Database Mining: A Performance Perspective

Rakesh Agrawal *Tomasz Imielinski*[*] *Arun Swami*

IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Abstract

We present our perspective of database mining as the confluence of machine learning techniques and the performance emphasis of database technology. We describe three classes of database mining problems involving classification, associations, and sequences, and argue that these problems can be uniformly viewed as requiring discovery of rules embedded in massive data. We describe a model and some basic operations for the process of rule discovery. We show how the database mining problems we consider map to this model and how they can be solved by using the basic operations we propose. We give an example of an algorithm for classification obtained by combining the basic rule discovery operations. This algorithm not only is efficient in discovering classification rules but also has accuracy comparable to ID3, one of the current best classifiers.

Index Terms. database mining, knowledge discovery, classification, associations, sequences, decision trees

[*]Current address: Computer Science Department, Rutgers University, New Brunswick, NJ 08903

1 Introduction

Database technology has been used with great success in traditional business data processing. There is an increasing desire to use this technology in new application domains. One such application domain that is likely to acquire considerable significance in the near future is *database mining* [12] [3] [5] [8] [9] [11] [15] [16] [18] [19]. An increasing number of organizations are creating ultra large data bases (measured in gigabytes and even terabytes) of business data, such as consumer data, transaction histories, sales records, etc. Such data forms a potential gold mine of valuable business information.

Unfortunately, the database systems of today offer little functionality to support such "mining" applications. At the same time, statistical and machine learning techniques usually perform poorly when applied to very large data sets. This situation is probably the main reason why massive amounts of data are still largely unexplored and are either stored primarily in an offline store or are on the verge of being thrown away.

We present in this paper our perspective of database mining as the confluence of machine learning techniques and the performance emphasis of database technology. We argue that a number of database mining problems can be uniformly viewed as requiring discovery of rules embedded in massive data. We describe a model and some basic operations for the process of rule discovery. We also show how these database mining problems map to this model and how they can be solved by using the basic operations we propose.

Our view of database mining complements the perspective presented in [9, 19]. Both of these papers argue for an iterative process for mining with a human in the loop. The user begins with a hypothesis and uses data to refute or confirm the hypothesis. The hypothesis is refined, depending on the response and this process continues until a satisfactory theory has been obtained. The emphasis in [19] is on having a declarative language that makes it easier to formulate and revise hypotheses. The emphasis in [9] is on providing a large bandwidth between the machine and human so that user-interest is maintained between successive iterations. Although we do not discuss this aspect in detail in this paper, we admit the possibility of human intervention in the mining process. This intervention can be in the form of domain knowledge to guide the mining process, or additional knowledge as the rules are mined. There has been work on quantifying the "usefulness" or "interestingness" of a rule [17]. These ideas may be built as filters on top of the kernel of the rule discovery techniques.

The rest of the paper is organized as follows. In Section 2, we present three classes of database mining problems involving classification, associations, and sequences. In Section 3, we present a unifying framework and show how these three classes of problems can be uniformly viewed as requiring discovery of rules. In Section 4, we introduce operations that may form the computational kernel for the process of rule discovery. We show how the database mining problems under consideration can be solved by combining these operations. To make the discussion concrete, we consider the classification problem in detail in Section 5, and present a concrete algorithm for classification problems obtained by combining these operations. We show that the classifier so obtained is not only efficient but has a classification accuracy comparable to the well-known classifier ID3 [14]. We present our conclusions and directions for future work in Section 6.

2 Database Mining Problems

We present three classes of database mining problems that we have identified by examining some of the often cited applications of database mining. These classes certainly do not exhaust all database mining applications, but do capture an interesting subset of them. In Section 3, we will present a unifying framework for studying and solving these problems.

2.1 Classification

The *classification* problem [6] [10] [11] [18] involves finding rules that partition the given data into disjoint groups. As an example of a classification problem, consider the store location problem. It is assumed that the success of the store is determined by the neighborhood characteristics, and the company is interested in identifying neighborhoods that should be the primary candidates for further investigation for the location of a proposed new store. The company has access to a neighborhood database. It first categorizes its current stores into successful, average, and unsuccessful stores. Based on the neighborhood data for these stores, it then develops a profile for each category of stores, and uses the profile for the successful stores to retrieve candidate neighborhoods. Other applications involving classification include credit approval, treatment-appropriateness determination, etc.

A variation of the classification problem is the *BestN* problem [1]. A company may be interested in finding the best N candidates to whom a ski package should be mailed. First a small number of ski packages are mailed to a selected sample of the population and then a profile of likely positive respondents is obtained. Such a profile is usually built as a disjunction of conjunctions of attribute value ranges characterizing individuals in the population. For instance, the profile of likely respondents to the ski package may be the union of all individuals with age between 30 and 40 and income above 40K a year with all individuals who drive a sports car. Notice that both conditions generate rules that have the given condition as the antecedent of the rule and "positive response" as the consequent. The confidence factor associated with each term of the disjunction can be used to develop an order in which the terms in the disjunction are applied to the data for obtaining the best N candidates.

2.2 Associations

Consider a supermarket setting where the database records items purchased by a customer at a single time as a transaction. The planning department may be interested in finding "associations" between sets of items with some minimum specified confidence. An example of such an association is the statement that 90% of transactions that purchase bread and butter also purchase milk. The antecedent of this rule consists of bread and butter and the consequent consists of milk alone. The number 90% is the confidence factor of the rule. Usually, the planner will be interested not in a single rule but rather in sets of rules satisfying some initial specifications. Here are some other examples of the problem of finding associations (we have omitted the confidence factor specification):

- Find all rules that have "Diet Coke" as consequent. These rules may help plan what the store should do to boost the sale of Diet Coke.

- Find all rules that have "bagels" in the antecedent. These rules may help determine what products may be impacted if the store discontinues selling bagels.
- Find all rules that have "sausage" in the antecedent and "mustard" in the consequent. This query can be phrased alternatively as a request for the additional items that have to be sold together with sausage in order to make it highly likely that mustard will also be sold.
- Find all the rules relating items located on shelves A and B in the store. These rules may help shelf planning by determining if the sale of items on shelf A is related to the sale of items on shelf B.
- Find the "best" k rules that have "bagels" in the consequent. Here, "best" can be formulated in terms of the confidence factors of the rules, or in terms of their support, i.e., the fraction of transactions satisfying the rule.

Note that a transaction need not necessarily consist of items bought together at the same point of time. It may consist of items bought by a customer over a period of time. Examples include monthly purchases by members of a book or music club.

2.3 Sequences

Another major source of database mining problems is ordered data, such as temporal data related to stock market and point of sales data. Here is an example of a rule over stock market data:

When AT&T stock goes up on 2 consecutive days and DEC stock does not fall during this period, IBM stock goes up the next day 75% of the time.

Another example of such a query in the retailing situation is: "What items are sold in sequence?", to which the response could be "dress followed by matching shoes."

We now present a unifying framework and show that the above three classes of problems can be studied in this framework of rule discovery.

3 A Unifying Framework

Let O be a set of objects. Denote by $D(m)$ and $R(m)$ the domain and range respectively of a method m. Let M be a set of methods whose domain is either O or $R(m)$ for some m in M. We denote by $o.m$ the result of the application of method m on object o.

A *formula* is of the form $p(o.l)$, where l is a composition of methods from M and p is a predicate defined on $R(m)$ for some m in M. An example of a predicate is *senior* where *senior(t.age)* is true when the age of the object t is above 65. Here *age* is a method in M. As another example, consider a method *year-of-birth* and a method *decade* that maps a year to the corresponding decade. Then, given a predicate *bohemian* that is true if the decade is the sixties and denoting method composition by "$\circ$", we have that *bohemian(t.decade* $\circ$ *year-of-birth)* is a formula.

By a *rule*, we mean a statement of the form $F(o) \Longrightarrow G(o)$ where F is a conjunction of formulas and G is a formula. The rule $r : F(o) \Longrightarrow G(o)$ is satisfied in the set of objects O with the confidence factor $0 \leq c \leq 1$ iff at least c% of objects in O that satisfy F also satisfy G.

Note that all formulas in our rules are unary, with a single variable ranging through the set of objects O. The reason for this restriction is that the business applications that we have considered so far lead to unary rules.

Given the set of objects O, we will be interested in generating all the rules that satisfy certain additional constraints of two different forms:

- *Syntactic Constraints*: These constraints involve restrictions on predicates and methods that can appear in the rule. For example, we may be interested only in rules that have method x appearing in the consequent, or rules that have a method y appearing in the antecedent. Combinations of the above constraints are also possible — we may request all rules that have methods from some predefined set X to appear in the consequent, and methods from some other set Y to appear in the antecedent.
- *Support Constraints*: These constraints concern the number of objects in O that support a rule. The *support* for a rule is defined to be the fraction of objects in O that satisfy the conjunction of the consequent and antecedent of the rule.
  Support should not be confused with confidence. While confidence is a measure of the rule's strength, support corresponds to statistical significance.
  Besides statistical significance, another motivation for support constraints comes from the fact that we are usually interested only in rules with support above some minimum threshold for business reasons. If the support is not large enough, it means that the rule is not worth consideration or that it is simply less preferred (and may be considered later).

3.1 Mapping Database Mining Problems

We now illustrate how the database mining problems under consideration can be mapped into the framework just described.

- *Classification*:
  The set of objects O consists of labeled data tuples comprising the training set for the classifier. The label identifies the group to which the object belongs. Other tuple attributes specify the properties of the object. Corresponding to each attribute in a tuple, there is an accessor method that returns the value of that attribute of the object. There is also a method that returns the tuple label.
  The goal of the classification problem is to discover rules for characterizing each of the groups in the training set, that is, to discover all rules with the consequent taking the form "o . label_method = k", where k ranges over the different label values. For example, the classification problem for target marketing involves rules with the method positive_response in the consequent, i.e. the consequent will have the form (o . positive_response = yes). For instance, one of the rules previously described in the context of the ski package example has the following form:

  $$(30 \leq u.Age \leq 40) \ and \ (u.Salary \geq 40K) \implies u.positive\_response = yes$$

  The restriction on the form of the consequent is an example of a syntactic constraint on the classifier rules. Another syntactic constraint is that the label method cannot appear in the antecedent of a rule. We can also have support constraints in the form of a requirement that a minimum number of tuples should satisfy the antecedent before a rule is acceptable.

- *Associations*:
  Here the set of objects O consists of customer transactions. Corresponding to each item in the transaction set, there is a binary-valued method that returns true/false depending whether the item is present or not present in the transaction. Association rules are subject to both syntactic and support constraints. Syntactic constraints cover the cases when the user is specifying additional restrictions which the rules should satisfy (all associations that have "milk" in the consequent, all associations that have "milk" in the consequent and "bread" in the antecedent, etc.). Support constraints are of primary importance, since the antecedents of rules should have some minimal support. This is critical both from the statistical as well as the business point of view (the larger the number of transactions supporting the antecedent of a rule, the more widely applicable the rule.)

- *Sequences*:
  Here the set of objects O consists of timestamps (possibly with different granularity such as days, minutes, hours etc). For example, the rule that IBM stock goes up the next day if AT&T stock goes up two consecutive days and DEC stock does not fall during this time can be described as follows: Let Stock(s, $t1$, $t2$) be a method, which when applied to a timestamp t returns whether the stock s has gone UP or DOWN between time $t + t1$ and $t + t2$. Our rule can then be formulated as:

  $$\begin{aligned} &t.\text{Stock(AT\&T,0,1)} = \text{UP} \;\wedge\; t.\text{Stock(AT\&T,1,2)} = \text{UP} \;\wedge \\ &\quad t.\text{Stock(DEC,0,1)} \neq \text{DOWN} \;\wedge\; t.\text{Stock(DEC,1,2)} \neq \text{DOWN} \;\wedge\; \Longrightarrow \\ &\qquad t.\text{Stock(IBM,2,3)} = \text{UP} \end{aligned}$$

  Sequence rules can be viewed as a special case of association rules. In these rules, antecedents and consequents contain literals that are related through the temporal component. In most cases, antecedents and consequents come in sequence in time. This can be viewed as a special case of a syntactic constraint. Support constraints will play a major role here as well. Thus if the number of timestamps for which the antecedent of a rule is satisfied is too small, the statistical value of such a rule is insignificant.

4 Basic Operations

Our objective is to provide efficient computational support for rule discovery problems. Our thesis is that all the problems that can be cast in the framework presented above require a small set of basic operations. By implementing these operations efficiently, we can solve a large number of database mining problems.

These operations use the concept of a *string*, which is an ordered sequence of (method, value) pairs. A value can be atomic, or it can be an interval in case of methods returning values that can be ordered. A method-value pair (m, v) for object t is a notational simplification of the predicate $t.m = v$ if v is atomic and $t.m \in v$ if v is an interval. A string is a conjunction of such predicates.

The computational process of discovering rules can be described using the following basic operations:

- `newstrings` $\leftarrow$ `Generate(seed, database)`

Takes the set of strings represented as the first parameter (seed) and builds from it a set of strings that are to be measured in a pass through the database. Sets of strings are formed from the strings in the seed set by extending them by (method, value) pairs according to the database schema. The actual number of new strings constructed in one pass through the database depends on a number of parameters, including the size of available memory, etc.

- `Measure(newstrings)`
  Measuring may involve simple counting of the number of objects supporting each of the new strings generated by Generate procedure. In these cases, Measure may be combined with Generate for efficiency reasons. Measuring may also involve more elaborate aggregation operations. For example, if the customer's id is stored together with the transaction number we may want to discover associations between items that were not bought in the same transaction, but perhaps within a certain period of time. In such a case, instead of the support set being defined as a number of all transactions with a particular item in them, we may define the support set to be the number of *customers* who bought a particular item. We could also be interested in calculating the total quantity or total price of a particular item bought by some customer. In such cases, further aggregation is necessary during the measurement phase.

- `combstrings` $\leftarrow$ `Combine(newstrings)`
  Combines some of the new strings. This combination could be done by creating intervals and partitions of the domain of contiguous attributes or using taxonomic information. In this way, multiple strings may be replaced by a single "combined string". For example, a number of strings describing individuals of the ages 20, 22, 24, 27, 28, 30 may be replaced by one string that describes individuals of ages between 20 and 30.

- `seed` $\leftarrow$ `Filter(combstrings)`
  Filters out strings from the set of combined strings to form a new seed set. The new seed set consists of strings that are good "prospects" to produce new strings for the Target set.

- `target` $+ \leftarrow$ `Select(seed)`
  Selects the strings that are to be stored in the Target set along with their measured values. The selected strings may or may not be retained in the seed set.

The initial seed set contains only the empty string. These four operations are evaluated repetitively in successive passes through the database until the seed set becomes empty. They construct as the output the final Target set of strings together with their measured values. Figure 1 shows the sequence in which the basic operations are applied.

4.1 Combining Operations

We now briefly illustrate how these operations can be combined to solve database mining problems. Later we consider the classification problem in more detail and describe the realization of an efficient algorithm for discovering classification rules using these operations. In the following we often use the term "attribute" for the accessor method associated with the attribute.

- *Classification*:
  We consider classifiers based on decision trees (see [7] [6] [11] for an overview.) We refer the reader to [1] for a discussion on why these classifiers (as opposed to, for example, neural nets [10]) are more appropriate for database mining applications.

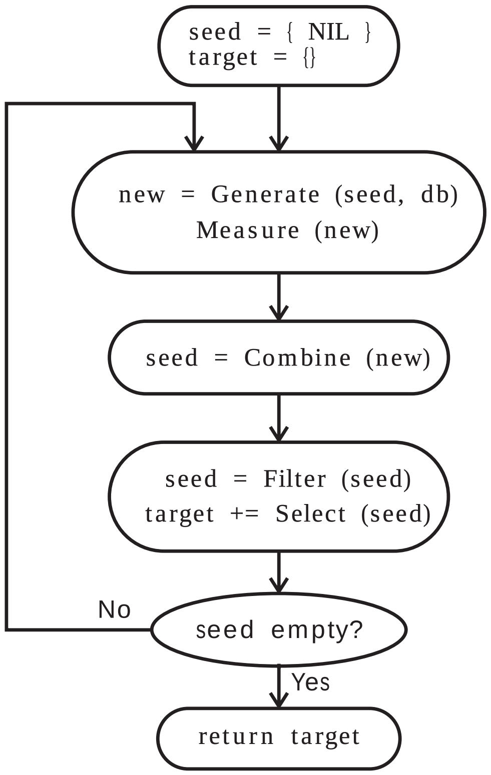

Figure 1: Basic Operations

The target set of strings includes all strings corresponding to the paths from the root to the leaves of the classification tree. Given a string s in the seed, we Generate all extensions of this string by adding all possible (attribute, value) pairs to it. Combination is performed on new strings generated through an extension by a continuous-valued attribute. ID3 [14] and CART [2] use binary splitting [13] for this purpose, whereas $\mathcal{IC}$ [1] partitions the domain of a continuous attribute into intervals. In the Filter operation, entropy [13] is computed for each attribute added, and only the strings containing the attribute that has the highest value of information gain [13] are retained and included in the next seed set. $\mathcal{IC}$ also computes for each expanded string an expansion merit, and a string is filtered out if its expansion merit is below an acceptable level. In the Select operation, ID3 picks those strings that have attained an information gain of 1 whereas $\mathcal{IC}$ picks those strings whose information gain exceeds a dynamic threshold function. The selected strings are not retained for further extension.

- *Associations*:
  To determine whether the rule $F(o) \Longrightarrow G(o)$ is satisfied with a confidence of at least $c\%$, we first need to count the total number n of objects o in O that support $F(o)$ and the total number m of objects that support both $F(o)$ and $G(o)$, and then divide m by n. If the ratio is greater than c, then the rule is satisfied with the confidence factor c; otherwise it is not.
  The target set of strings contains all strings that have support above a certain threshold (minimum support) and this is the criterion used in the Select operation. Such strings form a basis for the potential rules (in other words they can form potential antecedents of rules). In the Generate operation, new strings are generated by extending a seed string by items

not present in the seed. The Measure operation usually involves simply counting the total number of occurrences of a string in the database. Combination is a null operation. In the Filter operation, a string not meeting the minimum support requirement is discarded. Note that the strings picked in the Select operation are retained in the seed set for further extension.

- *Sequences*:
  Sequence rules are handled in a similar way to the association rules. The temporal nature of relationships between antecedents and consequents can be explored in the implementation in a number of ways that includes compressed storage and special purpose indexes. The global nature of the process is however analogous to discovery of associations.

4.2 Example

We give a simple example based on the classification application to illustrate how the algorithm shown in Figure 1 works. Section 5.2 gives a more elaborate example. The table below describes the XOR function. The algorithm is trying to determine what characterizes group 0 ($G = 0$) and group 1 ($G = 1$). The two attributes a and b are categorical attributes.

a	b	G
0	0	0
0	1	1
1	0	1
1	1	0

A string is put in the target set if the information gain [13] computed for the string is 1. In the Filter operation a single attribute is selected for extension and strings obtained by extending other attributes are eliminated.

We start with the empty string. In the first pass through the data, the following strings are generated by extending using all possible (attribute, value) pairs:

$(a, 0)$, $(a, 1)$, $(b, 0)$, $(b, 1)$

Since no continuous-valued attributes are present, no combination is done. The two attributes each have an information gain of 0. The algorithm breaks the tie by retaining strings with the first attribute. Thus the following strings are removed in the Filter operation:

$(b, 0)$, $(b, 1)$

No strings are put in Target in the Select operation since no string has an information gain of 1. The seed set at the beginning of the second pass contains the following strings:

$(a, 0)$ and $(a, 1)$

In the second pass, the following strings are generated:

$(a, 0) \wedge (b, 0)$, $(a, 0) \wedge (b, 1)$,
$(a, 1) \wedge (b, 0)$, $(a, 1) \wedge (b, 1)$

Here each string can only be extended by the attribute b. Hence no attribute selection is needed and no string is eliminated in the Filter operation. All the strings have an information gain of 1 and are hence selected for the target set. They are removed from `seed` which becomes empty. This terminates the algorithm. The strings in the target set can now be processed to generate the following rules:

$(a,0) \ \wedge \ (b,0) \Longrightarrow (G=0)$
$(a,0) \ \wedge \ (b,1) \Longrightarrow (G=1)$
$(a,1) \ \wedge \ (b,0) \Longrightarrow (G=1)$
$(a,1) \ \wedge \ (b,1) \Longrightarrow (G=0)$

4.3 Performance Considerations

Given that the data sets are expected to be massive, it is of paramount importance that the rule discovery algorithm be efficient. The following two factors should be kept in mind when combining basic operations:

- *Waste Ratio*:
  Consider the ratio of the number of strings in the target set to the total number of strings that were measured by the algorithm. Denote it by α. Then the waste ratio is defined to be $1-\alpha$. A large value for the waste ratio indicates generally poor performance due to possibly unnecessary additional work. For instance, we may build a classification tree either by building first a complete tree and then pruning it (as it is the case for instance for ID3 [14]) or take a dynamic approach and expand the tree only until estimated errors are reduced by a certain amount. The latter method will have a much better waste ratio since it will not generate and measure many strings that would be pruned later. Hence, if the classification accuracy is similar, the second method is a winner from the computational perspective.
- *Balancing I/O costs with CPU costs*:
  One way to minimize the waste ratio is to be conservative in the Generate part of the algorithm, and generate and measure only the most promising strings in one pass over the data. For large databases, however, this approach is unacceptable. Depending on the cost ratio between CPU and I/0 costs, different solutions may be appropriate for a given amount of memory. We have a choice of either making a small number of passes with significant processing per pass, or minimizing the computation per pass and make many passes instead. Similarly, we may precompute the whole set of rules and query it directly, or for each query we may make passes through the original data. An algorithm must determine which approach must be applied in each case.

5 $\mathcal{CDP}$: A Classifier Obtained Using Basic Operations

We now consider the classification problem in more detail and describe the realization of a specific classifier using the basic operations introduced in Section 4. We also describe the implementation of each of the basic operations. We refer to this classifier as $\mathcal{CDP}$, for classifier with dynamic pruning. $\mathcal{CDP}$ uses the binary partitioning of continuous attributes, proposed in ID3 [14] and CART [2], in the Combine operation. It uses the dynamic pruning scheme of $\mathcal{IC}$ [1] in the Filter operation. $\mathcal{CDP}$ belongs to the class of tree-classifiers [7] [6] [11], and hence can be used to generate rules that can easily be translated into SQL queries for efficient interfacing with relational databases [20].

5.1 Basic Operations in $\mathcal{CDP}$

We present the implementation of the basic rule discovery operations in $\mathcal{CDP}$. Starting with an empty string in the seed set, these operations are performed in sequence until the seed set become empty. The Target set then contains the desired classification rules.

5.1.1 Generate and Measure

Given a string s, new strings are generated by extending s with all possible (attribute, value) pairs for different groups. For efficiency reasons, Generate and Measure are combined into one operation. The initial seed for the Generate operation is the empty string.

The generation of new strings from a string s proceeds as follows. We read a tuple t from the training database. If s is present in t, new strings $\tilde{s} = s + (a, v)$ are generated, where (a, v) is an (attribute, value) pair present in t but not in s. If $\tilde{s}$ so generated is already in the new strings set, its count is incremented by one; otherwise, it is added to the new strings set with a count of 1. This operation is repeated for all tuples.

5.1.2 Combine

Let $\{\tilde{s_a}\}$ be the set of new strings generated by extending a seed s with (a, value) pairs of a continuous attribute a. We use binary partitioning, as proposed in ID3 [14] and CART [2], to determine a value u that partitions the range of atomic values of a into two intervals i_1 and i_2 such that the information gain is maximized. The interval i_1 is given by $a < u$ and the interval i_2 is given by $a \geq u$. Then the strings in $\{\tilde{s_a}\}$ are combined and replaced by two strings $s + (a, i_1)$ and $s + (a, i_2)$.

We do not combine strings generated by extending a seed with a categorical attribute. However, if taxonomical information is available for categorical attributes, this information can be used for combination of strings.

Thus, at the end of the Combine operation, corresponding to a seed string s, we have a set of new strings $\{\tilde{s}\}$ that are extensions of s. For a continuous attribute, there will be two strings in $\{\tilde{s}\}$. For a categorical attribute, there can be as many strings as the the number of distinct values that this attribute can have. The number of strings actually present depends on the number of distinct values present in the database.

5.1.3 Filter

We retain strings corresponding to the attribute that maximizes the *information gain ratio* [13] and eliminate all other strings. Let the database $\mathcal{D}$ of n objects contain n_k objects of group G_k. Then the entropy E of $\mathcal{D}$ is given by

$$E = -\textstyle\sum_k \frac{n_k}{n} \log_2 \frac{n_k}{n}$$

If attribute A_i with values $\{a_i^1, a_i^2, \ldots, a_i^w\}$ is used as the extension attribute, it will partition $\mathcal{D}$ into $\{\mathcal{D}_i^1, \mathcal{D}_i^2, \ldots, \mathcal{D}_i^w\}$ with $\mathcal{D}_i^j$ containing n_i^j objects of $\mathcal{D}$ that have value a_i^j of A_i. If the expected entropy for the substring of $\mathcal{D}_i^j$ is E_i^j, then the expected entropy for the string with A_i as the seed is the weighted average

$$E_i = \sum_j \frac{n_i^j}{n} E_i^j$$

The information gain by extending by A_i is therefore

$$\text{gain}(A_i) = E - E_i$$

Now, the information content of the value of the attribute A_i can be expressed as

$$I(A_i) = -\sum_j \frac{n_i^j}{n} \log_2 \frac{n_i^j}{n}$$

The information gain ratio for attribute A_i is then defined to be the ratio

$$\text{gain}(A_i)/I(A_i)$$

5.1.4 Select

We have eliminated all new strings that are not obtained due to extension by the selected attribute. We now compare the frequencies of different groups for the remaining strings. For a given string, the *winner* is the group with the largest frequency. The *strength* of the winner group is determined as follows. The winning group for a string is said to be *strong* if the ratio of the frequency of the winner group to the total frequency for the string across all groups in the database is above a certain *precision threshold*; the group is said to be *weak* otherwise.

The precision threshold is an adaptive function of the length of the string. The adaptive precision threshold we use is given by

$$1 - ((\text{string\_length - 1})/\text{max\_length})^2$$

where *max_length* is an algorithm parameter. This function is conservative in the beginning in declaring a winner strong, but loosens the criteria as the string length increases. The parameter *max_length* enables the user to bound the computational expense in the classification process since filtering takes place sooner for a smaller value of max_length. Some experiments with various smooth decay functions led us to use the quadratic function as having the best effect on classification accuracy.

If a string is found to have a strong winner, it is moved to the Target set. The string becomes the antecedent and the winner group becomes the consequent of the rule. This string is removed from the seed and hence not further extended. A string of *max_length* is also moved to the Target set. The winner-group (irrespective of its strength) becomes the consequent of the rule corresponding to this string. A final case is the string whose total frequency is found to be zero. Such a string is also moved to the Target set. However, the winner of its seed string becomes the consequent in this case. If the seed string is empty, the consequent is labeled unknown.

5.2 Example

We illustrate the basic operations in $\mathcal{CDP}$ with a simple example. Consider a people database in which every tuple has only three attributes:

- age (**age**) – non-categorical attribute – uniformly distributed from 20 to 80
- the zip code of the town the person lives in (**zipcode**) – categorical attribute – uniformly distributed between 9 available zipcodes
- level of education (**elevel**) – categorical attribute – uniformly distributed from 0 to 4

Group membership depends only on **age** and **elevel**, and is independent of **zipcode**. There are only two groups in the population:

$$\begin{array}{lcll}
\text{Group A} & \Longleftarrow & ((\mathbf{age} < 40) & \wedge\ (\mathbf{elevel} \in [0..1])) \vee \\
 & & ((40 \leq \mathbf{age} < 60) & \wedge\ (\mathbf{elevel} \in [0..3])) \vee \\
 & & ((60 \geq \mathbf{age}) & \wedge\ ((\mathbf{elevel} = 0))) \\
\text{Group B} & \Longleftarrow & \text{otherwise} &
\end{array}$$

where $(\mathbf{elevel} \in [1..k])$ is equivalent to $((\mathbf{elevel} = 1) \vee (\mathbf{elevel} = 2) \vee \ldots \vee (\mathbf{elevel} = k))$. We have a database of 10000 tuples that satisfy the above predicates. The parameter *max_length* is set to 10.

$\mathcal{CDP}$ starts with the seed of an empty string. In the Generate and Measure step, new strings are generated by extending strings in the seed set with the tuples in the database. The measurement involves counting the number of occurrences of each string for different groups. Thus, if the first tuple were

$$< (\mathbf{age} = 25), (\mathbf{zipcode} = 95120), (\mathbf{elevel} = 1), (\mathbf{group} = A) >$$

then 3 new strings are generated: (**age**=25), (**zipcode**=95120), and (**elevel**=1). The count for Group A is set to 1 and the count for Group B is set to 0 for all the three strings. Now if the next tuple were

$$< (\mathbf{age} = 25), (\mathbf{zipcode} = 95120), (\mathbf{elevel} = 2), (\mathbf{group} = B) >$$

then one new string is generated: (**elevel**=2). The count for Group B is set to 1 and the count for Group A is set to 0 for this string. Counts for Group B for strings (**age**=25) and (**zipcode**=95120) are incremented by 1. This process continues till all the tuples in the database have been exhausted.

$\mathcal{CDP}$ now combines the new strings for the continuous attributes using binary partitioning, as proposed in ID3 [14] and CART [2]. In this example, **age** is the only continuous attribute. We omit the details of partitioning and present only the results. We find that the partitioning replaces all the **age** strings with two strings: $(20 \leq \mathbf{age} < 59.5)$ and $(59.5 \leq \mathbf{age} < 80)$. Thus, at the end of Combine operation, we have 16 strings: 2 corresponding to **age**, 9 corresponding to **zipcode**, and 5 corresponding to **elevel**.

$\mathcal{CDP}$ now performs Filter operation, and eliminates all strings corresponding to attributes other than the one that maximizes the information gain ratio [13]. The following table shows the values obtained for the information gain ratio for the three attributes:

Attribute	Information Gain Ratio
age	.19682
zipcode	.00018
elevel	.13675

Therefore, all strings except ($20 \leq$ **age** < 59.5) and ($59.5 \leq$ **age** < 80) are eliminated, and the new seed set consists of these two strings.

$\mathcal{CDP}$ now examines if any of the strings in the seed set should be moved to the Target set on the basis of the strength of the winning group for the string. The following table shows the winner for the two strings in the seed set and the relative frequency of the winning group:

String	Winner	Relative Frequency	Strength
($20.0 \leq$ **age** < 59.5)	Group A	0.62	Weak
($59.5 \leq$ **age** < 80.0)	Group B	0.87	Weak

The precision threshold is 1 for these strings, and therefore the strength of both strings is weak. None of the strings is moved to the Target set.

$\mathcal{CDP}$ now makes another pass over the database. For brevity, we will only discuss the extensions of the string ($20.0 \leq$ **age** < 59.5). As before, new strings are generated by extending the seed string with (attribute:value) pairs found in tuples in which the seed string is present. Note that ($20.0 \leq$ **age** < 59.5) is also extended with a (**age**:v) string such that ($20.0 \leq v < 59.5$). Counts are also developed for each of the new strings.

New strings corresponding to **age** attribute are again combined, which results in two strings: ($20 \leq$ **age** < 59.5)($20 \leq$ **age** < 39.5) and ($20 \leq$ **age** < 59.5)($39.5 \leq$ **age** < 59.5). We also have 9 extensions of the seed string with strings corresponding to **zipcode** values and 5 extensions corresponding to **elevel** values.

The following table shows the information gain ratios for the attributes, using the count available with the new strings developed by extending ($20 \leq$ **age** < 59.5):

Seed string: ($20.0 \leq$ **age** < 59.5)	
Attribute	Information Gain Ratio
age	.19177
zipcode	.00028
elevel	.20476

Therefore, the filter operation eliminates all strings, except those generated by extending the seed string with an **elevel** value.

Since *max_length* is 10, the adaptive precision algorithm reduces the precision threshold to 0.99, and **winner_strength** finds the winning group to be strong for three strings:

String	Winner	Relative Frequency	Strength
($20 \leq$ **age** < 59.5)(**elevel** $= 0$)	Group A	1.00	Strong
($20 \leq$ **age** < 59.5)(**elevel** $= 1$)	Group A	1.00	Strong
($20 \leq$ **age** < 59.5)(**elevel** $= 2$)	Group A	0.51	Weak
($20 \leq$ **age** < 59.5)(**elevel** $= 3$)	Group A	0.50	Weak
($20 \leq$ **age** < 59.5)(**elevel** $= 4$)	Group B	1.00	Strong

The three strong strings are removed from the seed set and moved to the target set. The two weak strings remain in the seed set and are extended in the next pass over the data.

We omit the rest of the processing and show the final rules generated by the $\mathcal{CDP}$ in Figure 2 as a decision tree. It is a coincidence that the next attribute selected for the initial two **age**

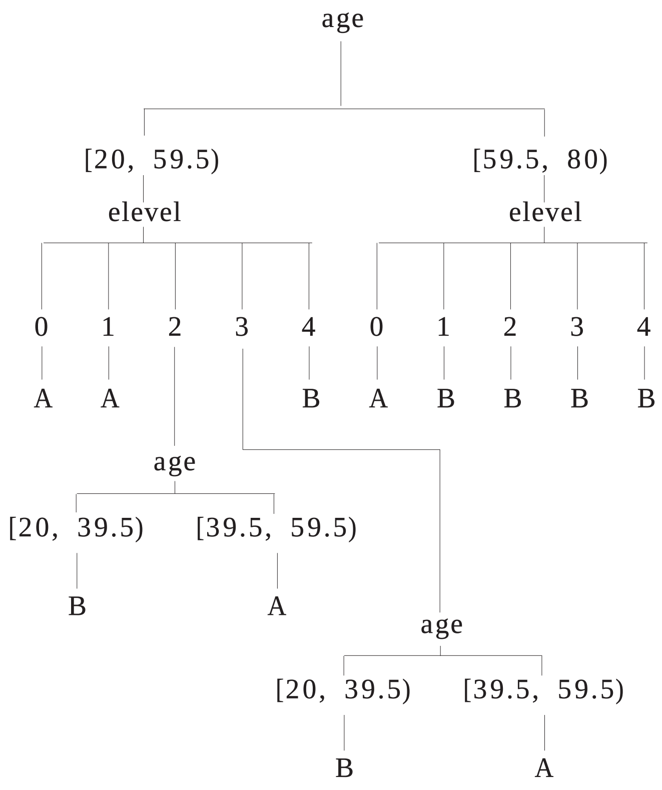

Figure 2: Example Decision Tree Generated by $\mathcal{CDP}$

strings turned out to be **elevel**. In general, the siblings may not be the same attribute and different attributes may be selected.

Thus, $\mathcal{CDP}$ infers the following set of classification rules

$$\begin{array}{rlll} \text{Group A} \Longleftarrow & ((20 \leq \textbf{age} < 59.5) & \wedge & (\textbf{elevel} \in [0..1])) \vee \\ & ((39.5 \leq \textbf{age} < 59.5) & \wedge & (\textbf{elevel} \in [2..3])) \vee \\ & ((59.5 \leq \textbf{age} < 80) & \wedge & ((\textbf{elevel} = 0))) \\ \\ \text{Group B} \Longleftarrow & ((20 \leq \textbf{age} < 39.5) & \wedge & (\textbf{elevel} \in [2..3])) \vee \\ & ((20 \leq \textbf{age} < 59.5) & \wedge & (\textbf{elevel} = 4)) \vee \\ & ((59.5 \leq \textbf{age} < 80) & \wedge & (\textbf{elevel} \in [1..4])) \end{array}$$

which is equivalent to the original set of rules. Note that the actual age range in the data set was from 20 to 80.

5.3 Performance Considerations

During the Generate-and-Measure operation, as we make a pass over the database, we would like to extend all the strings in the seed set and measure them to minimize I/O. However, all the strings in the seed set and their extensions may not fit in main memory. $\mathcal{CDP}$ takes a dynamic approach and starts by loading all the seed strings in memory. As the strings are expanded, memory may fill up. In that case, a victim seed string is selected (one with the maximum number of extensions) and this seed string and all its extensions are discarded. The discarded seed string is reconsidered in the next pass over the data.

$\mathcal{CDP}$ further seeks to improve the performance of the generation process by not expanding a string further if the winner strength of the string is above a certain precision threshold. An alternative would have been to expand all the strings fully and then prune them as is the case, for instance, in ID3 [14]. However, this approach will exhibit a bad waste ratio. Hence, if the classification accuracy is similar, using dynamic pruning is a winner from the computational perspective.

Classification Accuracy and Generation Efficiency The *classification error*, that is, the fraction of instances in the test data that are incorrectly classified, is the classical measure of the classification accuracy. To assess the accuracy of the rules discovered by $\mathcal{CDP}$, we compared it with ID3. We used the IND tree package [4] from the NASA Ames Research Center for this empirical evaluation. IND implements C4, which is a more recent, improved version of ID3. The experimental methodology, data sets, and the classification functions used for the experiments are described in the Appendix.

Figure 3 shows the average error rates for $\mathcal{CDP}$ and ID3. A maximum depth of 10 was used for the $\mathcal{CDP}$ algorithm. The 95% confidence intervals for the results indicate that differences in classification errors of less than 0.7% are not very significant. This means that the two algorithms are comparable in accuracy for functions 1, 3, and 10. ID3 does better than $\mathcal{CDP}$ for functions 6 and 7, but $\mathcal{CDP}$ obtains better accuracy than ID3 for functions 2, 4, 5, 8 and 9.

We now compare the rule generation efficiencies of $\mathcal{CDP}$ and ID3. We note that the two algorithms use identical attribute selection and partitioning procedures. Hence, any difference in generation efficiency is directly proportional to the difference in the number of strings generated. Figure 4 shows the number of strings generated by the two algorithms for each of the functions. These numbers were obtained by averaging the number of strings generated over several runs. We see that $\mathcal{CDP}$ generates only a third to a tenth as many strings as ID3. Thus we see that $\mathcal{CDP}$ can be 3 to 10 times faster than ID3 in discovering rules with comparable accuracy.

6 Summary

We presented our perspective of database mining as the confluence of machine learning techniques and the performance emphasis of database technology. We described three classes of database mining problems involving classification, associations, and sequences, and argued that these problems can be viewed within a common framework of rule discovery. We describe a model and four basic operations for the process of rule discovery. We also showed how these database mining problems map to this model and how they can be solved by using the basic operations we propose. Finally, we gave a concrete example of an algorithm suitable for discovering classification rules, and described the efficient implementation of the basic operations for this algorithm. This algorithm not only is efficient in discovering classification rules but also has accuracy comparable to the well known classification algorithm ID3 [13].

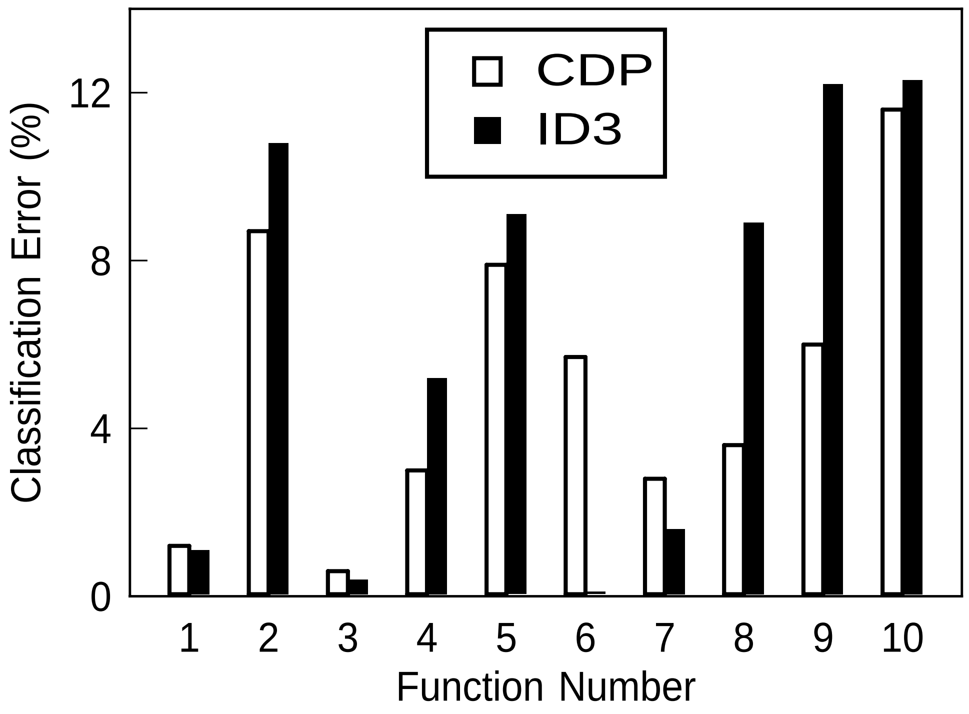

Figure 3: Classifier Accuracy: $\mathcal{CDP}$ and ID3

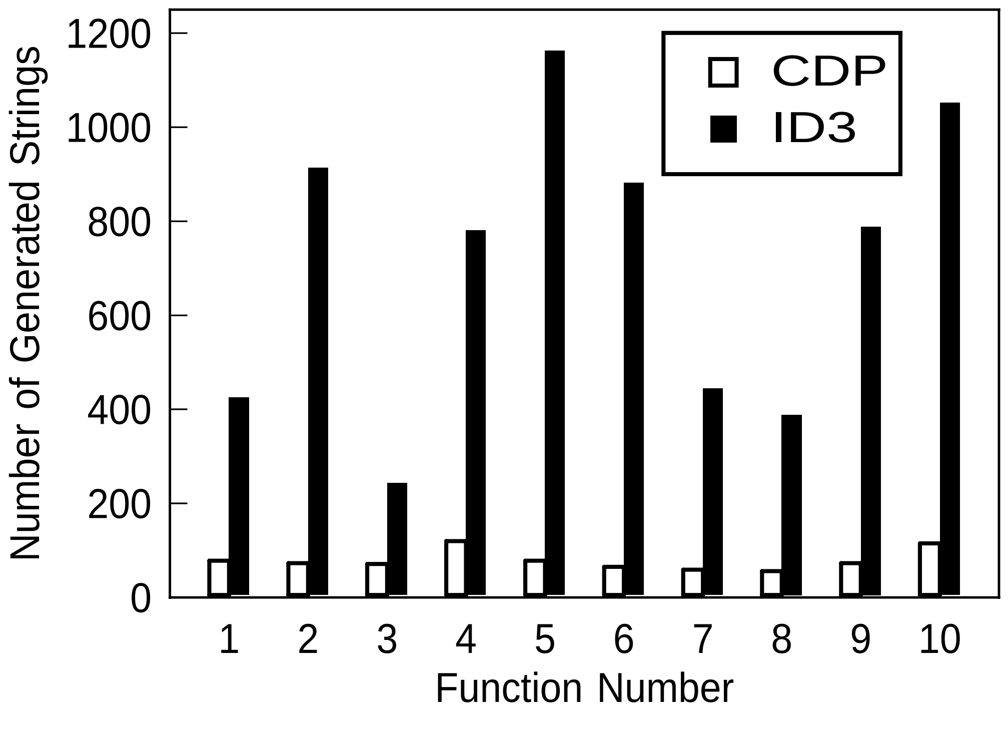

Figure 4: Strings Generated: $\mathcal{CDP}$ and ID3

The work reported in this paper has been done in the context of the Quest project at the IBM Almaden Research Center. In Quest, we are exploring the various aspects of the database mining problem. Our future plans include developing efficient implementations of the basic rule discovery operations described in this paper for the database mining problems involving associations and sequences. The eventual goal is to build an experimental system that can be used for mining rules embedded in massive databases. We believe that database mining is an important new application area for databases, combining commercial interest with intriguing research questions.

7 Appendix: Experimental Methodology

We used the evaluation methodology and the synthetic database proposed in [1] to assess the accuracy characteristics of $\mathcal{CDP}$. Every tuple in this database has the nine attributes given in Table 1. Attributes **elevel**, **car**, and **zipcode** are categorical attributes, all others are non-categorical attributes. Attribute values were randomly generated.

Attribute	Description	Value
salary	salary	uniformly distributed from 20000 to 150000
commission	commission	**salary** $>= 75000 \Longrightarrow$ **commission** $= 0$ else uniformly distributed from 10000 to 75000
age	age	uniformly distributed from 20 to 80
elevel	education level	uniformly chosen from 0 to 4
car	make of the car	uniformly chosen from 1 to 20
zipcode	zip code of the town	uniformly chosen from 9 available zipcodes
hvalue	value of the house	uniformly distributed from $0.5k100000$ to $1.5k100000$ where $k \in \{0 \cdots 9\}$ depends on **zipcode**
hyears	years house owned	uniformly distributed from 1 to 30
loan	total loan amount	uniformly distributed from 0 to 500000

Table 1: Description of Attributes

Using the above attributes, we developed 10 classification functions of different complexities. These functions are labeled 1 through 10 and involve 2 groups. Function 1 involves a predicate with ranges on a single attribute value. Functions 2 and 3 involve predicates with ranges on two attribute values. Function 2 uses two non-categorical attributes whereas function 3 uses a categorical and a non-categorical attribute. Similarly functions 4, 5 and 6 have predicates with ranges on three attribute values. Function 4 involves one categorical attribute. Function 5 involves only non-categorical attributes. Function 6 involves ranges on a linear function of two non-categorical attributes. Functions 7 through 9 are linear functions and function 10 is a non-linear function of attribute values. These functions are listed in Section 7. Note that these functions are a superset of the functions used in [1].

For every experiment, we generated a training set and a test data set. Tuples in the training set were assigned the group label by first generating the tuple and then applying the classification function on the tuple to determine the group to which the tuple belongs. Labels were also generated for tuples in the test data set as per the classification function to determine whether the classifier correctly identified the group for the tuple or not.

To model fuzzy boundaries between the groups, the data generation program takes a perturbation factor p as an additional argument. After determining the values of different attributes of a tuple and assigning it a group label, the values for non-categorical attributes are perturbed. If the value of an attribute A_i for a tuple t is v and the range of values of A_i is a, then the value of A_i for t after perturbation becomes $v + r \times p \times a$, where r is a uniform random variable between -0.5 and +0.5. In our experiments we used a perturbation factor of 5%.

For each experimental run, the errors for all the groups are summed to obtain the classification error. For each classification function, 200 replications were done with new training sets being generated. The replications were then used to calculate the mean error with 95% confidence intervals. Errors are reported as percentages of the total test data set. The intrinsic error in the test data due to perturbation was subtracted from the total error to arrive at the error due to misclassification.

We used training sets of 2500 tuples and test data sets of 10000 tuples. Before settling on these sizes, we studied the sensitivity of $\mathcal{CDP}$ to these sizes. The training set was reduced from 2500 tuples to 1000 tuples in steps of 500. As expected, the classification error increased with decreasing training set size, but the increase in mean error was small. In database mining applications involving databases in gigabytes, the training sets are likely to be fairly large, and training sets of 2500 tuples are not unreasonable. We increased the test data sizes from 10000 to 25000, 50000, and 100000 tuples. The results indicated that 10000 tuples provided almost identical error estimates as larger test data sets. Hence we decided to stay with test data sets of 10000 tuples to save on computing time.

Classification Functions

In the following, $(X \in [1..k])$ is equivalent to $((X = 1) \vee (X = 2) \vee \ldots \vee (X = k))$. Also, $P?Q : R$ is equivalent to the sequential conditional function, i.e., the expression is equivalent to $(P \wedge Q) \vee (\neg P \wedge R)$. There are two groups: Group A and Group B. We only specify the predicate function for Group A. All tuples not selected by this predicate function belong to Group B.

Function 1

Grp A: $((\textbf{age} < 40) \vee ((60 \leq \textbf{age})$

Function 2

$$
\begin{aligned}
\text{Grp A:}\quad & ((\textbf{age} < 40) \wedge (50K \leq \textbf{salary} \leq 100K)) \vee \\
& ((40 \leq \textbf{age} < 60) \wedge (75K \leq \textbf{salary} \geq 125K)) \vee \\
& ((\textbf{age} \geq 60) \wedge (25K \leq \textbf{salary} \leq 75K))
\end{aligned}
$$

Function 3

Grp A: $((\textbf{age} < 40) \ \wedge\ (\textbf{elevel} \in [0..1])) \vee$
$((40 \leq \textbf{age} < 60) \ \wedge\ (\textbf{elevel} \in [1..3])) \vee$
$((\textbf{age} \geq 60) \ \wedge\ (\textbf{elevel} \in [2..4]))$

Function 4

Grp A: $((\textbf{age} < 40) \wedge$
$(((\textbf{elevel} \in [0..1]) \ ?\ (25K \leq \textbf{salary} \leq 75K)) \ :\ (50K \leq \textbf{salary} \leq 100K)))) \vee$
$((40 \leq \textbf{age} < 60) \wedge$
$(((\textbf{elevel} \in [1..3]) \ ?\ (50K \leq \textbf{salary} \leq 100K)) \ :\ (75K \leq \textbf{salary} \leq 125K)))) \vee$
$((\textbf{age} \geq 60) \wedge$
$(((\textbf{elevel} \in [2..4]) \ ?\ (50K \leq \textbf{salary} \leq 100K)) \ :\ (25K \leq \textbf{salary} \leq 75K))))$

Function 5

Grp A: $((\textbf{age} < 40) \wedge$
$(((50K \leq \textbf{salary} \leq 100K) \ ?\ (100K \leq \textbf{loan} \leq 300K) \ :\ (200K \leq \textbf{loan} \leq 400K)))) \vee$
$((40 \leq \textbf{age} < 60) \wedge$
$(((75K \leq \textbf{salary} \leq 125K) \ ?\ (200K \leq \textbf{loan} \leq 400K) \ :\ (300K \leq \textbf{loan} \leq 500K)))) \vee$
$((\textbf{age} \geq 60) \wedge$
$(((25K \leq \textbf{salary} \leq 75K) \ ?\ (300K \leq \textbf{loan} \leq 500K) \ :\ (100K \leq \textbf{loan} \leq 300K))))$

Function 6

Grp A: $((\textbf{age} < 40) \ \wedge\ (50K \leq (\textbf{salary} + \textbf{commission}) \leq 100K)) \vee$
$((40 \leq \textbf{age} < 60) \ \wedge\ (75K \leq (\textbf{salary} + \textbf{commission}) \geq 125K)) \vee$
$((\textbf{age} \geq 60) \ \wedge\ (25K \leq (\textbf{salary} + \textbf{commission}) \leq 75K))$

Function 7

$\textbf{disposable} \ =\ (0.67 \times (\textbf{salary} + \textbf{commission}) \ -\ 0.2 \times \textbf{loan} \ -\ 20K)$
Grp A: $\textbf{disposable} > 0$

Function 8

$\textbf{disposable} \ =\ (0.67 \times (\textbf{salary} + \textbf{commission}) \ -\ 5000 \times \textbf{elevel} \ -\ 20K)$
Grp A: $\textbf{disposable} > 0$

Function 9

$\textbf{disposable} \ =\ (0.67 \times (\textbf{salary} + \textbf{commission}) \ -\ 5000 \times \textbf{elevel} \ -\ 0.2 \times \textbf{loan} \ -\ 10K)$
Grp A: $\textbf{disposable} > 0$

Function 10

$\textbf{hyears} < 20 \Longrightarrow \textbf{equity} = 0$
$\textbf{hyears} \geq 20 \Longrightarrow \textbf{equity} = 0.1 \times \textbf{hvalue} \times (\textbf{hyears} - 20)$

$\textbf{disposable} \ =\ (0.67 \times (\textbf{salary} + \textbf{commission}) \ -\ 5000 \times \textbf{elevel} \ +\ 0.2 \times \textbf{equity} \ -\ 10K)$
Grp A: $\textbf{disposable} > 0$

8 Acknowledgments

We wish to thank Ashar Mahboob for his help in implementing $\mathcal{CDP}$ and running performance experiments. We are thankful to Wray Buntine for the IND tree package that allowed us to compare $\mathcal{CDP}$ with ID3. Thanks are also due to Guy Lohman for his comments on an earlier version of this paper.

References

[1] Rakesh Agrawal, Sakti Ghosh, Tomasz Imielinski, Bala Iyer, and Arun Swami, "An Interval Classifier for Database Mining Applications", *VLDB 92*, Vancouver, British Columbia, Canada, 1992, 560–573.

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, 1984.

[3] R. Brice and W. Alexander, "Finding Interesting Things in Lots of Data", *23rd Hawaii International Conference on System Sciences*, Kona, Hawaii, January 1990.

[4] Wray Buntine, *About the IND Tree Package*, NASA Ames Research Center, Moffett Field, California, September 1991.

[5] Wray Buntine and Matha Del Alto (Editors), *Collected Notes on the Workshop for Pattern Discovery in Large Databases*, Technical Report FIA-91-07, NASA Ames Research Center, Moffett Field, California, April 1991.

[6] Philip Andrew Chou, "Application of Information Theory to Pattern Recognition and the Design of Decision Trees and Trellises", Ph.D. Thesis, Stanford University, California, June 1988.

[7] G. R. Dattatreya and L. N. Kanal, "Decision Trees in Pattern Recognition", In *Progress in Pattern Recognition 2*, L. N. Kanal and A. Rosenfeld (Editors), Elsevier Science Publishers B.V. (North-Holland), 1985.

[8] J. Han, Y. Cai, and N. Cercone, "Knowledge Discovery in Databases: An Attribute-Oriented Approach", *VLDB-92*, Vancouver, British Columbia, Canada, 1992, 547–559.

[9] Ravi Krishnamurthy and Tomasz Imielinski, "Practitioner Problems in Need of Database Research: Research Directions in Knowledge Discovery", *SIGMOD Record*, Vol. 20, No. 3, Sept. 1991, 76–78.

[10] Richard P. Lippmann, "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, April 1987, 4–22.

[11] David J. Lubinsky, "Discovery from Databases: A Review of AI and Statistical Techniques", *IJCAI-89 Workshop on Knowledge Discovery in Databases*, Detroit, August 1989, 204–218.

[12] Tarek M. Anwar, Howard W. Beck, and Shamkant B. Navathe, "Knowledge Mining by Imprecise Querying: A Classification-Based Approach", *IEEE 8th Int'l Conf. on Data Engineering*, Phoenix, Arizona, Feb. 1992.

[13] J. Ross Quinlan, "Induction of Decision Trees", *Machine Learning*, **1**, 1986, 81–106.

[14] J. Ross Quinlan, "Simplifying Decision Trees", *Int. J. Man-Machine Studies*, **27**, 1987, 221–234.

[15] G. Piatetsky-Shapiro and W. Frawley (Editors), *Proceedings of IJCAI-89 Workshop on Knowledge Discovery in Databases*, Detroit, Michigan, August 1989.

[16] G. Piatetsky-Shapiro (Editor), *Proceedings of AAAI-91 Workshop on Knowledge Discovery in Databases*, Anaheim, California, July 1991.

[17] G. Piatetsky-Shapiro, *Discovery, Analysis, and Presentation of Strong Rules*, In [18], 229–248.

[18] G. Piatetsky-Shapiro (Editor), *Knowledge Discovery in Databases*, AAAI/MIT Press, 1991.

[19] Shalom Tsur, "Data Dredging", *IEEE Data Engineering Bulletin*, **13**, 4, December 1990, 58–63.

[20] Jeffrey D. Ullman, *Principles of Database and Knowledge-Base Systems*, Volume I, Computer Science Press, 1988.