮	⋮	⋮	⋮	⋮

Table 1.2: Shop Database

Shop	Category	Country	State	City
Shop-1	Foods	US	California	San Jose
Shop-2	Liquor	Japan	Tokyo	Machida
Shop-3	Pharmacy	Australia	New South Wales	Sydney
⋮	⋮	⋮	⋮	⋮

in SQL statements, corresponding function names in SQL statements are in the parenthesis respectively. For example, the following SQL statement,

```
SELECT "Shop", SUM("Price") FROM "Sales"
GROUP BY "Shop",
```

computes the total sales of each shop. We can grasp some kinds of trends in the database by aggregating data in various aspects.

Assume another database, containing shop information, in Table 1.2. If we join the two relations, sales and shop, we can aggregate the databases in different views that could not be considered by each one, for example, aggregation by each category, aggregation by each country, and so forth. For each view point, i.e., group-by predicate, we can define a dimension in the database space. We can further define various *data cubes* by combining these dimensions [GBLP95]. For any position in a data cube, we can assign corresponding aggregated result.

The OLAP query functions have to respond to many ad hoc requests, which are coming online, within a short time period. In order to achieve better service level, various techniques have been proposed for the OLAP query functions [GHQ95, HRU96, AAD$^+$96], for example, precomputing intermediate results that are considered to be used frequently in many queries.

1.1.2 Knowledge Discovery and Data Mining Process

As mentioned above, constructing a data warehouse is one of the effective means for utilizing information property in large databases. Many enterprises expect to find useful knowledge for their decision making by data analysis services perhaps provided by a data warehouse.

Data analysis services such as OLAP queries, data minings, visualizations, statistical calculations, and reportings are considered to be kinds of data abstraction or summarization services. Figure 1.2 shows the conceptual analytical processes that are roughly separated into five steps

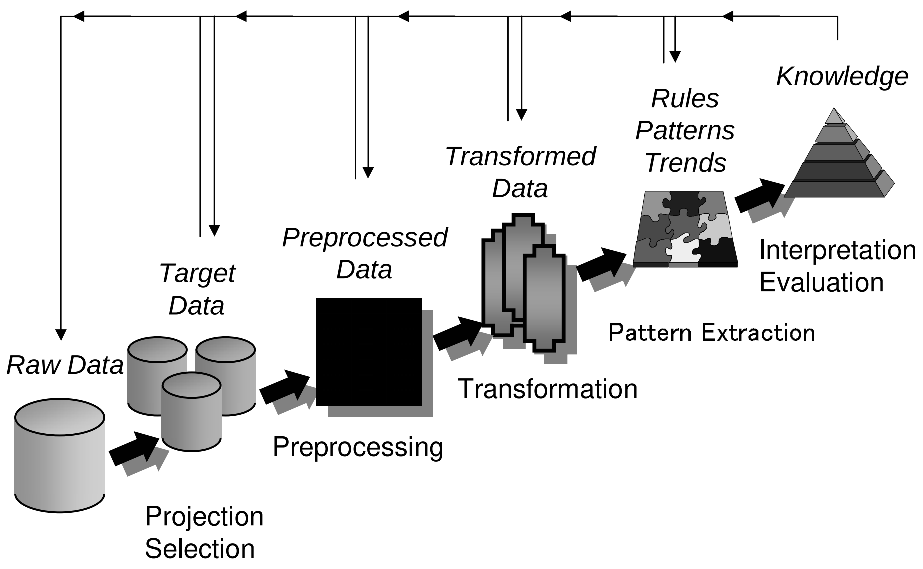


Figure 1.2: Knowledge Discovery and Data Mining Process

according to the level of abstraction. These analytical processes are often referred as **KDD (Knowledge Discovery and Data mining) processes**.

First of all, the raw data goes to the "projection / selection" step in which we identify the target dataset and relevant attributes. Then, in the "preprocessing" step and in the "transformation" step, we remove noise, transform values to common units, generate new attributes through combination of existing attributes, and bring data into a table that is used as an input to data mining tasks. In the "pattern extraction" step, we extracts the actual patterns in the dataset. Finally, in the "interpretation / evaluation" step, we present the patterns in an understandable form for example through data visualizations. The results of any step might feedback to an earlier step in order to redo the process with the new knowledge gained.

In general, data mining and statistical calculations are considered to be the higher level tasks in the KDD process, which roughly cover the pattern extraction step and the interpretation / evaluation step. Though many functions of data mining and statistical calculations are overlapped, the former tends to be "hypothesis finding" tasks while the latter tends to be "hypothesis testing" tasks. Data mining, which is new emerging technology that attracts many enterprises from the early 1990's, has hypothesis finding features in other word "knowledge discovery" features.

1.1.3 Data Mining

Data mining consists of technologies for finding hypothesis or knowledge in large datasets. The findings are expected to be utilized for decision making. Some of the findings might give a data analyst unexpected insights that can be more carefully investigated by using statistical hypothesis testing techniques.

A lot of technologies have been proposed, for example, for finding, frequent or rare patterns, modeling and predicting value of an attribute, grouping or clustering data, classifying data, and so forth. Most of these had been widely studied in the field of databases, statistics, and machine learning. Data mining, in general, is focusing on efficiency so that we can handle emerging huge databases whose size is too large to be processed by the conventional techniques.

Several data mining functions, which can efficiently handle huge databases, have been developed so far. Roughly speaking, all of these functions can be classified into one of four categories, 1) functions for **association analysis**, 2) functions for **classification and regression**, 3) functions for **cluster analysis**, and 4) functions for **evolution analysis**.

Association Analysis

Association analysis is used for discovering *association rules* that occur frequently together in a given database. If we can find a rule "a customer who purchases `A` and `B` tend to purchase `C`," the retailer can consider strategic sales promotions for the goods. For example, they can arrange the layout of goods in a store or in a catalog page so that `A`, `B`, and `C` are close to each other.

Such rules are called *association rules*. An association rule has the form:

$$LHS \Rightarrow RHS, (support, confidence)$$

where both LHS, left hand set, and RHS, right hand set, are sets of predicates.

Market basket analysis is the most typical applications of association analysis. In the market basket analysis, a collection of items purchased by a customer, called market basket, in purchases *transactions* is analyzed by retailers to identify items that frequently purchased together. In the market basket analysis, both LHS and RHS are sets of items. In addition, the form implies that a customer who purchases all items in LHS tends to purchase all items in RHS.

Each association rule has two measures *support* and *confidence*. The support is the probability of records, or transactions in market basket analysis, that satisfies all of predicates in the union of LHS and RHS. The confidence is the probability of records satisfying all of predicates in the union of LHS and RHS over records satisfying all predicates in LHS.

Among all possible association rules, we are interested in rules such that both support and confidence are large enough for users. Agrawal, Imielinski, and Swami [AIS93b] proposed a method for enumerating all association rules that have larger support value than a user specified minimal support and confidence value.

Classification and Regression

Classification, or regression, is the process of finding a set of models that describe and distinguish data classes, or data values (resp.), for the purpose of being able to use the models to predict the class, or the value (resp.), of a record whose class, or value (resp.), is unknown.

Assume that we have a weather forecast database that consists of attributes, `Weather`, actual weather of the next day, `Forecast`, a weather forecast for the next day, and `Pressure` and

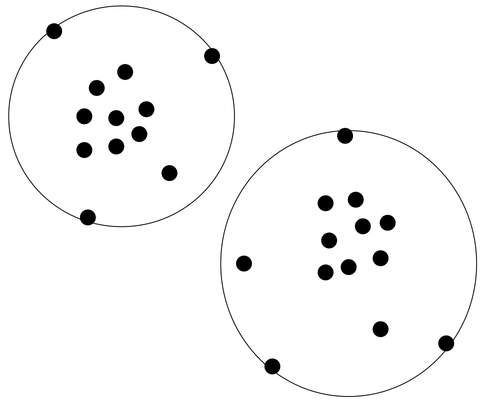


Figure 1.3: Minimize Diameter of the Maximal Cluster of Two Clusters

`Temperature` on today.

If we are interested in weather of the next day, we want to know which rules are important to know weather of the next day. However, we want to predict weather of future from observed data whose actual weather value is unknown. In such cases, we construct prediction models for `Weather` by using other attributes in the database, whose `Weather` value is known. Forms of the prediction models can be represented as rules, rule lists, trees, neural networks, or Bayesian networks.

Cluster Analysis

Clustering is to partition a set of records into groups such that records within a group are similar to each other. Each such group is called a *cluster*. Similarity between records is measured by a *distance function*. We can choose or define a distance function for each application.

In general, a distance function, $f(p_1, p_2)$, takes two records as input and returns a value that is a measure of their similarity. We often use a function based on Euclidean distance of two input records that have numerical attributes. The clustering is a kind of optimization problem to make the optimal clusters in a certain criterion for example to minimize sum of diameters of all clusters, to minimize the maximal diameter of clusters like Figure 1.3, and so forth.

Unlike classification and regression analysis, which need database records whose class or value is known, clustering analyzes database records without consulting a known class or a value. In general, classes or values are not present in database records when we use clustering analysis. Clustering can be used to generate such classes or values.

Evolution Analysis

Evolution analysis describes and models regularities or trends for database records whose behavior changes over time. Though this may include functions for association analysis, classification and regression, or cluster analysis of *time-related* databases, distinct features of the evolution analysis include time-series data analysis, sequence or periodical pattern matching, and similarity-based data analysis.

Time-series data mining and *sequential pattern* mining are typical examples of the evolution analysis. Assume we have the stock market (time-series) databases. Someone wants to identify stock evolution regularities for overall stocks and for the stocks of particular companies. Such regularities may help predict future trends in stock market prices. Sequential pattern mining is the mining of frequently occurring patterns related to time or other sequences. An example of the sequential pattern is "customers who bought `A` are likely to buy `B` within one month."

1.2 Accomplishments

As mentioned in the previous section, data mining covers technologies for association analysis, classification and regression, cluster analysis, and evolution analysis. Among these technologies, the author focused on the association analysis and the classification and regression in this dissertation. The author considered association rules on numerical attributes while conventional data mining can only effective for categorical attributes. The accomplishment significantly expanded applications of the association analysis. Then, the author explored classification and regression problems. By utilizing techniques developed for the numerical association rules, the author proposed accurate and comprehensive classification and regression trees. Among these accomplishments, primary contributions of the author are the works on the classification and regression problems.

1.2.1 Contributions on Association Analysis

In general, there are two types of attributes: *categorical* and *numerical* attributes in a database. Conventional algorithms for finding association rules assume that attributes, which are investigated, are all categorical. However, in many systems we have to analyze data of numerical attributes especially for financial applications. Association analysis for numerical attributes is necessary for such databases containing numerical attributes. Therefore, the author considered association rules for numerical attributes.

General form of the association rules is:

$$P_1(C_1) \wedge P_2(C_2) \cdots \wedge P_i(C_i) \Rightarrow P_{i+1}(C_{i+1}) \wedge P_{i+2}(C_{i+2}) \cdots \wedge P_k(C_k)$$

where C_i $(1 \leq i \leq k)$ are attributes that are analyzed. The $P_i(C_i)$ are predicates that involve attribute C_i.

For categorical attributes, the form of the predicates would be $t[C_i] = v_i$ where $t[C_i]$ is the value of the attribute C_i of a certain record and v_i is a value in the domain of C_i.

Conventional algorithms for finding association rules work efficiently if predicates has the form $t[C_i] = v_i$. If C_i is a numerical attributes, the number of distinct values becomes large and hence support value becomes much smaller. Such rules whose support is small are not significant. Therefore, for numerical attributes, the form of the predicate would be $t[C_i] \in [v_{i-lo}, v_{i-hi}]$ where

v_{i-lo} and v_{i-hi} ($v_{i-lo} < v_{i-hi}$) are threshold values in the domain of C_i. In the predicate, v_{i-lo} can be $-\infty$ or v_{i-hi} can be ∞.

Association rules on a numerical attribute are often called "numerical association rules" or "quantitative association rules." Srikant and Agrawal proposed an algorithm for finding rules [SA96] of the following form:

$$\{t[C_1] \in [v_{1-lo}, v_{1-hi}]\} \wedge \cdots \wedge \{t[C_i] \in [v_{i-lo}, v_{i-hi}]\} \Rightarrow t[C_k] = v_k, (support, confidence)$$

The rule implies that if the value of C_i of a record r falls in the range $[v_{i-lo}, v_{i-hi}]$, the record's value of C_k is likely to be v_k." The support of the rule is the probability of records that satisfy both $\{t[C_1] \in [v_{1-lo}, v_{1-hi}]\} \wedge \cdots \wedge \{t[C_i] \in [v_{i-lo}, v_{i-hi}]\}$ and $t[C_k] = v_k$. The confidence is the probability of records that satisfy $t[C_k] = v_k$ among the records satisfying $\{t[C_1] \in [v_{1-lo}, v_{1-hi}]\} \wedge \cdots \wedge \{t[C_i] \in [v_{i-lo}, v_{i-hi}]\}$.

Like association rules for categorical attributes, we are interested in rules whose support and confidence value are large. We have a freedom to choose the value range $[v_{i-lo}, v_{i-hi}]$ of a numerical attribute C_i when we search for rules. In general, if we choose a wider range on the numerical attribute, we will have larger support value, however, the corresponding rules tend to converge on the average confidence value, it means the rules are less significant. On the other hand, if we choose a narrower range, we can find rules whose confidence value is significantly high or low though the support values are low.

Optimized Numerical Association Rules

The authors considered *optimized numerical association rules* [FMMT96d, FMMT99] of the form

$$\{t[C_i] \in [v_{i-lo}, v_{i-hi}]\} \Rightarrow t[C_k] = v_k, (support, confidence).$$

In this work, we focused on computing two *optimized ranges*: one that maximizes the support on the condition that the confidence ratio is at least a given threshold value, and the other maximizes the confidence ratio on the condition that the support is at least a given threshold number. The former rules are called *optimized support rules* and the latter rules are called *optimized confidence rules.*

The optimized support rules can be used to identify the largest set of records on a numerical attribute whose minimum confidence is more than the specified value. On the other hand, the optimized confidence rules can be used to identify the densest set of records on a numerical attribute whose minimum support is more than the specified value.

Figure 1.4 shows the optimized confidence rule and the optimized support rule on numerical attribute A.

There are trivial ways of computing optimized support rules and optimized confidence rules in $O(n^2)$ time for each numerical attribute, where n is the number of all records in a database. We proposed a non-trivial linear time algorithm for each optimized rule, on the assumption that the data are sorted with respect to the numeric attribute, it means we spend $O(n \log n)$ time

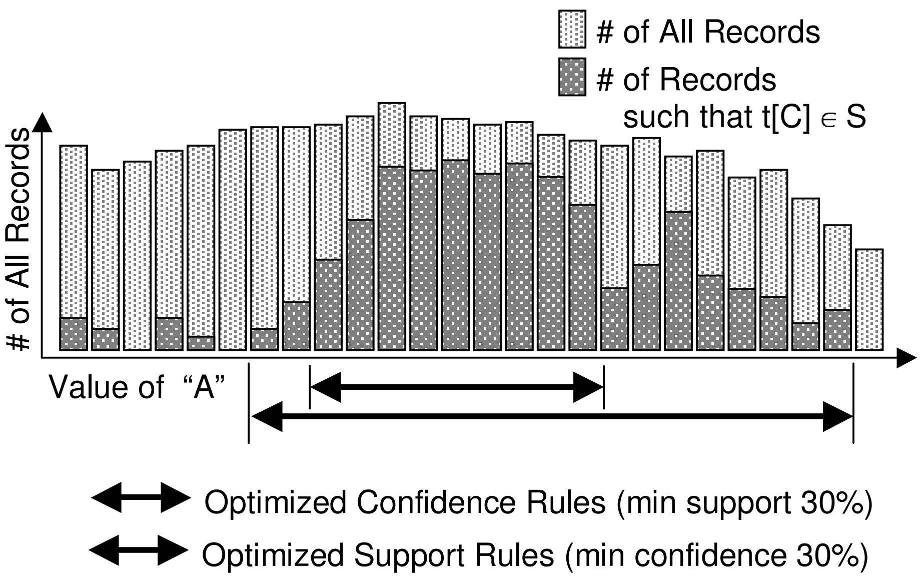


Figure 1.4: Optimized Numerical Association Rule

for the preprocessing. Each algorithm uses some computational geometry techniques to achieve the linear time complexity.

Optimized Two-Dimensional Numerical Association Rules

It would also be valuable to extend our framework to rules with two numeric attributes in the presumptive condition, and to find the region in the two-dimensional space of these attributes that represents a nice association rule between these two numeric attributes and the conclusion. For instance, we would like to find a rule such as

$$(Age, Balance) \in X \Rightarrow (CardLoan = yes),$$

where X is a rectangle or a connected region in two-dimensional space of Age and $Balance$. Optimized rules can also be defined naturally in this extension.

We considered regions that can be defined on a two-dimensional pixel grid plane G. While the problem of finding the optimal arbitrary connected pixel grid region is NP-hard, we proposed practical solutions for the cases where the regions are x-monotone, rectilinear convex, or rectangular [FMMT96b, YFM$^+$97, FMMT01].

Assume that we focus on two numerical attribute B and C. We distribute the values of B (resp. C) into N_B (N_C) buckets so that every bucket contains almost the same number of records. We then divide the Euclidean plane associated with B and C into $N_B \times N_C$ pixels (unit squares). For simplicity, we assume that $N_B = N_C = N$, without loss of generality, as regards our algorithms.

A *grid region* is a union of pixels in G that are connected. An *x-monotone* region is a grid region whose intersection with any vertical line is undivided. For example, Figure 1.5 (a) shows

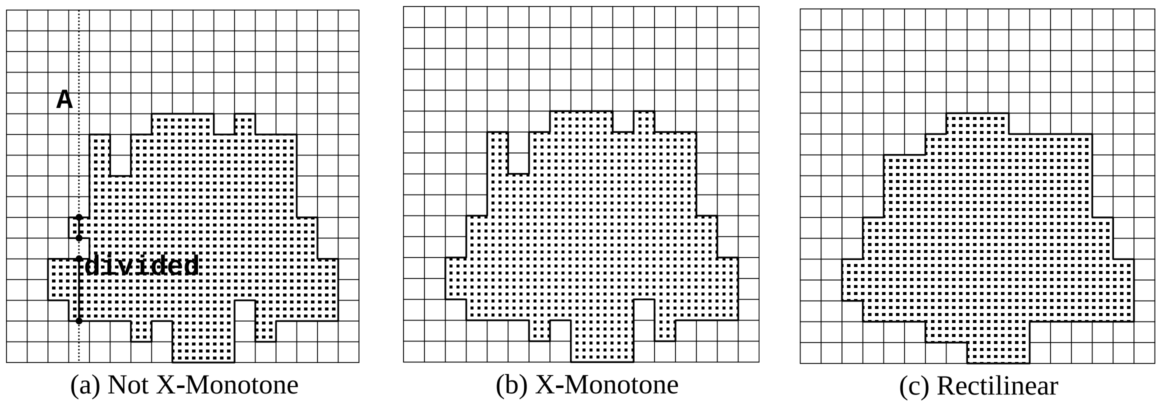


Figure 1.5: Region Families

a region that is not x-monotone, since the intersection of the vertical line `A` and the region is divided. In contrast, Figure 1.5 (b) shows an x-monotone region. We can express an x-monotone region by a disjunction of expressions, where each disjunct has the form $(a1 \le x \le a2)$ `and` $(b1 \le y \le b2)$, where the union of disjuncts does not divide a vertical column. A *rectilinear* convex region is an x-monotone region such that its intersection with any horizontal line is also undivided. Figure 1.5 (c) shows an example of a rectilinear convex region. A *rectangular* region is a rectangle on G, and is thus a rectilinear convex region. We will consider the class of x-monotone regions, the class of rectilinear convex regions, and the class of rectangular regions.

We considered algorithms for computing optimized two-dimensional association rules, i.e., both optimized support two-dimensional association rules and optimized confidence two-dimensional association rules. We found that the optimized two-dimensional association rules that optimize support or confidence can be computed in time proportional to $O(N^2 n)$, $O(N^3 n)$, and $O(N^3)$ where n is the number of records in the whole grid G for x-monotone regions, rectilinear convex regions, and rectangular regions, respectively [FMMT96b]. However, the n is unacceptably large in data mining applications. Therefore, for x-monotone and rectilinear convex regions, we considered another efficient algorithms that closely approximate the optimized two-dimensional regions. Those algorithms run in time proportional to $O(N^2 \log n)$ and $O(N^3 \log n)$ for x-monotone regions and rectilinear convex regions respectively.

1.2.2 Contributions on Classification and Regression

Assume that we have a weather forecast database that consists of attributes, `Weather`, actual weather of the next day, `Forecast`, a weather forecast for the next day, and `Pressure` and `Temperature` on today. In addition assume that we are interested in the `Weather` value, i.e., weather of the next day. Some association rules may affect the `Weather` value and some may not. We want to know which rule is the most important for the `Weather` value.

Classification is the process of finding a set of models that describe and distinguish data classes for the purpose of being able to use the models to predict the class of database records

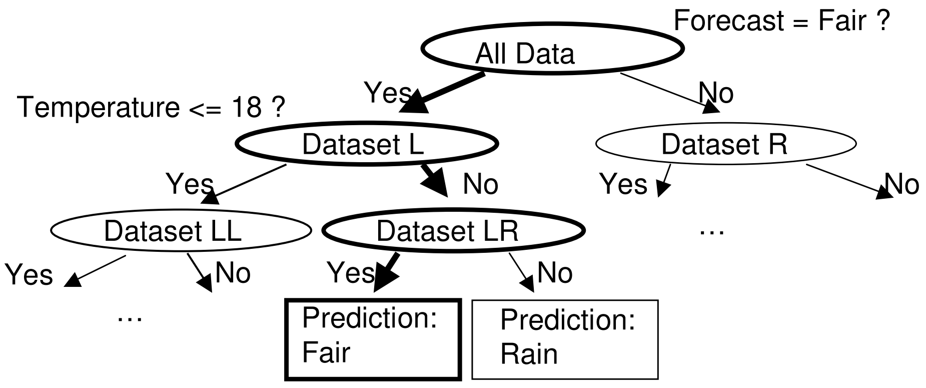


Figure 1.6: Decision Tree

whose class is unknown. In the example, the classes are values of the `Weather` attribute. In general, there is one designated attribute like `Weather` whose value of which we would like to predict or model in classification and regression. We call such an attribute the *target attribute*. When the target attribute is categorical, we call the modeling and prediction process classification. When the target attribute is numerical, on the other hand, we call the process regression.

Typical forms of the models are rules, rule lists, trees, neural networks, or Bayesian networks. Among them, the author focused on tree induction models in this dissertation.

A rooted tree, each of whose internal nodes is associated with a left-hand-side predicate of a rule, called a *discriminant rule* or a *test*. We associate each leaf node with the subset (called leaf-cluster) of records satisfying all tests on the path from the root to the leaf. Every leaf-cluster is labeled as one of the value of the target attribute on the basis of the target value distribution in the leaf-cluster.

For example, if we are interested in the `Weather` value, each leaf node has `Fair` or `Rain` as the label of the node. Such a tree-based prediction model is called a *decision tree* if the target attribute is categorical, while it is called a *regression tree* if the target attribute is numerical. Figure 1.6 shows an example of a decision tree which models and predicts the `Weather` value.

Decision trees and regression trees are very popular since they are easy to interpret. For example, highlighted leaf node in Figure 1.6 represents the rule: if "`Forecast` is Fair" and "`Temperature` is larger than 18" and "the test for LR node is satisfied" then "it is likely to be `Fair`." In addition, decision trees are accurate despite the limitations in structure. Therefore, many researches have been investigated after the pioneer works done by Breiman et al. [BFOS84] and Quinlan [Qui93].

In this dissertation, the author explores the tree induction models and proposes some significant improvements over the previous works on the literature. Though we can consider n-nary decision trees, we consider only binary trees like the example in the rest of this dissertation.

Multivariate Categorical Discriminant Rules

We associate each internal nodes of a tree model with a rule, which splits a dataset into two subsets. The choice of a rule at each internal node strongly affects the size and accuracy of the tree model. In general, the quality of a discriminant rule can be evaluated by how they discriminate a dataset on the view point of value distribution of the target value. We need to find as better rule as possible in order to construct compact and accurate tree models.

For a categorical conditional attribute C, the form of a rule is $t[C] \in V$ where V is a set of values on the domain of C. If C has n distinct values, *the conditional domain size*, there are $O(2^n)$ possible discriminant rules. It is not affordable to examine all possible rules exhaustively except for the cases where n is small.

Attributes in a database are often correlated. In such cases, we should consider rules on multiple categorical attributes, which are called *multivariate (categorical) rules*.

Let $C_1, C_2, \cdots, C_M$ be the conditional attributes. We can treat these attributes as a single attribute C whose domain is the Cartesian product of their domains, that is, $\mathrm{dom}(C) = \mathrm{dom}(C_1) \times \mathrm{dom}(C_2) \times \cdots \times \mathrm{dom}(C_M)$. If C_i, where $i = 1, 2, \cdots, M$, has n_i distinct values, the conditional domain size of C is $n = \prod_i n_i$ for $1 \leq i \leq M$.

If we consider multivariate (categorical) rules, the n tends to become large. Moreover, in a huge database, the n can be large even on a single categorical conditional attribute. Therefore, we have to develop algorithms that work even for large n.

Let k be the number of distinct values of the target categorical attributes, call it *the target domain size*. For the case where the target domain size is two, i.e., $k = 2$ or the target attribute is numerical, we can order the n values so that the best V is one "cut" of the ordered sequence [BFOS84]. Consequently, we have an $O(n \log n)$ algorithm. However, this algorithm is not applicable to cases in which the target domain size is greater than two.

For huge categorical databases, in which n is large and $k > 2$, there is no practical existing algorithm that can find the best discriminant rule. Despite the difficulty, there are some heuristics for handling the problem [BFOS84, MP91, Qui93] that are used in practice for constructing decision trees.

In this dissertation, the author proposes two algorithms, named the Random Enumeration Algorithm and the Probing Algorithm, which use computational geometry techniques. Both of the algorithms can feasibly compute a high quality V, which is much better than that of conventional heuristics's, for cases in which k is a small constant. The n is allowed to be very large.

Each possible V on a conditional attribute can be interpreted as a point in a k-dimensional space, and we can translate the problem of finding V into that of finding a point in the k-dimensional space. We proved that the best point must be on the convex hull of the point set of all possible value groups. Both of the algorithms compute a point on the convex hull efficiently, that is, in $O(n)$ time, without examining points inside the hull.

The Random Enumeration Algorithm examines points on the convex hull by using random sampling. It outputs the best point of the examined points, which are computed from a small sample. The time complexity is reduced to $O(s^{k-2}n)$, where $s(\ll n)$ is the sample size. We will empirically show that the Random Enumeration Algorithm finds a satisfactory V with a small sample. This algorithm can run with a small working space and can easily be parallelized.

The Probing Algorithm uses tangential hyperplanes to compute points on the convex hull. If we maintain a list of points that are examined during the search and incrementally construct inscribed and circumscribed convex polygons inside and outside the hull, we can find clues as to the next point to be examined. However, the total cost of the incremental convex polygon maintenance and probing of the convex hull is $O((n+m)|P|)$, if we have m points and $|P|$ facets on the convex hull. The m and P can be asymptotically as large as n^{k-1} and $m^{\lfloor \frac{k}{2} \rfloor}$, respectively, in a pathological input. Though they are known to be much smaller in a normal input, the incremental polygon maintenance is still costly when k becomes large.

The Probing Algorithm maintains only promising facets of the inscribed convex polygon, using some heuristics, and works within a limited working space. It can find a satisfactory value group in an earlier step of the algorithm. At every incremental step of the algorithm, it can report the best solution found so far, and thus the solution gradually converges to the best point. In cases where a quick response is required, the Probing Algorithm can return the best V found in the required time, and it is thus suitable for online applications.

Two-Dimensional Numerical Discriminant Rules

Quinlan [Qui93] pointed out that the approach that use discriminant rules on a single numerical conditional attribute has a serious problem if a pair of attributes is correlated. For example, let us consider two numerical attributes, "height (cm)" and "weight (kg)," in a health check database. Obviously, these attributes have a strong correlation. Indeed, the region $0.85 * 22 * \mathit{height}^2 < \mathit{weight} < 1.15 * 22 * \mathit{height}^2$ and its complement provide a popular criterion for separating healthy patients from patients who need dietary cures. In the left chart of Figure 1.7, the enclosed gray region shows the "healthy" region. However, if we construct a decision tree for classifying patients by using rules on a single attribute, its subdivision is complicated and the size of the tree becomes very large, and hence, it becomes hard to recognize the substantial rule.

Therefore, it is very important to propose a better scheme for handling numerical conditional attributes with strong correlations in order to make an efficient diagnostic system based on a decision trees.

In this dissertation, the author proposes the following scheme for the above problem, applying the two-dimensional association rules, region rules in short, of [FMMT96b, FMMT01] and an image segmentation algorithm of [ACKT96]. The scheme has been implemented as a subsystem of SONAR, which stands for System for Optimized Numeric Association Rules, developed by the authors [FMMT96e].

Let n be the number of records in the database. First, for each numeric attribute, we create an equi-depth bucketing so that records are uniformly distributed into $N \leq \sqrt{n}$ ordered buckets

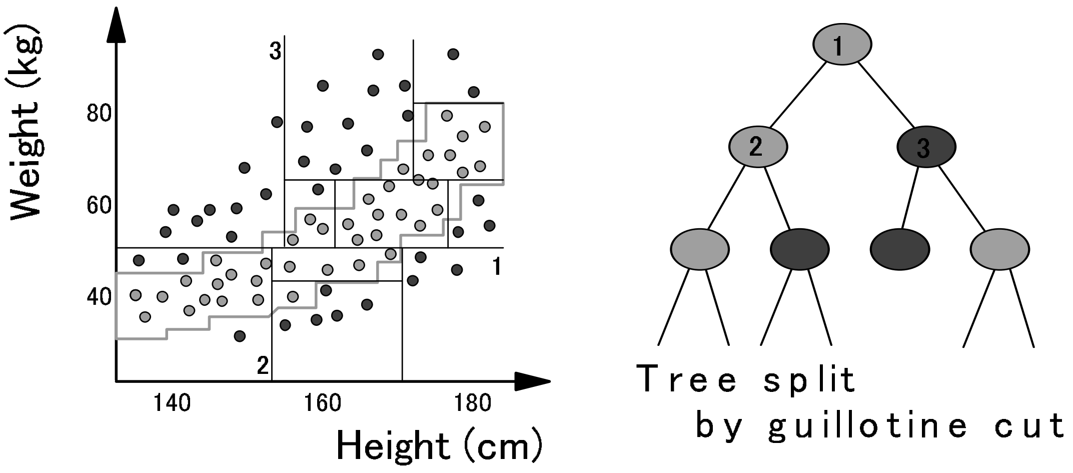


Figure 1.7: Healthy Region and Guillotine-cut Subdivision

according to the values of the attribute.

Next, we find all pairs of strongly correlated numeric attributes. For each such a pair A and A', we create an $N \times N$ pixel grid G according to the Cartesian product of the buckets of each numeric attribute. The problem of finding the optimal arbitrary connected pixel grid region as a discriminant rule is NP-hard like the optimized two-dimensional association rule. Therefore, we consider a family $\mathcal{R}$ of grid regions; in particular, we consider the set $\mathcal{R}(xmono)$ of all *x-monotone* connected regions and the set $\mathcal{R}(recti)$ of all *rectilinear convex* regions.

The author presents algorithms for computing the optimal two-dimensional discriminant rule in worst case times of $O(nN^2)$ and $O(nN^3)$ for $\mathcal{R}(xmono)$ and $\mathcal{R}(recti)$ respectively. Moreover, in practical instances, our algorithms run in $O(N^2 \log n)$ time and $O(N^3 \log n)$ time. Since $N \leq \sqrt{n}$, the those time complexities are $O(n \log n)$ and $O(n^{1.5} \log n)$ respectively.

Accurate Tree Induction

The author used the optimal two-dimensional discriminant rule, region rule in short, to construct tree models. Diverse experiments show that trees with region rules are compact and accurate compared to conventional trees. Following two applications are examples of a decision tree and a regression tree. One is for estimating credit risk of companies. The other is for predicting market performances.

Credit Risk Analysis

Table 1.3 shows parts of the financial statements of some Japanese companies from 1992 to 1996. It contains 69 numeric attributes such as "`ID`" (ID of a company), "`Net Income / Sales`," and "`Equity Ratio`." It also contains the attribute "`Default`," which indicates whether the company defaulted within a year or not. Companies marked "`D`" defaulted, while companies marked "`N`" did not default within a year. In order to discover tests that discriminate between

Table 1.3: Financial Statements

ID	Net Income / Sales (%)	Equity Ratio (%)	...	Default
xxx	5.119	25.876	...	N
xxx	1.248	3.847	...	N
xxx	0.355	8.941	...	D
xxx	1.235	38.886	...	N
...	...	...	...	...
xxx	-0.096	4.111	...	D

"D" and "N," we collected 1036 samples of defaulted companies and another 1036 of non-defaulted companies.

Figure 1.8 shows the most important two-dimensional x-monotone region that was found from the table. Note that there are 69 attributes in the table and more than 2300 permutations (pairs). We examined each pair to find the optimal region, and chose the most important one based on the gini index value. The region on the plane whose x-axis is "`EBIT / Sales`" and whose y-axis is "`Equity Ratio`" divides the original data "S" into two subsets "S_1" (inside the region) and "S_2" (outside the region) as follows:

	No. of companies	No. of default	No. of non-default
S: All data	2072	1036	1036
S_1: Inside	969	797	172
S_2: Outside	1103	239	864

We divided the data recursively by using such two-dimensional tests, and constructed a decision tree to evaluate the credit risk of each company. Figure 1.9 shows the decision tree constructed from the example. In each leaf node (square node), the relative frequency of default companies is larger (or smaller) than that of non-default ones. Each company whose credit risk is unknown can be classified into one of the leaf nodes in the tree.

We want to estimate credit risk according to the frequency of default companies in each leaf node. In order to estimate credit risk of companies, we have to take account of macro conditions like market indices, exchange ratios, and so forth. However, such macro conditions change on a daily basis, while companies disclose their financial statements on a yearly basis. We first use all the daily macro conditions to compute average probability of default (for each industry). Then, we use the average probability to compute the probability for each leaf node in the constructed decision tree.

In Japanese market, the Nikkei average is one of the important macro conditions that affect credit risk of companies. To simplify the explanation, let us focus on only the Nikkei average, we found that there is strong linear correlation between the index and probability of default in all industries. Therefore, we can use a simple function for computing average default probability p as follows:

$$p = A * Nikkei + B$$

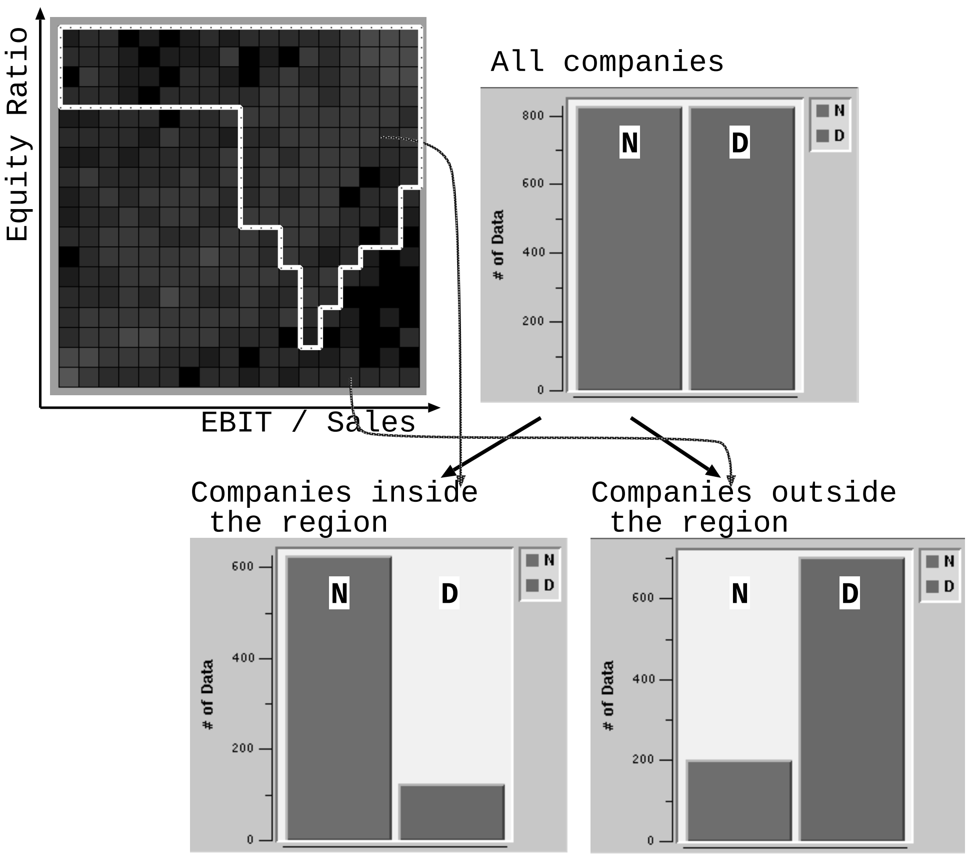


Figure 1.8: Tests for Analyzing Credit Risk

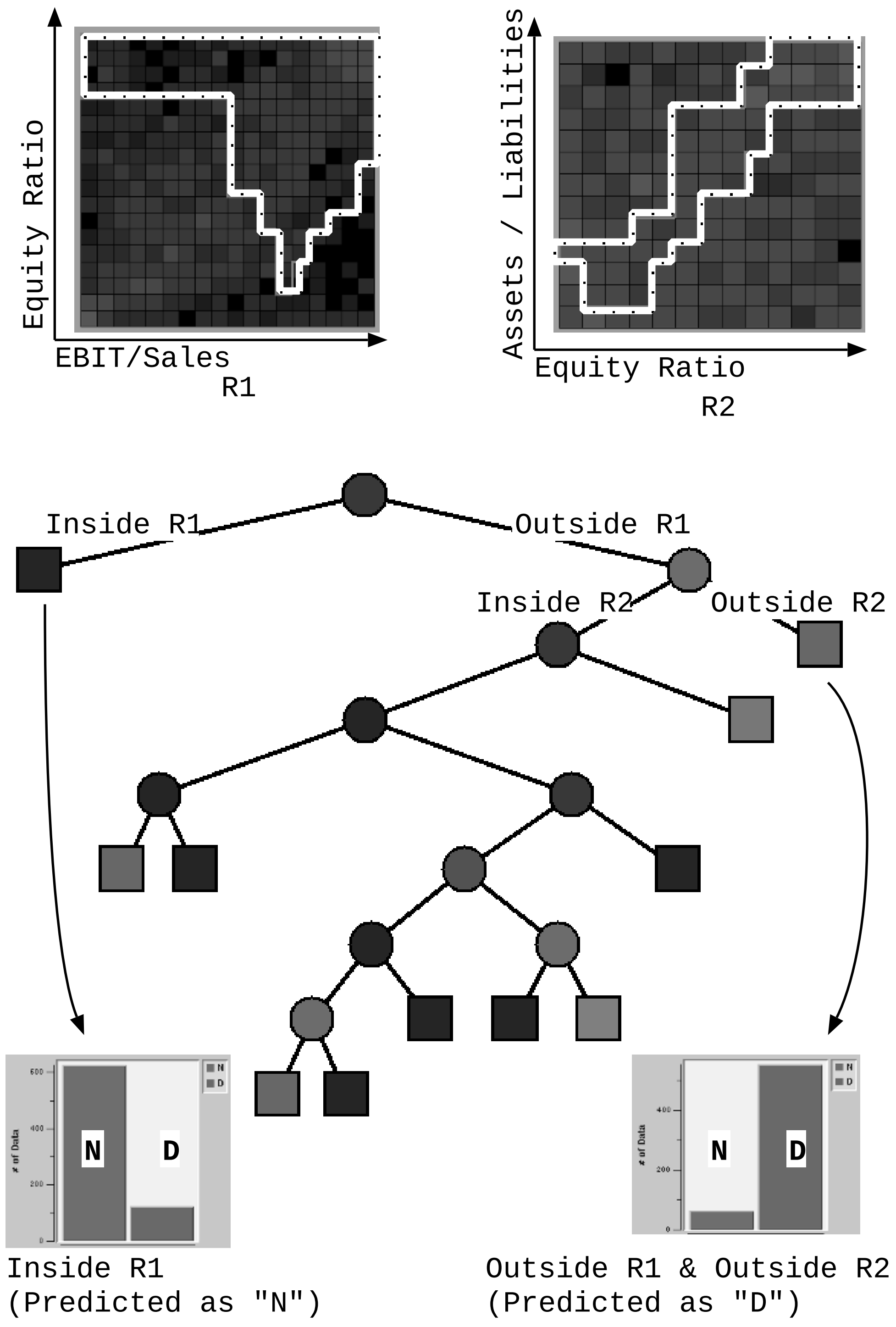


Figure 1.9: Decision Tree for Analyzing Credit Risk

Table 1.4: NY Market Database

Y	M	W	BPS	GDM	YEN	TB3M	TB30Y	GOLD	SP500
85	12	52	1.44353	.40746	.00498	7.02	9.31	326.00	210.88
86	1	1	1.44612	.40805	.00495	7.04	9.28	339.45	205.96
86	1	2	1.43794	.40485	.00494	7.13	9.37	357.25	208.43
86	1	3	1.40470	.40879	.00498	7.17	9.49	355.25	206.43
...	...	...	...	...	...	...	...	...	...
93	5	18	1.56875	.63384	.00907	2.91	6.89	357.50	442.31

where A and B are constants. We use p as prior probability to compute posterior probability after we have known which leaf node a company belongs.

Assume that we used D_{all} default samples and N_{all} non-default samples for training the decision tree. And assume a leaf node, say "l," of the decision tree contains D_l and N_l default and non-default samples respectively. Based on the Bayes theorem, probability of default for companies that belong to the node, p_l, can be computed as follows:

$$p_l = \frac{p\frac{D_l}{D_{all}}}{p\frac{D_l}{D_{all}} + (1-p)\frac{N_l}{N_{all}}}.$$

Market Analysis

Table 1.4 shows the relation collected from the New York markets every Monday from the last Monday of 1985 through the first Monday of May 1993. It contains 384 records and 10 numeric attributes: "W" (week), "M" (month), "Y" (year), "BPS" (British pound sterling, i.e., US$/pound), "GDM" (deutschmark, i.e., US$/mark), "YEN" (Japanese yen, i.e., US$/yen), "TB3M" (3-month Treasury bill yields), "TB30Y" (30-years Treasury bill yields), "GOLD" (US$/ounce), and "SP500" (Standard and Poors index). Suppose we are interested in the "SP500," which is one of the indices for measuring the overall market performance.

Figure 1.10 shows the most important two-dimensional x-monotone region that can be found from the table. The region on the plane whose x-axis is "GDM" and whose y-axis is "GOLD" divides the data "S" into "S_1" and "S_2" so that the variance of the "SP500" is minimized. The number of data, the mean, and the variance are as follows:

	No. of data	SP500 mean	SP500 variance
All Data S	384	324	4431
Inside Region S_1	157	391	1118
Outside Region S_2	227	278	1489

We divided the data recursively by using such two-dimensional tests, and constructed a regression tree to predict the "SP500." Figure 1.11 is the regression tree. The variance of the "SP500" value in each leaf node (square node) is much smaller than that of "S."

Note that tests found in the examples are non-linear correlations.

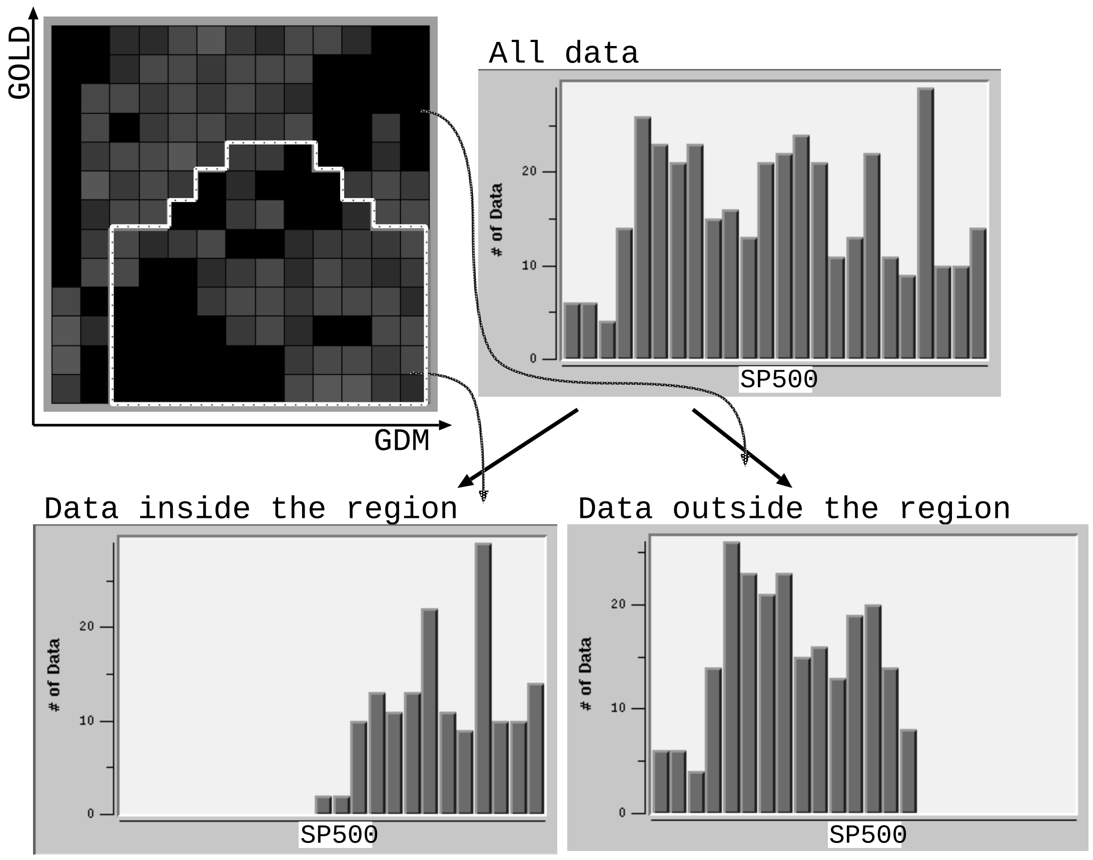


Figure 1.10: Tests for Analyzing the Market

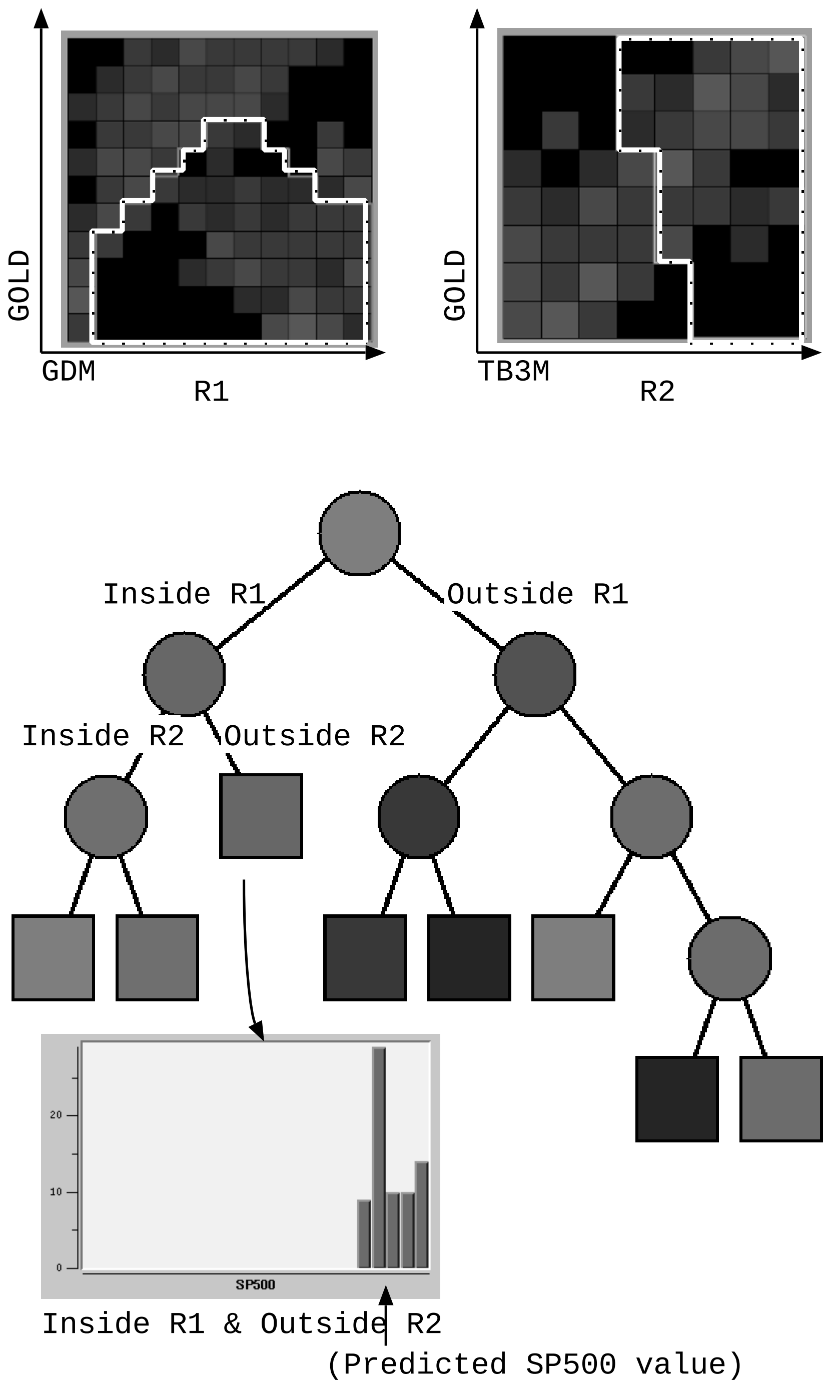


Figure 1.11: Regression Tree for Analyzing the Market

1.3 Organization of Dissertation

In Chapter 2, the author discussed association rules, which is the most fundamental functionality of data mining. For numerical attributes, conventional data mining techniques are not applicable. The author considered optimized numerical association rules and proposed efficient algorithms for finding such optimal rules, in particular, optimized numerical association rules (range rules) and optimized two dimensional association rules (region rules).

Next, the author explored classification problems in Chapter 3. How to find high quality discriminant rules is the key for modeling and predicting unknown value, which we are focusing on. The author proposed efficient algorithms for finding rules on categorical attributes by using techniques of computational geometry. For numerical attributes, the author modified the efficient algorithms for optimized numerical association rules (range rules) and optimized two dimensional association rules (region rules) to find discriminant rules.

Then, in Chapter 4, the author consider effective ways of utilizing discriminant rules in decision trees. Conventional decision trees that use univariate discriminant rules used to suffer from the correlation problem. The author proposed to use multivariate rules, such as region rules, for handling correlation in a database.

Finally, in Chapter 5, regression problems are discussed. The author utilized the algorithms of multivariate rules for regression problems and proposed regression trees that use such multivariate rules.

Chapter 2

Association Analysis

Data mining consists of technologies for finding hypothesis or knowledge in large datasets. Several data mining functions, which can efficiently handle huge databases, have been developed so far. Among them, functions for finding *association rules* that occur frequently together in a database have been intensively investigated and utilized.

2.1 Rules on Categorical Attributes

2.1.1 Frequent Itemsets

One of the most typical data mining application is *market basket analysis*, where a collection of items purchased by a customer, called market basket, in a purchases *transactions* is analyzed by retailers to identify items that frequently purchased together.

Assume a database of purchases transactions as shown in Table 2.1. Each records has customer ID (Customer), transaction ID (TID), name of purchased item (Item), and date and time of the transaction (Time). We call a set of items an *itemset*. For each itemset S, we can count the frequency of transactions that contain all items in the itemset, and let the number be $cnt(S)$. Let N be the number of all transactions in the transaction database. We call the number defined by $sup(S) = cnt(S)/N$ as *support* of S. In the example database, the support of the itemset {A}, $sup(\{A\})$, is 2/4. Similarly, $sup(\{A, B\}) = 1/4$.

If we look at the itemset $\{B, E\}$, its support is 3/4. Thus, we can consider that B and E are frequently purchased together. We usually interested in such frequent itemsets. We define itemsets whose support value is more than a user specified value, call it *minimum support*, as *frequent itemsets*.

Agrawal and Srikant proposed an efficient algorithm, called *a priori algorithm*, for enumerate all frequent itemsets in a large transaction database [AS94]. We begin by defining terminology to explain the a priori algorithm. Assume we have a transaction database, say $\mathcal{D}$, like the one in Table 2.1. Let $\mathcal{I}$ be a set of all items in $\mathcal{D}$. We assume, without loss of generality, we can order all items in $\mathcal{I}$. We call an itemset that consists of k items a k-itemset and express it I_k. Let $item_i(I)$ be the i-th item in the itemset I. Let C_k be a set of k-itemsets and let L_k be a set

Table 2.1: Purchases Transactions

Customer	TID	Item	Time
397	00001	A	99/02/23:10:00
397	00001	C	99/02/23:10:00
397	00001	D	99/02/23:10:00
147	00002	B	99/02/26:18:10
147	00002	C	99/02/26:18:10
147	00002	E	99/02/26:18:10
921	00003	A	99/02/27:18:30
921	00003	B	99/02/27:18:30
921	00003	C	99/02/27:18:30
921	00003	E	99/02/27:18:30
397	00004	B	99/03/02:14:20
397	00004	E	99/03/02:14:20

of frequent k-itemsets in which each member itemset I_k has support value greater than or equal to the minimum support *minsup*. Figure 2.1 is the a priori algorithm.

The problem for enumerating all frequent itemsets has inherent difficulty in terms of computational complexity, because if a target transaction database has k items there are 2^k possible itemsets that can be frequent. The frequent itemsets have a property that $sup(I_k) \geq sup(I_{k+1})$ if I_k is a k-subset of I_{k+1}. Therefore, if any k-subset of an itemset I_{k+1} has less support value than the minimum support, the itemset I_{k+1} must not be a frequent itemset. Figure 2.2 shows an itemset lattice to be examined for frequent itemsets. The pruning phase of the algorithm use this property to prune away candidate itemsets to be examined as shown in the figure. Consequently, it can work even for large databases within a limited working space in practice.

The a priori algorithm and many variants of this algorithm have been widely used in this literature. Park et al. utilized a hashing scheme to improve efficiency of the algorithm [PCY95] and Brin et al. pipelined the recursions, called dynamic itemset counting, of the algorithm [BMUT97]. Morishita and Sese focused on the statistical significance, in stead of frequence, of itemsets and proposed an algorithm for enumerating significant itemsets [MS00].

While most of a priori style algorithms search the itemset lattice, like Figure 2.2, in breadth first manner, some algorithms search the lattice in depth first manner and find local maximal (deepest) itemsets. Every itemsets that are descended from the local maximal itemsets must be frequent and we do not have to examine. For example, if we find a maximal itemset $\{A, B, C\}$, all of its descendants, $\{A, B\}$ $\{A, C\}$, $\{B, C\}$, $\{A\}$, $\{B\}$, and $\{C\}$ must be frequent itemsets. MaxMiner [Bay98], DepthProject [AAP98], MaxEclat/MaxClique [ZPOL97] are such algorithms.

While all of the mentioned algorithms traverses the itemset lattice, Han et al. [HPY00] proposed a method that uses a prefix tree, they call it a frequent pattern (FP) tree, like the one often used in pattern matching literature.

```
Algorithm Apriori($\mathcal{D}$, $minsup$) {
    $C_1 := \{\{I_1\} \mid I_1 \in \mathcal{I}\}$;
    $k := 1$;
    while ($C_k \neq \emptyset$) {
        scan all transactions $t \in \mathcal{D}$ to compute $cnt(I_k)$ for all $I_k \in C_k$
        $L_k := \{I_k \in C_k \mid cnt(I_k)/N \geq minsup\}$;
        $C_{k+1}$ := Apriori-Gen($L_k$);
        $k := k + 1$;
    }
    return $\bigcup_k L_k$;
}
Function Apriori-Gen($L_k$) {
    /* Generation Phase */
    $C_{k+1} := \emptyset$;
    foreach $p, q \in L_k$ such that
        $item_i(p) = item_i(q)$ for $i = 1 \ldots k-1$ & $item_k(p) < item_k(q)$
    {
            $I := p \cup item_k(q)$;
            $C_{k+1} := C_{k+1} \cup \{I\}$;
    }
    /* Pruning Phase */
    foreach $I_{k+1} \in C_{k+1}$ {
        foreach $k$-subsets $I_k$ of the $I_{k+1}$ {
            if ($I_k \notin L_k$)
                remove $I_{k+1}$ from $C_{k+1}$;
        }
    }
    return $C_{k+1}$;
}
```

Figure 2.1: A Priori Algorithm for Frequent Itemsets

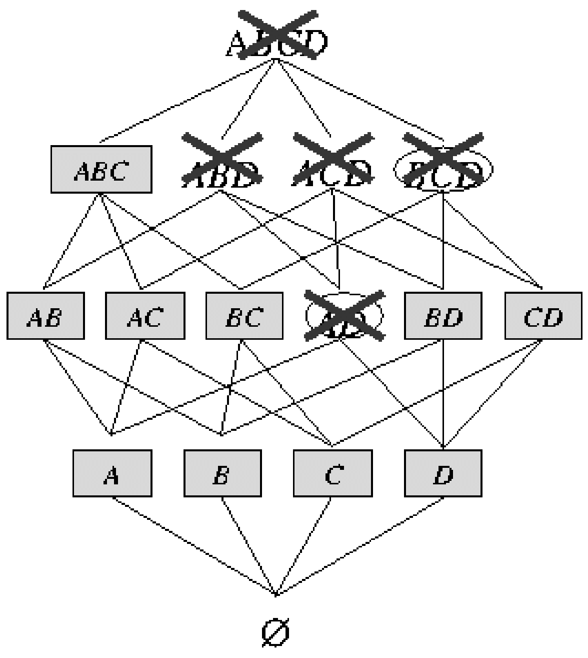


Figure 2.2: Itemset Lattice

2.1.2 Association Rules

If we can find a rule "a customer who purchases A and B tend to purchase C," the retailer can consider strategic sales promotions for the goods. For example, they can arrange the layout of goods in a store or in a catalog page so that A, B, and C are close to each other.

Such rules are called *association rules*. An association rule has the form:

$$LHS \Rightarrow RHS, (support, confidence)$$

where both LHS, left hand set, and RHS, right hand set, are sets of items. The form implies that a customer who purchases all items in LHS tends to purchase all items in RHS. Each association rule has two measures *support* and *confidence*. The support is the probability of transactions that contains all of items in the union of LHS and RHS. The confidence is the probability of transactions containing all of items in the union of LHS and RHS over transactions containing all items in LHS.

Definition 2.1 The *support* of the rule $LHS \Rightarrow RHS$ is $cnt(LHS \cup RHS)/N$, and the *confidence* is $cnt(LHS \cup RHS)/cnt(LHS)$. **[EOD]**

The support value is an indication of the applicability of the rule, while the confidence value is an indication of the strength.

Association rules that can be found from the transaction database in Table 2.1 are as follows:

$$\{B\} \Rightarrow \{E\}, (support = 3/4, confidence = 3/3)$$
$$\{A\} \Rightarrow \{C\}, (support = 2/4, confidence = 2/2)$$

$\{C\} \Rightarrow \{A\}, (support = 2/4, confidence = 2/3)$
$\{B, E\} \Rightarrow \{C\}, (support = 2/4, confidence = 2/3)$

Among all possible association rules, we are interested in rules such that both support and confidence are large enough for users. Agrawal, Imielinski, and Swami [AIS93b] proposed a method for enumerating all association rules that have larger support value than a user specified minimal support and confidence value. Agrawal and Srikant improved time and space efficiency by using the a priori algorithm [AS94].

Assume a frequent itemset S and its subset $s \in S$. The support and confidence of the rule $s \Rightarrow (S - s)$ are $sup(S)$ and $sup(S)/sup(s)$ respectively. Let $conf(s \Rightarrow (S - s))$ be confidence of the rule. Figure 2.3 is an algorithm for finding all association rules whose support and confidence are larger than user specified values *minsup* and *minconf*, respectively. For each frequent itemset the GENERATE-RULE examine rules whose RHS contains only one item and the subroutine AP-GENRULE examines other rules that can be generated from the itemset.

This algorithm uses a property that is derived from the a priori property of itemsets, $sup(s) \geq sup(S)$ if $s \in S$. Suppose we focus on an itemset S. Assume two subsets, say $s1$ and $s2$, of S such that $s1 \in s2 \in S$. Assume that we are examining all rules that can be generated from S. If the rule whose RHS is $s1$ $(S - s1 \Rightarrow s1)$ does not satisfy the minimum confidence criterion, i.e., $minconf > sup(S)/sup(S - s1)$, we do not have to examine rules that contains $s1$ in their RHS like $S - s2 \Rightarrow s2$ because the confidence must be smaller than $minconf$[*]. Notice that all rules generated from frequent itemsets satisfy the minimum support criterion.

Since the algorithms of [AIS93b, AS94] have proposed, many data mining applications developed based on their algorithms and utilized in market basket analysis. Several parallel implementations have been proposed to handle huge transaction databases [AS96, SK96]. Sawaragi et al. considered an integration into relational database systems [STA98].

Some researchers pointed out a problem that data mining algorithms often find too many rules to analyze each rule. Brin et al. and Morishita et al. considered improvements to eliminate rules whose statistical significance is low [BMS97, MS00]. Bayardo and Agrawal introduced several alternative measures in addition to support and confidence [BA99]. Liu et al. considered further implications of rules to filter unnecessary rules [LHM99].

2.2 Rules on Numerical Attributes

In general, there are two types of attributes: *categorical* and *numerical* attributes in a database. Conventional algorithms for finding association rules assume that attributes, which are investigated, are all categorical. However, in many systems we have to analyze data of numerical attributes especially for financial applications. Association analysis for numerical attributes is necessary for such databases containing numerical attributes. Therefore, the author considered

[*]Notice that $minconf > sup(S)/sup(S - s1) > sup(S)/sup(S - s2)$ because $sup(s1) > sup(s2)$ and $sup(S - s1) < sup(S - s2)$.

```
0)  Algorithm Generate-Rules() {
1)      foreach frequent itemsets $l_k$ (with $k \geq 2$ items) $\in L_k$ {
2)          $RHS_1 := \emptyset$;
3)          foreach item $r_1 \in l_k$ {
4)              $conf := sup(l_k)/sup(l_k - r_1)$;
5)              if ($conf \geq minconf$) {
6)                  output $(l_k - r_1) \Rightarrow r_1$;
7)                  $RHS_1 := RHS_1 \cup \{r_1\}$;
8)              }
9)          }
10)         call Ap-GenRule($l_k, RHS_1$);
11)     }
12) }
13) Procedure Ap-GenRule($l_k$, $RHS_m$) {
14)     if ($k > m + 1$) {
15)         $RHS_{m+1} :=$ Apriori-Gen($RHS_m$);
16)         foreach $r_{m+1} \in RHS_{m+1}$ {
17)             $conf := sup(l_k)/sup(l_k - r_{m+1})$;
18)             if ($conf \geq minconf$)
19)                 output $(l_k - r_{m+1}) \Rightarrow r_{m+1}$;
20)             else
21)                 $RHS_{m+1} := RHS_{m+1} - \{r_{m+1}\}$;
22)         }
23)         Ap-GenRules($l_k$, $RHS_{m+1}$);
24)     }
25) }
```

Figure 2.3: Algorithm Generate-Rule

association rules for numerical attributes.

General form of the association rules is:

$$P_1(C_1) \wedge P_2(C_2) \cdots \wedge P_i(C_i) \Rightarrow P_{i+1}(C_{i+1}) \wedge P_{i+2}(C_{i+2}) \cdots \wedge P_k(C_k)$$

where C_i $(1 \leq i \leq k)$ are attributes that are analyzed. The $P_i(C_i)$ $(1 \leq i \leq k)$ are predicates that involve attribute C_i.

For categorical attributes, the form of the predicates would be $t[C_i] = v_i$ where $t[C_i]$ is the value of the attribute C_i of a certain record and v_i is a value in the domain of C_i.

Conventional algorithms for finding association rules work efficiently if predicates has the form $t[C_i] = v_i$. If C_i is a numerical attributes, the number of distinct values becomes large and hence support value becomes much smaller. Such rules whose support is small are not significant. Therefore, for numerical attributes, the form of the predicate would be $t[C_i] \in [v_{i-lo}, v_{i-hi}]$ where v_{i-lo} and v_{i-hi} $(v_{i-lo} < v_{i-hi})$ are threshold values in the domain of C_i. In the predicate, v_{i-lo} can be $-\infty$ or v_{i-hi} can be ∞.

2.2.1 Numerical Association Rules

Association rules on a numerical attributes are often called "numerical association rules" or "quantitative association rules." Srikant and Agrawal proposed an algorithm for finding rules [SA96] of the following form:

$$\{t[C_1] \in [v_{1-lo}, v_{1-hi}]\} \wedge \cdots \wedge \{t[C_i] \in [v_{i-lo}, v_{i-hi}]\} \Rightarrow t[C_k] = v_k, (support, confidence)$$

The rule implies that if the value of C_j of a record r falls in the range $[v_{j-lo}, v_{j-hi}]$ for $j = 1, ..., i$, the record's value of C_k is likely to be v_k. The support of the rule is the probability of records that satisfy both $\{t[C_1] \in [v_{1-lo}, v_{1-hi}]\} \wedge \cdots \wedge \{t[C_i] \in [v_{i-lo}, v_{i-hi}]\}$ and $t[C_k] = v_k$. The confidence is the probability of records that satisfy $t[C_k] = v_k$ among the records satisfying $\{t[C_1] \in [v_{1-lo}, v_{1-hi}]\} \wedge \cdots \wedge \{t[C_i] \in [v_{i-lo}, v_{i-hi}]\}$.

Like association rules for categorical attributes, we are interested in rules whose support and confidence value are large. We have a freedom to choose the value range $[v_{i-lo}, v_{i-hi}]$ of a numerical attribute C_i when we search for rules. In general, if we choose a wider range on the numerical attribute, we will have larger support value, however, the corresponding rules tend to converge on the average confidence value, it means the rules are less significant. On the other hand, if we choose a narrower range, we can find rules whose confidence value are significantly high or low though the support values are low.

2.2.2 Optimized Numerical Association Rules

The *optimized numerical association rules* of the form

$$\{t[C] \in [v_{lo}, v_{hi}]\} \Rightarrow t[A] = a, (support, confidence).$$

are considered for numerical attributes[FMMT96d, FMMT99].

Definition 2.2 A rule is *confident* if its confidence is not less than the given minimum confidence threshold. Among confident rules, the *optimized support rule* maximizes $sup(v_{lo} \leq t[C] \leq v_{hi})$. A rule is *ample* if $\sup(v_{lo} \leq t[C] \leq v_{hi})$ is not less than the given minimum support threshold. Among ample rules the *optimized confidence rule* maximizes (or minimizes) the confidence. **[EOD]**

Ordered Equi-Depth Buckets

We assume all distinct values on a numerical attribute C are sorted. This preprocess takes $O(n \log n)$ time where n is the number of records in a database. If n is unacceptably large, we make M *buckets* on the domain of C.

Definition 2.3 *Buckets* of the domain C are a sequence of disjoint ranges

$$B_1, B_2, \ldots, B_M$$
$$(B_i = [x_i, y_i] \text{ and } x_i \leq y_i < x_{i+1})$$

such that the value of C for all records are covered by the buckets; namely, for an arbitrary record $t \in R$, there exists a bucket B_j that contains $t[C]$. We say that a bucket B_i is *finest* if $B_i = [x, x]$ for a value x. **[EOD]**

Linking consecutive buckets $B_s, B_{s+1}, \ldots, B_t$ creates a range $[x_s, y_t]$. Observe that if all buckets are finest, the combination of consecutive finest buckets gives the range of an optimized association rule. Given a large number of buckets that may not be finest, an approximation of the range of an optimized rule can be obtained by joining consecutive buckets.

Definition 2.4 We call the number of records in $\{t \in R \mid t[C] \in B_i\}$ the *size* of B_i, and denote it by u_i. We assume that each bucket B_i contains at least one record; that is, $u_i \geq 1$. B_i's are called *equi-depth* if the size of any B_i is the same or almost same. Let v_i denote the number of records in $\{t \in R \mid t[C] \in B_i, t[A] = a\}$, and let n be the number of all records. $(\sum_{i=s}^{t} v_i)/(\sum_{i=s}^{t} u_i)$ gives the confidence of rule $(C \in [x_s, y_t]) \Rightarrow t[A] = a$, and the support of $A \in [x_s, y_t]$ is $(\sum_{i=s}^{t} u_i)/n$. **[EOD]**

Though there are many ways of making M buckets, we make *equi-depth buckets*, which divide the n data into almost evenly, because we empirically learned that the equi-depth buckets have suitable properties, for instance robustness against noise [FMMT99], for data mining applications.

Since we had better avoid sorting data, the bucketting algorithm in Figure 2.4 is one of the good way for making equi-depth buckets. In the algorithm, D is the database and C is a conditional numerical attribute in D. The s is the number of records for a random sample and M is the number of buckets. The MAKEBUCKETS function takes those parameters and makes M equi-depth buckets for attribute C in the database D.

```
0)  Algorithm MakeBuckets($D$, $C$, $s$, $M$) {
1)      Choose a random sample $S$ having $s$ records
        from the database $D$ having $n$ records.
2)      Sort $S$ in ascending order of $t[C]$. (It takes $O(s \log s)$ time.)
3)      $v_0 := -\infty$, $v_M := \infty$.
4)      For each $i = 1, \cdots, M-1$ {
5)          Set the $i(s/M)$-th data in $S$ to $v_i$.
6)      }
7)      For each record $t \in D$ {
8)          Find $i$ such that $v_{i-1} < t[C] \leq v_i$.
9)          Assign $t$ to the $i$-th bucket.
10)     }
```

Figure 2.4: Randomized Algorithm for Equi-Depth Buckets

In the algorithm in Figure 2.4, the step 8 can be done in $O(logM)$ time by using a binary search tree for the buckets. In the ordinal inputs, s is much smaller than n and the time for the step 2 is not dominant factor. Therefore, for all i, $(x_1(\{B_i\}), x_2(\{B_i\})[, \cdots])$, the coordinate values of B_i can be computed in $O(n \log M)$ time.

In the bucketting algorithm, we use an s-sized random sample and the size affects the quality of buckets. We intensively investigated the quality of buckets with respect to s and found that $40 \cdot M$ is enough for s [FMMT99].

Given a sequence of buckets $B_1, B_2, \cdots, B_M$, such that $B_i = [x_i, y_i]$ and $x_i \leq y_i < x_{i+1}$, we focus on rules of the form

$$(C \in [x_s, y_t]) \Rightarrow t[A] = a,$$

where $[x_s, y_t]$ is a combination of consecutive buckets $B_s, B_{s+1}, \cdots, B_t$.

Definition 2.5 Since any range $[x_s, y_t]$ is specified by a pair of indexes $s \leq t$, for simplicity, we denote $sup(A \in [x_s, y_t])$ by $sup(s,t)$ and denote $conf((A \in [x_s, y_t]) \Rightarrow t[A] = a)$ by $conf(s,t)$ throughout this section. **[EOD]**

Then, among ample rules such that $sup(s,t)$ is not less than a given threshold, the optimized confidence rule maximizes (or minimizes) $conf(s,t)$. On the other hand, among confident rules such that $conf(s,t)$ is not less than a given threshold, the optimized support rule maximizes $sup(s,t)$.

2.2.3 Optimized Confidence Rules

Definition 2.6 Let $B_1, \cdots, B_M$ be buckets. Let N denote the number of all records. Let u_i denote the number of records in $\{t \in R \mid t[C] \in B_i\}$. We assume that $u_i \geq 1$. Let v_i denote a real number associated with B_i. Consider the sequence of points $Q_k = (\sum_{i=1}^{k} u_i, \sum_{i=1}^{k} v_i)$ for

$k = 1, \cdots, M$, and let Q_0 be $(0, 0)$. Let m and n be non-negative integers such that $m < n$. Observe that the x-coordinate of Q_n minus the x-coordinate of Q_m is equal to $N \times sup(m+1, n)$.

We call $s \leq t$ an *ample* pair if $sup(s,t)$, which is $\sum_{i=s}^{t} u_i/N$, is no less than the given minimum support threshold. We call m and n an *optimal slope pair*, if $m+1$ and n are an ample pair that maximizes the slope of $Q_m Q_n$. If more than one pair has the same maximum slope, select a pair that maximizes $sup(m+1, n)$. **[EOD]**

In the special case when v_i is the number of records in $\{t \in R | t[C] \in B_i, t[A] = a\}$, the slope of the line $Q_m Q_n$ gives $conf(m+1, n)$. Thus, if m and n are the optimal slope pair, $(C \in [x_{m+1}, y_n]) \Rightarrow t[A] = a$ is the optimized confidence rule. We will therefore present an algorithm for computing the optimal slope pair.

To compute the optimal slope pair, we use a technique of handling convex hulls, for which we introduce some special terms.

Definition 2.7 Let S be a set of distinct points. A *convex polygon* of S has the property that any line connecting any two points of S must itself lie entirely inside the polygon. The *convex hull* of S is the smallest convex polygon of S. Let v_{min} be the node in S with the minimum x-coordinate, and let v_{max} be the node in S with the maximum x-coordinate. Observe that v_{min} and v_{max} are on the convex hull of S. From v_{min} we can visit nodes on the convex hull of S in clockwise (counterclockwise) order until we hit v_{max}, and we call the set of nodes visited the *upper* (*lower*) hull of S.

Let U_m denote the upper hull of $\{Q_m, \cdots, Q_M\}$, and let $r(m)$ be

$$min\{i \mid m+1 \leq i \text{ is an ample pair}\}.$$

Now consider the tangent of Q_m and $U_{r(m)}$, and suppose that the tangent touches $U_{r(m)}$ at Q_t, as illustrated in Figure 2.5. Q_t is called the *terminating* point of the tangent (if the tangent touches more than one node of $U_{r(m)}$, select the node with the maximum x-coordinate as Q_t). **[EOD]**

It is easy to see that if $m \leq n$ is the optimal slope pair, Q_n is the terminating point of the tangent of Q_m and $U_{r(m)}$. Thus, we need to find the tangent of Q_m and $U_{r(m)}$ with the maximum slope among all m.

Online Maintenance of Convex Hulls

Online stack maintenance algorithm in Figure 2.6 constructs a data structure that represents *the convex hull tree* of $Q_0, \cdots, Q_M$, such as illustrated in Figure 2.7. While various implementations are possible [PS85], we use stacks S and D_i ($i = 0, \cdots, M$) in the data structure. We use S to store the sequence of nodes of the convex hull that we are focusing on, and we use D_i to store a branch of the convex hull tree; namely, the nodes that belong to U_{i+1}, but do not belong to U_i.

The algorithm consists of Preparatory Phase and Restoration Phase. Given a sequence of nodes $Q_0, \cdots, Q_M$ that is the sorted list with respect to the x-coordinate value, Preparatory

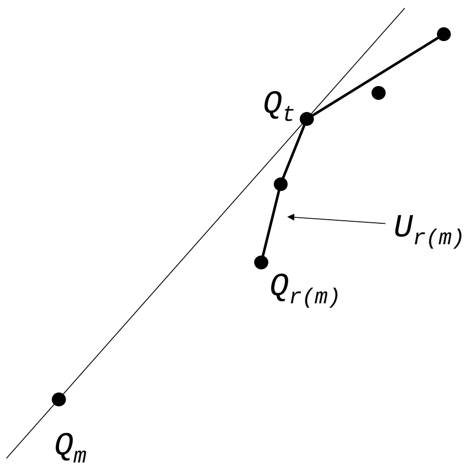


Figure 2.5: The Inner Tangent of Q_m and $U_{r(m)}$

Phase sets each branch of the convex hull tree to D_i, for $i = M-1, \cdots, 0$. After the execution of each step of the Preparatory Phase, the top-to-bottom order of nodes in S corresponds to the clockwise order of nodes on U_i (the upper hull of $\{Q_i, \cdots, Q_M\}$), which enables us to access the neighbors of each node Q on U_i by looking at the next node and the previous node of Q in S.

Restoration Phase makes $U_{r(m)}$ on S, for each $m = 0, \cdots, M-1$, using D_is.

Since throughout the execution, at most M nodes are popped from S, and at most M nodes are pushed back from D_is to S, the overall computation time is $O(M)$.

Example 2.1 Consider nodes $Q_0, Q_1, \cdots, Q_9$ in Figure 2.7. The dotted line from Q_i to Q_9 shows the upper hull of $\{Q_i, \cdots, Q_9\}$. Let us apply the preparatory phase of the algorithm in Figure 2.6 to $\{Q_0, \cdots, Q_9\}$. Each column of the upper table in Figure 2.8 illustrates the content of S for each $i = 9, \cdots, 0$. Observe that each column contains the upper convex hull of $\{Q_i, \cdots, Q_9\}$. Each column of the lower table in Figure 2.8 shows D_i for $i = 9, \cdots, 0$. We can also see how the restoration phase works by observing the columns from $i = 0$ to 9. **[EOE]**

Computing Tangents

Next, we search the tangent of Q_m and $U_{r(m)}$ with the maximum slope among all m. The algorithm in Figure 2.9 finds the maximum slope efficiently.

Figure 2.10 illustrates an example of the case of the step 8). We do not compute the tangent of Q_m and $U_{r(m)}$, and leave L untouched because the slope of the tangent of Q_m and $U_{r(m)}$ is not greater than that of L.

Figure 2.11 illustrates the step 11) to 14). Among all nodes on both $U_{r(k)}$ and $U_{r(m)}$, let X denote the node with the minimum x-coordinate. The clockwise search of the step 12) only scans edges from $Q_{r(m)}$ to at most X; otherwise, $U_{r(k)}$ cannot be convex, since the terminating point is above Q_mX, and at least one node on the left-hand side of X on $U_{r(k)}$ is also above

```
Algorithm UpperHull($Q_0, Q_1, \cdots, Q_M$) {
    Let $S$ and $D_i$ ($i = 1, \cdots, M$) be empty.
    **Preparatory Phase:**
    For each $i = M, \cdots, 0$ {
        If $i = M$ {
            Push $Q_M$ onto $S$, which trivially makes $S$ store $U_M$.
        }
        Else{
            **Clockwise Search:**
            If the slope of $Q_i$ and the top node of $S$ is less than or equal to
            the slope of $Q_i$ and the node that is the second from the top of $S$,{
                Pop the top node from $S$. (the top node is no longer a node on $U_i$)
                Push it onto $D_i$. ($D_i$s are used for recording the nodes deleted at each step.)
                Repeat the Clockwise Search.
            }
            Else{
                Push $Q_i$ onto $S$. (the slope of $Q_i$ and the top node is maximum)
            }
        }
    }
    **Restoration Phase:**
    Set 1 to $i$. (We use $i$ to search for $r(m)$.)
    For each $m = 0, \cdots, M - 1$ {
        While $(m + 1, i)$ is not an ample pair,
        pop the top node $Q_i$ from $S$,
        push back all nodes of $D_i$ in top-to-bottom order onto $S$,
        which makes $S$ store $U_{i+1}$.
        Increment $i$.
        If $i > M$, stop the restoration phase.
    }
}
```

Figure 2.6: Algorithm for Computing Upper Hull

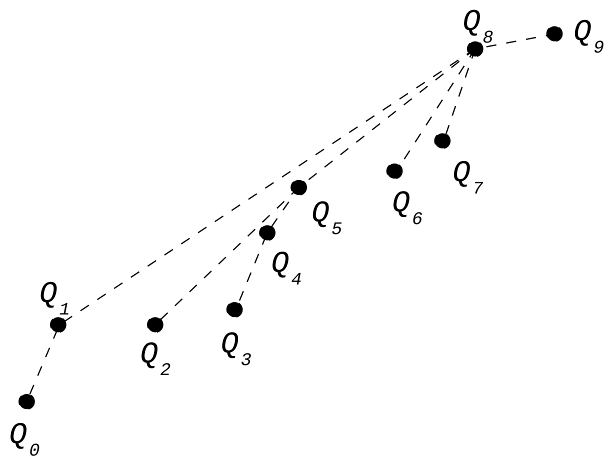


Figure 2.7: Upper Hulls

							Q_3			
						Q_4	Q_4	Q_2		Q_0
			Q_7	Q_6	Q_5	Q_5	Q_5	Q_5	Q_1	Q_1
		Q_8	Q_8	Q_8	Q_8	Q_8	Q_8	Q_8	Q_8	Q_8
	Q_9	Q_9	Q_9	Q_9	Q_9	Q_9	Q_9	Q_9	Q_9	Q_9
$i =$	9	8	7	6	5	4	3	2	1	0

							Q_4	Q_5	
			Q_7	Q_6			Q_3	Q_2	
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Figure 2.8: Computing Upper Hulls

```
0)  Algorithm MaxSlopeSearch($Q_0, Q_1, \cdots, Q_M$){
1)      Run UpperHull($Q_0, Q_1, \cdots, Q_M$) to make $U_{r(m)}$ for each $m = 0, \cdots, M$.
2)      **Base Step:**
3)      Find the terminating point of the tangent of $Q_0$ and $U_{r(0)}$ by clockwise search;
        that is, visit each node $Q$ on $U_{r(0)}$ from $Q_{r(0)}$ in clockwise order.
4)      Set the tangent $Q_0 Q$ with the maximum slope to $L$.
5)      **Inductive Step:**
6)      For each $m = 1, \cdots, M$ {
7)          While $U_{r(m)}$ is not empty {
                (Assume that $L$ stores the tangent of $Q_k$ and $U_{r(k)}$ for some $k < m$.)
8)              If $Q_m$ is above or on $L$, leave $L$ untouched.
9)              If $Q_m$ is below $L$ {
10)                 Let $Q_t$ be the terminating point of $L$.
11)                 If $L$ does not touch $U_{r(m)}$ ($Q_t$ is on the left-hand side of $Q_{r(m)}$){
12)                     find the terminating point of $Q_m$ and $U_{r(m)}$ by clockwise search;
                        that is, visit each node $Q$ on $U_{r(m)}$ from $Q_{r(m)}$ in clockwise order.
13)                     Find the $Q_m Q$ with the maximum slope.
14)                 }
15)                 Else{ ($L$ touches $U_{r(m)}$ at $Q_t$.)
16)                     Find the terminating point of $Q_m$ and $U_{r(m)}$ by counter-clockwise search;
                        that is, visit each node $Q$ on $U_{r(m)}$ from $Q_t$ in counter-clockwise order.
17)                     Find the $Q_m Q$ with the maximum slope.
18)                 }
19)                 Set the tangent of $Q_m$ and $U_{r(m)}$ to $L$.
20)             }
21)         }
22)     }
23)     **Final Step:**
24)     Among all tangents that have been set to $L$,
        select the ones with the maximum slope.
25) }
```

Figure 2.9: Algorithm for Finding Maximum Slope

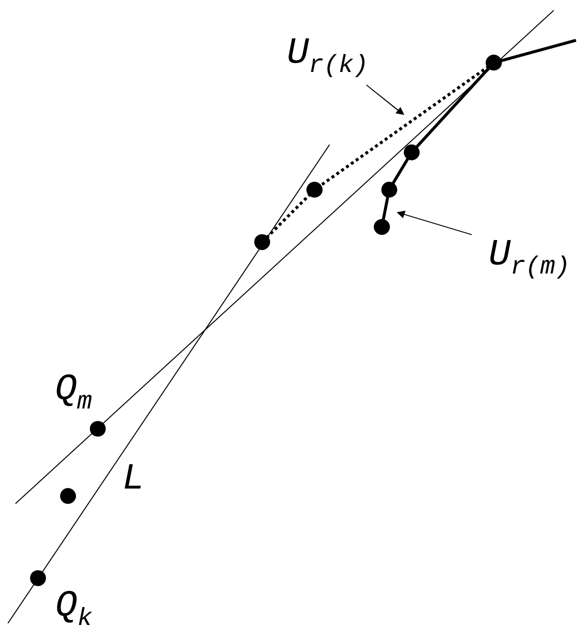


Figure 2.10: Leaving L Untouched

or on Q_mX. Since edges between $Q_{r(m)}$ and X are hidden inside $U_{r(k)}$, those edges have never been scanned in this algorithm.

Figure 2.12 illustrates the case of the step 15) to 18). Note that edges between $Q_{r(m)}$ and Q_t have never been scanned before in this algorithm.

Theorem 2.1 *Algorithm in Figure 2.9 computes all optimal slope pairs in $O(M)$ time.* **[EOT]**

Proof: Let S denote the set of edges on $U_{r(m)}$ for all m. Both the clockwise search and the counter-clockwise search in the algorithm scan each edge in S at most once. Since the number of edges in S is at most $M - 1$, the algorithm computes tangents with the maximum slope in $O(M)$ time. **[EOP]**

2.2.4 Optimized Support Rules

Definition 2.8 Let $B_1, \cdots, B_M$ be buckets such that $B_i = [x_i, y_i]$ and $x_i \leq y_i < x_{i+1}$. Let N denote the number of all records. Let u_i denote the number of records in $\{t \in R \mid t[C] \in B_i\}$. Let v_i denote a real number associated with B_i. Let s and t be non-negative integers such that $s \leq t$. Let $avg(s,t)$ denote $(\sum_{i=s}^{t} v_i)/(\sum_{i=s}^{t} u_i)$. Let θ be a minimum threshold for $avg(s,t)$. We call (s,t) an *optimal support pair* if $avg(s,t) \geq \theta$, and (s,t) maximizes $sup(s,t)$ $(= (\sum_{i=s}^{t} u_i)/N)$. **[EOD]**

In the special case when v_i is the number of records in $\{t \in R | t[C] \in B_i, t[A] = a\}$, $avg(s,t)$ is equal to $conf(s,t)$. Suppose that θ is the minimum confidence threshold. If (s,t) is an optimal support pair, $(C \in [x_s, y_t]) \Rightarrow t[A] = a$ is an optimized support rule.

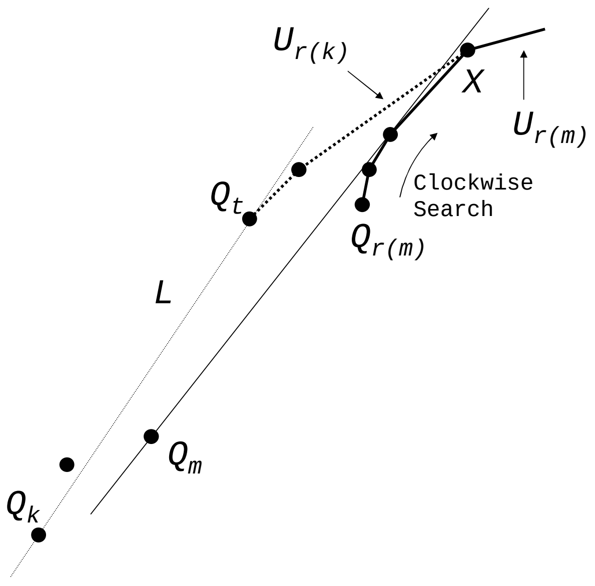


Figure 2.11: Clockwise Search

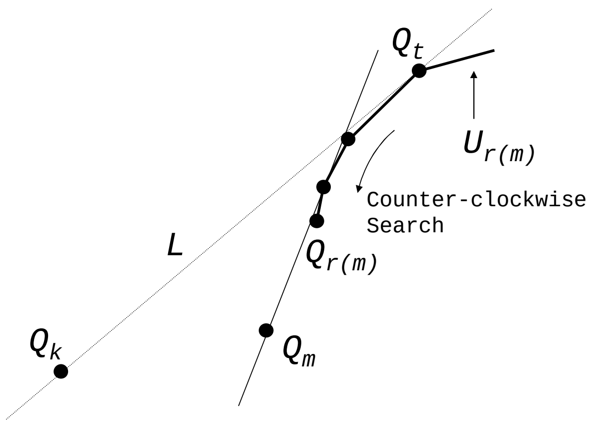


Figure 2.12: Counter-clockwise Search

```
0) Algorithm EnumEffective(){
1)       1 is effective
2)       $w := 0$
3)       For each $s := 2$ to $M${
4)             $w := v_{s-1} - \theta u_{s-1} + \max\{0, w\}$
5)             If ($w < 0$) then $s$ is effective
6)       }
7) }
```

Figure 2.13: Algorithm for Enumerating Effective Ranges

Definition 2.9 Let us call s is *effective* if $avg(j, s-1) < \theta$ for every $j < s$. **[EOD]**

Lemma 2.1 *If $s \leq t$ is an optimal support pair, s is effective.* **[EOL]**

Proof: Otherwise, there exists j such that $avg(j, s-1) \geq \theta$. Since $s \leq t$ is an optimal support pair, $avg(s,t) \geq \theta$, and hence $avg(j,t) \geq \theta$, which contradicts the optimality of s and t. **[EOP]**

From the above lemma, we will find all effective indices and choose an optimal support pair. Let w be $max_{j<s} \sum_{i=j}^{s-1}(v_i - \theta u_i)$ for each index s. Then, note that s is effective iff $w < 0$. The algorithm in Figure 2.13 computes w for all indices in $O(M)$ by scanning buckets forwards, and gives the set of all effective indices.

Let $top(s)$ denote the largest index t, such that $s \leq t$ and $avg(s,t) \geq \theta$. The final step is to choose a value of s that maximizes $\sum_{i=s}^{top(s)} u_i$.

Lemma 2.2 *If $s < s'$ are effective, $top(s) \leq top(s')$.* **[EOL]**

Proof: Since s' is effective, $avg(s, s'-1) < \theta$. From the definition of $top(s)$, $avg(s, top(s)) \geq \theta$. Then, it follows that $avg(s', top(s)) \geq \theta$, which implies that $top(s) \leq top(s')$. **[EOP]**

Thanks to this property, we only need to scan backwards through the list of effective indices $(s(1), \cdots, s(q))$ and the list of all indices $(1, \cdots, M)$ alternately to find $top(s(i))$. We can do this by means of the algorithm in Figure 2.14.

In the algorithm in Figure 2.14, we pre-compute a cumulative table $F(j) = \sum_{i=1}^{j} v_i - \theta \sum_{i=1}^{j} u_i$. Since

$$avg(s(j), i) < \theta \text{ iff } F(i) - F(s(j)-1) < 0,$$

we can check $avg(s(j), i) < \theta$ in a constant time ($F(0)$ is defined as 0). Thus both algorithms in Figure 2.13 and 2.14 run in $O(M)$ time.

Theorem 2.2 *All optimal support pairs can be computed in $O(M)$ time.* **[EOT]**

```
0) Algorithm Top(){
1)     $i := M$
2)     For each $j$ from $q$ to 1 {
3)         While $(avg(s(j), i) < \theta)${
4)             $i := i - 1$
5)         }
6)         $top(s(j)) := i$
7)     }
8) }
```

Figure 2.14: Algorithm for Finding Top Index

2.2.5 Two Dimensional Numerical Association Rules

It would also be valuable to extend the optimized numeric association rules to rules with two numeric attributes in the presumptive condition, and to find the region in the two-dimensional space of these attributes that represents a nice association rule between these two numeric attributes and the conclusion. For instance, we would like to find a rule such as

$$(Age, Balance) \in X \Rightarrow (CardLoan = yes),$$

where X is a rectangle or a connected region in two-dimensional space of *Age* and *Balance*. Optimized rules can also be naturally defined in this extension.

We considered regions that can be defined on a two-dimensional pixel grid plane G. While the problem of finding the optimal arbitrary connected pixel grid region is NP-hard, we proposed practical solutions for the cases where the regions are x-monotone, rectilinear convex, or rectangular [FMMT96b, YFM$^+$97, FMMT01].

Assume that we focus on two numerical attribute B and C. We distribute the values of B (resp. C) into N_B (N_C) buckets so that every bucket contains almost the same number of records. We then divide the Euclidean plane associated with B and C into $N_B \times N_C$ pixels (unit squares). For simplicity, we assume that $N_B = N_C = N$, without loss of generality, as regards our algorithms.

A *grid region* is a union of pixels in G that are connected. We considered three class of region families, *x-monotone*, *rectilinear convex*, and *rectangular*. (Examples of these regions are shown in Figure 1.5 in Section 1.2.1.) We considered algorithms for computing optimized two-dimensional association rules, i.e., both optimized support two-dimensional association rules and optimized confidence two-dimensional association rules.

Optimized Rectangles

There are $O(N^4)$ rectangular regions of G. Thus, a naive algorithm examines these $O(N^4)$ rectangles and outputs the optimal one. The time complexity of this algorithm is $O(N^4)$, which is too expensive.

It can be easily reduced to $O(N^3)$ by transforming the problem into the computation of optimized ranges. We choose a pair $r < r'$ of rows in G, and consider only rectangles whose horizontal edges are on these rows. For each column index j, we define $u_j = \sum_{i=r}^{r'} u_{i,j}$ and $v_j = \sum_{i=r}^{r'} v_{i,j}$.

Consider buckets B_1, B_2,..., B_N such that the number of records in B_j is u_j and the number of success records in B_j is v_j. From Theorem 2.1 and Theorem 2.2, we can compute the optimized confidence range and the optimized support range in $O(N)$ time. Since there are $O(N^2)$ candidate pairs of rows, the optimized confidence rectangle and the optimized support rectangle can be computed in $O(N^3)$ time.

Theorem 2.3 *Each of the optimized confidence rectangle and the optimized support rectangle can be computed in $O(N^3)$ time.* **[EOT]**

Non-Rectangular Optimized Regions

In our experience, the use of non-rectangular regions often yields useful, i.e., accurate and comprehensive, rules. For example, let us consider "Age" and "Salary" as the numeric attributes, and "GoldCard" as the objective condition. Here, we would expect to find a rule that, among people on the same salary, younger ones are more likely to pay an annual fee for premium credit cards. This expectation is confirmed if we find a two-dimensional association whose region resembles a triangle, which is a rectilinear convex and an x-monotone region, but a rectangle. Consequently, we believe that rectilinear convex or x-monotone regions are better than rectangles for our class of regions in two-dimensional association rules.

We found that the optimized two-dimensional association rules that optimize support or confidence can be computed in time proportional to $O(N^2 n)$ and $O(N^3 n)$ where n is the number of records in the whole grid G for x-monotone regions and rectilinear convex regions, respectively [FMMT96b]. However, n is unacceptably large in data mining applications. Therefore, another efficient algorithms that closely approximate the optimized two-dimensional regions are explored in [FMMT96b]. Those algorithms run in time proportional to $O(N^2 \log n)$ and $O(N^3 \log n)$ for x-monotone regions and rectilinear convex regions respectively.

The author also considered algorithms for computing the optimal two-dimensional regions for discriminant rules of classification and regression problems. The details of each algorithm for classification and regression are discussed in Section 3.4. Since the problems for the two-dimensional optimized support or confidence rules has similar properties of the problems in Section 3.4, the author omited the details of the two-dimensional optimized support or confidence rules in this dissertation.

Chapter 3

Classification

We can find unexpected patterns from rules extracted by data mining technologies. Such patterns, or rules, are further investigated to utilize in businesses. For example, some rules may affect whether a certain product is sold or not by a customer. We want to know which rule is the most important for the product sales. And, we naturally would like to know such information to identify rules that predict whether the product will be sold by new customers.

Assume that we have a following schema of relations concerning weather.

```
(Weather, Pressure, Temperature, Humidity)
```

If we are interested in the `Weather`, useful rule is, for example, "if the `Pressure` is larger than 980 and the `Humidity` is smaller than 60% then it is likely to be `Fair`." In such case, we use the `Pressure`, the `Temperature`, and the `Humidity` to predict or model the `Weather`. Thus, there is one designated attribute whose value of which we would like to predict or model, and we call this attribute the *target attribute*. The other attributes are called *conditional attributes*.

General form of the rules for such modeling and prediction tasks is:

$$P_1(C_1) \wedge P_2(C_2) \cdots \wedge P_k(C_k) \Rightarrow t[A] = a$$

where C_1, ... , C_k are conditional attributes that are used to predict or model the value of the target attribute A. In this dissertation, the author use a notation $t[C]$ for denoting the value of the attribute C of a certain record. The $P_i(C_i)$ $(i = 1, ..., k)$ are predicates that involve attribute C_i. There are two types of attributes: *categorical* and *numerical* attributes. For categorical attributes, the form of the predicates would be $t[C_i] \in V$ where V is a subset of values in the domain of C_i. For numerical attributes, the form would be $t[C_i] \in [v_{lo}, v_{hi}]$ where v_{lo} and v_{hi} $(v_{lo} < v_{hi})$ are threshold values in the domain of C_i.

The rules whose target attribute is categorical are called *classification* rules. On the other hand, the rules whose target attribute is numerical are called *regression* rules. In this chapter, the author explores classification rules.

3.1 Discriminant Rules

If we applied a rule (or predicates of a rule) for a set of records, each record of the set can be divided into two segments: one is the set of records that satisfy the predicates and the other is the set of records that do not satisfy the predicates. Therefore, rules (or predicates) for classification problems are often called *discriminant rules* and they can be characterized by how they discriminate records in a database on the view point of value distribution of the target attribute.

In this dissertation, the author explores the classification problems as geometrical problems. Therefore, we now characterize rules geometrically and define *stamp points* of rules (or predicates).

3.1.1 Stamp Points

Let A be the target attribute that is categorical. Assume A has k distinct values, $a_1, \cdots, a_k$. A rule $P(C)$ that is defined on a conditional attribute C divides the database relation R into two segments, a segment S containing records that satisfy the predicate $P(C)$ and its complement segment $\bar{S}$ containing records that do not satisfy the predicate. We can characterize any rules by the corresponding segment S or its complement $\bar{S}$.

Definition 3.1 Assume a predicate divides the database relation R into two segment S and its complement. For a segment S, let $x_i(S)$ be the number of records in S for which the value of the target attribute is a_i. Thus, each segment S of the relation R can be mapped to a point $\mathbf{x}(S) = (x_1(S), x_2(S), \cdots, x_k(S))$ in the k-dimensional Euclidean space, which is referred to as a *stamp point* of S, or of the predicate. **[EOD]**

A stamp point represents the distribution of the target attribute of interest. In the rest of the dissertation, we use the k-dimensional stamp points, $\mathbf{x}(S) = (x_1(S), x_2(S), \cdots, x_k(S))$, for classification problems where k is the number of distinct values that the target attribute can take.

3.1.2 Rules on Categorical Conditional Attribute

Definition 3.2 Let C be a categorical conditional attribute and $\mathrm{dom}(C)$ be the *domain* of C, i.e., a set of all values that $t[C]$ can take. **[EOD]**

Let $V \subset \mathrm{dom}(C)$ be a subset of values on the domain of C. Rules on C can be characterized by V. As mentioned above, the subset divides the database relation R into two segments $S = \{t \in R \,|\, t[C] \in V\}$ and $\bar{S} = \{t \in R \,|\, t[C] \notin V\}$. The subset can further be mapped into a stamp point.

As the author argues later in this chapter, we can compare the quality of rules based on the coordinate values of stamp points. For each categorical conditional attribute, we will examine all possible subsets on the domain and their corresponding stamp points.

When changing the subset V, we will frequently compute stamp points $\mathbf{x}(V)$, i.e., $\mathbf{x}(S)$. If we scan the database relation R to find each stamp point, it will always take $O(|R|)$ time. To speed up this process, we preprocess the relation as follows.

Definition 3.3 Among all possible subsets on $\text{dom}(C)$, we call subsets that consist of only one element *atomic* subsets and denote each such subset V_{atom}. **[EOD]**

We can construct an arbitrary V for the categorical conditional attribute by making a union of atomic subsets.

We compute a stamp point $\mathbf{x}(V_{\text{atom}}) = (x_1, x_2, \cdots, x_k)$ for each atomic subset beforehand. To find the stamp point of V, we simply need to sum up the points $\mathbf{x}(V) = \sum_{V_{\text{atom}} \in V} \mathbf{x}(V_{\text{atom}})$, which will take $O(|\text{dom}(C)|)$ time. We also compute the stamp point of the entire relation $\mathbf{x}(R)$, since the stamp point of $\bar{S} = R \setminus S$, the complement of S, can be easily computed from $\mathbf{x}(\bar{S}) = \mathbf{x}(R) - \mathbf{x}(S)$.

Example 3.1 Suppose we are given a relation R with categorical attributes A and C. Let A be a target attribute having k distinct values, and C be a conditional attribute having n distinct values. The following SQL query will count the number of records for each distinct value of A and value of C to generate the stamp points of the atomic subsets:

```
SELECT A, C, COUNT(*) FROM R GROUP BY A, C.
```

A	C	count(*)	
a_1	c_1	26	$= x_1(\{c_1\})$
a_1	c_5	15	$= x_1(\{c_5\})$
a_2	c_2	31	$= x_2(\{c_2\})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_k	c_n	17	$= x_k(\{c_n\})$

3.1.3 Rules on Numerical Conditional Attribute

Let C be a numerical conditional attribute. Let the minimal and maximal value of $t[C]$ in the database relation R be v_{min} and v_{max} respectively. We call the value range between v_{min} and v_{max}, $[v_{min}, v_{max}]$, be the domain of C. Let v_{lo}, v_{hi} ($v_{lo} < v_{hi}$) be the threshold values on the domain of C that divides the database relation R into two segments $S = \{t \in R \mid t[C] \in [v_{lo}, v_{hi}]\}$ and $\bar{S} = \{t \in R \mid t[C] \notin [v_{lo}, v_{hi}]\}$. The v_{lo} can be $-\infty$ or the v_{hi} can be ∞.

Rules on the numerical conditional attribute C are all possible combinations of v_{lo} and v_{hi} in the domain of C. And all rules can be mapped into stamp points. For each numerical conditional attribute, we will examine all possible v_{lo} and v_{hi} combinations on the domain and their corresponding stamp points. Therefore, we will frequently compute a stamp point $\mathbf{x}(S)$, taking $O(|R|)$ time for each.

For each numerical conditional attribute C, we make adequate number of buckets on the domain. We precompute stamp points for all the buckets, like atomic points in the previous

Table 3.1: Weather Forecast Database (Training Data)

Weather	Forecast	Pressure	Temperature	···
Fair	Fair	1012	24.6	···
Fair	Fair	998	28.0	···
Rain	Rain	986	24.2	···
Rain	Fair	968	21.7	···
···	···	···	···	···
Fair	Fair	1004	24.5	···

section. Assume that C takes n distinct numerical values and we are trying to make M buckets. If n is small enough to fit in a main memory, we use $M = n$, that is, we make buckets for all the n values. In such cases, we can compute all stamp points of buckets by simple SQL query like the previous example and sort all the buckets. However, for large databases, n might be large and sometimes we do not allow to sort the n value. Therefore, we need smaller buckets.

Though there are many ways of making M buckets, we make *equi-depth buckets*, which divide n records into almost evenly, because we empirically learned that the equi-depth buckets have suitable properties, for instance robustness against noise [FMMT99], for data mining applications. In Section 2.2.2, the author presented an efficient algorithm, which runs in $O(n \log M)$ time, for making such equi-depth buckets. Note that since $M \ll n$, the running time is almost linear to n.

To find the stamp point of value range $B = [B_{lo}, B_{hi}](lo \leq hi)$, we simply need to sum up the points $\mathbf{x}(B) = \sum_{i=lo}^{hi} \mathbf{x}(B_{\mathrm{i}})$, which will take $O(M)$ time.

3.2 Criteria of Classification Rules

Assume that we have a weather forecast database in Table 3.1. In the database, `Weather={Fair, Rain}` is actual weather of the next day, `Forecast={Fair, Rain}` is a weather forecast for the next day, and `Pressure` and `Temperature` on today. Assume that we found following two association rules that lead to actual weather of the next day.

- `Rule 1:` $Y1$
  `Forecast=Fair` $\Rightarrow$ `Weather=Fair (support=54%, confidence=90%)`
- `Rule 2:` $Y2$
  `Pressure < 980` $\Rightarrow$ `Weather=Rain (support=30%, confidence=50%)`

If we are interested in weather of the next day, we want to know which rules are important to know weather of the next day. In the database, actual weather values are given, we call such data *training data*. However, in general, we want to predict weather of future from observed data like Table 3.2 whose next day's weather is unknown. In such cases, we use classification rules for predicting the `Weather` by using other given conditional attributes.

Table 3.2: Observed Data

Weather	Forecast	Pressure	Temperature	⋯
?	Fair	970	24.6	⋯
?	Fair	1004	24.5	⋯
⋯	⋯	⋯	⋯	⋯

Let us focus on the first record of Table 3.2. Since its `Forecast` is `Fair`, we can apply the `Rule 1` and therefore we can predict the `Weather` is `Fair`. However, its `Pressure` value is less than 980. Therefore, the `Rule 2` is also applicable for the record and we can predict the `Weather` is `Rain`. Both the support and the confidence are not suitable measures to compare rules for modeling and prediction tasks.

The significance of the discovered rules depends on the user's objective, and hence there is no universal criterion for measuring the significance. A useful segmentation should divide data into segments whose target distribution is more skewed than that of the data as a whole. Therefore, for classification problems, we often use *mutual information*, *GINI index*, or χ^2 for the modeling and prediction tasks. All of the above criteria indicate the extent to which the divided data distributions are skewed and differ from the original data distribution.

Example 3.2 Let us consider mutual information criterion as an example. The following is the *entropy* function, which indicates the extent of uncertainty of information.

$$H(S) = -\sum_{j=1}^{k} p_j \log_k p_j$$

In the formula, S is a segment $S = \{x_1, ..., x_k\}$ where x_j is the number of records in S whose target value is the j-th value in the target domain. The p_j is the probability of the j-th target value in S, i.e., $x_j/|S|$ where $|S|$ is the number of records in S. If we have a segment whose probability of `Fair` and `Rain` are 0.95 and 0.05 respectively, the entropy value of the segment is $-0.95 \log_2 0.95 - 0.05 \log_2 0.05 = 0.286$. Similarly, if we have a segment whose corresponding probabilities are both 0.5, the entropy value is $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1.0$. Notice that if we predict `Fair` since `Fair` is the majority value in the former segment, we can answer the `Weather` value with 95% accuracy. On the other hand, we can only predict with 50% accuracy for the latter segment. Thus, the latter segment has much uncertainty with respect to the `Weather`.

Assume that we have historical results for the weather records regarding the `Rule 1`, say $Y1$, and the `Weather`, say X, as in Table 3.3. From the table, the entropy value regarding the `Weather` (X) is

$$\begin{aligned} H(X) &= -0.64 \log_2 0.64 - 0.36 \log_2 0.36 \\ &= 0.943. \end{aligned}$$

If we discriminate records that satisfy the `Rule 1` ($Y1 = Yes$), the probabilities of `Fair` and `Rain` in the discriminated segment are $0.54/0.6 = 0.9$, $0.06/0.6 = 0.1$, respectively. Therefore,

Table 3.3: Information of the Rule 1

		`Forecast = Fair:` $Y1$		
		`Yes`	`No`	
`Weather` X	`Fair`	`0.54`	`0.1`	`0.64`
	`Rain`	`0.06`	`0.3`	`0.36`
		`0.6`	`0.4`	

Table 3.4: Information of the Rule 2

		`Pressure` $\leq$ `980:` $Y2$		
		`Yes`	`No`	
`Weather` X	`Fair`	`0.3`	`0.34`	`0.64`
	`Rain`	`0.3`	`0.06`	`0.36`
		`0.6`	`0.4`	

the entropy value is

$$\begin{aligned} H(X|Y1 = Yes) &= -0.9 \log_2 0.9 - 0.1 \log_2 0.1 \\ &= 0.469. \end{aligned}$$

As for the other segment, it can be computed as

$$\begin{aligned} H(X|Y1 = No) &= -0.25 \log_2 0.25 - 0.75 \log_2 0.75 \\ &= 0.811. \end{aligned}$$

Since the probability of `Fair` and `Rain` in the `Forecast` attribute is 0.6 and 0.4, respectively, the `Rule 1` ($Y1$) reduces the entropy value regarding the `Weather` (X) to

$$\begin{aligned} H(X|Y1) &= 0.6 * H(X|Y1 = Yes) + 0.4 * H(X|Y1 = No) \\ &= 0.606. \end{aligned}$$

The reduction of the entropy value by the `Rule 1` ($Y1$), i.e., $H(X) - H(X|Y1) = 0.337$, is the mutual information regarding the `Weather` (X) of the `Rule 1` ($Y1$). Similarly from Table 3.4, the mutual information of the `Rule 2` ($Y2$) is $H(X) - H(X|Y2) = 0.099$. **[EOE]**

3.2.1 Classification Criteria

All of the mentioned criteria can be defined as functions of stamp points, $\mathbf{x}(S) = (x_1, \cdots, x_k)$ for a classification rule whose target attribute has k distinct values, $a_1, \cdots, a_k$. Followings are criteria of classification rules. In each definition, let R be set of all records in the database and $R = S \cup \bar{S}$ where $\bar{S}$ is the complement of S. Let $p_i(S)$ be the probability of the i-th target value, a_i, in a set S, i.e., $x_i(S)/|S|$. And let $x_i(S)$ be the number of records in S whose target value is the i-th value in the target domain.

Mutual Information

The following entropy gain function compares the mutual information gained by a segmentation, S. It indicates "how much information is given by the segmentation" like the above example.

$$\begin{aligned} Ent(\mathbf{x}(S)) &= Ent(S;\bar{S}) \\ &= -\sum_{i=1}^{k} p_i(R)\log p_i(R) \\ &\quad +\frac{|S|}{|R|}\sum_{i=1}^{k} p_i(S)\log p_i(S) + \frac{|\bar{S}|}{|R|}\sum_{i=1}^{k} p_i(\bar{S})\log p_i(\bar{S}) \end{aligned}$$

GINI Index

The implication of the "gini" criterion, which is used in the CART system [BFOS84], is "how much the mean squared error of the target values is decreased by a segmentation, S." The optimal segmentation according to this criterion minimizes the mean squared error. The error is the sum of the number of misclassified records in each segment, that is, the number of records that are not the majority value in each segment. Let $|S| = \sum_{i=1}^{k} x_i(S)$. The GINI index criterion is defined as follows:

$$\begin{aligned} Gini(\mathbf{x}(S)) &= Gini(S;\bar{S}) \\ &= \left(1-\sum_{i=1}^{k} p_i(R)^2\right) \\ &\quad -\frac{|S|}{|R|}\left(1-\sum_{i=1}^{k} p_i(S)^2\right) - \frac{|\bar{S}|}{|R|}\left(1-\sum_{i=1}^{k} p_i(\bar{S})^2\right) \end{aligned}$$

χ^2 (Correlation)

The following χ^2 function indicates how strongly the statistical hypothesis that "S and $\bar{S}$ are not different from R" is denied.

$$\begin{aligned} Chi(\mathbf{x}(S)) &= Chi(S;\bar{S}) \\ &= \sum_{i=1}^{k} \frac{|S|(p_i(S)-p_i(R))^2 + |\bar{S}|(p_i(\bar{S})-p_i(R))^2}{p_i(R)} \end{aligned}$$

The quality of a classification rule with respect to mutual information criterion can be evaluated by the value of $Ent(\mathbf{x}(S)) = Ent(S;\bar{S})$ if the value group splits data into two segments, say "S" and "$\bar{S}$." The higher the value of the objective function is, the better the quality of the rule is with respect to this criterion. Similarly, we prefer higher values of $Gini(\mathbf{x}(S)) = Gini(S;\bar{S})$ and $Chi(\mathbf{x}(S)) = Chi(S;\bar{S})$.

3.2.2 Optimal Rule

For classification problems, the quality of a rule can be evaluated by one of the objective functions for the classification criteria. The higher the value of the objective function is, the better the quality of the rule is with respect to the corresponding criterion. Among all possible rules on a conditional attribute, we call the rule that has the highest value of the specified objective function the *optimal rule*.

The optimal rule can be considered as the most important rule for modeling and prediction tasks of a certain target attribute. Especially, for constructing decision trees, the optimal rule is frequently computed. Therefore, many researches have been focused on the optimization problem.

For a categorical conditional attribute that has n distinct values, there are $O(2^n)$ possible rules or stamp points. It is not affordable to examine all possible rules exhaustively except for the cases when n is small. And for a numerical conditional attribute that has M buckets (and n values), there are $O(M^2)$ (or $O(n^2)$) possible rules or stamp points, if we use a value range as rules. Though it is much smaller compared to the case for a categorical conditional attribute, it is sometimes too expensive for large n.

3.2.3 Convexity of Classification Criteria

In this dissertation, we consider the optimization problem in computational geometry context. Therefore, we discuss efficient algorithm for finding the optimal stamp point from the set of $O(2^n)$ stamp points on a categorical conditional attribute or the set of $O(n^2)$ stamp points on a numerical conditional attribute. If we consider the problem in computational geometry context, we can utilize a property of the objective functions. An objective function, say $f(\mathbf{x})$, is *convex* if it satisfies following condition.

Definition 3.4 $f(\mathbf{x})$ is *convex* if

$$max\{f(\mathbf{x1}), f(\mathbf{x2})\} \geq f((1-\gamma)\mathbf{x1} + \gamma\mathbf{x2})$$

for $0 \leq \gamma \leq 1$ and arbitrary points $\mathbf{x1}$ and $\mathbf{x2}$ in the domain of f. **[EOD]**

All of the objective functions, the mutual information $Ent(\mathbf{x})$, the gini index $Gini(\mathbf{x})$, and the χ^2 function $Chi(\mathbf{x})$ are all convex on $\mathbf{x}$ in the k-dimensional space.

Mutual Information Function

Theorem 3.1 The function $Ent(\mathbf{x})$ is convex in the k-dimensional space; that is,

$$max\{Ent(\mathbf{x1}), Ent(\mathbf{x2})\} \geq Ent((1-\gamma)\mathbf{x1} + \gamma\mathbf{x2})$$

for $0 \leq \gamma \leq 1$ and arbitrary points $\mathbf{x1} = (x1_1, ..., x1_k)$ and $\mathbf{x2} = (x2_1, ..., x2_k)$ satisfying $x1_i > 0$ and $x2_i > 0$ for $i = 0, ..., k$. **[EOT]**

Proof: Let $f(\mathbf{x}) = \sum_{i=1}^{k} x_i \log(x_i/s(\mathbf{x}))$, where $s(\mathbf{x}) = \sum_{i=1}^{k} x_i$, i.e., $|S|$. Since $x_i = p_i|S|$, we have

$$Ent(\mathbf{x}(S)) \quad = \quad Ent(S; \bar{S})$$

$$
\begin{aligned}
&= -\sum_{i=1}^{k} p_i(R)\log p_i(R) \\
&\quad + \frac{|S|}{|R|}\sum_{i=1}^{k} p_i(S)\log p_i(S) + \frac{|\bar{S}|}{|R|}\sum_{i=1}^{k} p_i(\bar{S})\log p_i(\bar{S}) \\
&= -\sum_{i=1}^{k} p_i(R)\log p_i(R) \\
&\quad + \frac{1}{|R|}f(\mathbf{x}(S)) + \frac{1}{|R|}f(\mathbf{x}(\bar{S})).
\end{aligned}
$$

In the function, $\sum_{i=1}^{k} p_i(R)\log p_i(R)$ is invariant to the choice of $\mathbf{x}$. Therefore, it suffices to show that the convexity of $f(\mathbf{x})$.

For a vector $\Delta = (\delta_1, ..., \delta_k)$, let $Y(\Delta) = (\Delta, x) = \sum_{i=1}^{k} \delta_i x_i$. In order to show the convexity of $Ent(\mathbf{x})$, it suffices to show that for any Δ,

$$f''(\mathbf{x}) = \frac{\partial^2 f(\mathbf{x})}{\partial Y(\Delta)^2} \geq 0.$$

Since

$$f'(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial Y(\Delta)} = \sum_{i=1}^{k} \frac{\partial f(\mathbf{x})}{\partial x_i}\frac{1}{\delta_i},$$

$$f'(\mathbf{x}) = \sum_{i=1}^{k} \delta_i^{-1}\log x_i - s(\mathbf{t})\log s(\mathbf{x}),$$

where $\mathbf{t} = (\delta_1^{-1}, .., \delta_k^{-1})$. Hence,

$$f''(\mathbf{x}) = \sum_{i=1}^{k} \delta_i^{-2} x_i^{-1} - s(\mathbf{t})^2 s(\mathbf{x})^{-1}.$$

Now, it suffices to show that

$$(*) = s(\mathbf{x})\sum_{i} \delta_i^{-2} x_i^{-1} - s(\mathbf{t})^2 \geq 0.$$

We consider vectors $A = (x_1^{1/2}, .., x_k^{1/2})$ and $B = (x_1^{-1/2}\delta_1^{-1}, .., x_k^{-1/2}\delta_k^{-1})$. Then, from the Cauchy-Schwarz inequality, $(*) = |A|^2|B|^2 - (A, B)^2 \geq 0$. **[EOP]**

GINI Index Function

Theorem 3.2 The function $Gini(\mathbf{x})$ is convex in the k-dimensional space; that is,

$$max\{Gini(\mathbf{x1}), Gini(\mathbf{x2})\} \geq Gini((1-\gamma)\mathbf{x1} + \gamma\mathbf{x2})$$

for $0 \leq \gamma \leq 1$ and arbitrary points $\mathbf{x1} = (x1_1, ..., x1_k)$ and $\mathbf{x2} = (x2_1, ..., x2_k)$. **[EOT]**

Proof: For any vector $\mathbf{\Delta} \neq 0$, the second derivative of the functions along with $\mathbf{\Delta}$ is non-negative.

Let $\mathbf{r} = (r_1, \cdots, r_k)$ be the set of all records to be split, and let $\|\mathbf{r}\| = \sum_{i=1}^{k} r_i$, i.e., $|R|$ and let $\|\mathbf{x}\| = \sum_{i=1}^{k} x_i$, i.e., $|S|$.

$Gini(\mathbf{x}(S))$ is transformed as follows:

$$Gini(\mathbf{x}(S)) \quad = \quad Constant + \frac{1}{|R|}G(\mathbf{x}(S)) + \frac{1}{|R|}G(\mathbf{x}(R) - \mathbf{x}(S)),$$

where

$$G(\mathbf{x}(S)) = G(\mathbf{x}) = \sum_{i=1}^{k} \frac{x_i^2}{\|\mathbf{x}\|}.$$

Let $\mathbf{\Delta} = (\delta_1, \delta_2, \cdots, \delta_k)$, $Y = (\mathbf{\Delta}, \mathbf{x}) = \sum_{i=1}^{k} \delta_i x_i$. The first derivative of $G(\mathbf{x})$ is

$$\begin{aligned} G'(\mathbf{x}) &= \frac{dG(\mathbf{x})}{dY} \\ &= \sum_{i=1}^{k} \frac{1}{\delta_i} \frac{\partial G(\mathbf{x})}{\partial x_i} \\ &= \frac{1}{\|\mathbf{x}\|} \sum_{i=1}^{k} \frac{2x_i}{\delta_i} - \frac{\|\mathbf{t}\|}{\|\mathbf{x}\|^2} \sum_{i=1}^{k} x_i^2, \end{aligned}$$

where $\mathbf{t} = (\delta_1^{-1}, \cdots, \delta_k^{-1})$, and the second derivative is

$$G''(\mathbf{x}) \quad = \quad \frac{2}{\|\mathbf{x}\|} \sum_{i=1}^{k} (\frac{\|\mathbf{t}\| x_i}{\|\mathbf{x}\|} - \frac{1}{\delta_i})^2 \geq 0.$$

Therefore, $Gini''(\mathbf{x}(S)) \geq 0$. **[EOP]**

χ^2 Function

Theorem 3.3 The function $Chi(\mathbf{x})$ is convex in the k-dimensional space; that is,

$$max\{Chi(\mathbf{x1}), Chi(\mathbf{x2})\} \geq Chi((1-\gamma)\mathbf{x1} + \gamma\mathbf{x2})$$

for $0 \leq \gamma \leq 1$ and arbitrary points $\mathbf{x1} = (x1_1, ..., x1_k)$ and $\mathbf{x2} = (x2_1, ..., x2_k)$. **[EOT]**

Proof: $Chi(\mathbf{x}(S))$ is transformed as follows:

$$Chi(\mathbf{x}(S)) \quad = \quad C(\mathbf{x}(S)) + C(\mathbf{x}(R) - \mathbf{x}(S)),$$

where

$$C(\mathbf{x}(S)) = C(\mathbf{x}) = \sum_{i=1}^{k} \frac{(x_i - r_i \frac{\|\mathbf{x}\|}{\|\mathbf{r}\|})^2}{r_i \frac{\|\mathbf{x}\|}{\|\mathbf{r}\|}}.$$

The first derivative of $C(\mathbf{x})$ is

$$\begin{aligned} C'(\mathbf{x}) &= \frac{dC(\mathbf{x})}{dY} \\ &= \sum_{i=1}^{k} \frac{1}{\delta_i} \frac{\partial C(\mathbf{x})}{\partial x_i} \\ &= \frac{\|\mathbf{r}\|}{\|\mathbf{x}\|} \sum_{i=1}^{k} \frac{2x_i}{\delta_i r_i} - \frac{\|\mathbf{t}\|\|\mathbf{r}\|}{\|\mathbf{x}\|^2} \sum_{i=1}^{k} \frac{x_i^2}{r_i} - \|\mathbf{t}\|, \end{aligned}$$

and the second derivative is

$$\begin{aligned} C''(\mathbf{x}) &= \frac{d^2C(\mathbf{x})}{dY^2} \\ &= \frac{2\|\mathbf{r}\|}{\|\mathbf{x}\|}\sum_{i=1}^{k}\frac{1}{r_i}\left(\frac{1}{\delta_i} - x_i\frac{\|\mathbf{t}\|}{\|\mathbf{x}\|}\right)^2 \geq 0. \end{aligned}$$

Therefore, $Chi''(\mathbf{x}(S)) \geq 0$. **[EOP]**

3.3 Optimal Rule on Categorical Attribute

Definition 3.5 Let A be the target attribute, and let $\mathrm{dom}(A) = \{a_1, a_2, \cdots, a_k\}$ be the domain of A, where k is the *target domain size*, that is, the number of distinct values that A has. Let $x_i(S)$ denote the number of records in $S \subseteq R$ for which the value of the target attribute A is a_i $(1 \leq i \leq k)$. Let C be a conditional attribute C whose domain is $\mathrm{dom}(C) = \{c_1, c_2, \cdots, c_n\}$, where n is the *conditional domain size.*

To make a binary segmentation of R, we use a set of attribute values, which we call a *value group*, on the conditional attribute C. Let $V \subset \mathrm{dom}(C)$ be a value group that divides the database relation R into two segments $S = \{t \in R \,|\, t[C] \in V\}$ and $\bar{S} = \{t \in R \,|\, t[C] \notin V\}$, where t denotes a record in R and $t[C]$ denotes a value of t for attribute C. We say "V splits R into $(S; \bar{S})$." Such a value group V is a discriminant rule on conditional categorical attribute. **[EOD]**

Our ideal goal is to find, among all possible value groups, a value group V that optimize an objective function $f(S; \bar{S}) = f(x_1(S), \cdots, x_k(S)) = f(\mathbf{x}(S))$. The $f(\mathbf{x}(S))$ is one of $Ent(\mathbf{x}(S))$, $Gini(\mathbf{x}(S))$, and $Chi(\mathbf{x}(S))$.

We can also consider a value group on multiple conditional categorical attributes. Let $C_1, C_2, \cdots, C_M$ be the conditional attributes. We can treat these attributes as a single attribute C whose domain is the Cartesian product of their domains, that is, $\mathrm{dom}(C) = \mathrm{dom}(C_1) \times \mathrm{dom}(C_2) \times \cdots \times \mathrm{dom}(C_M)$. If C_i, where $i = 1, 2, \cdots, M$, has n_i distinct values, the conditional domain size of C is $n = \prod_i n_i$ for $1 \leq i \leq M$.

Note that we do not have to treat all conditional categorical attributes as a single attribute. In many applications, it is better to treat each attribute separately. In such cases, there is a binary segmentation problem for each attribute.

Example 3.3 Table 3.5 shows an example of a relation projected from the sales log of a car rental company. In the relation, `Color`, and `Size` show the characteristics of each car rented, and `Age` shows the customer's age. Assume that there are five colors (`white`, `black`, `red`, `blue`, and `silver`) and three sizes (`compact`, `medium`, and `large`) in the relation.

Assume that the company wants to classify customers according to the value of `Age` by using the characteristics of cars. We use the `Age` as a target attribute and the other attributes as conditional attributes. If we treat the two conditional attributes as a single attribute, a value group is a set of values like "`Color=white` and `Size=compact`" or "`Color=red` and `Size=compact`."

Table 3.5: Car Rental Sales Log

Color	Size	**Age**
white	compact	**young**
black	compact	**young**
black	medium	**middle-aged**
red	large	**old**
blue	large	**middle-aged**
white	medium	**old**
...	...	...

Finding a high quality value group on the conditional domain gives us clues to understanding what characteristics attracted young, middle-aged, or old customers.

If there are five colors and three sizes, there are at most $15 = 5 \times 3$ values in the conditional domain of the relation. Therefore, we have to examine at most 2^{15-1} possible value groups in this small example to find the optimal one. [**EOE**]

In general, if the conditional domain size is n, there are $2^{n-1} - 1$ possible value groups. Hence, a naive exhaustive search for the optimal binary segmentation requires $O(2^n)$ time, which is not practical. Therefore, we need feasible heuristics that can find high quality value groups approximating the optimal one with respect to the quality of the binary segmentation.

Definition 3.6 In the rest of this section, we use P as a set of all stamp points of value groups and $P_{\text{atom}} \subset P$ as a set of all stamp points of atomic value groups. We define $Conv(P)$ to be the convex hull of P. [**EOD**]

3.3.1 Related Works

For the case in which the target domain size is two, i.e., $k = 2$, we can order the n values so that the optimal value group is one "cut" of the ordered sequence, if we can assume convexity of the objective criterion [BFOS84], and all of the mentioned criteria do have this property. Consequently, we have an $O(n \log n)$ algorithm. However, this algorithm is not applicable for the cases in which the target domain size is greater than two.

For categorical databases, in which the conditional domain size is large and $k > 2$, there is no practical existing algorithm that can find the optimal value group. Despite the difficulty, there are some heuristics for handling the problem [BFOS84, MP91, Qui93] that are used in practice for constructing decision trees.

A heuristic called "two-ing" [BFOS84] divides the target domain into two classes, called superclasses, and applies the $O(n \log n)$ algorithm for $k = 2$ to create the optimal subdivision for each of the 2^{k-1} possible divisions into superclasses, and finds the best one among them. This runs in $O(2^{k-1} n \log n)$ time, which is efficient for a small constant k.

```
0) Algorithm GREEDY-ENUMERATION() {
1)     Sort points in $P_{\text{atom}}$ in descending order of $y = x_1/(x_1 + x_2)$.
       (It takes $O(n \log n)$ time.)
2)     $\mathbf{p}_0 = (0, 0)$
3)     For $i = 1$ to $n$ {
4)         $\mathbf{p}_i = \mathbf{p}_{i-1} + \mathbf{x}_i$
           ($\mathbf{x}_i$ is the $i$-th point of the ordered point sequence.)
5)         Examine the stamp point $\mathbf{p}_i$ by an objective function.
6)     }
7) }
```

Figure 3.1: Greedy Enumeration Algorithm

Another heuristic [Qui93] greedily merges two conditional values from the conditional domain to reduce the conditional domain size to $n - 1$, so that the objective function is maximized. It repeats this greedy merging process until $n = 2$ and then returns the final two groups. This $O(n^3)$ heuristic can be used even if k is large.

The above heuristics are known to be practical for constructing decision trees. However, neither of them has a guarantee on the optimality of the result.

3.3.2 Greedy Enumeration Algorithm

Let us consider the case in which the target domain size is two. Any stamp point of P can be characterized as a point in the 2-dimensional space, i.e., $\mathbf{x} = (x_1, x_2)$. In this case, there must be an $O(n \log n)$ algorithm for the optimal value group problem if the objective function is convex, as proved in [BFOS84]. The GREEDY-ENUMERATION algorithm in Figure 3.1 is one such algorithm.

Theorem 3.4 The GREEDY-ENUMERATION algorithm in Figure 3.1 scans all points of $Conv(P)$ and finds the optimal answer in $O(n \log n)$ time. **[EOT]**

Proof: We will show that the iteration in Step 3 to Step 6 scans the upper chain of $Conv(P)$ (see Figure 3.2). Let $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$ be the ordered point sequence of P_{atom} in descending order of $x_1/(x_1 + x_2)$, and let $\mathbf{p}_0$ be the origin $(0, 0)$.

$\mathbf{p}_0$ is on the convex hull. $\mathbf{p}_1 = \mathbf{p}_0 + \mathbf{x}_1$ is also on the convex hull, since if there is a point above $\mathbf{p}_0\mathbf{p}_1$, there must be a stamp point whose $x_1/(x_1 + x_2)$ is larger than that of $\mathbf{x}_1$. This is a contradiction. Similarly, it is easy to see that if $\mathbf{p}_i$ is on the upper convex hull, $\mathbf{p}_{i+1}$ is also on the convex hull.

We do not have to scan the lower chain of the convex hull, since the upper and the lower chains are symmetric. There are n points on the upper chain of $Conv(P)$ and it takes a constant

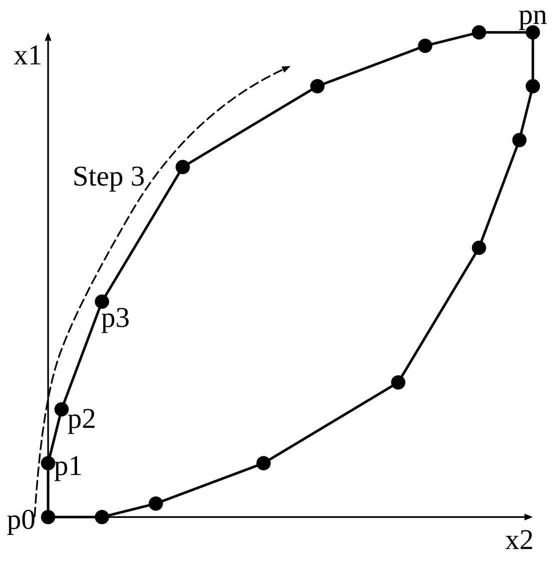


Figure 3.2: Convex hull of Two Class Problem

time to examine the objective function for each point. Hence, the $O(n \log n)$ cost for sorting points in P_{atom} dominates the time complexity of this algorithm. [**EOP**]

Example 3.4 To illustrate this algorithm by using Figure 3.2, we assumes following stamp points of eight atomic value groups.

P_{atom}	x_1	x_2	$x_1/(x_1+x_2)$	$\sum x_1$	$\sum x_2$
$\mathbf{x}_1$	3	0	1	3	0
$\mathbf{x}_2$	3	1	.75	6	1
$\mathbf{x}_3$	6	3	.66	12	4
$\mathbf{x}_4$	9	5	.64	21	9
$\mathbf{x}_5$	5	8	.38	26	17
$\mathbf{x}_6$	3	7	.3	29	24
$\mathbf{x}_7$	1	3	.25	30	27
$\mathbf{x}_8$	0	3	0	30	30

Those stamp points are ordered by $x_1/(x_1+x_2)$. Let $\mathbf{p}_0$ be the origin and let $\mathbf{p}_i = \mathbf{p}_{i-1} + \mathbf{x}_i$ for $i = 1, ..., 8$. The point sequence of $\mathbf{p}_i$ $(i = 1, ..., 8)$ is the upper chain of the convex hull of all stamp points. [**EOE**]

This algorithm is not applicable to cases in which the target domain size k is greater than two.

3.3.3 Enumeration Algorithm

If $k > 2$, we project all points of P_{atom} in the k-dimensional space into points $\mathbf{y} = (x_1/\|\mathbf{x}\|, x_2/\|\mathbf{x}\|, \ldots, x_{k-1}/\|\mathbf{x}\|)$ in the $(k-1)$-dimensional space where $\|\mathbf{x}\| = \sum_{i=1}^{k} x_i$. Let H be a $(k-2)$-dimensional hyperplane that contains at least $k-1$ linearly independent points. H splits the projected space into two halfspaces, i.e., an upper side and a lower side of H. We can define a value group $V(H)$ that corresponds to the union of the projected points in the upper (or lower) halfspace of H. Let $x(V(H))$ be the stamp point of $V(H)$.

Theorem 3.5 The stamp point $x(V(H))$ must be a point on $Conv(P)$. Conversely, each vertex of $Conv(P)$ can be represented as $x(V(H))$ for a suitable hyperplane H. **[EOT]**

Proof: We first prove the first statement of the theorem. Assume a $(k-2)$-dimensional hyperplane $H : a_1y_1 + a_2y_2 + \cdots + a_{k-1}y_{k-1} = a_k$ in the projected space. Each projected point in the upper halfspace of H must be $a_1y_1 + a_2y_2 + \cdots + a_{k-1}y_{k-1} \geq a_k$. That is equivalent to $(a_1 - a_k)x_1 + (a_2 - a_k)x_2 + \cdots + (a_{k-1} - a_k)x_{k-1} - a_kx_k \geq 0$ in the original k-dimensional space.

Any point of $Conv(P)$ must have a $(k-1)$-dimensional tangential hyperplane. If the tangential hyperplane has a normal vector Θ, the hyperplane maximizes (or minimizes) the inner product $(\Theta, \mathbf{x})$. If a point maximizes (or minimizes) the inner product, it must be the tangential point of the hyperplane and $Conv(P)$.

Let V^+ be the value group that is defined by the upper halfspace of H. And let $\Theta_V = (a_1 - a_k, a_2 - a_k, \cdots, a_{k-1} - a_k, -a_k)$. The stamp point of V^+ maximizes $(\Theta_V, \mathbf{x})$ among P, since if there is a point whose $(\Theta_V, \mathbf{x})$ is larger than V^+, there must be an atomic point whose $(\Theta_V, \mathbf{x})$ is non-negative except points in V^+ or there must be an atomic point whose $(\Theta_V, \mathbf{x})$ is negative in V^+. It is contradiction. Therefore, the stamp point of V^+ is a point of $Conv(P)$. The second half can be proved in a similar manner. **[EOP]**

Theorem 3.6 We can enumerate all of the vertices on $Conv(P)$ by examining all combination of $k-1$ atomic points. **[EOT]**

Proof: A set of $k-1$ linearly independent points in the projected space identifies a $(k-2)$-dimensional hyperplane $H : a_1y_1 + a_2y_2 + \cdots + a_{k-1}y_{k-1} = a_k$ that corresponds to a $(k-1)$-dimensional hyperplane $S : (a_1 - a_k)x_1 + (a_2 - a_k)x_2 + \cdots + (a_{k-1} - a_k)x_{k-1} - a_kx_k = 0$ which contains the origin. The hyperplane S, which contains $k-1$ linearly independent points, splits P_{atom} into two groups. Note that S and two tangential hyperplanes, each of which contains the stamp point of the corresponding group, have identical normal vector Θ. Since we are focusing on enumerating different value groups, it is enough to examine all combinations of $k-1$ atomic points and corresponding values of Θ. **[EOP]**

Thanks to the Theorem 3.5 and 3.6, the ENUMERATION algorithm in Figure 3.3 enumerates all vertices on $Conv(P)$:

Theorem 3.1, 3.2, and 3.3 prove that one of the points on the convex hull gives the optimal value group. Therefore, we can concentrate on enumerating all the combinations of $k-1$ projected points in order to find the optimal value group.

Example 3.5 To explain the ENUMERATION algorithm, let us consider the following data, whose target domain size k is three and conditional domain size n is four:

```
0)  Algorithm ENUMERATION() {
1)      Project $P_{\text{atom}}$ into the $(k-1)$-dimensional space.
        $(\mathbf{x} = (x_1, \cdots, x_k) \mapsto \mathbf{y} = (x_1/\|\mathbf{x}\|, \cdots, x_{k-1}/\|\mathbf{x}\|))$
2)      For each combination of $k-1$ projected points, {
3)          If the $k-1$ points are linearly independent {
4)              Define a $(k-2)$-dimensional hyperplane $H$ containing the $k-1$ points.
5)              Initialize a stamp point for a value group, $\mathbf{p} = 0$.
6)              For each atomic point $\mathbf{x}_i$ of the projected points, {
7)                  if $\mathbf{x}_i \mapsto \mathbf{y}_i$ is in the upper halfspace associated with $H$ {
8)                      $\mathbf{p} = \mathbf{p} + \mathbf{x}_i$
9)                      Evaluate $\mathbf{p}$ by means of an objective function.
10)                 }
11)             }
12)         }
13)     }
14) }
```

Figure 3.3: Enumeration Algorithm

	c_1	c_2	c_3	c_4
$x_1(A = a_1)$	20	20	20	30
$x_2(A = a_2)$	20	10	20	10
$x_3(A = a_3)$	40	10	0	40

For each stamp point $\mathbf{x}(\{c_i\}) = (x_1, x_2, x_3)$ of P_{atom} in three-dimensional space, we consider a projected point $\mathbf{y} = (y_1 = x_1/(x_1+x_2+x_3),\ y_2 = x_2/(x_1+x_2+x_3))$ in two-dimensional space. Figure 3.4 illustrates the two-dimensional space. A straight line H in the projected space, which can be identified by two points, splits the projected points $\mathbf{y}$ into two groups.

The broken line H in Figure 3.4, which is $0.0y_1 + 1.0y_2 = 0.25$, splits the atomic points into two groups: $V^+ = \{c_1, c_2, c_3\}$ and $V^- = \{c_4\}$. H corresponds to a plane $S : -0.25x_1 + 0.75x_2 - 0.25x_3 = 0$ in the original three-dimensional space. The value group V^+ (resp. V^-) maximizes (resp. minimizes) the inner product with the normal vector $\Theta = (-0.25, 0.75, -0.25)$ of S (See also the table in Section 3.3.5). **[EOE]**

Let n be a conditional domain size of C. Since one splitting is defined by $k-1$ projected points of n atomic values, there are $O(n^{k-1})$, i.e., $\begin{pmatrix} n \\ k-1 \end{pmatrix}$, different combinations, and it will take $O(n)$ time to obtain the coordinates of a stamp point of a value group. Therefore, the time complexity of the ENUMERATION algorithm 3.3 is $O(n^k)$. There is a way to improve this complexity to $O(n^{k-1})$ by using a sophisticated computational geometry algorithm [AT94b]. However, if k and n become large, it is still costly.

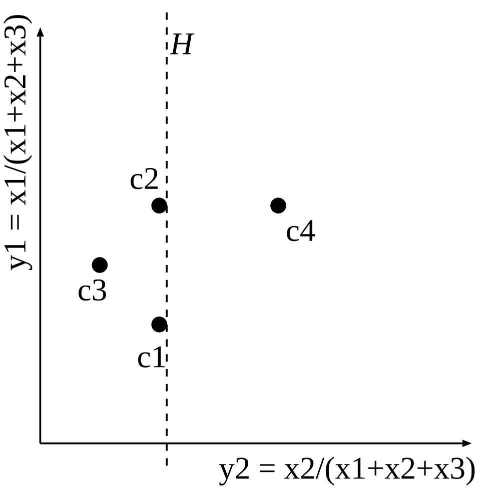


Figure 3.4: A Segmentation in a Projected Space

3.3.4 Random Enumeration Algorithm

To reduce the complexity of the ENUMERATION algorithm, we (1) take an s-sized random sample from P_{atom}, (2) project the sample into a $(k-1)$-dimensional space, and (3) apply the ENUMERATION algorithm to the sample. Figure 3.5 shows such randomized algorithm.

The time complexity of the randomized algorithm becomes $O(s^{k-1}n)$, because it is proportional to $\begin{pmatrix} s \\ k-1 \end{pmatrix}$, and can be further improved to $O(s^{k-2}n)$. The RANDOM-ENUMERATION algorithm needs only a small working space and is easy to parallelize.

Quality Analysis on Sample Size

As we have shown above, the optimal solution is given as a subdivision of atomic stamp points by a hyperplane cut in $(k-1)$-dimensional space. From the PAC learning theory, such a subdivision can be closely approximated by using a small number of samples. Let Y be the set of points in $(k-1)$-dimensional space and Z be a random sample from Y. We say that Z is an ϵ-net for a region family if $|X \cap Y|/|Y| \leq \epsilon$ holds for every region X of the family satisfying $X \cap Z = \emptyset$.

Note that our sample strategy is weighted sampling. First, we consider the unweighted case where $|\mathbf{x}| = 1$ for each x. Suppose that the optimal value group V_{opt} is given by a hyperplane H_{opt}. Let us consider the family of wedges bounded by H_{opt} and another hyperplane. This family of wedges defines at most $O(n^k)$ different value groups of n points, roughly speaking, because the Vapnik-Chervonenkis dimension is k, and hence it is known [HW87, BEHW89] that a random sample of size

$$s(\epsilon) = \epsilon^{-1} \max\{6k \log(16k\epsilon^{-1}), 4 \log(2\delta^{-1})\}$$

is an ϵ-net with a probability of at least $1-\delta$. Note that $s(\epsilon)$ is independent of $n = |Y|$.

Let us take a sample that is an ϵ-net for our wedges, and let V_{sample} be the value group obtained by the sample maximizing the objective function (e.g., Gini($\mathbf{x}$)) F. There exists a

```
0)  Algorithm Random-Enumeration() {
1)      Choose $\{\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_s}\}$ atomic values at random,
        so that each atomic value $\mathbf{x}$ is chosen with probability $\|\mathbf{x}\|s/|\mathbf{x}(R)|$.
        ($|\mathbf{x}(R)|$ is the total number of records in the database.)
2)      Project the sample into the $(k-1)$-dimensional space.
        ($\mathbf{x} = (x_1, \cdots, x_k) \mapsto \mathbf{y} = (x_1/\|\mathbf{x}\|, \cdots, x_{k-1}/\|\mathbf{x}\|)$)
3)      For each combination of $k-1$ points of the $s$ sample points, {
4)          If the $k-1$ points are linearly independent {
5)              Define a $(k-2)$-dimensional hyperplane $H$ containing the $k-1$ points.
6)              Initialize the stamp point of a value group, $\mathbf{p} = 0$.
7)              For each atomic value $\mathbf{x}_i$ of $P_{\text{atom}}$, {
8)                  If $\mathbf{x}_i \mapsto \mathbf{y}_i$ is in the upper halfspace associated with $H$ {
9)                      $\mathbf{p} = \mathbf{p} + \mathbf{x}_i$
10)                     Evaluate $\mathbf{p}$ by means of an objective function.
11)                 }
12)             }
13)         }
14)     }
15) }
```

Figure 3.5: Random Enumeration Algorithm

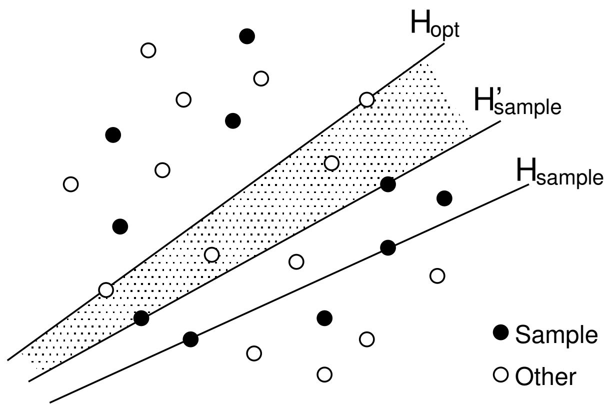


Figure 3.6: Sample and Subdivision

value group V'_{sample} obtained as the sample such that the edge bounded by H_{opt} and H'_{sample} (the hyperplane associated with V'_{sample}) contains no sample point. Since our sample is an ϵ-net, the set difference between the value groups V'_{sample} and V_{opt} contains at most ϵn points. Figure 3.6 shows a sample that is an ϵ-net when $k = 2$. The gray wedge in the figure shows a subdivision bounded by H_{opt} and H'_{sample}.

For the weighted case, we consider each value group $\mathbf{x}$ as a set of $|\mathbf{x}|$ copies of the point to obtain the result that if we take $s(\epsilon)$ samples then the set difference between the value groups V'_{sample} and V_{opt} contains points whose total weight is at most $\epsilon|\mathbf{x}(R)|$.

By definition, $F(\mathbf{x}(V_{sample})) \geq F(\mathbf{x}(V'_{sample}))$, and hence we have a lower bound of $F(\mathbf{x}(V_{sample}))$. For example, we can show that

$$Gini(\mathbf{x}(V_{opt})) - Gini(\mathbf{x}(V_{sample})) \leq 2\epsilon + \alpha\epsilon^2$$

where $\alpha = |R|(|V_{opt}|^{-1} + |\bar{V}_{opt}|^{-1})$. Since we do not want to find a subdivision with a very large α, this is a good approximation if ϵ is small.

We can use the theory of ϵ-approximation for the k-labeled space given by Hasegawa et al. [HII95] to avoid introducing α into the analysis. Theoretically, if we want to make $\epsilon = 0.01$, $s(0.01) = 600k\log(1600k)$, which is very large. However, the theoretical bound is very pessimistic, and a much smaller sample is sufficient, as we will show by experiment later.

3.3.5 Probing Algorithm

Hand Probing for Value Group

The Probing Algorithm also searches for stamp points on $Conv(P)$. Computing a stamp point on $Conv(P)$ and its corresponding value group without knowing the coordinates of the point is called *hand probing* in the field of computational geometry [DEY86]. Geometrically, hand probing in a k-dimensional space means computing a tangential point of a $(k-1)$-dimensional hyperplane and $Conv(P)$.

Any tangential hyperplane in a k-dimensional space has a normal vector $\Theta = (\theta_1, \theta_2, \cdots, \theta_k)$.

By giving a k-dimensional vector Θ, we can compute the tangential point $\mathbf{p}$ of the hyperplane and $Conv(P)$ by maximizing (or minimizing) the inner product $(\Theta, \mathbf{p})$.

Example 3.6 Let us consider once again the Example 3.5. The tangential point of a hyperplane with a normal vector $\Theta = (-0.25, 0.75, -0.25)$ maximizes (or minimizes) the inner product $(\Theta, \mathbf{x}) = -0.25x_1 + 0.75x_2 - 0.25x_3$. We compute the inner product $(\Theta, \mathbf{x})$ for each stamp point of $P_{\text{atom}} = \{c_1, \cdots, c_4\}$.

	c_1	c_2	c_3	c_4
$(\Theta, \mathbf{x})$	0	0	10	-10

The value group corresponding to the hand probing is the union of those terms whose inner product is non-negative (resp. negative). The coordinates of the tangential point (value group) can be obtained by summing up the coordinates of those terms. In this example, $C = c_1 \vee c_2 \vee c_3$ (resp. $C = c_4$) is the value group, and the tangential point is $(60, 50, 50)$ (resp. $(30, 10, 40)$). **[EOE]**

The time complexity to compute a stamp point $\mathbf{p}$ maximizing $(\Theta, \mathbf{p})$ for a given Θ is $O(n)$, where n is the conditional domain size.

Convex Hull Searching

We have shown that stamp points on $Conv(P)$ can be computed efficiently by using hand probing. Now, let us consider how to find the optimal points on $Conv(P)$.

Various convex hull algorithms have been studied intensively [PS85], since many problems, such as optimized numeric association rules [FMMT96d, FMMT96c, FMMT99], two dimensional association rules [FMMT96b, YFM+97] in data mining, classification and regression trees [FMMT96a, MFMT97, MIM97], can be interpreted as convex hull problems. The Probing Algorithm uses an online convex hull maintenance algorithm called the "beneath-beyond" method.

First of all, we compute k different points, which are linearly independent, by using hand probings with k different random vectors. Consequently, we have $2k$ points. Empirically, a vector Θ_{init} satisfying $(\Theta_{init}, \mathbf{x}(R)) = 0$ determines a satisfactory point as an initial stamp point with respect to a convex criterion. The line containing the origin and $\mathbf{x}(R)$ is the central line of the convex hull, and a hyperplane whose normal vector is Θ_{init} is parallel to the line. Therefore, we include such vectors in the initial set of k vectors. Strictly speaking, we may not be able to find k linearly independent points in a k-dimensional space by any hand probing. In this case, we project all points into a $(k-1)$-dimensional space and solve the problem as a $(k-1)$-dimensional one.

We can define an inscribed convex polygon whose vertices are the initial $2k$ points, inside $Conv(P)$. We can also define a circumscribed polygon whose facets are tangential hyperplanes used by hand probings. Figure 3.7 shows examples of inscribed and circumscribed polygons of $Conv(P)$ in two-dimensional space, i.e., $k = 2$.

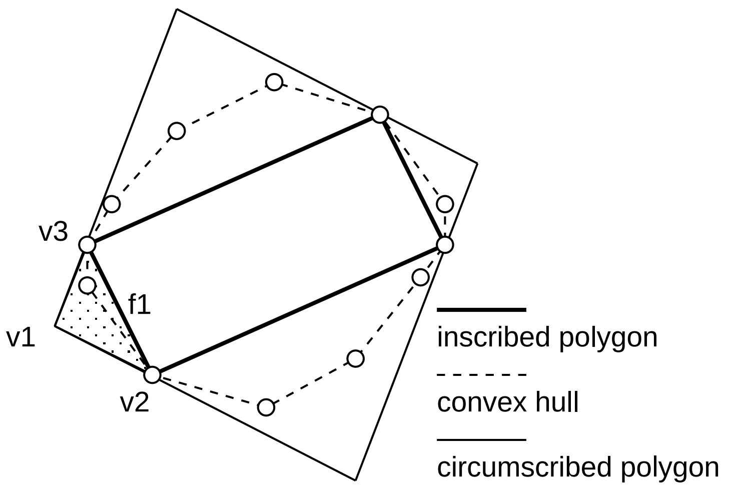


Figure 3.7: Inscribed Polygon and Circumscribed Polygon

Let $I \subseteq P$ be a set of vertices of an inscribed convex polygon, and let C be a set of intersecting points of tangential hyperplanes of $Conv(P)$ that are vertices of a circumscribed convex polygon. Thanks to Theorem 3.1, 3.2, and 3.3, we know stamp points inside $Conv(I)$ are not better than the best vertex of I. If we consider a facet of $Conv(I)$, the stamp points between $Conv(C)$ and the facet are not better than the corresponding vertex of $Conv(C)$ and the vertices of the facet. For example, if we consider the facet f_1 in Figure 3.7, the stamp points inside the gray triangular region can not be better than the best of $v1$, $v2$, and $v3$. Therefore, the value of a vertex of C gives a lower bound of corresponding points on $Conv(P)$ inside $Conv(C)$ and $Conv(I)$.

The guided branch-and-bound search method [FMMT96a, MFMT97, MIM97] recursively refines the inscribed and circumscribed polygons. It efficiently finds the optimal point on $Conv(P)$ by using those lower bounds to order and drop candidate facets. However, there are as many as $(n^{k-1})^{\lfloor \frac{k}{2} \rfloor}$ facets on $Conv(P)$. The method needs to maintain $Conv(I)$ and $Conv(C)$ that may have as many facets as $Conv(P)$ has. Therefore, it is only applicable when k is small enough for the available working space.

In the Probing Algorithm, we predefine the size of the working space that is used to maintain facets of $Conv(I)$ and drop facets so that it works within the limited working space.

The algorithm in Figure 3.8 is the essential part, which contains refinement procedure of facet queue of the inscribed polygon, of the Probing Algorithm.

In each refinement procedure, we can make at least k facets outside of the inscribed convex polygon. Figure 3.9 is an example for a three-dimensional case. If we find a point x by hand probing with the normal vector of the facet $\{1, 2, 3\}$, we add three facets $\{x, 2, 3\}$, $\{1, x, 3\}$, $\{1, 2, x\}$. If k is large, the limited working space may run out after several refinement procedures. We maintain Q as a priority queue. We decide the priority of new facets based on the value of the new point. In the example, the value of x is used for the three new facets.

We empirically find that stamp points for which the value of the objective criterion exceeds the best value tend to lie near the best point or near the points that are closest to the best

```
0)  Algorithm PROBING() {
1)      Compute initial $2k$ points by $k$ vectors.
2)      Construct the initial inscribed polygon $I$.
3)      Initialize a facet queue $Q$ from $I$.
4)      Repeat until $Q$ becomes empty or stopping conditions are satisfied {
5)          REFINE($Q$)
6)      }
7)  }
8)  Function REFINE($Q$) {
9)      Let $f$ be the first facet of $Q$.
10)     Compute the normal vector $\Theta$ of $f$.
11)     Compute a tangent point of $Conv(P)$ by using hand probing with $\Theta$.
12)     If a new point $\mathbf{x}$ is found {
13)         Refine $I := I \cup \mathbf{x}$.
14)         Create new facets by using $\mathbf{x}$.
15)         Replace the facets in $Q$.
16)     }
17) }
```

Figure 3.8: Probing Algorithm

value. The above heuristics help to find a high quality result in an earlier step of the algorithm.

Furthermore, in order to prevent too many excessively small expansions of the convex polygon, we use a distance threshold. If the distance between a facet and a new point is smaller than a given threshold, the new facets that contain the new point are pruned away. This heuristic also helps to speed up the searching.

Though the expected running time of the Probing Algorithm is still substantial, it can find a satisfactory point in a period much shorter than the expected running time, and can return the intermediate value group interactively.

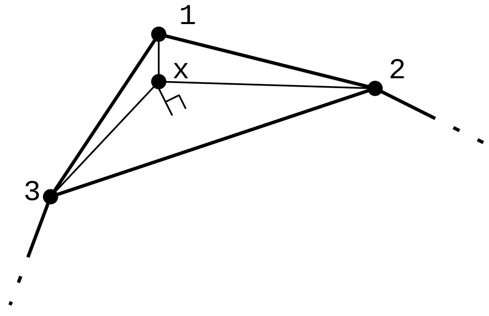


Figure 3.9: Inscribed Polygon Refinement

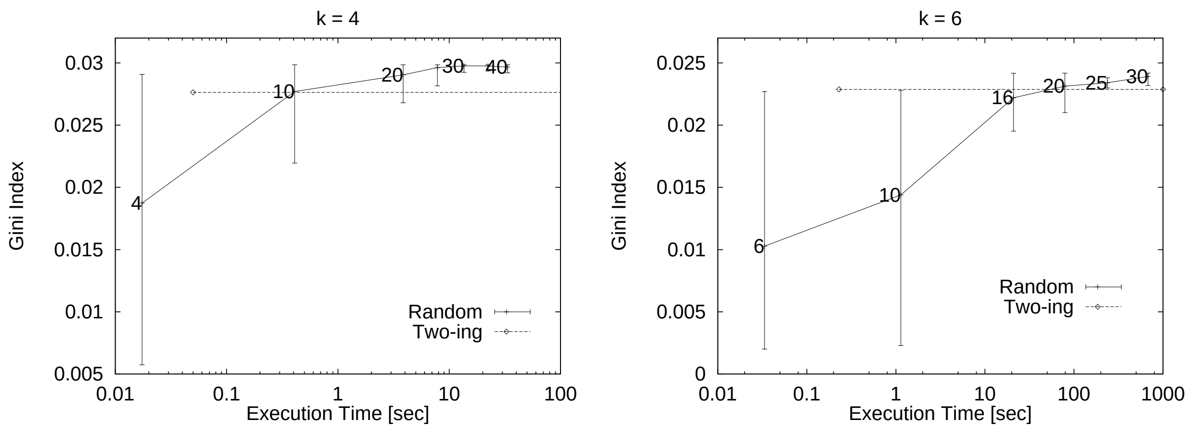


Figure 3.10: Performance of the Random Enumeration Algorithm (1)

3.3.6 Experiments

We implemented the proposed algorithms for rules on categorical conditional attribute and performed several experiments to evaluate their performance. All experiments were performed on an IBM RS/6000 workstation with a POWER2 processor running at 66 MHz with 2 MB of L2 cache and 256 MB of real memory.

In this experiment, we generated synthetic data with $n = 1000$ (conditional domain size) and various values of k (target domain size) to simulate a very large categorical database. Each type of dataset has 10,000 records and two categorical attributes, C and A. The conditional attribute C takes $c_1, \cdots, c_n$ distinct values. The target attribute A takes $a_1, \cdots, a_k$ distinct values. For each record of the synthetic data, we randomly and independently assign a value of C and a value of A. For purposes of comparison, we use the "two-ing" which is used in the CART system to compute a binary segmentation. Note that another well known conventional heuristic used in C4.5 is unsuitable for large n and does not perform well in this condition.

Random Enumeration Algorithm

Figure 3.10 shows the relationship between the execution time for a single run and the improvement in the gini index gained by a value group obtained by the Random Enumeration Algorithm for various sample sizes. The target domain sizes k are 4 and 6. The numbers in the graph represent the sample sizes. Each error-bar (vertical line) indicates the range between the best and worst results of 32 runs for each sample size. The results of the 32 runs are distributed throughout the range, and each point on the range shows the average value of these results.

The "two-ing" method deterministically computes a value group for each problem. Each diamond mark in the figures indicates the time taken to compute the value group and the improvement in the gini index achieved by the value group. We draw a horizontal broken line for each diamond mark so that our algorithms can be easily compared with the "two-ing" method. Though the "two-ing" gives the optimal value group of a certain superclass obtained by grouping k classes, the heuristic is known to find a relatively good approximated value group.

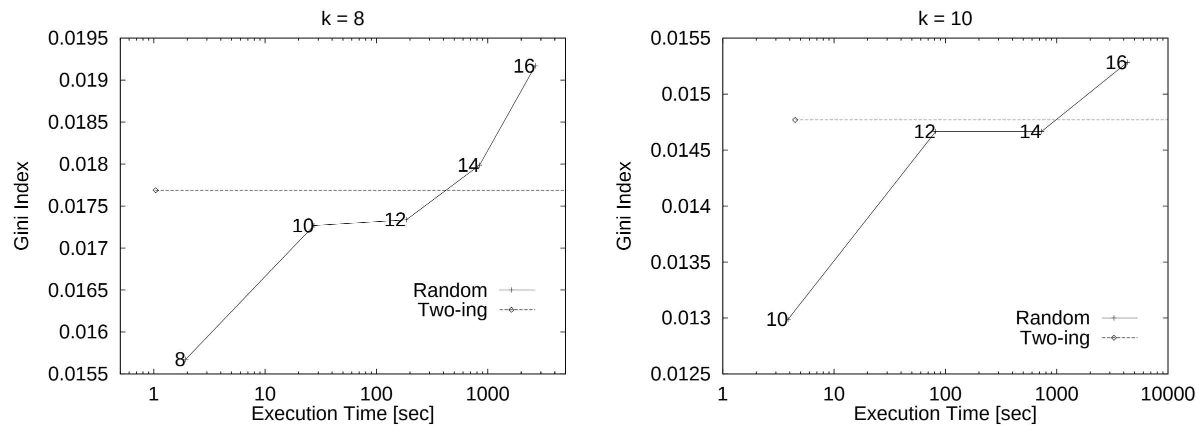


Figure 3.11: Performance of the Enumeration Random Algorithm (2)

From this experiment, we can see that the Random Enumeration Algorithm generates a result of satisfactory quality within a practical time when the sample size is around 20. Note that the best result of the 32 runs is better than the result of the "two-ing" method, even if we use a small sample size. The CPU time taken for a single run is almost proportional to $\begin{pmatrix} s \\ k-1 \end{pmatrix}$.

When k becomes larger, we have to use a small sample size s so that the algorithm can terminate in a practical amount of time. However, a small sample often gives low-quality results. To overcome this problem, we run this algorithm with a small sample a number of times, and take the best result. Figure 3.11 shows the relationship between the execution time for 32 runs and the best improvement in the gini index of the 32 runs when $k = 8$ and 10.

The Random Enumeration Algorithm achieved better or comparable results with a small sample in 32 runs. Those multiple trials can be executed independently. Therefore, we can expect better results if we can use a parallel environment.

Probing Algorithm

Figure 3.12 shows the extent to which the gini index value is improved by the Probing Algorithm, along with the time taken in seconds. Thanks to the heuristics for maintaining the facet queue, we can observe that when the Probing Algorithm found a better result, it often found a still better result within a short period of time. As a result, the Probing Algorithm tends to find a satisfactory value group relatively quickly. Such a value group, which is much better than the result given by the "two-ing" method, is satisfactory for most applications. Thus, the Probing Algorithm returns an intermediate result in a practical time.

One defect of the Probing Algorithm is its required working space when k becomes large. However, it can usually find a satisfactory result before its working space becomes too large. In the experiment for $k = 10$, the working space of the Probing Algorithm is less than 64 MB, which is acceptable on most workstations, when the best result in the graph was obtained. In these experiments, we limited the working space of the Probing Algorithm to 130 MB.

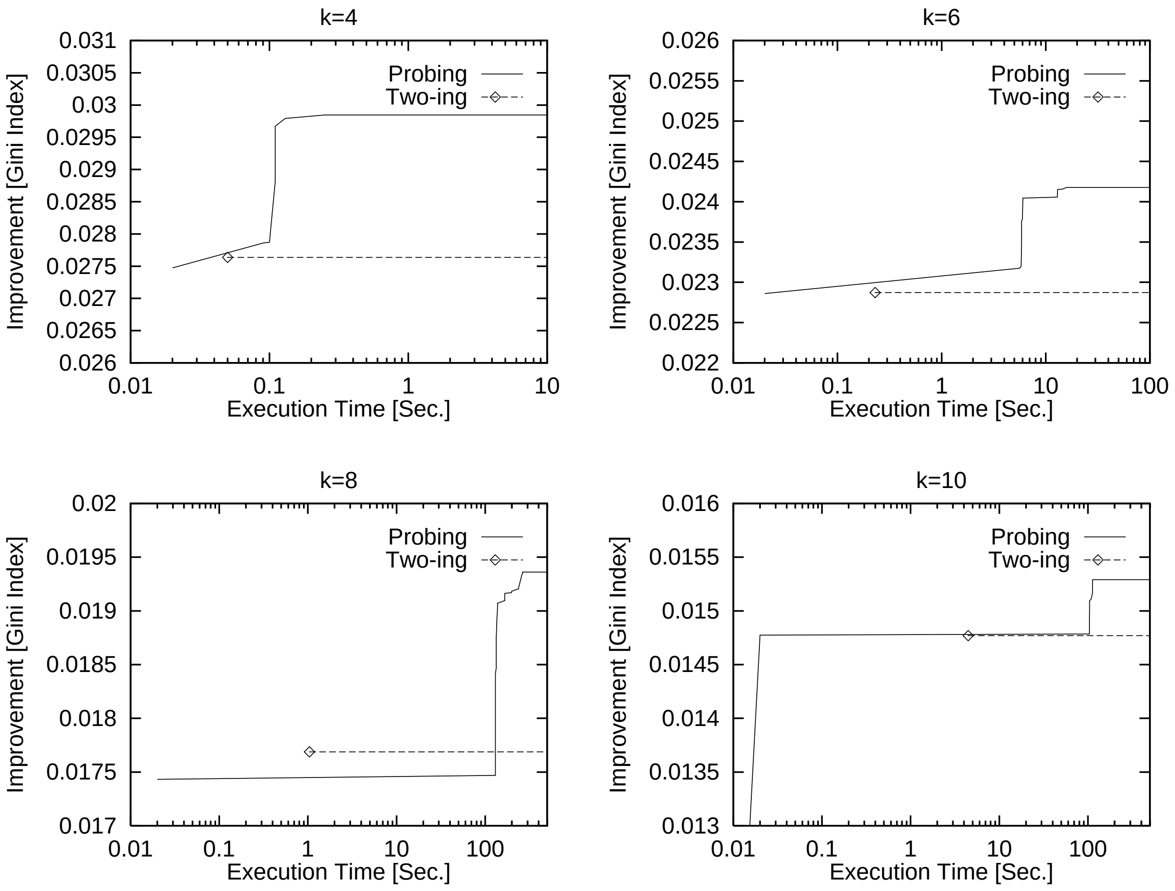


Figure 3.12: Performance of the Probing Algorithm

These experiments used synthetic databases whose values are randomly assigned. We have tried similar experiments by using other synthetic and real databases whose values are skewed. We observed that the performance of our algorithms is basically the same for such skewed cases.

The results of the experiments show that both the Random Enumeration Algorithm and the Probing Algorithm found a better value group than the "two-ing" method in a practical time, and that both are feasible.

Though the complexities are $O(s^{k-2}n)$ for the Random Enumeration Algorithm and $O((n+m)|P|)$ for the Probing Algorithm, we derived practical implementations by using randomization and strategic facet maintenance. Above various experiments confirmed that the algorithms could find satisfactory value groups within reasonable computation times. The quality of the results obtained for various samples by the Random Enumeration Algorithm differs dramatically if we use a small sample size. Therefore, multiple trials are needed to obtain better value groups. Those trials can be executed independently and in parallel; thus, the Random Enumeration Algorithm is suitable for a parallel environment. On the other hand, the quality of the Probing Algorithm becomes stable after a certain amount of execution time. However, it requires a large working space compared to the Random Enumeration Algorithm. Since the available memory size is still increasing dramatically, the practical applicability of the Probing Algorithm seems to be promising.

3.4 Optimal Rule on Numerical Attribute

For a numerical conditional attribute that has M buckets (and n values), there are $O(M^2)$ (or $O(n^2)$) possible rules or stamp points, if we use a range as a rule. Though the time complexity is much smaller compared to the case for a categorical conditional attribute, it is sometimes too expensive for large n.

Note that many classification and regression systems use a sole threshold value in the numerical domain instead of range and split records into two segments based on the threshold value. Such splitting method is called "guillotine-cut" splitting. And there are $O(M)$ (or $O(n)$) possible rules or stamp points for the guillotine-cut rules. The guillotine-cut splitting can be a special case of range splitting where one of the lower or the higher threshold is $-\infty$ or ∞, respectively. Therefore, for a numerical conditional attribute, we analyze the problem as range splitting.

When it comes to handling numerical attributes, conventional methods are inefficient if any numerical attributes are strongly correlated. The author offers one solution to this problem. For each pair of numerical conditional attributes with strong correlation, we compute a two dimensional association rule with respect to these attributes and the target attribute.

In particular, we consider a family $\mathcal{R}$ of grid-regions in the plane associated with the pair of numerical attributes. For $R \in \mathcal{R}$, the data can be split into two segments: data inside R and data outside R. We compute the region $R_{opt} \in \mathcal{R}$ that optimizes one of the criteria for a classification or a regression problem. We give efficient algorithms for cases in which $\mathcal{R}$ is x-monotone connected regions, rectilinear convex regions, and rectangles.

3.4.1 Range Rules

In this section, we consider classification rules on a numerical conditional attribute C and a target categorical attribute A whose target domain size is k. As mentioned above, if C has n values, there are $O(n^2)$ possible (range) rules or stamp points. Though we often divide the domain of C into M equi-sized buckets for large n, we consider the problem of n values without loss of generality.

In this dissertation, we consider the optimization problem in computational geometry context. Therefore, we discuss efficient algorithm for finding the optimal stamp point from the set of the $O(n^2)$ stamp points.

Hand Probing for Range

As proved in Section 3.2.3, all of criteria mentioned for classification problems are convex in the k-dimensional space. Therefore, we can concentrate on stamp points on the convex hull, $Conv(P)$, of the set of all $O(n^2)$ points, P.

We can compute a stamp point on $Conv(P)$ and its corresponding range without knowing the coordinates of the point by hand probing algorithm for range. Geometrically, hand probing in a k-dimensional space means computing a tangential point of a $(k-1)$-dimensional hyperplane and $Conv(P)$, as same as the cases for value groups in Section 3.3.5.

Any tangential hyperplane in a k-dimensional space has a normal vector $\Theta = (\theta_1, \theta_2, \cdots, \theta_k)$. By giving a k-dimensional vector Θ, we can compute the tangential point $\mathbf{p}$ of the hyperplane and $Conv(P)$ by maximizing (or minimizing) the inner product $(\Theta, \mathbf{p})$.

Assume there are n buckets B_i for $i = 1, \cdots, n$ on the domain of C. We precompute stamp points of each buckets, $\mathbf{p}(B_i)$. Then, for a vector Θ, we compute $\sum_{j=0}^{i}(\Theta, \mathbf{p}(B_i))$ for all $i = 0, \cdots, n$. During the computation, we maintain two indexes, min and max that minimizes and maximizes $\sum_{j=0}^{i}(\Theta, \mathbf{p}(B_i))$, respectively. The tangential point that minimizes (or maximizes) the inner product $(\Theta, \mathbf{p})$ is the range $[B_{max+1}, B_{min}]$ (resp. $[B_{min+1}, B_{max}]$) when $\sum_{j=0}^{i}(\Theta, \mathbf{p}(B_i))$ is minimized (resp. maximized).

Example 3.7 The tangential point of a hyperplane with a normal vector $\Theta = (-0.25, 0.75, -0.25)$ maximizes (or minimizes) the inner product $(\Theta, \mathbf{x}) = -0.25x_1 + 0.75x_2 - 0.25x_3$. We compute $\sum_{j=0}^{i}(\Theta, \mathbf{p}(B_i))$ for each $i = 0, \cdots, n$. Assume we have 10 buckets as follows:

i	1	2	3	4	5	6	7	8	9	10
a_1	4	0	8	4	2	8	12	16	0	10
a_2	8	4	4	12	0	0	8	0	10	0
a_3	0	4	12	12	0	4	0	12	8	0
$\sum_{j=0}^{j}(\Theta, \mathbf{B_j})$	5	7	5	**10**	9.5	6.5	5.5	**-1.5**	4	1.5
max	1	2	2	**4**	4	4	4	**4**	4	4
min	0	0	0	**0**	0	0	0	**8**	8	8

The range corresponding to the hand probing is $[B_5, B_8]$ and $[B_1, B_4]$. **[EOE]**

The time complexity to compute a stamp point $\mathbf{p}$ minimizing or maximizing $(\Theta, \mathbf{p})$ for a given Θ is $O(n)$, where n is the conditional domain size.

Probing Algorithm for Range

Thanks to Theorem 3.1, 3.2, and 3.3, we can apply the PROBING algorithm in Section 3.3.5 for searching the optimal stamp point. If k is small, we can prune unnecessary stamp points on *conv*(P) effectively and hence the expected running time for the PROBING algorithm find the optimal range can be $O(n \log n)$. However, if k is large, examining all $O(n^2)$ stamp points is faster than the PROBING algorithm.

3.4.2 Region Rules

There are many correlated numerical attributes in a database. This is one of the important reasons why multivariate analysis in the field of statistics is widely used for analyzing databases that contain several numerical attributes.

In the statistics literature, multivariate analysis has been used to handle correlated data. "Principal component analysis," "factor analysis," and so forth, is categorized as multivariate analysis. Most of the methods in the multivariate analysis assume a linear correlation. Such conventional techniques are effective for data that have linear correlations. However, databases contain various types of correlations that cannot be handled by the conventional methods.

3.4.3 Related Works for Handling Correlations

One approach is as follows: consider each pair of numeric attributes as a two-dimensional attribute. Then, for each such two-dimensional attribute, compute a line partition of the corresponding two dimensional space so that the corresponding objective function is maximized. One (minor) defect of this method is that it is not cheap to compute the optimal line. Although some works have been done on this problem in computational geometry [AT94a, DE93], the worst time complexity remains $O(n^2)$ if there are n records. Another (major) defect is that the decision tree may still be too large even if we use line partition. Although some multivariate decision tree classifiers [BFOS84, BU95] can find multivariate tests of the form $\sum_i a_i B_i < c$, where a_i and c are constants, in more practical ways, the latter problem, which is inherent in linear partitioning methods, still remains.

To handle non-linear correlations, Ittner and Schlosser [IS96] considers composite attributes that include all possible pairwise products and squares of numerical attributes, and finds a linear partitioning on those composite attributes. This approach is an alternative to our use of regions.

3.4.4 Optimized Two-Dimensional Region

In this dissertation, the author proposes the following scheme for the above problem, applying the two-dimensional association rules, region rules in short, of [FMMT96b, FMMT01] and an image segmentation algorithm of [ACKT96]. The scheme has been implemented as a subsystem of SONAR, which stands for System for Optimized Numeric Association Rules, developed by the authors [FMMT96e].

Let n be the number of records in the database. First, for each numeric attribute, we create an equi-depth bucketing so that records are uniformly distributed into $N \leq \sqrt{n}$ ordered buckets according to the values of the attribute.

Next, we find all pairs of strongly correlated numeric attributes. For each such a pair A and A', we create an $N \times N$ pixel grid G according to the Cartesian product of the buckets of each numeric attribute. The problem of finding the optimal arbitrary connected pixel grid region, which optimizes one of classification criteria mentioned in Section 3.2.1, is NP-hard. Therefore, we consider a family $\mathcal{R}$ of grid regions; in particular, we consider the set $\mathcal{R}(xmono)$ of all *x-monotone* connected regions, the set $\mathcal{R}(recti)$ of all *rectilinear convex* regions, and the set $\mathcal{R}(recta)$ of all *rectangular regions*. (Examples of these regions are shown in Figure 1.5 in Section 1.2.1.)

An *x-monotone* region is a grid region whose intersection with any vertical line is undivided. For example, Figure 1.5 (a) shows a region that is not x-monotone, since the intersection of the vertical line `A` and the region is divided. In contrast, Figure 1.5 (b) shows an x-monotone region. We can express an x-monotone region by a disjunction of expressions, where each disjunct has the form ($a1 \leq x \leq a2$) `and` ($b1 \leq y \leq b2$), where the union of disjuncts does not divide a vertical column. A *rectilinear* convex region is an x-monotone region such that its intersection with any horizontal line is also undivided. Figure 1.5 (c) shows an example of a rectilinear convex region. A *rectangular* region is a rectangle on G, and is thus a rectilinear convex region. We will consider the class of x-monotone regions, the class of rectilinear convex regions, and the class of rectangular regions.

Assume that we focus on two numerical attribute B and C. We distribute the values of B (resp. C) into N_B (N_C) buckets so that every bucket contains almost the same number of records. We then divide the Euclidean plane associated with B and C into $N_B \times N_C$ pixels (unit squares). For simplicity, we assume that $N_B = N_C = N$ without loss of generality as regards our algorithms.

Regarding the pair of attributes as a two-dimensional attribute, we compute the region R_{opt}

in $\mathcal{R}$ and consider the discriminant rule $(t[A], t[A']) \in R_{opt}$. The author presents algorithms for computing R_{opt} in worst case times of $O(nN^2)$, $O(nN^3)$, and $O(nN^3)$ for $\mathcal{R}(xmono)$, $\mathcal{R}(recti)$, and $\mathcal{R}(recta)$ respectively. Moreover, in practical instances, our algorithms run in $O(N^2 \log n)$ time, $O(N^3 \log n)$ time, and $O(N^3 \log n)$ time. Since $N \leq \sqrt{n}$, those time complexities are $O(n \log n)$, $O(n^{1.5} \log n)$, and $O(n^{1.5} \log n)$, respectively.

3.4.5 Hand Probing for Region

If a region rule divides the database relation R into two segments S and $\bar{S}$. The rule can be characterized by a stamp point of the segment S, $\mathbf{x}(S) = (x_1(S), x_2(S), \cdots, x_k(S))$, as same as the cases of value groups or ranges.

There are $O(N^{2N})$ possible x-monotone or rectilinear regions on an $N \times N$ pixel grid and it is not affordable to examine all of the possible regions. As proved in Section 3.2.3, all criteria for classification problems are convex. Therefore, we can concentrate on stamp points on the convex hull, $Conv(P)$, of the set of all $O(N^{2N})$ points, P. We use the hand probing algorithm for the optimal region for find a stamp point on the convex hull.

A point of the convex hull and its corresponding region can be computed efficiently by using “hand probing” technique, invented by Asano et al. [ACKT96] for image segmentation and modified by Fukuda et al. [FMMT96b] for extraction of the two-dimensional optimized association rules.

Any tangential hyperplane in a k-dimensional space has a normal vector $\Theta = (\theta_1, \theta_2, \cdots, \theta_k)$. By giving a k-dimensional vector Θ, we can compute the tangential point $\mathbf{p}$ of the hyperplane and $Conv(P)$ by maximizing (or minimizing) the inner product $(\Theta, \mathbf{p})$.

Definition 3.7 Assume two numerical conditional attributes C_1 and C_2 that are divided into N_1 and N_2 buckets, respectively. Let us consider a two-dimensional $N_1 \times N_2$ pixel-grid G, consisting of $N_1 \times N_2$ unit squares called *pixels*. $G(r,c)$ is the (r,c)-th pixel, where r and c are called the *row number* and *column number*, respectively. The c-th *column* $G(*,c)$ of G is its subset consisting of all pixels whose column numbers are c. Geometrically, a column is a vertical stripe. We use the notation $n = N_1 \times N_2$. For a set of pixels, the union of pixels in it forms a planar region, which we call a *pixel region*. **[EOD]**

For each record t, $t[C_1]$ and $t[C_2]$ are values of the numeric attributes C_1 and C_2 at t. If $t[C_1]$ is in the r-th bucket and $t[C_2]$ is in the c-th bucket in the respective bucketings, we define $f(t) = G(r,c)$. Then, we have a mapping f from the set of all records to the grid G.

Let $x_i(r,c)$ $(i = 1, \cdots, k)$ denote the number of records whose value of the target attribute is a_i and are mapped to $G(r,c)$. Given a region X, we define

$$\mathbf{x}(X) = (\sum_{G(r,c) \in X} x_1(r,c), \cdots, \sum_{G(r,c) \in X} x_k(r,c)).$$

Given a vector $\Theta = (\theta_1, \cdots, \theta_k)$, we define

$$g_\Theta(r,c) \quad = \quad \theta_1 \times x_1(r,c) + \cdots + \theta_k \times x_k(r,c)$$

$$Gain(X) \quad = \sum_{G(r,c)\in X} g_\Theta(r,c).$$

We want to compute a region X that maximizes $Gain(X)$ for a vector Θ.

Hand Probing for X-monotone Region

For each $m = 1, 2, \ldots, N$, we pre-compute the indices $bottom_m(s)$ and $top_m(s)$ for all $1 \le s \le N$, where $bottom_m(s)$ and $top_m(s)$ are defined so that

$$\sum_{i=bottom_m(s)}^{s} g_\Theta(i,m) \quad \text{and} \quad \sum_{i=s}^{top_m(s)} g_\Theta(i,m)$$

are maximized, respectively.

Theorem 3.7 The indices $top_m(s)$ and $bottom_m(s)$ for all $s = 1, 2, \ldots, N$ can be computed in $O(N)$ time. **[EOT]**

Proof: Define $Sum_m[< j] = \sum_{i=1}^{j-1} g_\Theta(i,m)$ for $2 \le j \le N$, and $Sum_m[< 1] = 0$. $bottom_m(s)$ maximizes

$$\sum_{i=bottom_m(s)}^{s} g_\Theta(i,m) = Sum_m[< (s+1)] - Sum_m[< bottom_m(s)],$$

which is equivalent to the property that $bottom_m(s)$ minimizes $Sum_m[< bottom_m(s)]$. Define $bottom_m(1) = 1$. For each $s = 2, \ldots, N$, compute $bottom_m(s)$ from $bottom_m(s-1)$ in the following manner:

- If $Sum_m[< s] < Sum_m[< bottom_m(s-1)]$, $Sum_m[< bottom_m(s)]$ is the minimum when $bottom_m(s) = s$.
- Otherwise, $Sum_m[< bottom_m(s)]$ is the minimum when $bottom_m(s) = bottom_m(s-1)$.

During the above step, we only need to scan $g_\Theta(i,m)$ once for each $i = 1, \ldots, N$.

The computation of $top_m(s)$ is analogous. **[EOP]**

For two indices s and s', we define $cover_m(s, s')$ as follows:

$$cover_m(s,s') = \begin{cases} \sum_{i=bottom_m(s)}^{top_m(s')} g_\Theta(i,m) & \text{if } s \le s' \\ \sum_{i=bottom_m(s')}^{top_m(s)} g_\Theta(i,m) & \text{if } s > s' \end{cases}$$

Let $G(*, \le m)$ be the part of the grid on the left of the m-th column, including $G(*, m)$. We define $F(i, \le m)$ to be the corresponding (intermediate) x-monotone region X' with a vector Θ that maximizes $Gain(X')$ and contains the pixel $G(i,m)$ and are contained in the region $G(*, \le m)$. Then, we have the following formula:

$$F(i, \le m+1) = \max_{1\le j\le N} \{\max(F(j, \le m), 0) + cover_{m+1}(i,j)\}. \tag{3.1}$$

When $F(j, \le m)$ is negative, we do not connect the m-th and the $(m+1)$-th columns. Figure 3.13 illustrates this formula and the corresponding region. By using Formula 3.1, we can compute $\max_m\{\max_i F(i, \le m)\}$ and the corresponding region X with a vector Θ, which maximizes $Gain(X)$.

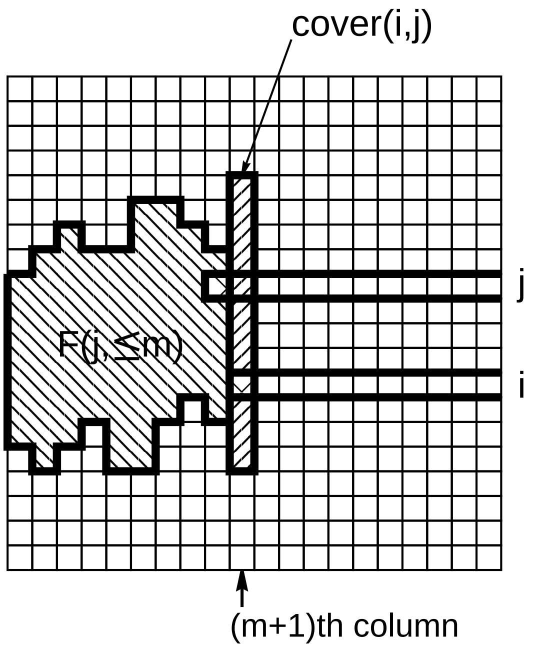


Figure 3.13: $F(i, \leq m+1)$ and the Corresponding Region

Lemma 3.1 Suppose that $F(j, \leq m)$ is given for $j = 1, 2, \cdots, N$. We can compute $F(i, \leq m{+}1)$ for all $i = 1, 2, \cdots, N$ in $O(N)$ time. **[EOL]**

Proof: Define $D(i,j) = F(j, \leq m){+}cover_m(i,j)$ and assume an $N{\times}N$ matrix D whose elements are $D(i,j)$. We can see that the upper and lower triangle parts (D^+ and D^-, respectively) of the matrix D satisfies the following equation

$$D(i_1, j_1) + D(i_2, j_2) = D(i_2, j_1) + D(i_1, j_2)$$

for every $i_1 \leq i_2$ and $j_1 \leq j_2$.

Thanks to the property, in D^- where $j_2 \leq i_1$, if the k-th column is the maximum in the m-th row, the maximum column of the $(m+1)$-th row must be either the k-th or the $(m+1)$-th column. Note that since $D(m,k){+}D(m{+}1,k') = D(m{+}1,k){+}D(m,k')$ and $D(m,k) \geq D(m,k')$, $D(m+1,k') \leq D(m+1,k)$ for all $k' \leq m$.

Thus, in $O(N)$ time, we can compute all the row maxima of D^+ and D^-, and consequently, of D. By definition, $F(i, \leq m+1)$ is the i-th row maximum of D. **[EOP]**

Theorem 3.8 *The corresponding x-monotone region with a vector* Θ *can be computed in* $O(N^2) = O(n)$ *time.* **[EOT]**

Proof: We solve Formula 3.1 for $m = 1, 2, \ldots, N$. This requires $O(N^2) = O(n)$ time. **[EOP]**

Hand Probing for Rectilinear Region

For x-monotone regions, we can compute a stamp point and its corresponding region on $Conv(P)$ from a set of points (x-monotone regions) P by using hand probing. However, one defect was pointed out by several people who have observed our output x-monotone region rules is that although the output region usually gives an intuitive idea on the association between attributes, it sometimes happens that the region is wildly notched and that it is difficult to speculate on the meaning of the rule. Since speculation through visualization is very important for users who require decision support knowledge, this is a serious problem. Moreover, if we use a very fine bucketing, the shape of the region tends to be very sensitive to the sampling if we use a sample subset of the database to construct the region [YFM$^+$97]. This dependency on the sample should be avoided, since we want to make the rule applicable not only to data in the database but also to unknown data in general; however, it is often tedious to tune the size of bucketings.

In order to resolve these defects, one idea is to design a tool to smooth the shape of grid regions, and apply the tool to the x-monotone region output by our algorithm. However, this may result in a loss of optimality, and we would like to avoid this kind of heuristic as far as possible. Our solution keeps the optimality formulations but replaces the family of x-monotone regions by the family of *rectilinear convex* regions. A region is rectilinear convex if it is both x-monotone and y-monotone. We show below that the corresponding region with a vector Θ in this family can be computed in $O(N^3) = O(n^{1.5})$ time.

This type of regions gives a more intuitive region and also it is stable for the data sampling even if we use a fine bucketing (reported in [YFM$^+$97]). As a trade-off, it is more expensive to compute the rectilinear convex regions than x-monotone regions; therefore, we should provide both functions so that the user can flexibly choose whichever is better suited to the application and data.

Let X be a rectilinear convex region. Let m_1 and m_2 respectively denote the indices of the first and last columns. Let $s(i)$ and $t(i)$ denote the indices of the bottom and the top pixels of the i-th column. Since X is a rectilinear convex region, the sequence $(t(m_1), t(m_1 + 1), \cdots, t(m_2 - 1), t(m_2))$ of indices of top pixels from left to right increases monotonically up to some column and then decreases monotonically (possibly one of these two monotone parts is empty). Similarly, the sequence of indices of bottom pixels decreases monotonically up to some column and then increases monotonically.

Therefore, we can cut a rectilinear convex region vertically into at most three *monotone* pieces each of which is among the following four types:

- A region that gets wider from left to right: This type of region is named W.
- Regions that slant upward and downward: These types of regions are named U and D, respectively.
- A region that gets narrower from left to right: This type of region is named N.

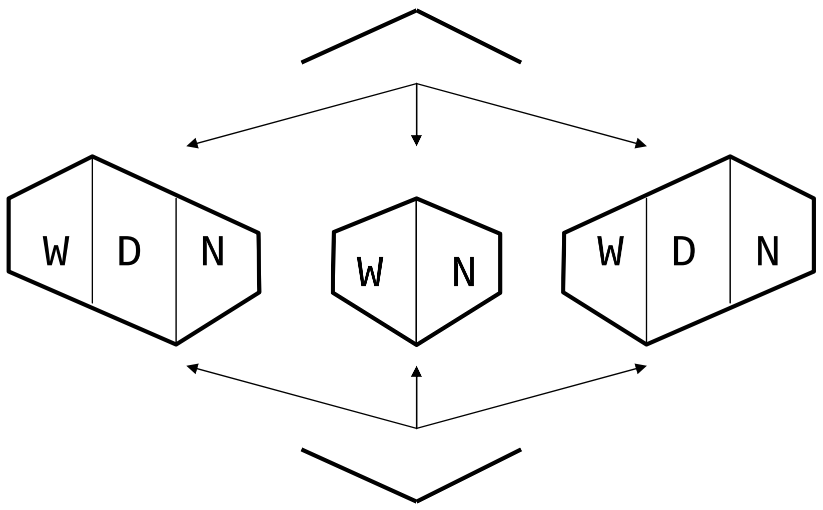


Figure 3.14: Partition of Rectilinear Region into Monotone Parts

Theorem 3.9 *The corresponding rectilinear convex region X with a vector Θ that maximizes $Gain(X)$ can be computed in $O(N^3)$ time.* **[EOT]**

Proof: There are some possible combinations as shown in Figure 3.14, but we consider only the WUN-type region that starts with a W-type part, continues to a U-type part, and ends with an N-type part, since other cases can be handled similarly. However, we also use W and WU types to explain our algorithm.

Our algorithm is based on the dynamic programming paradigm. Before running the dynamic programming program, we pre-compute $\sum_{j\in[s,t]} g_\Theta(m,j)$, which will be denoted by $g_\Theta(m,[s,t])$, for $m = 1,\cdots,N$ and $1 \leq s \leq t \leq N$. This computation takes $O(N^3)$ time.

Now, let $R_W(m,[s,t])$ (resp. $R_{WU}(m,[s,t])$, $R_{WUN}(m,[s,t])$) be the (intermediate) corresponding rectilinear convex regions with Θ of W-type (resp. WU-type, WUN-type) rectilinear convex regions whose last column is the m-th one, and their intersections with the m-th column range from the s-th pixel to the t-th pixel. Let $f_W(m,[s,t])$, $f_{WU}(m,[s,t])$ and $f_{WUN}(m,[s,t])$ be their inner product values, i.e., $(\mathbf{x}(R_W(m,[s,t])),\Theta)$, $(\mathbf{x}(R_{WU}(m,[s,t])),\Theta)$, and $(\mathbf{x}(R_{WUN}(m,[s,t])),\Theta)$, respectively.

First, we consider W-type regions. For $m = 1$, $f_W(1,[s,t]) = g_\Theta(1,[s,t])$. For $m > 1$, if $s = t$, $f_W(m,[s,s]) = \max\{g_\Theta(m,s), f_W(m-1,[s,s]) + g_\Theta(m,s)\}$. Consider the case $s < t$. Since the region is of type W, the $(m-1)$-th column of $R_W(m,[s,t])$ must be a subinterval $[s,t]$. If it is a subinterval of $[s,t-1]$ (resp. $[s+1,t]$), we can observe that the region must contain $R_W(m,[s,t-1])$ (resp. $R_W(m,[s+1,t])$. Hence, if $R_W(m,[s,t])$ contains neither $R_W(m,[s+1,t])$ nor $R_W(m,[s,t-1])$, its $(m-1)$-th column must be the interval $[s,t]$.

Hence, the following recurrence holds:

$$f_W(m,[s,t]) = \max\begin{pmatrix} f_W(m-1,[s,t]) + g_\Theta(m,[s,t]) \\ f_W(m,[s+1,t]) + g_\Theta(m,s) \\ f_W(m,[s,t-1]) + g_\Theta(m,t) \end{pmatrix}$$

On the basis of this recursion formula, we can compute $f_W(m, I)$ for all m and I in $O(N^3)$ time.

Next, consider $f_{WU}(m, [s, t])$. For $m = 1$, $f_{WU}(1, [s, t]) = g_\Theta(1, [s, t])$. For $m > 1$, we pre-compute $\max_{i \le s} f_W(m-1, [i, t])$ and $\max_{i \le s} f_{WU}(m-1, [i, t])$ for all $s \le t$ in $O(N^2)$ time. Since the region is of type U, the $(m-1)$-th column of $R_{WU}(m, [s, t])$ should be an interval $[i, j]$ for $i \le s$ and $j \le t$. If $j \ne t$, $j \le t-1$ holds, and we can observe that the region must contain $R_W(m, [s, t-1])$. Hence, we have the following recurrence, which leads us to an $O(N^3)$-time dynamic programming algorithm:

$$f_{WU}(m, [s, t]) = \max \begin{pmatrix} \max_{i \le s} f_W(m-1, [i, t]) + g_\Theta(m, [s, t]) \\ \max_{i \le s} f_{WU}(m-1, [i, t]) + g_\Theta(m, [s, t]) \\ f_{WU}(m, [s, t-1]) + g_\Theta(m, t) \end{pmatrix}$$

Finally, consider WUN type. For $m = 1$, $f_{WUN}(1, [s, t]) = g_\Theta(1, [s, t])$. For $m > 1$, we have the following recurrence (a mirror formula to that for the W-type):

$$f_{WUN}(m, [s, t]) = \max \begin{pmatrix} f_{WU}(m-1, [s, t]) + g_\Theta(m, [s, t]) \\ f_{WUN}(m-1, [s, t]) + g_\Theta(m, [s, t]) \\ f_{WUN}(m, [s-1, t]) - g_\Theta(m, s-1) \\ f_{WUN}(m, [s, t+1]) - g_\Theta(m, t+1) \end{pmatrix}$$

In this case, we need to compute $f_{WUN}(m, [s, t])$ by using $f_{WUN}(m, [s-1, t])$ and $f_{WUN}(m, [s, t+1])$. Thus, we first compute $f_{WUN}(m, [1, N]) = \max\{f_{WUN}(m-1, [1, N]) + g_\Theta(m, [1, N]), g_\Theta(m, [1, N])\}$, and run the dynamic programming to compute the values of $f_{WUN}(m, I)$ from larger intervals I to smaller ones.

Consequently these recursion formulas provide a dynamic programming method of obtaining an $O(N^3)$-time solution for computing $f_{WUN}(m, [s, t])$ for all m and $s \le t$. **[EOP]**

Hand Probing for Rectangular Region

As for rectangular regions, there are $O(N^2)$ ranges on one of the two axes. For each $O(N^2)$ range, we can apply range optimization technique, which spends $O(N)$ time, given in Section 2.2. Therefore, the time complexity of computing a rectangular region is also $O(N^3)$.

3.4.6 Search on Convex Hull

To search for the stamp point associated with the region R that optimizes one of criteria for classification problems, Theorem 3.1, 3.2, and 3.3 imply that the optimal stamp point must be on the convex hull of all the stamp points. By using the hand probing technique repeatedly, we can scan all the points on the hull and find the optimal point. However, when the number of records is n, the number of points on the hull is at most n. Therefore, in the worst case, we may need to try hand probing n times in order to find the optimal stamp point, which is costly in practice. To avoid searching for unnecessary points, we use a guided branch and bound strategy. If we rewrite the hand probing procedure into the one of the three region families, we can use probing algorithm mentioned in Section 3.3.5.

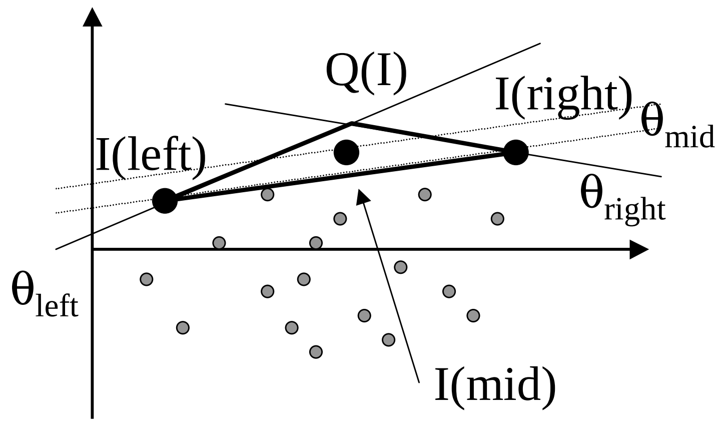


Figure 3.15: Guided Branch-and-Bound Search

Example 3.8 Let us consider the searching algorithm for a two dimensional, i.e., $k = 2$ problem. While searching for the optimal region, we maintain the current maximal value V_{max} corresponding to the points examined so far. Suppose we have examined two tangent points, say $I(left)$ and $I(right)$, and consider the interval $I = [I(left), I(right)]$ in Figure 3.15. Let $Q(I) = (x_{Q(I)}, y_{Q(I)})$ be the point of intersection of the two tangent lines that are used to compute $I(left)$ and $I(right)$. We can compute the value of an objective function, say "f," of $Q(I)$, $f(Q(I)) = f(x_{Q(I)}, y_{Q(I)})$.

Lemma 3.2 *For any point $Q' = (x', y')$ inside the triangle $I(left)$, $I(right)$, and $Q(I)$,*

$$f(x', y') \leq max\{f(x_{Q(I)}, y_{Q(I)}), V_{max}\}$$

[EOL]

Proof: Immediate from Theorem 3.1, 3.2, and 3.3. **[EOP]**

This lemma gives an upper bound of points on the convex hull between these two points. Hence, we can find the optimal region effectively by hand probing together with a branch and bound strategy guided by the values $f(Q(I))$. We examine the subinterval with the maximum value of $f(Q(I))$ first. In addition, subintervals whose $f(Q(I))$ is less than V_{max} are pruned away. During the process, V_{max} is monotonically increases while each $f(Q(I))$ is monotonically decreases. Most of the subintervals are expected to be pruned away during the computation, and therefore the number of hand probings is expected to be $O(\log n)$, where n is the number of points on the convex hull. Algorithm in Figure 3.16 sketches the branch and bound search.

Recall that the time complexities of performing hand probing for an x-monotone region, a rectilinear convex region, and a rectangular region are $O(N^2)$, $O(N^3)$, and $O(N^3)$, respectively. Thus, we can experimentally compute the region minimizing the mean squared error in time proportional to $O(N^2 \log n)$, $O(N^3 \log n)$, and $O(N^3 \log n)$ for x-monotone regions, rectilinear convex regions, and rectangular regions, respectively. **[EOE]**

```
0)  Algorithm PROBE2D() {
1)      Compute the initial interval $I_{init} = [I_{init}(left), I_{init}(right)]$ using $\Theta = (\infty, 1)$.
2)      $f(Q(I_{init})) = \infty$
3)      $V_{max} = max\{f(I_{init}(left)), f(I_{init}(right))\}$
4)      Insert $I_{init}$ into $S$.
5)      Initialize the set of sorted candidate intervals $S$.
6)      Repeat until $S$ becomes empty or stopping conditions are satisfied {
7)          REFINE2D($S$)
8)      }
9)  }
10) Function REFINE2D($S$) {
11)     Let $I$ be the interval that has the maximal $f(Q(I))$ from $S$.
12)     IF $V_{max} < f(Q(I))$ {
13)         Let $\Theta_{mid}$ be the slope of the line containing $I(left)$ and $I(right)$.
14)         Compute a new point $I(mid)$ using $\Theta_{mid}$.
15)         If $V_{max} > f(I(mid))$ then $V_{max} = f(I(mid))$.
16)         Divide $I$ into $I_{left} = [I(left), I(mid)]$ and $I_{right} = [I(mid), I(right)]$.
17)         Compute $f(Q(I_{left}))$ and $f(Q(I_{right}))$.
18)         Insert $I_{left}$ and $I_{right}$ into $S$.
19)     }
20) }
```

Figure 3.16: Branch and Bound Search for Two Classes Problems

Table 3.6: Time for Computing the Optimal Region in Classification (1)

	X-monotone		Rectilinear		Rectangular	
Resolution	sec.	#hand-probe	sec.	#hand-probe	sec.	#hand-probe
10^2	0.08	26	0.05	24	0.01	16
20^2	0.36	27	0.30	24	0.05	23
30^2	1.03	32	1.30	31	0.14	25
40^2	1.78	33	3.19	30	0.37	29
50^2	2.83	34	6.97	31	0.73	30

3.4.7 Experiments

In this subsection, we examine the time taken to compute the optimal region. In this experiments, we generated our test data whose target attributes takes two values, in the form of an $N \times N$ grid, as follows: We first generated random numbers uniformly distributed in $[N^2, 2N^2]$ and assigned them to the number of records in each pixel. We then assigned a value in $[1, N^2]$ to the number of records that take one of the boolean values of the target attribute, from a corner pixel to the central one, proceeding in a spiral fashion. These test data were generated so that the number of points on the convex hull increases sub-linearly to N, the square root of the number of pixels.

We examined the CPU time taken to compute the optimal region and the number of hand probing needed to find the region. All the experiments were performed on an IBM RS/6000 workstation with a 112 MHz PowerPC 604 chip and 512 MB of main memory, running under the AIX 4.1 operating system.

Table 3.6 shows the time (sec.) and the number of hand probing that were required in the guided branch-and-bound algorithm to find the optimal x-monotone (resp. rectilinear, or rectangular) region that maximizes the entropy function. It shows that the number of hand probing increases very slowly thanks to the guided branch-and-bound algorithm. Figure 3.17 confirms that the CPU time follows our scale estimation. Although the asymptotic time complexity for computing the optimal x-monotone region is better than that for computing the optimal rectilinear region, in practice the optimal rectilinear region is computed faster, because the constant factor is smaller.

We assume the optimal region will be used for classifying data, for example, as a decision tree model. At the root of a decision tree, we may need a large number of pixels to guarantee the specified pixel density. The number of records, however, decreases in lower parts of the tree, and the number of pixels soon becomes less than 30×30 for most datasets; hence, computing the optimal rectilinear region is not costly for the purposes, according to Table 3.6.

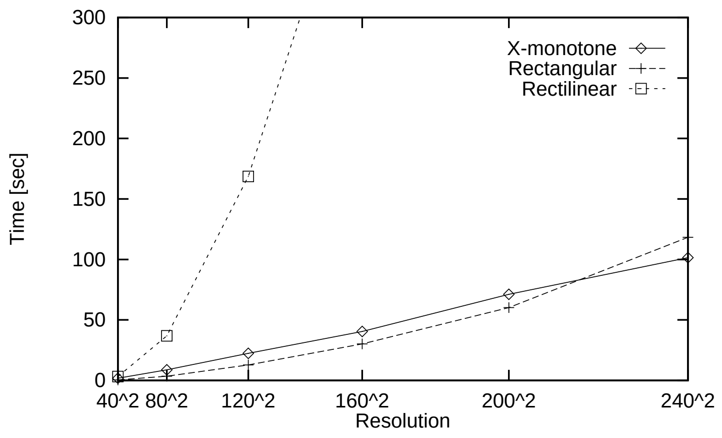


Figure 3.17: Time for Computing the Optimized Region in Classification (2)

Chapter 4

Decision Tree

Classification is the process of finding a set of models that describe and distinguish data classes for the purpose of being able to use the models to predict the class of database records whose class is unknown. Typical forms of the models are rules, rule lists, trees, neural networks, or Bayesian networks. Among them, the author focused on tree induction models, called *decision trees*, in this dissertation.

As mentioned above, among all attributes in a database, we treat one attribute as special, and call it the target attribute. The other attributes are called conditional attributes. A decision tree is a rooted (binary) tree structure[*] for modeling and prediction of the target categorical attribute, each of whose internal nodes is associated with a discriminant rule, called a *test.* We associate each leaf node with the subset (called leaf-cluster) of records satisfying all tests on the path from the root to the leaf. Every leaf-cluster is labeled as one of the value of the target attribute on the basis of the target value distribution in the leaf-cluster.

For each record, at an intermediate node (initially at the root node), we check the test of the node to find out which branch a given record should move to. If a record satisfies a test, it goes down one branch. Otherwise, it goes down the other branch. The record is recursively checked at intermediate nodes and finally reaches a leaf node. Each leaf lists a predicted value for the target attribute that is appropriate to the test conditions along the path to that leaf. We can predict the value of the target attribute for a given record as the value given at the associated leaf.

Example 4.1 Assume a customer profile database in Table 4.1. In the table, the "`Item A`" indicates whether the customer purchase the item or not, "`O`," indicates the customer purchased the item while "`X`" indicates the customer did not. Figure 4.1 is a decision tree that models and predicts the `Item A` of the customer profile database. Each leaf lists a predicted value for the `Item A`. Each test conditions along the path to a leaf is one of typical model for the `Item A`. We can use the tree to predict the (unknown) value of the `Item A` for a given record whose values for conditional attributes, which are used for predicates in the tests, are known. For example, a

[*]The author assumes a tree is a binary tree structure in this dissertation.

Table 4.1: Customer Profile Database (Training Data)

CustomerID	Sex	Age	Occupation	···	Item A	···	Amount Purchase
001	Female	20	Office Worker	···	O	···	30000
002	Male	20	Other	···	X	···	24000
003	Male	40	Teacher	···	O	···	78000
004	Female	30	Office Worker	···	X	···	12000
···	···	···	···	···	···	···	···

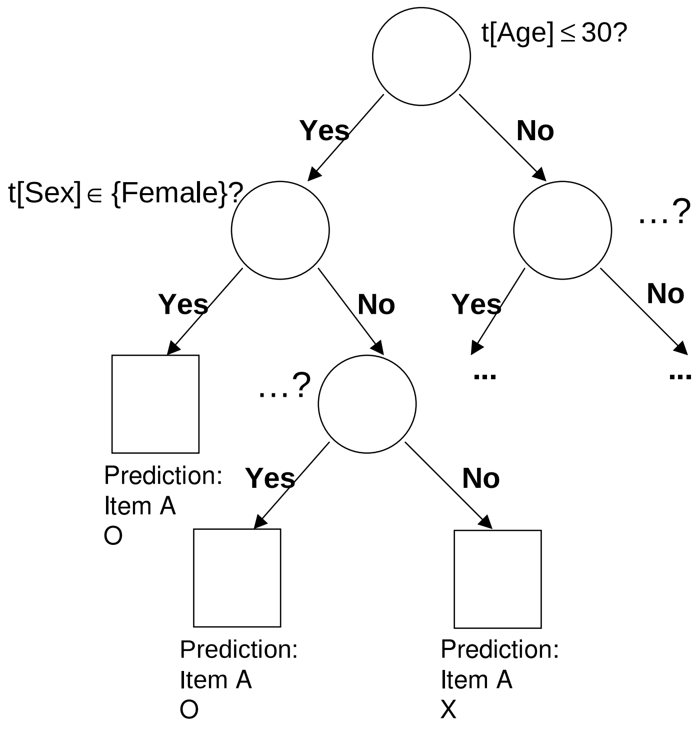


Figure 4.1: Decision Tree for ”Item A”

customer whose `Age` is 15 and `Sex` is `Female` goes to the left node at the test associated with the root node. Then, at the next node, it goes to the left and reaches the leaf node whose predicted value is “`O`.” [**EOE**]

4.1 Construction of Tree Models

If a training data whose value of the target attribute is given, we can construct a decision tree by using discriminant rules for the target attribute that can be found from the training data. For a given training data, we want to construct a compact tree as possible. Unfortunately, if we want to minimize the total sum of the lengths of exterior paths, the problem of constructing the minimum decision tree which completely classify a given data is known to be NP-hard [HR76, GJ79]. It is also believed that it is NP-hard if the minimization objective is the “size,” i.e., the number of nodes of the tree.

```
0)  Algorithm GreedyTreeConst($\mathcal{D}$) {
1)      Read all records in $\mathcal{D}$ to $R$.
2)      Split($R$)
3)  }
4)  Function Split($S$) {
5)      If ($S$ satisfies stopping condition) {
6)          Set a label of this leaf node.
7)          Return.
8)      }
9)      Examine all possible tests on each categorical attribute.
10)     Examine all possible tests on each numeric attribute.
11)     Set the optimal test to the test of this node.
12)     Split $S$ into $S^{left}$ and $S^{right}$ by the optimal test.
13)     Split($S^{left}$)
14)     Split($S^{right}$)
15) }
```

Figure 4.2: Algorithm of Greedy Tree Construction

However, in practical applications, classification accuracy for unknown data is more important than the complete classification of a given data. Therefore, despite the NP-hardness of the problem, many practical solutions [BFOS84, Qui86b, QR89, Qui93] have been proposed in the literature. Among them, the C4.5 program [Qui93] applies a heuristic, which greedily constructs a decision tree in a top-down manner according to a criterion based on the mutual information. At each internal node, the heuristic examines all the possible tests, and chooses the one for which the associated splitting of the set of records attains the minimum entropy value.

The greedy construction method proposed by Quinlan can also be applied for various criteria like the ones in Section 3.2.1. The classic greedy recursive partitioning strategy for constructing a decision tree can be summarized as the algorithm GreedyTreeConst in Figure 4.2.

In the greedy tree construction, we find the optimal discriminant rule for test at each node. The author explored how we can find the optimal rule in Section 3.3 and 3.4. We stop splitting a tree if a set, S, satisfies one of stopping conditions. In general, the stopping conditions are as follows:

- **Set Size Condition**
  The number of records in S becomes negligible.

- **Accuracy Condition**
  For a decision tree, all values of the target attribute becomes the same or one dominant value in S is much more frequent than the other values.

- **Dividability Condition**
  We can not find any predicate that divide S on the domain of all conditional attributes.

Most of the tree construction algorithms assume a binary tree structure. Some commercial systems like C4.5 [Qui93] construct n-nary tree structure, some nodes in which are divided into n segments, one for each distinct value of the categorical conditional attribute that is used for corresponding predicate in the node.

4.2 Prediction Accuracy

One of the most important purpose to construct decision trees from given training database is to predict unknown value of the target attribute. For a record whose value of the target attribute is unknown but values of other necessary conditional attributes are known, we can apply a decision tree to find which leaf node of the tree the record is classified. In each leaf node of a tree, there is a representative value for the target attribute and the value is used for prediction of records that are classified into the node.

Now the problem is how credible the predicted value is. One of a good measure for estimating the credibility is *prediction accuracy* of decision trees.

One of the most popular method to evaluate prediction accuracy is so called *cross validation.* In the cross validation, we divide a training database whose value of target attribute are known into two datasets. Then, we construct a tree from one of the two datasets. We predict value of the target attribute for each record in the other dataset, we call the set "validation set," and examine the prediction.

For a decision tree, we compute the following ratio of the number of records that are misclassified over the number of all data in the validation set, and call the ratio the *error ratio* of the decision tree. A misclassified record is a record such that predicted value for the record is different from actual value of the record.
Accuracy of Decision Trees:

$$\frac{number\ of\ misclassified\ records\ in\ the\ validation\ set}{number\ of\ records\ in\ the\ validation\ set}$$

By evaluating accuracy for a validation set, we can analogically estimate the credibility of the predicted value of decision trees for records whose value of the target attribute is unknown.

4.3 Pruning

We have examined the prediction accuracy of a decision tree by using public datasets, "diabetes" provided in the following WWW site:

`http://www.ics.uci.edu/~mlearn/MLRepository.html`

In the experiment, we made a huge tree for each data and examined its accuracy against both training set and validation set. Then, we pruned its leaves little by little and examined

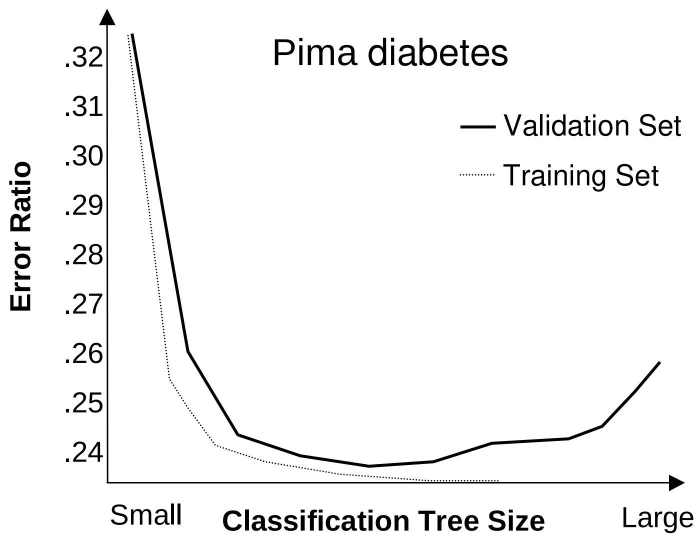


Figure 4.3: Tree Size and Prediction Accuracy

the smaller tree of each size. Figure 4.3 shows how the accuracy of trees change depending on the size.

In general, training data contains some noisy records caused by input failures, strange instance and so forth. If we strictly classify such training data and make large trees, trees tend to fit even for such noisy records. As the results, such trees *overfit* for the training data and prediction accuracy for validation set become worse. As in the figure, the accuracy for the validation set becomes better if we prune some leaves, while the accuracy for training set becomes better if we increase the size of the tree.

In order to avoid the overfitting, many techniques, which we call *pruning*, have been proposed in the literature [Min89]. There are roughly two types of pruning techniques. One, called *prepruning* or *stopping*, is to stop expanding leaves earlier before a tree becomes too large. The other, called *postpruning*, is to remove some of the expanded leaves from a large tree. The former method saves wasteful time for unnecessary expansions. On the other hand, the latter achieves more reliable results in many practical cases even though it needs the unnecessary expansions.

4.3.1 Prepruning

Quinlan proposed a prepruning method for classification problems in [Qui86a]. In the greedy construction of a tree, the method stop expanding a node based on χ^2 test. Assume we focus on a node whose corresponding dataset is R consists of $|R|$ records. Let $p_i(S)$ denote the relative frequency in a set S with which the target attribute takes the i-th value. Suppose that R is divided into two subsets named S and $\bar{S}$. The χ^2 denote

$$\sum_i \frac{|S|(p_i(S) - p_i(R))^2 + |\bar{S}|(p_i(\bar{S}) - p_i(R))^2}{p_i(R)}.$$

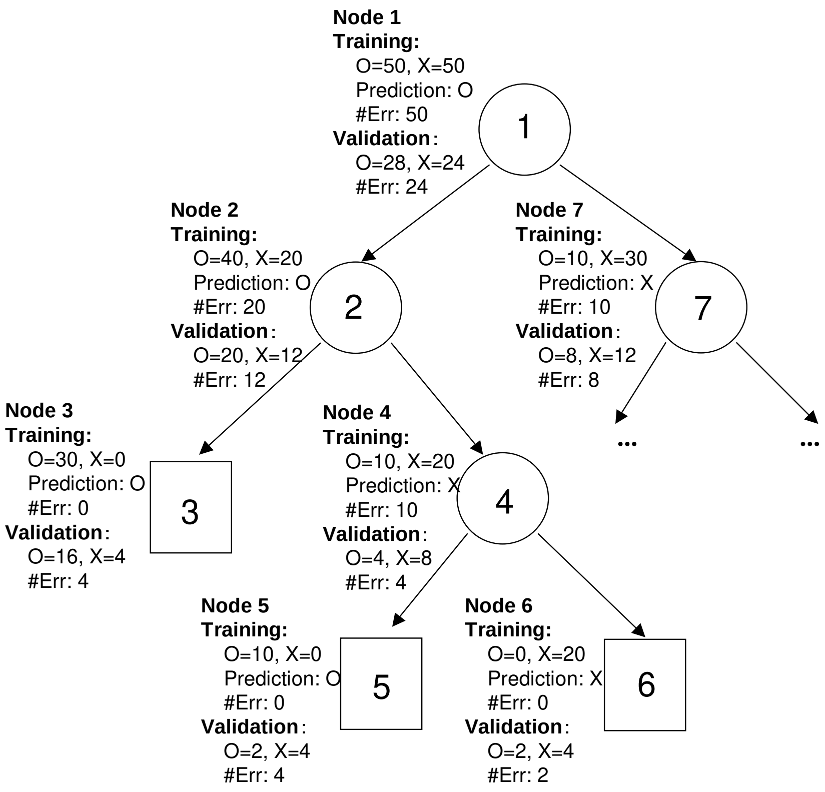


Figure 4.4: Classification of Validation Set

Lower χ^2 values support with higher confidence the hypothesis that the data distributions in R and in R's two child subsets are independent. Put another way, lower χ^2 values mean that the splitting of R is less effective. We, therefore, give a (user defined) threshold for χ^2 so that we stop expanding the node if χ^2 is less than the threshold. Higher thresholds are more likely to stop generating subtrees and thereby producing smaller decision trees.

Though the prepruning methods can reduce computation time, the accuracy of the trees produced by postpruning is, in general, higher than that of prepruned trees. Moreover, recent computational power allows us to spend time for unnecessary expansion of nodes. Therefore, postpruning methods becomes very popular.

4.3.2 Reduced Error Pruning

One of the most popular postpruning method is *reduced-error pruning* [Qui87]. To explain the method, assume a decision tree in Figure 4.4. The tree predicts a target attribute that has two values, "`O`" and "`X`." We apply a validation set to the tree and the prediction result for each node is shown in the figure.

For each intermediate node, the reduced-error pruning compares the error at the node to the sum of the error of its all descendant leaf nodes. For example, if we examine the node 4, the error of the node (for the validation set) is 4 and the sum of error of the node 5 and the node 6 is 6. Therefore, we can reduce the error for the validation set by pruning the node 5 and the node 6. The reduced-error pruning eliminate all of such nodes.

4.3.3 Cost Complexity Pruning

Breiman et al. considered another popular postpruning method called *cost-complexity pruning* that compares the complexity of a tree and the error rate [BFOS84]. In the cost-complexity pruning, we use a following function evaluating error cost and complexity of subtree for each node:

$$\frac{sum\ of\ error\ in\ descendant\ leaf\ nodes}{number\ of\ records} + \alpha * (number\ of\ descendant\ leaf\ nodes)$$

where α is a weight parameter for the number of leaf nodes. We can change the weight for the complexity in the function by the parameter.

For example, the cost of the node 4, the node 5, and the node 6 in Figure 4.4 is

$$\left(\frac{0}{10}\frac{10}{100} + \frac{0}{20}\frac{20}{100}\right) + \alpha * 2.$$

If we prun the node 5 and the node 6, the cost becomes

$$\left(\frac{10}{30}\frac{30}{100}\right) + \alpha * 1.$$

In the example, we will prun the node 5 and the node 6 if $\alpha \geq 0.1$. We construct trees with various size with various α. We apply the validation set to the trees and find the most accurate tree and its α value.

In addition to the mentioned postpruning methods, Quinlan proposed a method, which does not use a validation set to find adequate size of a tree, called *pessimistic pruning* [Qui87, Qui93]. Mehta et al. proposed a method based on the minimum description length (MDL) principle [MRA95].

4.4 Multivariate Decision Trees

The author explores an improvement to the standard strategy of selecting internal-node tests during the decision tree construction. In particular, conventional tree construction algorithms often use a test on a numerical conditional attribute with a cut value within the range of the conditional attribute that splits data into those below the cut value and those above. Such a test is called a "guillotine-cut." However, for the case when two numerical conditional attributes have a non-linear correlation with respect to the target attribute, trees with guillotine-cuts tend to become large and such large trees are in general inaccurate. In such a case, a test with a two-dimensional region for splitting data into two subsets may be more natural and effective.

4.4.1 Trees with Region Rules

Quinlan [Qui93] pointed out that the approach that use discriminant rules on a single numerical conditional attribute has a serious problem if a pair of attributes is correlated. For example, let us consider two numerical attributes, "height (cm)" and "weight (kg)," in a health check database. Obviously, these attributes have a strong correlation. Indeed, the region $0.85 * 22 * height^2 < weight < 1.15 * 22 * height^2$ and its complement provide a popular criterion for separating healthy patients from patients who need dietary cures. As the example shown in Figure 1.7 in Section 1.2.2, the enclosed gray region shows the "healthy" region. However, if we construct a decision tree for classifying patients by using guillotine cutting, its subdivision is complicated and the size of the tree becomes very large, and hence, it becomes hard to recognize the substantial rule.

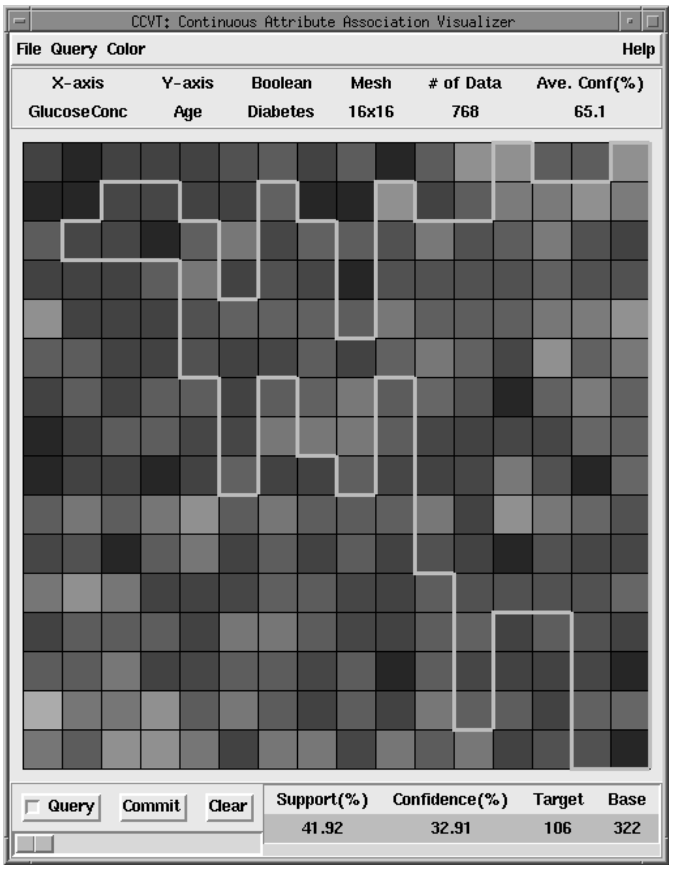
In order to handle the correlation problems, the author added region rules for all pairs (A, A') of correlated attributes, and construct a decision tree by using the optimal region rule at each step of the greedy tree construction. As a special case of region rules, we also consider rules of the form $(t[A] \in I)$ for a range I in order to develop our system.

We specify a number $N \leq \sqrt{n}$, and construct an almost equi-depth ordered bucketing of records for each numeric attribute A. That is, we construct buckets $B_1^A, \cdots, B_N^A$ each of which contains approximately n/N records, satisfying $t[A] \leq t'[A]$ for every $t \in B_i^A, t' \in B_j^A$ and $i < j$. We can use the efficient randomized algorithm mentioned in Section 2.2.2 for constructing such a bucketing.

For a pair of numeric attributes A and A', we have a pixel grid G of size $N \times N$ generated as a Cartesian product of the buckets. We consider a family $\mathcal{R}$ of grid regions of G. For each $R \in \mathcal{R}$, we consider a splitting that divides data into those inside the region R and those outside the region. Let R_{opt} be the region of $\mathcal{R}$ that maximizes one of objective functions defined in Section 3.2. The region R_{opt} and the associated splitting are called the *optimal region* with respect to $\mathcal{R}$ and the pair of attributes (A, A'). The author proposed decision trees that use such optimal regions for tests at internal nodes of the trees.

Since the regions separated by guillotine cutting and those separated by line cutting are very special cases of connected x-monotone regions, our method can find a region rule with better objective function values at each step of the greedy tree construction. Hence, we can almost always create a smaller tree. In the example in Figure 1.7, the rule $0.85*22*height^2 < weight < 1.15 * 22 * height^2$ itself defines an x-monotone region, and hence we can create a nice decision tree of height two, i.e., with the root and two leaves.

One defect of our approach is that the rule $(t[A], t[A']) \in R$ is sometimes hard to describe. However, we can describe the rule by combining a visualization system. Figure 4.5 is a graphical view of an x-monotone region in a decision tree which was constructed from a "diabetes" diagnosis dataset, which is in the UCI repository [BM98]. The visualization system uses red color level and brightness to show characteristics of each pixel. The red level indicates the probability of a positive, or negative, patient in each pixel, and the brightness indicates the number of patients in each pixel. The data in the node are partitioned according to whether they are in the x-monotone region R_{opt} or not. In this example, the near-triangle region R_{opt} corresponds to the cluster of patients less likely to be positive for diabetes.



X-axis	Y-axis	Boolean	Mesh	# of Data	Ave. Conf(%)
GlucoseConc	Age	Diabetes	16x16	768	65.1

Support(%)	Confidence(%)	Target	Base
41.92	32.91	106	322

Figure 4.5: X-monotone Region Splitting

4.4.2 Record Density of a Pixel

If we consider regions for tests in a decision tree, we have to adjust some inherent tuning parameters of regions in addition to the choice of region families.

We uniformly distribute each numeric attribute into N ordered buckets. Next, for each pair of numeric attributes, we create an $N \times N$ pixel grid G. The average number of records in each pixel on G, which we call the *record density*, influences the accuracy of the trees. Using a lower record density is likely to make the trees overfit the training dataset, while a coarse grid with a higher record density often fails to find variously shaped regions.

We empirically found that a record density ranging from five to ten gives accurate results for test datasets, and we therefore use a record density of five or ten in the later experimental section. The experimental results for a record density of five show the effects of relatively fine grid regions, while those for a record density of ten show the effects of coarse grid regions.

In the process of generating trees, the number of records becomes smaller at nodes lower in the tree. In order to guarantee a record density of five or ten, we are forced to use a coarse grid, say 2×2, which is too rough to generate interesting regions. If we have to use a coarse grid whose size is less than 5×5, we employ guillotine-cut splitting instead.

4.5 Experiments

4.5.1 Prediction Accuracy

In this subsection, we describe several experimental results to examine classification accuracy of trees with region splitting by using cross validation test.

Ten-fold Cross-Validation Test

We performed the following ten-fold cross validation test:

- Randomly divide the original dataset into ten almost equal-sized subsets.
- Take the union of nine subsets and use the union as the training dataset to generate a decision tree that splits data by guillotine cutting, x-monotone regions, rectilinear regions, or rectangular regions.
- Use the remaining one subset as the test dataset to evaluate the decision tree generated by the training dataset. Compute the ratio of the number of test records that are misclassified to the number of all data in the test dataset, and call the ratio the *error ratio* of the decision tree *against* the test dataset.
- Repeat the above steps ten times, and then calculate the average of all the error ratios.

Pruning

From a training dataset, we can generate larger decision trees by expanding leaves as much as possible. Larger decision trees can correctly classify data in the training dataset with higher accuracy, but they are likely to *overfit* the training dataset, and therefore provide higher error ratios against the test dataset.

We need some criterion for when to stop expanding a decision tree, and we employ the χ^2 test to control tree size [Qui86a]. We used χ^2 thresholds ranging from 0 to 45. The accuracy critically depends upon the χ^2 critical value that is selected. Unfortunately, we do not have a theoretical method to predict the optimal, or critical, value of χ^2. To the best of our knowledge, this problem is neither solved nor well studied in the literature of research on decision trees. We adopt a naive heuristic method, named *multi-trial method*, in which we construct trees for a few number of χ^2 values, test the trees using the ten-fold cross validation, and select the one with the best accuracy. Although our method uses a portion, 1/10 in our experiment, of the dataset as the test data for the decision trees, we think it is practically sufficient to use 9/10 of the dataset for the training data of the decision trees.

Datasets

We used several public datasets, summarized in Table 4.2, which were acquired from the UCI Machine Learning Repository [BM98]. Since this experiment was to examine our method for handling numeric rules and its effectiveness, we chose these datasets because all the conditional attributes were numeric. For records that have null value, we assigned the average value of the attribute to those missing values.

If we use small datasets, we have to use small pixel grids, say 5×5 or a little larger. Such small pixel grids are inadequate for comparing various region families. Moreover, such notchy grid regions reduce the accuracy even if they can capture correlations. Therefore, we eliminated small datasets from the repository. However, we included some datasets that are small in the above sense, to obtain as broad an evaluation as possible. Note that numerical values in the "german credit" dataset are discrete.

For the datasets whose number of categories in the target attribute is more than three, we employed hand probing in the multidimensional space.

Classification Capability

Figure 4.6 shows the ten-fold cross validation result of the "breast-cancer-wisconsin" dataset in Table 4.2. In the figure, line labeled "Rectilinear (dens5)," for example, shows the average error ratio against the test dataset of decision trees using rectilinear convex region splitting with a pixel density of 5, for various χ^2 stopping values ranging from 0 to 45. For smaller χ^2 values, we have decision trees so much larger that their error ratios are relatively high and overfit the training dataset. In general, we have to make larger decision trees as the number of categories

Table 4.2: Summary of Classification Datasets

Dataset	#categories	#records	#attributes
breast-cancer-wisconsin	2	699	9
german credit	2	1000	24
liver disorder	2	345	6
pima diabetes	2	768	8
balance scale	3	625	4
waveform	3	5000	20
waveform-+noise	3	5000	40
vehicle	4	846	18
segmentation	7	2310	19

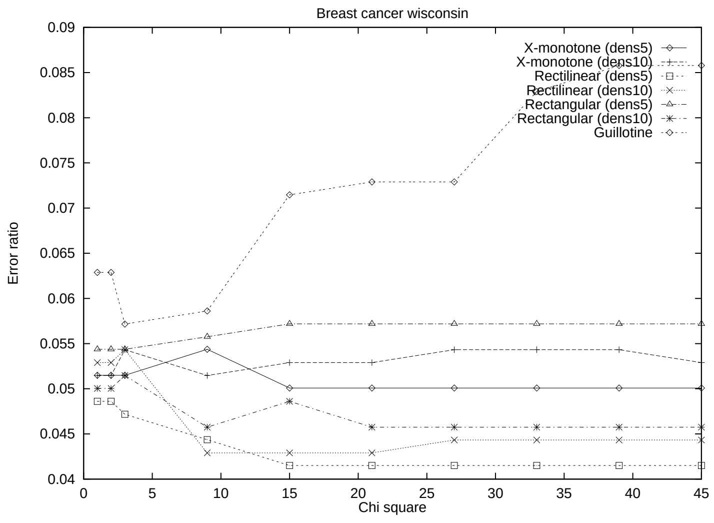


Figure 4.6: Accuracy for "breast cancer wisconsin"

in the target attribute increases.

In the figure, the error ratio of the decision tree for $\chi^2 = 45$ has the lowest value, 4.15%. On the other hand, the lines labeled "guillotine" show the cases in which we use guillotine cutting instead of region splitting. For the "breast-cancer-wisconsin" dataset, the lowest error ratio of those decision trees with guillotine cutting is 5.72%.

For the other datasets, we performed the same analysis, which shows the same general trends as Figure 4.6 (i.e., generally, a lower error for rectilinear (dens 5) over alternatives across weight values, and similar shape curves for each method due to under and overfitting). All the graphs of the cross validation results are shown in Appendix A.1.

Table 4.3 and 4.4 summarize the results of the ten-fold cross validation results of the datasets in Table 4.2. As regions for splitting datasets, we used x-monotone regions, rectilinear convex regions, and rectangular regions. The pixel density was set to 5 or 10. We compute the lowest error ratio for each type of splitting and the average number of leaves in the accurate trees. In the table, the lowest error ratio for each dataset is underlined. Observe that results of region

Table 4.3: Summary of Classification Accuracy (1)

	Rectilinear				X-monotone	
	dens. 5		dens. 10			
Dataset	Err(%)	Size	Err(%)	Size	Err(%)	Size
breast-cancer-wisconsin	<u>4.15</u>	3.3	4.29	3.5	5.01	3.4
german credit	<u>23.80</u>	3.6	27.90	4.3	27.30	3.8
liver disorder	38.83	3.2	33.36	11.4	34.81	2.0
pima diabetes	25.12	2.1	26.02	7.0	24.47	4.5
balance scale	<u>15.52</u>	34.7	18.24	14.5	<u>15.52</u>	34.7
waveform	<u>20.98</u>	33.2	21.18	27.4	21.74	46.9
waveform-+noise	21.80	34.6	<u>21.32</u>	38.6	22.54	30.3
vehicle	28.47	12.2	29.68	77.1	30.02	17.0
segmentation	<u>4.37</u>	53.7	5.06	58.2	4.81	57.9

splitting are more accurate than guillotine cutting in most cases.

All of the cross validation results from datasets using diverse conditions can be summarized as follows:

- Table 4.3 and 4.4 show that the rectilinear convex region splitting with pixel density 5 won over the guillotine cutting on eight of nine datasets. Moreover, it won over all of other region splittings except the rectilinear convex region splitting with pixel density 10 on six data sets.
- The graphs show that if we use x-monotone regions with a low pixel density, it tends to give high error ratio because of overfitting. In contrast, rectilinear convex regions are robust even if we use a low pixel density.
- The χ^2 value giving the most accurate result highly depend on both datasets and construction methods.

Another Pruning Method

A defect of constructing the decision tree by using χ^2 based pruning together with the multi-trial method is that we need to construct trees for several χ^2 values. In our actual implementation, we first construct the decision tree for the smallest χ^2 value, and then prune it to obtain trees with larger χ^2 values; therefore, construct unnecessarily huge tree for the smallest χ^2 value, and slow down the processing time.

Although it is ideal to find out a more sophisticated method to predict a critical value of χ^2 from the experiment; however, the experimental results, given in Figure 4.6 and other figures in Appendix A.1, show that the accuracy curve highly depends on the dataset, and behaves too wild to guess a nice function on the data parameters such as data size, number of attributes,

Table 4.4: Summary of Classification Accuracy (2)

	Rectangular		Guillotine	
Dataset	Err(%)	Size	Err(%)	Size
breast-cancer-wisconsin	4.58	4.0	5.72	19.2
german credit	26.90	4.6	25.60	12.8
liver disorder	<u>31.08</u>	2.0	34.87	3.0
pima diabetes	<u>23.69</u>	4.6	26.82	4.4
balance scale	19.34	44.7	20.95	48.8
waveform	22.36	65.1	22.74	91.7
waveform-+noise	22.94	67.7	24.36	91.7
vehicle	27.65	38.9	<u>26.23</u>	94.0
segmentation	4.89	65.3	4.50	71.1

and number of categories, to predict the optimal value of χ^2. The investigation into such a prediction function remains a major open problem in both theory and practice.

In this subsection, we describe our experiment on another pruning strategy, in which we need not construct a unnecessarily huge tree, and compare the accuracy to the multi-trial method.

Training and Validation Set Method

A popular approach for the problem to determine whether to expand a node or not during construction of a decision tree is the *training and validation set* method, in which we separate an available dataset into two distinct subsets and then use one to expand a leaf and the other to evaluate the expansion [Mit97]. Among many possible implementations of the training and validation set method, we adopt the following algorithm:

- We separate the available records into two distinct sets, a training set and a validation set. The validation set contains about one third of all the available records.
- We construct the decision tree on the training set top-down using the entropy heuristic, in which we stop the expansion using the validation given below.
- Suppose that a node (parent node) in the decision tree is split into two children. We compute the number of misclassified records in the validation set for each of the parent node and two children. If the splitting does not reduce the number of misclassified records, then we abandon the splitting and stop expanding at the parent node. Otherwise, we continue expanding the tree recursively.

We evaluate the effectiveness of this approach by using the cross validation test. Table 4.5 shows the error ratio and the average number of leaves of the pruned trees. The result shows that although the training and validation set method is sometimes better or competitive, the multiple trial method using χ^2 validation clearly wins on several datasets.

Table 4.5: Classification Accuracy for Pruned Trees

	X-monotone		Rectilinear		Rectangular		Guillotine	
Dataset	Err	Size	Err	Size	Err	Size	Err	Size
breast-cancer-wisconsin	3.86	4.0	3.72	4.1	6.14	3.9	7.29	5.6
german credit	28.10	4.2	28.40	4.3	27.60	5.3	26.40	8.2
liver disorder	38.00	2.0	38.00	2.0	38.00	2.0	37.44	4.5
pima diabetes	28.67	5.4	29.59	4.1	29.70	5.0	24.86	8.4
balance scale	19.98	15.8	19.98	15.8	22.40	11.1	26.24	12.5
waveform	23.28	64.5	23.98	65.2	23.84	82.6	29.50	63.4
waveform-+noise	23.82	54.6	22.96	60.5	25.02	66.7	25.08	92.3
vehicle	34.65	17.3	33.69	18.2	33.80	27.5	47.10	11.7
segmentation	24.46	21.4	21.95	22.0	22.12	22.3	32.25	14.2

4.5.2 Performance Results

In this section, we examine the overall performance in the tree construction. At each node of a decision tree, we first prepare the grid, and then compute the optimal region. The grid preparation is not expensive, because it can be done by scanning all the records at a node just once. The problem is that we have to calculate the optimal regions for all pairs (permutations for x-monotone regions) of two numerical attributes. Thus, the number of attributes dramatically affects the overall performance.

Table 4.6 compares the time (sec.) taken to construct trees by using datasets with different numbers of records. We randomly selected records from the "waveform" dataset to generate datasets having different numbers of records, and used those datasets to construct decision trees by performing region splitting with a pixel density of 5 and conventional guillotine cutting. We used the first eight numerical attributes as the conditional attributes, in order to simplify the experiment, and compared the time taken to construct pruned trees. The results show that the tree construction time is a little more than our scale estimation, because trees from larger datasets tend to become bigger.

Table 4.7, on the other hand, compares the performance using datasets with different numbers of attributes. We used 3000 records from the "waveform" dataset, and constructed trees using the first N numerical attributes. Observe that the time complexity is almost linear in the square of the number of attributes. In case of guillotine cutting trees, the test optimization cost is so small that time taken to construct a tree depends mainly on its final tree size.

Table 4.6: Decision Tree Construction Time (1)

#records	X-monotone	Rectilinear	Guillotine
1000	58	27	8
2000	163	66	14
3000	273	106	19
4000	425	165	26
5000	613	244	49

Table 4.7: Decision Tree Construction Time (2)

#attributes	X-monotone	Rectilinear	Guillotine
4	87	35	14
6	163	70	23
8	273	106	19
10	410	152	15
12	522	204	18

Chapter 5

Regression Tree

Like a decision tree, a regression tree is a tree structured model for values of a target attribute in a relational table. We call it a *regression tree* if a target attribute is numerical, while a decision tree is a model for a categorical target attribute. Figure 5.1 is a regression tree that models and predicts the `Amount Purchase` of the customer profile database in Table 4.1.

5.1 Regression Rules

If a training data whose value of the target attribute is given, we can construct a regression tree by using discriminant rules for the target attribute that can be found from the training data. For a given training data, we want to construct a compact tree as possible. However, same as the decision tree construction, problem of constructing the minimum regression tree is NP-hard. Therefore, we greedily construct a regression tree in a top-down manner according to a criterion that is designed for regression problems.

5.1.1 Criterion of Regression Rules

The target attribute A of regression problems is numerical. For regression problems, criterion of rules can be defined as functions of two dimensional stamp points, $\mathbf{x}(S) = (|S|, \sum_{t \in S} t[A])$.

Mean Squared Error

To predict or model the numerical target attribute, we prefer the rule that most reduces variance of the target attribute and use an objective function called the *mean square error function*. If a rule discriminates S from $\bar{S}$ in R, the mean square error function is defined as follows:

$$\begin{aligned} MSE(\mathbf{x}(S)) &= MSE(S; \bar{S}) \\ &= \frac{\sum_{t \in R}(t[A] - \mu(R))^2}{|R|} \\ &\quad - \frac{\sum_{t \in S}(t[A] - \mu(S))^2 + \sum_{t \in \bar{S}}(t[A] - \mu(\bar{S}))^2}{|R|} \end{aligned}$$

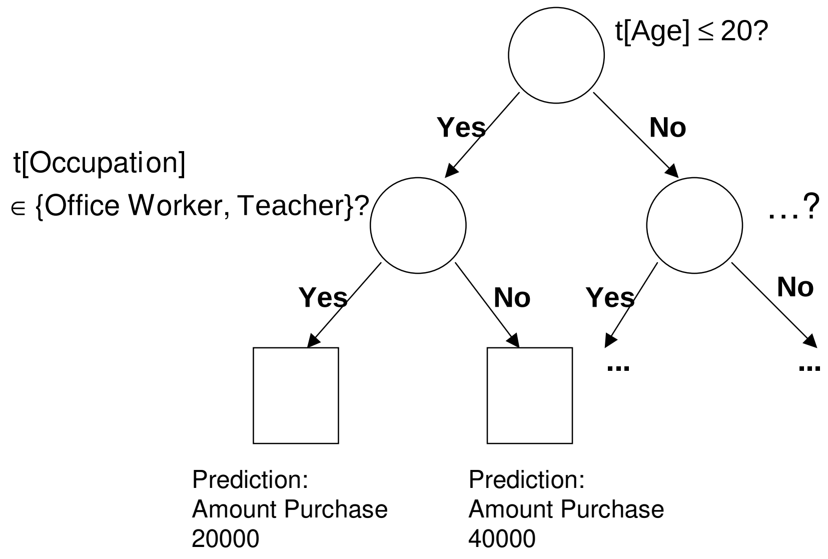


Figure 5.1: Regression Tree for "Amount Purchase"

In the definition, $\mu(S)$ is the mean value of the target attribute A in S, i.e., $\mu(S) = (1/|S|)\sum_{t\in S} t[A]$ assuming $|S| \neq 0$.

For regression problems, we prefer higher values of the mean squared error function. Among all possible rules on a conditional attribute, we call the rule that has the highest value of the function the *optimal rule*. If we assume a categorical conditional attribute C that has n distinct values, there are $O(2^n)$ rules, and thus, there are $O(2^n)$ stamp points in the two dimensional space. Similarly, there are $O(n^2)$ stamp points for a numerical conditional attributes that has n values. The optimization problem is to find a stamp point whose mean squared error is the smallest among all possible points.

In the definition of the mean squared error function, $\frac{\sum_{t\in R}(t[A]-\mu(R))^2}{|R|}$ is invariant of a choice of $\mathbf{x}(S)$. Therefore, we focus on

$$\frac{\sum_{t\in S}(t[A]-\mu(S))^2 + \sum_{t\in \bar{S}}(t[A]-\mu(\bar{S}))^2}{|R|},$$

which we denote by $U(\mathbf{x}(S))$, or $U(\mathbf{x})$ in short.

In order to examine the value of $U(\mathbf{x})$ for a stamp point $\mathbf{x}$, we have to scan entire relation R. Since we have to compute $U(\mathbf{x})$ frequently and $|R|$ can be huge in data mining applications, the computation cost for $U(\mathbf{x})$ is not affordable.

Interclass Variance Function

For easier computation of the value group $V \subset \mathrm{dom}(C)$ that minimizes $U(\mathbf{x})$, we introduce

$$Var(\mathbf{x}(S)) = Var(S;\bar{S}) = |S|(\mu(S)-\mu(R))^2 + |\bar{S}|(\mu(\bar{S})-\mu(R))^2,$$

which we call the *interclass variance*. The following theorem shows that the value group V, whose stamp point is $\mathbf{x}$, that maximizes $Var(\mathbf{x})$ also minimizes $U(\mathbf{x})$.

Theorem 5.1 *The maximization of $Var(\mathbf{x}(S))$ is equivalent to the minimization of $U(\mathbf{x}(S))$.* **[EOT]**

Proof:

$$\begin{aligned} Var(\mathbf{x}(S)) &= |S|(\mu(S)-\mu(R))^2 + |\bar{S}|(\mu(\bar{S})-\mu(R))^2 \\ &= -|R|\mu(R)^2 + (|S|\mu(S)^2 + |\bar{S}|\mu(\bar{S})^2) \end{aligned}$$

Since $-|R|\mu(R)^2$ is invariant with respect to the choice of $\mathbf{x}$, the maximization of $Var(\mathbf{x}(S))$ is equivalent to the maximization of $(|S|\mu(S)^2 + |\bar{S}|\mu(\bar{S})^2)$.

$$\begin{aligned} U(\mathbf{x}(S)) &= \frac{\sum_{t\in S}(t[A]-\mu(S))^2 + \sum_{t\in\bar{S}}(t[A]-\mu(\bar{S}))^2}{|R|} \\ &= \frac{\sum_{t\in R} t[A]^2 - (|S|\mu(S)^2 + |\bar{S}|\mu(\bar{S})^2)}{|R|} \end{aligned}$$

Since $\sum_{t\in R} t[A]^2$ and $|R|$ are independent of the choice of $\mathbf{x}$, the minimization of $U(\mathbf{x}(S))$ is equivalent to the maximization of $(|S|\mu(S)^2 + |\bar{S}|\mu(\bar{S})^2)$, and therefore the maximization of $Var(\mathbf{x}(S))$ is equivalent to the minimization of $U(\mathbf{x}(S))$. **[EOP]**

Notice that we do not have to scan entire relation R to examine the value of $Var(\mathbf{x}(S))$, if we precompute the coordinate values of all atomic stamp points.

5.1.2 Convexity

Next we consider how to compute the value group $V \subset \mathrm{dom}(C)$ that maximizes $Var(\mathbf{x}(S))$. Observe that $Var(\mathbf{x}(S))$ is invariant if we replace $t[A]$ by $t[A]-\mu(R)$ for each $t \in R$. Thus, after this replacement, the value group maximizing $Var(\mathbf{x}(S))$ still gives the solution to the original problem. Furthermore, this modification makes $\mu(R) = 0$, and hence

$$Var(\mathbf{x}(S)) = |S|\mu(S)^2 + |\bar{S}|\mu(\bar{S})^2.$$

Let x denote $|S|$, let y be $\sum_{t\in S} t[A]$, and let M be $|R|$. Since $|S|\mu(S) + |\bar{S}|\mu(\bar{S}) = |R|\mu(R) = 0$, we have

$$Var(\mathbf{x}(S)) = Var(x,y) = y^2(\frac{1}{x} + \frac{1}{M-x}),$$

which we will denote by $f(x,y)$.

Theorem 5.2 *$f(x,y) = y^2(\frac{1}{x} + \frac{1}{M-x})$ is convex in the region $M > x > 0$; namely,*

$$(1-\gamma)f(x_1,y_1) + \gamma f(x_2,y_2) \geq f((1-\gamma)x_1 + \gamma x_2, (1-\gamma)y_1 + \gamma y_2)$$

for $0 \leq \gamma \leq 1$ and arbitrary points (x_1,y_1) and (x_2,y_2) such that $M > x_1, x_2 > 0$. **[EOT]**

Proof: Let Δ denote a vector (δ_1,δ_2), and let Y be $\delta_1 x + \delta_2 y$. It is sufficient to show that for any Δ, $\partial^2 f(x,y)/\partial Y^2 \geq 0$. First let us consider the case in which $\delta_1, \delta_2 \neq 0$.

$$\begin{aligned} \frac{\partial f(x,y)}{\partial Y} &= \frac{\partial f(x,y)}{\partial x}\cdot\frac{1}{\delta_1} + \frac{\partial f(x,y)}{\partial y}\cdot\frac{1}{\delta_2} \\ &= y^2(\frac{-1}{x^2} + \frac{1}{(M-x)^2})\frac{1}{\delta_1} + 2y(\frac{1}{x} + \frac{1}{M-x})\frac{1}{\delta_2} \end{aligned}$$

Next,

$$\begin{aligned}
\frac{\partial^2 f(x,y)}{\partial Y^2} &= \{y^2(\frac{2}{x^3}+\frac{2}{(M-x)^3})\frac{1}{\delta_1}+2y(\frac{-1}{x^2}+\frac{1}{(M-x)^2})\frac{1}{\delta_2}\}\frac{1}{\delta_1} \\
&\quad +\{2y(\frac{-1}{x^2}+\frac{1}{(M-x)^2})\frac{1}{\delta_1}+2(\frac{1}{x}+\frac{1}{(M-x)})\frac{1}{\delta_2}\}\frac{1}{\delta_2} \\
&= \frac{2}{x}(\frac{y}{x\delta_1}-\frac{1}{\delta_2})^2+\frac{2}{M-x}(\frac{y}{(M-x)\delta_1}+\frac{1}{\delta_2})^2 \\
&\geq 0
\end{aligned}$$

Next, when $\delta_1 \neq 0$ and $\delta_2 = 0$,

$$\frac{\partial^2 f(x,y)}{\partial Y^2} = y^2(\frac{2}{x^3}+\frac{2}{(M-x)^3})\frac{1}{{\delta_1}^2} \geq 0.$$

We can similarly prove the case in which $\delta_1 = 0$ and $\delta_2 \neq 0$. **[EOP]**

Thanks to this property, the optimal point that gives the optimal rule must be a point of the convex hull of all stamp points in the two dimensional space. The problem now becomes how to find the optimal point of the convex hull efficiently.

In classification problems where $k = 2$, there is an efficient $O(n \log n)$ algorithm for finding the optimal value group V. For the optimal range or region, there are efficient convex hull search algorithms mentioned in Section 3.4. We can simply apply this efficient algorithm for the optimization problem of the regression problems.

5.1.3 Experiments

Rules on categorical conditional attributes can efficiently computed in $O(n \log n)$ time by the Greedy-Enumeration algorithm in Figure 3.1. Therefore, in this section, we examined performance of our algorithm for finding the optimal region that minimizes the mean squared error.

Computing One Optimal Region

We generated our test data, in the form of an $N \times N$ grid, as follows: We first generated random numbers uniformly distributed in $[N^2, 2N^2]$ and assigned them to the number of records in each pixel. We then assigned $1, \cdots, N^2$ as the sum of the target values in a pixel, from a corner pixel to the central one, proceeding in a spiral fashion. These test data were generated so that the number of points on the convex hull increased sub-linearly to N, by the square root of the number of pixels. We examined the CPU time taken to compute the optimal region and the number of hand probing needed to find the region. We performed all experiments on an IBM RS/6000 workstation with a 604e3 332 MHz CPU and 768 MB of main memory, running under the AIX 4.2 operating system.

Table 5.1 shows the time (sec.) and number of hand probing that were required in the guided branch and bound algorithm to find the optimal x-monotone (or respectively rectilinear convex, or rectangular) region that minimizes the mean squared error. It shows that the number of hand

Table 5.1: Time for Computing the Optimal Region in Regression (1)

	X-monotone		Rectilinear		Rectangular	
#pixel	time	#hand-probe	time	#hand-probe	time	#hand-probe
10^2	0.0117	29	0.0148	22	0.00282	17
20^2	0.0522	36	0.120	30	0.0146	23
30^2	0.112	35	0.444	33	0.0429	24
40^2	0.193	34	1.04	32	0.0951	24
50^2	0.372	42	2.27	35	0.199	27

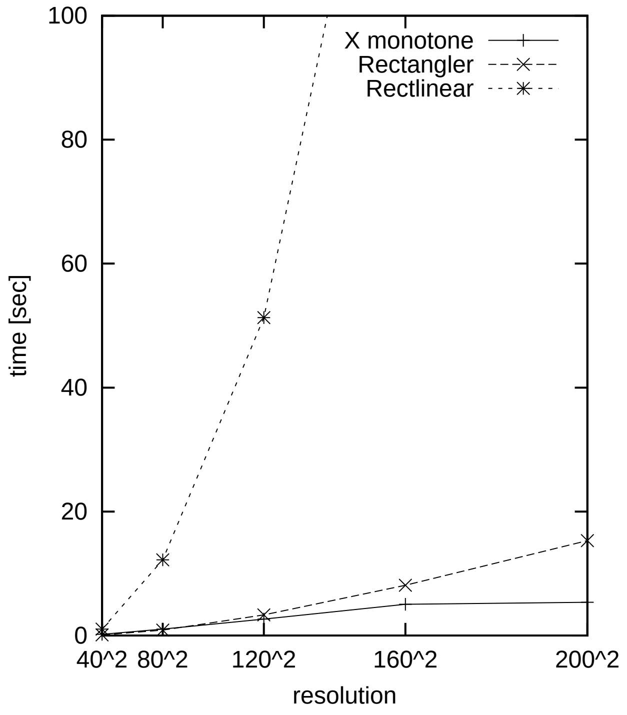


Figure 5.2: Time for Computing the Optimal Region in Regression (2)

probing increases very slowly thanks to the guided branch and bound algorithm. Figure 5.2 confirms that the CPU time follows our scale estimation.

At the root of a regression tree, we may need a large number of pixels to guarantee the specified record density. The number of records, however, decreases in the lower parts of the tree, and the number of pixels soon becomes less than 20×20 for most datasets; hence computing the optimal rectilinear convex region is generally not costly according to Table 5.1.

5.2 Multivariate Regression Trees

The author explores an improvement to the standard strategy of selecting internal-node tests during the regression tree construction. Conventional tree construction algorithms often use a test on a numerical conditional attribute with a cut value called "guillotine-cut." However, for the case when two numerical conditional attributes have a non-linear correlation with respect to the target attribute, trees with guillotine-cuts tend to become large and such large trees are in general inaccurate. In such a case, a test with a two-dimensional region for splitting data into two subsets may be more natural and effective. Therefore, the author proposed a regression tree by using the optimal region rule at each step of the tree construction.

Same as the multivariate decision trees in the previous chapter, we specify a number $N \leq \sqrt{n}$, and construct an almost equi-depth ordered bucketing of records for each numeric attribute A. For a pair of numeric attributes A and A', we have a pixel grid G of size $N \times N$ generated as a Cartesian product of the buckets. We consider a family $\mathcal{R}$ of grid regions of G. For each $R \in \mathcal{R}$, we consider a splitting that divides data into those inside the region R and those outside the region. Let R_{opt} be the region of $\mathcal{R}$ that maximizes the interclass variance function. The region R_{opt} and the associated splitting are called the *optimal region* with respect to $\mathcal{R}$ and the pair of attributes (A, A'). The author proposed regression trees that use such optimal regions for tests at internal nodes of the trees.

5.2.1 Spline Interpolation of Grid Regions

We compute the optimal region on the discretized pixel grid. However, such grid regions tend to be too coarse to describe regions if there are not enough records. In such cases, we can use an interpolation technique to get smooth regions. Since the number of records becomes smaller at nodes lower in trees, interpolation techniques are indispensable. Especially for regression problem, such interpolation techniques are effective as we empirically proved in Section 5.3.

The optimal region on a pixel grid is expressed by a disjunction of the form `(a1 < x < a2) and (b1 < y < b2)`. Therefore, we can plot two points $(\frac{a1+a2}{2}, b1)$ $(\frac{a1+a2}{2}, b2)$ for each column of the grid region. Then, we compute a smooth region based on those points.

We compute two kinds of spline curves, spline1 and spline2, of interpolated regions from a set of points on a pixel grid. Figure 5.3 shows examples of the two kinds of interpolated regions. In each pixel grid, the gray pixel grid region shows the optimal region. The points on the border of the gray region are used to compute the interpolated region.

Figure 5.4 illustrates how to compute a spline curve based on a set of points. We first order points of the set by x values and make a point sequence. Then, we split the point sequence into several intervals as in the figure. For spline1, we cut the sequence at the x value for each point. For spline2, we use the middle value of each two consecutive points in the sequence. For each interval i, we define a polynomial function of the form $f_i(x) = ax^3 + bx^2 + cx + d$ where a, b, c, and d are constants, and $x_i \leq x \leq x_{i+1}$ for spline1, and $\frac{x_i + x_{i+1}}{2} \leq x \leq \frac{x_{i+1}+x_{i+2}}{2}$ for spline2.

Each polynomial function, say $f_i(x)$, of spline1 is defined so that the following conditions are satisfied.

- $f_i(x)$ must pass through $\mathbf{x_i}$ and $\mathbf{x_{i+1}}$. ($\mathbf{x_i} = (x_i, y_i)$ is the i-th point of a point sequence.)
- $f_i'(x_{i+1}) = f_{i+1}'(x_{i+1})$. ($f_i'(x)$ is the 1st derivative of $f_i(x)$.)
- $f_i''(x_{i+1}) = f_{i+1}''(x_{i+1})$. ($f_i''(x)$ is the 2nd derivative of $f_i(x)$.)

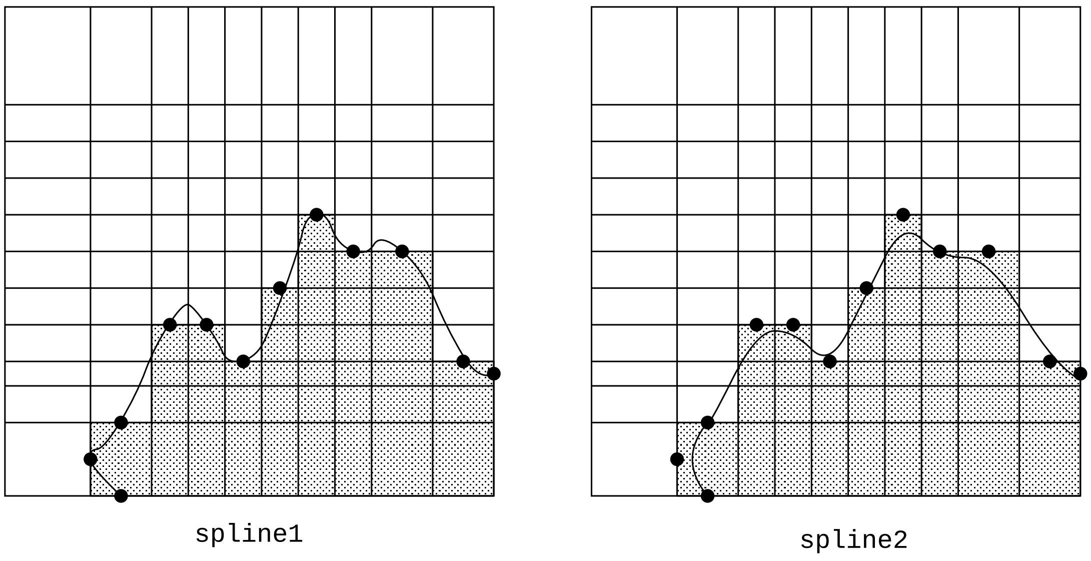


Figure 5.3: Interpolated Regions

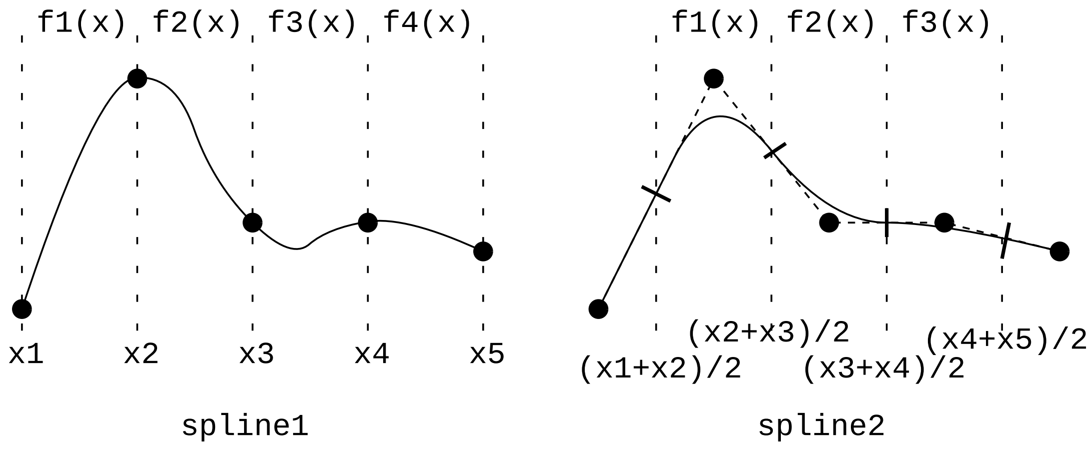


Figure 5.4: Spline Interpolation

Table 5.2: Summary of Regression Datasets

Dataset	#records	#attrs
add10	9792	10
abalone	4177	8
kin-8fh	8192	8
kin-8fm	8192	8
kin-8nh	8192	8
kin-8nm	8192	8
pumadyn-8fh	8192	8
pumadyn-8fm	8192	8
pumadyn-8nh	8192	8
pumadyn-8nm	8192	8

Each polynomial function of spline2 satisfies the following conditions.

- $f_i(x)$ must pass through $\frac{\mathbf{x_i+x_{i+1}}}{2}$ and $\frac{\mathbf{x_{i+1}+x_{i+2}}}{2}$.
- $f_i'(\frac{x_{i+1}+x_{i+2}}{2}) = f_{i+1}'(\frac{x_{i+1}+x_{i+2}}{2}) = \frac{y_{i+2}-y_{i+1}}{x_{i+2}-x_{i+1}}$.

Though interpolated regions may be worse than the original pixel grid regions with respect to the mean squared error of training datasets, those interpolated regions are much more accurate for test datasets, as we will show in the experimental results.

5.3 Experiments

5.3.1 Prediction Accuracy

We performed a ten-fold cross validation test for regression trees.

Datasets

We used several public datasets, summarized in Table 5.2, which were acquired from the following WWW site:

`http://www.cs.utoronto.ca/~ delve/data/datasets.html.`

We chose these datasets because almost all the attributes are numerical. As regions for splitting the datasets, we used x-monotone regions, rectilinear regions, and rectangular regions, and we assigned a value of 5 or 10 for the pixel density.

Minimal Potential Error

To examine the minimal potential error of trees, we performed the following ten-fold cross validation tests for each of the 10 datasets.

1. We randomly divided the original dataset into ten almost-equal-sized subsets.
2. We excluded one subset and used the other nine subsets as the training dataset to generate regression trees with each of 4 types of splitting: guillotine cuts, x-monotone regions, rectilinear convex regions, and rectangular regions.
3. We used the excluded subset as the test dataset to evaluate the regression tree, and computed the mean squared error against the test dataset. The absolute value of the mean squared error varies depending on the sample dataset. To compare the mean squared errors of different dataset, we used a "normalized" version called the *relative mean squared error* of a regression tree, which is defined as the mean squared error divided by the variance of the target attribute.
4. We returned to Step 2 and excluded a different subset. If there is no remaining subset for the test dataset, then we calculated the average and standard deviation of the ten results.

To examine the effectiveness of region splitting and eliminate the effect of pruning, we generated a number of regression trees for various values of the weight parameter, and then we compared the best accuracy of those various-sized regression trees. We call the best accuracy the minimal potential error.

For instance, let us consider the "abalone" dataset. Figure 5.5 shows how the relative mean squared error changes according to the value of the weight parameter. The graph for "Rectilinear (dens 5)" shows the average relative mean squared errors of the regression trees with rectilinear convex regions for a record density of five, and we see that the relative mean squared error is the smallest when the weight parameter is 0.006. Observe that the value increases for weight parameters less than 0.006, which indicates that the regression tree becomes larger and overfits the training data. Similarly, for x-monotone region splitting, rectangular region splitting, and guillotine cut splitting, we can find the respective regression tree with the smallest relative mean squared error. For the other datasets, we performed the same analysis (See Figures A.3, A.4 and A.5 in Appendix A.2).

Tables 5.3 to 5.5 summarize the results, showing the pairs of the average relative mean squared error and the average number of leaves for trees constructed with the best parameter values, i.e., weight parameter α and pixel density. The number in each parenthesis is the standard deviation over the ten results of cross validation. We underlined the smallest relative mean squared error among all split strategies (x-monotone, rectilinear convex, rectangular, and guillotine) for each dataset. Almost universally, the rectilinear, spline2 method yields the lowest error rates across all domains. While we do not indicate the significance of differences between methods within a domain because cross validation violates sample independence assumptions, the number of "wins" across domains for rectilinear, spline2 with respect to any other method is significant by a sign test at the 0.05 level ($N = 10$), though if kin-8XX and pumadyn-8XX are not considered independent, then the number of wins for rectilinear, spline2 is only marginally significant by a sign test ($N = 4$) with respect to each alternative strategy.

Table 5.3: Minimal Potential Error (X-monotone)

	Pixel Grid			
Dataset	Err		Size	
abalone	.523	(.0397)	10.0	(0.63)
add10	.141	(.00854)	117.6	(4.34)
kin-8fh	.450	(.0331)	49.5	(3.20)
kin-8fm	.224	(.0132)	157.8	(8.35)
kin-8nh	.661	(.0538)	35.7	(1.79)
kin-8nm	.497	(.0270)	57.0	(3.19)
pumadyn-8fh	.410	(.0187)	8.0	(0.00)
pumadyn-8fm	.0607	(.00726)	15.9	(0.49)
pumadyn-8nh	.350	(.0272)	7.8	(0.40)
pumadyn-8nm	.0526	(.00415)	20.0	(0.63)
	Spline1			
Dataset	Err		Size	
abalone	.523	(.0481)	9.1	(0.83)
add10	.135	(.00870)	147.7	(12.1)
kin-8fh	.449	(.0281)	38.8	(2.32)
kin-8fm	.222	(.0172)	95.2	(3.46)
kin-8nh	.648	(.0520)	43.0	(4.88)
kin-8nm	.487	(.0212)	63.4	(4.92)
pumadyn-8fh	.405	(.0232)	8.0	(0.00)
pumadyn-8fm	.0588	(.00587)	16.0	(0.00)
pumadyn-8nh	.344	(.0260)	8.3	(0.64)
pumadyn-8nm	.0487	(.00362)	39.4	(3.72)
	Spline2			
Dataset	Err		Size	
abalone	.505	(.0490)	8.6	(.917)
add10	.127	(.00969)	126.5	(9.87)
kin-8fh	.444	(.0190)	39.0	(1.55)
kin-8fm	.211	(.0127)	95.2	(3.54)
kin-8nh	.625	(.0374)	32.2	(1.89)
kin-8nm	.465	(.0206)	62.8	(4.02)
pumadyn-8fh	.403	(.0206)	8.0	(0.00)
pumadyn-8fm	.0574	(.00585)	17.0	(0.77)
pumadyn-8nh	.341	(.0250)	10.3	(0.90)
pumadyn-8nm	.0480	(.00338)	35.1	(2.70)

Table 5.4: Minimal Potential Error (Rectilinear)

	Pixel Grid			
Dataset	Err		Size	
abalone	.522	(.0436)	8.8	(0.98)
add10	.121	(.00618)	121.3	(8.00)
kin-8fh	.433	(.0241)	59.8	(2.14)
kin-8fm	.196	(.0106)	138.7	(4.45)
kin-8nh	.613	(.0437)	30.1	(2.07)
kin-8nm	.450	(.0365)	93.1	(3.27)
pumadyn-8fh	.401	(.0208)	8.0	(0.00)
pumadyn-8fm	.0600	(.00628)	15.8	(0.40)
pumadyn-8nh	.339	(.0287)	8.1	(0.30)
pumadyn-8nm	.0501	(.00373)	26.6	(1.36)
	Spline1			
Dataset	Err		Size	
abalone	.500	(.0427)	9.9	(0.94)
add10	.118	(.00719)	127.4	(6.83)
kin-8fh	.419	(.0274)	55.7	(1.85)
kin-8fm	.197	(.00865)	115.2	(5.31)
kin-8nh	.608	(.0422)	33.7	(2.57)
kin-8nm	.451	(.0234)	51.7	(1.19)
pumadyn-8fh	.400	(.0199)	8.2	(0.40)
pumadyn-8fm	.0581	(.00545)	16.3	(0.64)
pumadyn-8nh	.335	(.0260)	8.0	(0.00)
pumadyn-8nm	.0471	(.00428)	40.5	(4.15)
	Spline2			
Dataset	Err		Size	
abalone	.489	(.0463)	8.8	(0.75)
add10	.114	(.00458)	123.9	(6.76)
kin-8fh	.412	(.0223)	55.3	(2.93)
kin-8fm	.193	(.00793)	150.6	(6.59)
kin-8nh	.610	(.0388)	47.7	(2.41)
kin-8nm	.440	(.0265)	67.1	(3.91)
pumadyn-8fh	.400	(.0178)	8.0	(0.00)
pumadyn-8fm	.0572	(.00575)	19.5	(1.36)
pumadyn-8nh	.333	(.0274)	8.3	(0.46)
pumadyn-8nm	.0468	(.00456)	39.4	(2.33)

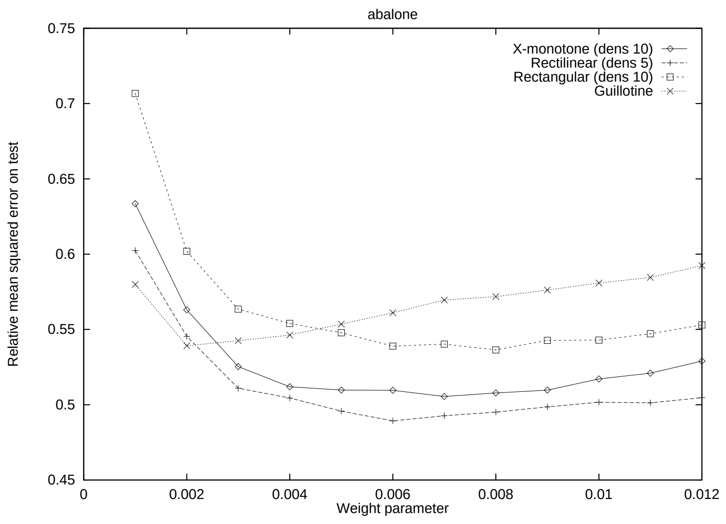


Figure 5.5: Accuracy for “abalone”

Table 5.5: Minimal Potential Error (Conventional)

	Rectangular			
Dataset	Err		Size	
abalone	.536	(.0352)	10.1	(0.700)
add10	.156	(.00932)	148.9	(5.36)
kin-8fh	.460	(.0256)	69.9	(3.33)
kin-8fm	.257	(.0186)	162.1	(3.88)
kin-8nh	.618	(.0398)	37.9	(1.81)
kin-8nm	.477	(.0217)	59.5	(3.47)
pumadyn-8fh	.409	(.0223)	19.8	(1.36)
pumadyn-8fm	.0655	(.00494)	62.6	(3.61)
pumadyn-8nh	.353	(.0304)	27.3	(2.00)
pumadyn-8nm	.0541	(.00335)	99.9	(2.47)

	Guillotine			
Dataset	Err		Size	
abalone	.539	(.0480)	38.9	(2.51)
add10	.185	(.00746)	540.2	(7.19)
kin-8fh	.479	(.0299)	128.3	(3.41)
kin-8fm	.249	(.0164)	919.9	(7.91)
kin-8nh	.655	(.0389)	88.4	(6.23)
kin-8nm	.541	(.0499)	203.6	(7.49)
pumadyn-8fh	.410	(.0184)	26.2	(2.09)
pumadyn-8fm	.0632	(.00683)	153.9	(5.09)
pumadyn-8nh	.355	(.0317)	44.1	(1.45)
pumadyn-8nm	.0535	(.00367)	185.0	(4.94)

Error Rates of Pruned Trees

For minimal potential error, the weight parameter value and record density value giving the minimum error ratio greatly depends on the dataset and construction method. In comparison of the error rate, we use a separate dataset, designated as the validation set, to find the adequate weight and density values for pruning.

We reserved one subset as the validation set, which is arbitrarily chosen from the nine training subsets during each trial of the ten-fold cross validation. That is, we used eight subsets for tree building and used one subset for finding the parameters, i.e., the weight parameter and the record density. We pruned a regression tree by the best parameter setting for the validation set and examined its error rate for the remaining test sets.

Tables 5.6-5.8 summarizes the results of the ten-fold cross validation experiments.

The cross validation results can be summarized as follows:

- X-monotone and rectilinear convex regions attain more accurate and smaller trees than conventional trees (at marginal significance for error rate by a sign test).
- Spline interpolation makes region splitting more accurate. Even though the spline2 sacrifices training accuracy (see Section 5.2.1), it attains the most accurate results (for tests) in most of the cases.
- Rectilinear regions are robust even if we use a low record density. Therefore, the results of a record density of five were more accurate than those of a record density of ten in most of the cases. As a result, rectilinear convex regions won over all of other splitting methods on all datasets.
- In case of x-monotone regions, on the other hand, using a low record density tends to give a higher error ratio because of overfitting. Therefore, the results for a record density of ten were more accurate than those for a record density of five in most of the cases.
- By using a validation set, we found parameter values that attained to satisfactory results as regards accuracy.

Table 5.6: Error of Pruned Tree (X-monotone)

	Pixel Grid			
Dataset	Err		Size	
abalone	.535	(.0573)	10.6	(4.27)
add10	.141	(.00986)	119.7	(26.1)
kin-8fh	.465	(.0233)	45.8	(18.0)
kin-8fm	.240	(.0194)	148.1	(89.9)
kin-8nh	.683	(.0348)	31.7	(11.4)
kin-8nm	.503	(.0315)	46.4	(11.9)
pumadyn-8fh	.416	(.0243)	7.90	(1.14)
pumadyn-8fm	.0611	(.00348)	17.9	(3.75)
pumadyn-8nh	.349	(.0242)	7.90	(1.04)
pumadyn-8nm	.0546	(.00568)	31.2	(9.31)
	Spline1			
Dataset	Err		Size	
abalone	.530	(.0615)	10.5	(4.30)
add10	.138	(.0145)	134.1	(23.9)
kin-8fh	.453	(.0187)	42.5	(9.70)
kin-8fm	.240	(.0129)	122.4	(34.8)
kin-8nh	.655	(.0303)	32.8	(10.0)
kin-8nm	.493	(.0322)	53.6	(17.5)
pumadyn-8fh	.410	(.0246)	8.90	(1.04)
pumadyn-8fm	.0601	(.00515)	19.1	(6.04)
pumadyn-8nh	.343	(.0190)	9.1	(2.66)
pumadyn-8nm	.0508	(.00529)	41.3	(11.3)
	Spline2			
Dataset	Err		Size	
abalone	.519	(.0432)	8.6	(1.56)
add10	.131	(.0108)	108.7	(18.2)
kin-8fh	.448	(.0148)	47.1	(20.8)
kin-8fm	.234	(.0126)	144.8	(63.4)
kin-8nh	.644	(.0393)	34.5	(21.3)
kin-8nm	.476	(.0241)	70.1	(36.6)
pumadyn-8fh	.402	(.0232)	8.30	(0.46)
pumadyn-8fm	.0584	(.00398)	16.4	(2.54)
pumadyn-8nh	.343	(.0195)	9.1	(2.07)
pumadyn-8nm	.0503	(.00409)	39.7	(16.0)

Table 5.7: Error of Pruned Tree (Rectilinear)

	Pixel Grid			
Dataset	Err		Size	
abalone	.511	(.0585)	9.00	(1.73)
add10	.122	(.00880)	115.6	(12.9)
kin-8fh	.439	(.0268)	55.6	(20.7)
kin-8fm	.213	(.0130)	137.2	(58.3)
kin-8nh	.620	(.0246)	36.9	(10.0)
kin-8nm	.459	(.0218)	54.6	(17.3)
pumadyn-8fh	.406	(.0253)	8.10	(0.54)
pumadyn-8fm	.0605	(.00396)	17.3	(2.05)
pumadyn-8nh	.338	(.0208)	8.2	(0.75)
pumadyn-8nm	.0524	(.00546)	37.1	(12.5)
	Spline1			
Dataset	Err		Size	
abalone	.501	(.0584)	9.90	(4.46)
add10	.121	(.00634)	111.8	(17.0)
kin-8fh	.437	(.0209)	46.1	(10.8)
kin-8fm	.208	(.0168)	115.8	(26.1)
kin-8nh	.620	(.0169)	40.8	(18.1)
kin-8nm	.461	(.0142)	63.2	(14.2)
pumadyn-8fh	.402	(.0257)	8.30	(0.90)
pumadyn-8fm	.0595	(.00378)	22.6	(6.30)
pumadyn-8nh	.338	(.0200)	9.1	(1.58)
pumadyn-8nm	.0483	(.00540)	33.9	(9.36)
	Spline2			
Dataset	Err		Size	
abalone	.511	(.0472)	14.9	(13.5)
add10	.117	(.00673)	124.3	(19.5)
kin-8fh	.431	(.0256)	52.8	(16.8)
kin-8fm	.211	(.0195)	130.2	(57.2)
kin-8nh	.615	(.0182)	37.9	(8.67)
kin-8nm	.456	(.0273)	60.3	(10.1)
pumadyn-8fh	.400	(.0231)	7.80	(0.60)
pumadyn-8fm	.0580	(.00345)	17.5	(1.86)
pumadyn-8nh	.336	(.0228)	10.3	(2.05)
pumadyn-8nm	.0482	(.00453)	33.7	(9.00)

Table 5.8: Error of Pruned Tree (Conventional)

	Rectangular			
Dataset	Err		Size	
abalone	.554	(.0859)	15.0	(3.13)
add10	.164	(.00747)	177.6	(36.4)
kin-8fh	.473	(.0353)	60.0	(14.5)
kin-8fm	.257	(.0193)	156.6	(43.2)
kin-8nh	.633	(.0290)	36.4	(12.9)
kin-8nm	.490	(.0281)	58.2	(10.6)
pumadyn-8fh	.414	(.0307)	18.8	(2.75)
pumadyn-8fm	.0653	(.00409)	56.8	(5.60)
pumadyn-8nh	.353	(.0188)	23.9	(4.59)
pumadyn-8nm	.0584	(.00592)	84.7	(18.9)

	Guillotine			
Dataset	Err		Size	
abalone	.545	(.0701)	30.2	(9.31)
add10	.191	(.00748)	425.6	(124.2)
kin-8fh	.490	(.0200)	121.7	(62.2)
kin-8fm	.252	(.0133)	800.0	(289.3)
kin-8nh	.641	(.0210)	87.2	(20.3)
kin-8nm	.533	(.0326)	125.7	(55.8)
pumadyn-8fh	.408	(.0239)	24.9	(4.18)
pumadyn-8fm	.0646	(.00467)	99.8	(21.4)
pumadyn-8nh	.365	(.0203)	46.6	(14.9)
pumadyn-8nm	.0566	(.00512)	190.1	(68.3)

Table 5.9: Regression Tree Construction Time (1)

#records	X-monotone	Rectilinear	Guillotine
2000	33.3	31.5	4.10
4000	80.2	80.1	9.65
6000	130	136	15.9
8000	180	188	22.8

Table 5.10: Regression Tree Construction Time (2)

#attributes	X-monotone	Rectilinear	Guillotine
4	58.7	64.4	17.7
6	85.8	91.8	19.3
8	128	136	20.8
10	180	186	22.3

5.3.2 Performance Results

The next experiment examines the overall performance, CPU time taken, in tree construction. At each node of a regression tree, we first prepared grid regions for each combination of numeric attributes, and then computed the optimal region. The grid preparation can be done by scanning all the records at a node just once. However, we have to examine all pairs (permutations for x-monotone regions) of two numeric attributes to find the optimal region. Thus, in general, the number of attributes dramatically affects the performance if we use region splitting.

Table 5.9 compares the time (sec.) taken to construct trees by using datasets with different numbers of records. We randomly selected records from the "add10" dataset to generate datasets having different numbers of records, and used those datasets to construct a regression tree by performing region splitting with a record density of five. We compared the time taken to construct unpruned large trees using x-monotone (spline2) regions, rectilinear convex (spline2) regions, and guillotine cuts. The result shows that tree construction time is a little more than our scale estimation, because trees from larger datasets tend to become larger.

Table 5.10, on the other hand, compares the performance using datasets with different numbers of attributes. We used 8000 records from the "add10" dataset, and constructed trees using N numerical attributes in the dataset. Though the time complexity for finding the optimal region in each split is almost linear of the combinations (permutations) of attributes, the overall tree construction time seems to increase sub-linearly relative to the number of combinations. One of the main reasons is that the cost of computing the optimal region is not dominant if there are few attributes.

Chapter 6

Conclusion

In many businesses, information or knowledge that can be extracted from databases are important with no doubt. Therefore, data mining technologies have attracted many enterprises that use databases or data warehouses.

Data mining covers technologies for finding, frequent or rare patterns, modeling and predicting value of an attribute, grouping or clustering data, classifying data, and so forth. Most of these have been widely studied in the field of databases, statistics, and machine learning. Data mining, in general, is focusing on efficiency so that we can handle emerging huge databases whose size is too large to be processed by the conventional techniques.

In this dissertation, the author explores modeling and prediction tasks for classification and regression problems. The most important problem in such tasks is how to find good discriminant rules. Many researches have been investigated in machine learning, database, and statistics literatures.

In general, a database contains several attributes and among them, there are correlated attributes. Especially for numerical attributes, we have to handle correlated attributes. Therefore, some researches have been focusing on rules that can be defined on multiple attributes, i.e., multivariate rules. Most of the researches considered linear multivariate discriminant rules and are effective only for linear correlations. However, we have various correlations that can not be handled by conventional methods. In addition, recent databases or data warehouses contain a lot of attributes and there are many correlated numerical and categorical attributes. Therefore, methods for handling correlated attributes become much more important.

The author proposed efficient algorithms for finding discriminant multivariate rules that can be used for huge databases. In particular, for each pair of correlated numerical attributes, the author proposed to use regions on the two dimensional plain of the pair of attributes. By using regions, we can handle correlations with various shapes, which can be non-linear ones. Moreover, we can find and grasp how the data is correlated through the two dimensional visualization, which is important for data analysts who want some insights in the correlations. The author also considered efficient algorithms for finding multivariate rules on multiple categorical conditional attributes.

The author used such multivariate discriminant rules for classification and regression trees. Diverse experiments confirm that the use of region splitting gives compact and accurate regression trees in many domains.

6.1 Remarks from Experiments

Experiments using diverse datasets confirmed that classification and regression trees with region splitting appear to be more accurate and smaller than conventional ones. However, in order to use region splitting, we have to spend additional computation time that is proportional to the number of combinations of numerical conditional attributes.

In the experiments for classification, as can be seen in Figures A.1 and A.2, the results show the substantial capability of our method. Moreover, in experiments, the pruned trees shown in Table 4.5, achieved greater error reduction, especially in cases where there were a sufficient number of records. As for the experiments for regression problems, region splitting trees generally reduced the error ratio by around 10 to 20%. One case of which achieves 48% reduction. Those reductions suggest the advantages of region splitting trees. In many applications, the improvements will be worth the computational cost if there are not too many numeric attributes.

Another important advantage of region splitting is comprehensibility of correlations among conditional attributes, which we did not evaluate objectively. Though the optimal region in each node of a tree itself is complicated, arguably we can easily understand two-dimensional visual representations like examples in Section 1.2.2.

By introducing region splitting in classification and regression trees, we have to consider additional parameters: record density, type of regions, and type of splines. Choice of those parameters affects construction time and prediction accuracy of the trees.

As for the types of splines, the computational costs of the three types, including no interpolation, are small and negligible. The spline2 achieved better results than the others in almost all datasets and experimental conditions. We examined the effectiveness of interpolation in classification problems. Since target attribute has categorical values, in most of the cases there are only a few values, we could not find notable difference in accuracy.

As for the other experimental conditions, there are tradeoffs. If we use lower record density, we can find finer pixel regions. However, it takes much time and it tends to overfit if record density is too fine or if the number of records is too small. According to the experimental results, we should use a record density of more than five. Since the number of records becomes smaller in the lower nodes of a tree, the pixel grid becomes too coarse with less than five by five. In such cases, we abandoned region splitting and used conventional guillotine cutting.

An x-monotone region can express more variety of shapes than a rectilinear region. However, it often overfits, especially when we use a low record density. In the experiments, because of overfitting, the accuracy results of x-monotone regions are worse than that of rectilinear regions. If we use an x-monotone region, we should use a record density of more than ten. A notable advantage of using x-monotone regions is their fast computation time. If the number of records becomes huge, say more than a million, it is better to use x-monotone regions.

6.2 Recent Progress in Modeling and Prediction Tasks

6.2.1 Fast Tree Construction Algorithms

Classification and regression trees have been one of the most popular models for approximating value of the target attribute. Therefore, some researchers have been focusing on fast tree construction algorithms, which work for huge databases.

Recently, Mehta et al. [MAR96] proposed an efficient scalable implementation called SLIQ. It maintains a data structure called attribute lists, one for each conditional attribute, and it computes the optimal rule for a node by scanning the attribute lists once. Once we construct the attribute lists, we do not have to scan the database. Therefore, SLIQ can handle a database with 10 million records and 400 attributes. Later, Shafer et al. parallelized implementation of SLIQ [SAM96] and called it SPLINT.

Gehrke et al. [GRG98] proposed another scalable implementation called RainForest. RainForest adaptively changes a data management method based on the size of main memory and training data during the tree construction. Gehrke et al. [GGRL99] also considered another efficient tree construction method that constructs an initial tree by using small sample data and then refines the tree by using other sample. The idea of using small sample for a huge database was also proposed by Breiman [Bre99] before their method.

Rastogi and Shim [RS98] considered a method to construct and prune classification trees to eliminate time for pruning phase of conventional tree construction methods.

6.2.2 Toward Accurate Predictions

Technologies for accurate prediction are also popular research area in machine learning, data mining, and statistics. Among them, as technologies that are strongly related to classification and regression trees, Freund and Schapire proposed an accurate classification system AdaBoost that used a method called *boosting* [FS97]. Boosting is a prediction method based on weighted majority voting. It constructs strong classifier, i.e., classification system, by combining a collection of weak classifiers. AdaBoost initially assign an equal weight to each record. Then, it generates a classifier whose error ratio of which is smaller than that of random guessing. Next, it increases weights of misclassified records to relatively higher than the others. AdaBoost iterates the weight process several times. Then, it performs weighted majority voting by classifiers and predicts unknown target value.

Though the boosting is not a mechanism for classification and regression trees, trees are suitable for weak classifiers of the boosting because trees can be easily constructed. Breiman [Bre96], he called it *bagging*, proposed such a strong classifier based on classification and regression trees. Quinlan also applied the technique and proposed C5.0 [Qui96].

6.2.3 Other Progresses

There are some progresses for handling correlated attributes. Since those are related works of this dissertation, the author summarized the related works and compared with the proposed algorithms in Section 3.3.1 and 3.4.3. Notice that the problem for finding discriminant rules for multiple categorical conditional attributes are essentially the same as the problem for finding rules for single categorical conditional attribute that has many distinct values. Therefore, the author also compared with the technologies for such equivalent problem in Section 3.3.1.

All of the related works and the proposed works in this dissertation assumed correlations on multiple categorical conditional attributes or multiple numerical conditional attributes. Brim et al. [BRS99] considered multivariate rules on both categorical and numerical conditional attributes. However, they did not consider optimization based on classification and regression criteria. The problem of how to find such multivariate rule on categorical and numerical attributes efficiently is still open.

6.3 Future Direction

The proposed algorithms for finding discriminant rules on categorical conditional attributes can find high quality rules than any other conventional methods for small number of classes, i.e., the number of distinct values of the target categorical attribute. In this case, the high quality rules are the rules that achieve high values of objective functions for classification and regression.

If the number of classes becomes larger, for example more than 10, the algorithms do not perform well. No other algorithm can perform well in such condition so far. Therefore, the problem is still open for the case when the number of classes is large. However, we do not often use such categorical attribute as the target attribute for classification tasks in practice. For a regression problem, we can treat the problem as a two classes classification problem. Therefore, we have efficient algorithms such as the method of Breiman et al. [BFOS84].

As for numerical conditional attributes, the author proposed to use pixel grid regions as discriminant rules for classification and regression problems. The author proposed efficient algorithms for computing high quality x-monotone, rectilinear, and rectangular regions. We can express various types of non-linear correlation on two numerical conditional attributes. If we consider non-linear correlation on more than two numerical conditional attributes, the complexity of the optimization problem becomes exponentially harder than the case of two attributes. A heuristic proposed by Ittner and Schlosser [IS96] is an alternative to the problem, though it abandons the optimality. Same as the case of categorical conditional attributes, when the number of classes becomes large, the optimization problem becomes much harder and the problem is still open so far.

Bibliography

[AAD$^+$96] Sameet Agrawal, Rakesh Agrawal, Prasad M. Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi. On the computation of multidimensional aggregates. In *Proceedings of the 22nd VLDB Conference*, pages 506–521, 1996.

[AAP98] Ramesh Agarwal, Charu Aggarwal, and V. V. V. Prasad. Efficiently mining long patterns from databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 85–93, 1998.

[ACKT96] Tetsuo Asano, Danny Chen, Naoki Katoh, and Takeshi Tokuyama. Polynomial-time solutions to image segmentations. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 104–113, 1996.

[AIS93a] Rakesh Agrawal, Tomasz Imielinski, and Arum Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, December 1993.

[AIS93b] Rakesh Agrawal, Tomasz Imielinski, and Arum Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 207–216, 1993.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, pages 487–499, 1994.

[AS96] Rakesh Agrawal and John Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6), December 1996.

[AT94a] Tetsuo Asano and Takeshi Tokuyama. Partial construction of an arrangement of lines and its application to optimal partitioning of bichromatic point set. *IEICE Transactions E*, 77:595–600, 1994.

[AT94b] Tetsuo Asano and Takeshi Tokuyama. Topological walk revisited. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, pages 1–6, 1994.

[BA99] Roberto J. Bayardo and Rakesh Agrawal. Mining the most interesting rules. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 145–154, 1999.

[Bay98] Roberto J. Bayardo. Efficiently mining long patterns from databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 85–93, 1998.

[BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM*, 36:929–965, 1989.

[BFOS84] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

[BM98] Catherine L. Blake and Christopher J. Merz. UCI repository of machine learning databases, 1998.

[BMS97] Sergay Brin, Rajeev Motowani, and Craig Silverstein. Beyond market basket: Generalizing association rules to correlations. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 265–276, 1997.

[BMUT97] Sergay Brin, Rajeev Motowani, Jeffery Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 255–264, 1997.

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996.

[Bre99] Leo Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1/2):85–103, 1999.

[BRS99] Sergey Brin, Rajeev Rastogi, and Kyuseok Shim. Mining optimized gain rules for numeric attributes. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 135–144, 1999.

[BU95] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Machine Learning*, 19:45–77, 1995.

[DE93] David P. Dobkin and David Eppstein. Computing the discrepancy. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 47–52, 1993.

[DEY86] David Dobkin, Herbert Edelsbrunner, and Chee-Keng Yap. Probing convex polytopes. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 387–392, 1986.

[FMMT96a] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Constructing efficient decision trees by using optimized association rules. In *Proceedings of the VLDB Conference*, pages 146–155, 1996.

[FMMT96b] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 13–23, 1996.

[FMMT96c] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Finding optimal intervals using comptational geometry. In *Proceedings of the International Symposium on Algorithm and Computing*, pages 55–64, 1996.

[FMMT96d] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Mining optimized association rules for numeric attributes. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 182–191, 1996.

[FMMT96e] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Sonar: System for optimized numeric association rules. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, page 553, 1996.

[FMMT99] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Mining optimized association rules for numeric attributes. *Journal of Computer and System Sciences*, 58(1):1–12, February 1999.

[FMMT01] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Data mining using two-dimensional optimized association rules. *ACM Transactions on Database Systems*, 26:179–213, 2001.

[FPSSU96] Usama Fayyad, Gregory Piatetsky-Shapiro, Padhr Smyth, and Ramasamy Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* MIT Press, Cambridge, MA., 1996.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[GBLP95] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Technical report, Microsoft, November 1995.

[GGRL99] Johannes Gehrke, Venkatesh Ganti, Raghu Ramakrishnan, and Wei-Yin Loh. BOAT - optimistic decision tree construction. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 169–180, 1999.

[GHQ95] Ashish Gupta, Venky Harinarayan, and Dallan Quass. Aggregate-query processing in data warehousing environments. In *Proceedings of the 21st VLDB Conference*, pages 358–369, 1995.

[GJ79] Michael R. Garey and David S. Johnson. *Computer and Intractability. A Guide to NP-Completeness.* W. H. Freeman, 1979.

[GRG98] Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. RainForest - a framework for fast decision tree construction of large datasets. In *Proceedings of the VLDB Conference*, pages 416–427, 1998.

[Han98] David J. Hand. Data mining: Statistics and more? *The American Statistician*, 52(2), May 1998.

[HII95] Susumu Hasegawa, Hiroshi Imai, and Masaki Ishiguro. ϵ-approximations of k-label spaces. *Theoretical Computer Science*, 137:145–157, 1995.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 1–12, 2000.

[HR76] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5:15–17, 1976.

[HRU96] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implementing data cubes efficiently. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 205–216, 1996.

[HW87] David Haussler and Emo Welzl. Epsilon-nets and simplex range queries. *Discrete and Computational Geometry*, 2:127–151, 1987.

[Inm92] William H. Inmon. *Building the Data Warehouse.* John Wiley & Sons, 1992.

[IS96] Andreas Ittner and Michael Schlosser. Non-linear decision trees – NDT. In *Proceedings of the International Conference on Machine Learning*, pages 252–258, 1996.

[LHM99] Bing Liu, Wynne Hsu, and Yiming Ma. Pruning and summarizing the discovered associations. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 125–134, 1999.

[MAR96] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proceedings of the International Conference on Extending Database Technology EDBT*, pages 18–32, 1996.

[MFMT97] Yasuhiko Morimoto, Takeshi Fukuda, Shinichi Morishita, and Takeshi Tokuyama. Implementation and evaluation of decision trees with range and region splitting. *Constraint, An International Journal*, 2(3/4):401–427, December 1997.

[MIM97] Yasuhiko Morimoto, Hiromu Ishii, and Shinichi Morishita. Efficient construction of regression trees with range and region splittng. In *Proceedings of the VLDB Conference*, pages 166–175, 1997.

[Min89] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.

[Mit97] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[MP91] Patrick M. Murphy and Michael J. Pazzani. Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In *Proceedings of the International Wrokshop on Machine Learning*, pages 183–187, 1991.

[MRA95] Manish Mehta, Jorma Rissanen, and Rakesh Agrawal. MDL-based decision tree pruning. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 216–221, 1995.

[MS00] Shinichi Morishita and Jun Sese. Traversing itemset lattice with statistical metric pruning. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS-00)*, pages 226–236, 2000.

[PCY95] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash-based algorithm for mining association rules. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 175–186, 1995.

[PS85] Franco P. Preparata and Michael I. Shamos. *Computational Geometry, An Introduction.* Springer-Verlag, 1985.

[PS91] Gregory Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In *Knowledge Discovery in Databases*, pages 229–248, 1991.

[PSF91] Gregory Piatetsky-Shapiro and William J. Frawley, editors. *Knowledge Discovery in Databases.* AAAI Press, 1991.

[QR89] J. Ross Quinlan and Ronald L. Rivest. Inferring decision trees using minimum description length principle. *Information and Computation*, 80:227–248, 1989.

[Qui86a] J. Ross Quinlan. The effect of noise on concept learning. *Machine Learning An Artificial Intelligence Approach*, 2:149–166, 1986.

[Qui86b] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui87] J. Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Qui96] J. Ross Quinlan. Bagging, boosting, and C4. 5. In *Proceedings of the National Conference on Artificial Intelligence and the Innovative Applications of Artificial Intelligence Conference*, pages 725–730. AAAI Press / MIT Press, 1996.

[RS98] Rajeev Rastogi and Kyuseok Shim. PUBLIC: A decision tree classifier that integrates building and pruning. In *Proceedings of the VLDB Conference*, pages 404–415, 1998.

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 1–12, 1996.

[SAM96] John Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proceedings of the VLDB Conference*, pages 544–555, 1996.

[SK96] Takayuki Shintani and Masaru Kitsuregawa. Hash based parallel algorithms for mining association rules. In *Proceedings of International Conference on Parallel and Distributed Information Systems*, pages 19–30, 1996.

[STA98] Sanita Sawaragi, Shiby Thomas, and Rakesh Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 343–354, 1998.

[YFM$^+$97] Kunikazu Yoda, Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Computing optimized rectilinear regions for association rules. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pages 96–103, 1997.

[ZPOL97] Mohammed J. Zaki, Srinivansan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pages 283–286, 1997.

Appendix A

Detailed Experimental Results

A.1 Cross Validation Results for Classification

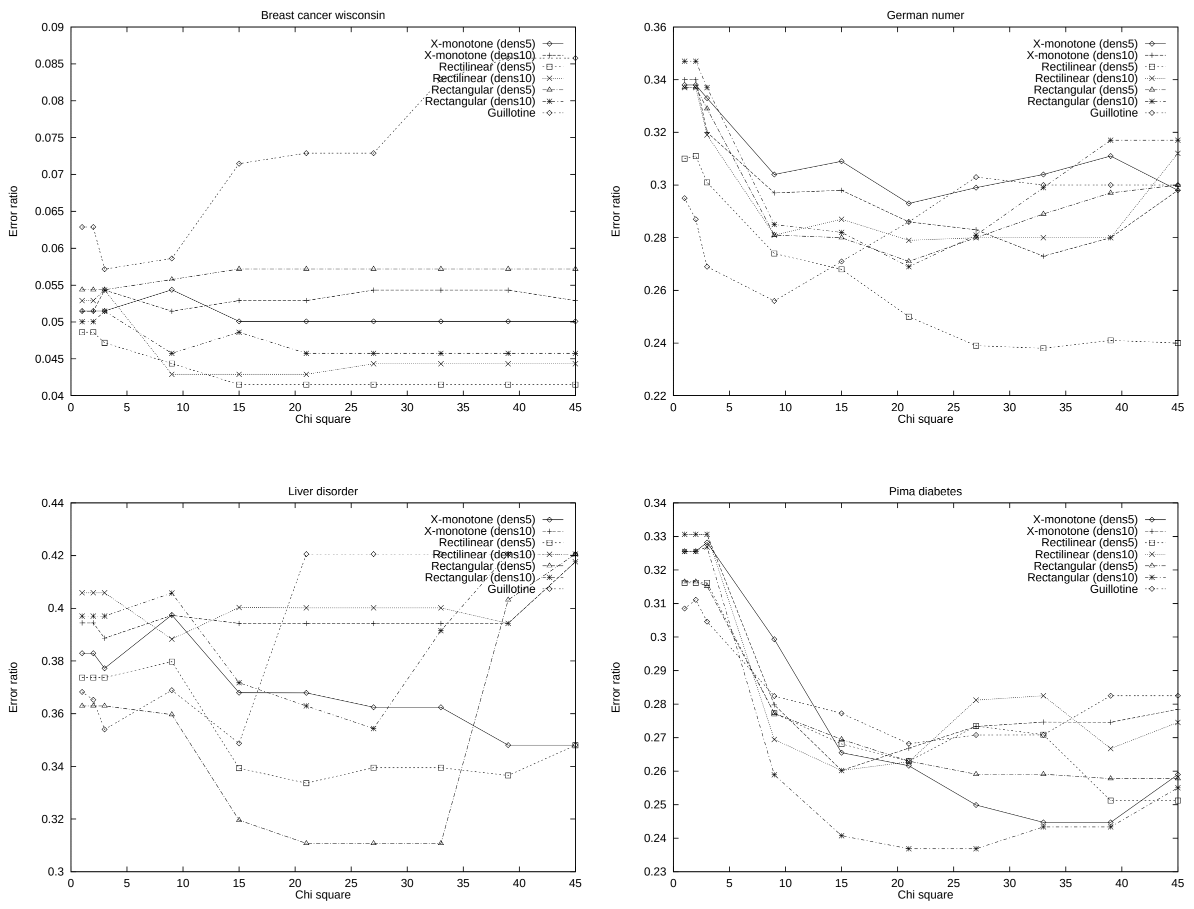


Figure A.1: Prediction Accuracy for Classification Problem (1)

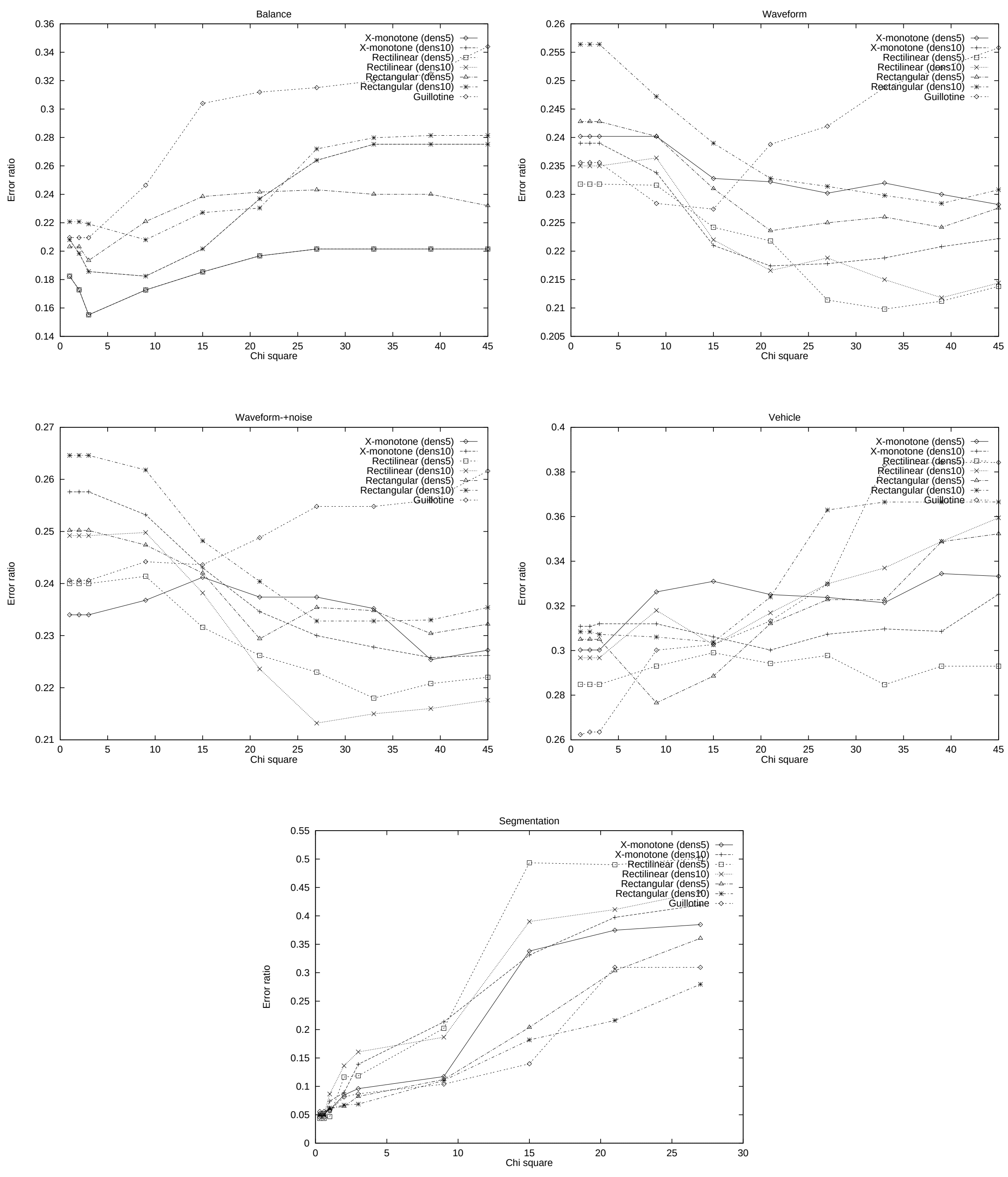


Figure A.2: Prediction Accuracy for Classification Problem (2)

A.2 Cross Validation Results for Regression

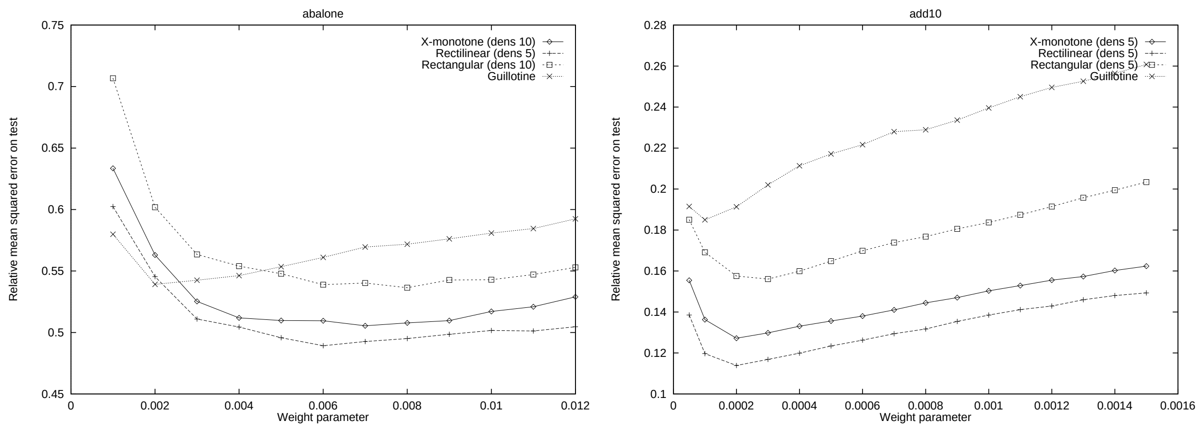


Figure A.3: Prediction Accuracy for Regression Problem (1)

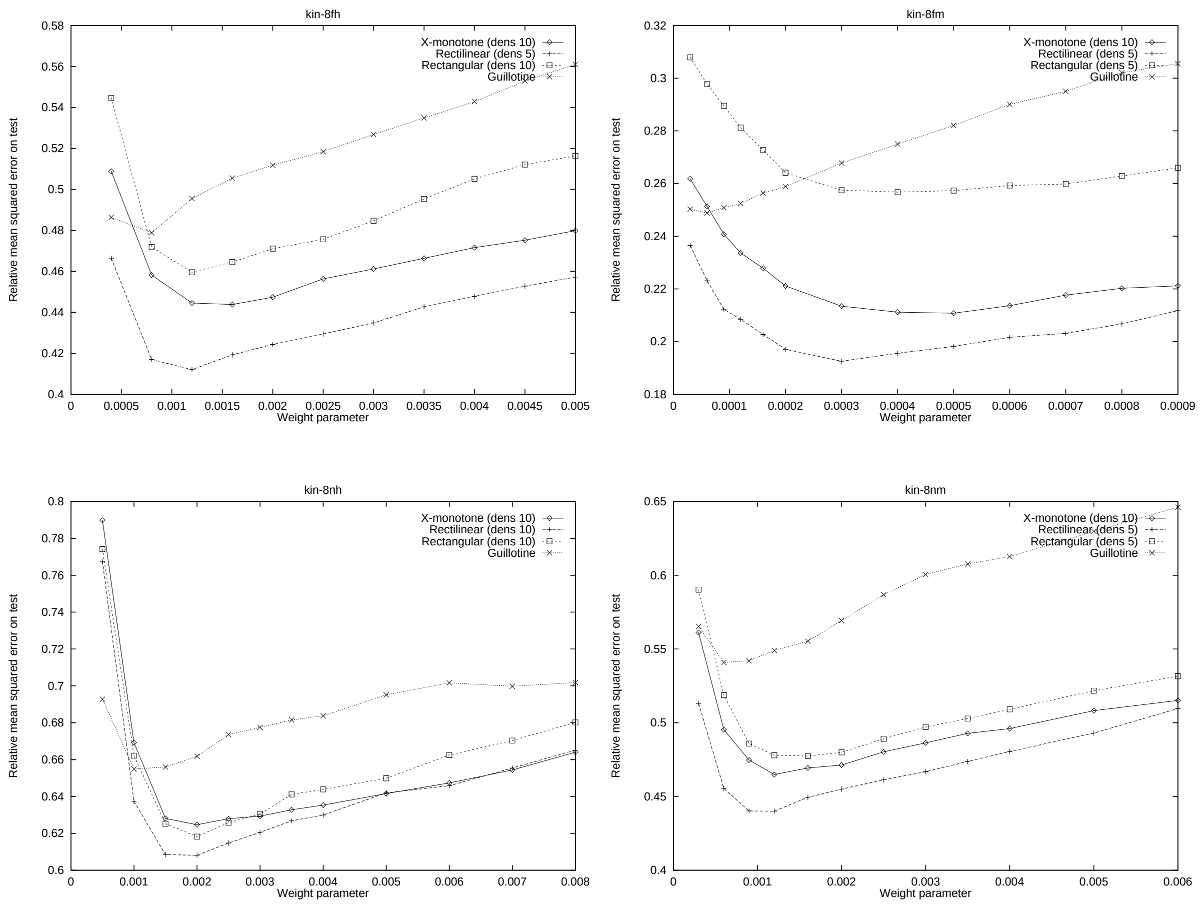


Figure A.4: Prediction Accuracy for Regression Problem (2)

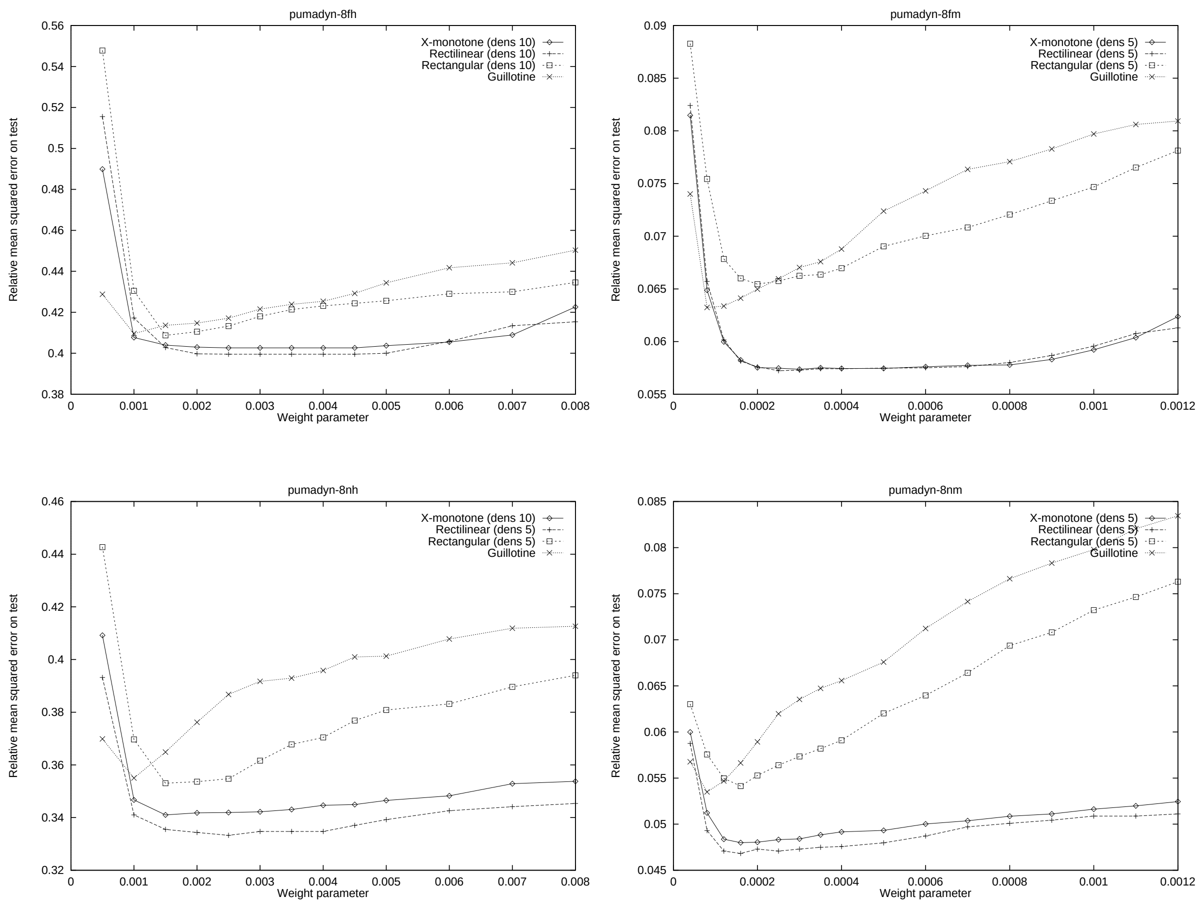


Figure A.5: Prediction Accuracy for Regression Problem (3)