Search Framework for Mining Classification Decision Trees[‡]

Eui-Hong (Sam) Han[†], Shashi Shekhar, Vipin Kumar, M. Ganesh, Jaideep Srivastava
Dept. of Computer & Information Sciences
4-192 EECS Bldg., 200 Union St. SE
University of Minnesota
Minneapolis, MN 55455, USA

Technical Report 96-023 (April 3, 1996)

Abstract

Classification–rule–learning task is presented as a search process of finding a classification–decision tree that meets users' preferences and requirements. Users can control the efficiency of the mining process and the quality of the final decision tree through the search parameters. This search framework allows users to easily adapt to different domains and different sets of data by modifying different search parameters. The mining process starts with a specific set of values for the search parameters and the process can be repeated with different search parameters values until a satisfactory result is obtained. This framework also allows the development of new algorithms. A set of search parameters that is frequently and successfully used can define a new algorithm. We present two new algorithms developed in this search framework and compare them to well–known algorithms. BF uses best–first ordering to expand the frontier nodes of the decision trees, instead of the conventional depth–first or breadth–first criteria. $\mathcal{CDP}+$ dynamically adjusts the depth pruning criteria for a node. Experimental results show that BF has a new capability to guide the construction of decision trees based on the overall error–rate criteria. In addition, $\mathcal{CDP}+$ often outperforms several decision–tree learning algorithms in error rate and number of nodes generated.

Keywords: Data mining, user interaction, classification, decision trees, algorithm development.

[†]**Contact author:** Eui-Hong (Sam) Han
Tel: (612) 626–7515
Fax: (612) 624–6539
email: han@cs.umn.edu

[‡] This work is sponsored (in whole or in part) by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, ARO Grant DA/DAAH04-95-1-0538, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

1 Introduction

One of the important problems in data mining [SAD$^+$93] is the classification–rule learning. The classification–rule learning involves finding rules or decision trees that partition the given data into predefined classes. For any realistic problem domain of the classification–rule learning, the set of possible decision trees is too large to be searched exhaustively. In fact, the computational complexity of finding an optimal classification decision tree is *NP–hard*. All of the existing algorithms, like $C4.5$ [Qui93] and $\mathcal{CDP}$ [AIS93], are greedy in nature, since a decision to choose an attribute for classification at a node in the decision tree is based on local heuristics. As the result of this greedy nature, it is entirely possible that one algorithm performs well with a specific set of data, but the same algorithm performs poorly with the other sets of data.

Another important issue in the classification–rule learning is to allow users to control the computational efficiency of the mining process and the quality of the knowledge to be discovered. Traditional machine–learning [Lan96] research has addressed the issues on the quality of the learned knowledge. However, given the huge volume of data in databases and the computational complexity of the learning algorithms, the computational efficiency of mining processes is also a critical issue. An algorithm with somewhat worse error rate may be more desirable if it takes up significantly less computation time.

In this paper, we address the above problems in the context of mining for classification rules. We describe the classification–rules–learning task as a search process of finding a final classification–decision tree that meets users' preferences and requirements. Users can control the efficiency of the mining process and the quality of the final decision tree through the search parameters. This search framework allows users to easily adapt to different domains and different set of data by modifying different search parameters. The mining process can start with a specific set of values of the search parameters and the process can be repeated with different values of the search parameters until a satisfactory result is obtained. This framework also leads to the development of new algorithms. A set of search parameters that is frequently used and successful can define a new algorithm. We show two new algorithms developed in this search framework and compare them to the well–known algorithms $C4.5$ [Qui93] and $\mathcal{CDP}$ [AIS93].

In recent years, substantial advances have been made in the development of data–mining techniques that deal with accuracy, efficiency and robustness. The $C4.5$ [Qui93] algorithm generates a decision tree for the given training data set by recursive partitioning of the data. The algorithm tries to generate a full decision tree which can classify the entire training data set correctly. The over–fitted tree is then pruned bottom–up fashion, using a predicted error criterion, by replacing some subtrees with leaves or some intermediate nodes with their subtrees. The pruning technique has been used to improve the accuracy and robustness. The algorithm tries to find the most accurate decision tree and does not provide any explicit control over efficiency or accuracy. $\mathcal{CDP}$ has been developed as part of the Quest project [AIS93] at the IBM Almaden Research Center. The $\mathcal{CDP}$ algorithm uses the dynamic pruning criterion to stop the generation of decision nodes if the error rate at a node of the tree is below some adaptive precision threshold. The dynamic pruning technique has been used to improve the efficiency of the mining process. User parameter *maxlength* is used to limit the maximum height of the decision tree. However, $\mathcal{CDP}$ does not have a way to specify the desired accuracy of a final decision tree.

The rest of the paper is organized as follows. Section 2 defines the search space of the classification–learning process, and shows the development of new classification–learning algorithms in the search paradigm defined. A performance evaluation of the algorithms developed in section 2 is shown in section 3. Section 4 presents our conclusions and future work.

2 Search Framework for Classification–Rule Learning

We have modeled the problem of classification–rule–learning as a search [KK88] process in the state space of different models. Learning a classification decision tree from a training data set can be regarded as a process of searching for the best decision tree that meets user–provided goal constraints. The problem space of this search process consists of *Model Candidates*, a *Model Candidate Generator*, and *Model Constraints*. Many existing classification–learning algorithms like $C4.5$ [Qui93] and $\mathcal{CDP}$ [AIS93] fit nicely within this search framework. New learning algorithms that fit users' requirements can be developed by defining the components of the problem space. We discuss the components of the search problem space in the context of $ID3$-like decision-tree learning.

2.1 Components of the Search Framework

A *Model Candidate* is a partial classification decision tree with some additional information. Each node of the decision tree is a *Model Atom*. The root node of the decision tree, called root model atom, contains all of the training data set. Each non–leaf model atom has an attribute associated with it that is used to assign the training data set of the model atom to its child model atoms. If the attribute has k values, then the non–leaf model atom has k child model atoms, each corresponding to a value of the attribute. The arc from a non–leaf model atom to its child model atom is labeled with the corresponding value of the attribute. Leaf model atoms have an attached label that indicates the majority class. Each model atom contains a subset of the training data set, such that the values of the attributes on the path from the root to the model atom are consistent with the data contained in it. An initial model candidate corresponds to a decision tree that has a single leaf model atom containing all of the training data set. The search process can be defined as a process to find a final model candidate starting from the initial model candidate, such that the final model candidate meets user goal specifications.

Table 1 shows a training data set with four data attributes and two classes. Figure 1 shows different model candidates that are constructed during the search process of finding a final model candidate with the training data set given in Table 1. Figure 1(a) shows the initial model candidate and Figure 1(c) shows the final model candidate. The leftmost leaf model atom of the final model candidate labeled "Play" contains the subset of the training data set for which the value of *Outlook* is sunny and the value of *Humidity* is less than or equal to 75.

The model candidate generator transforms the current model candidate into a new model candidate by selecting one model atom to expand from the expandable leaf model atoms.

Outlook	Temp(F)	Humidity(%)	Windy?	Class
sunny	75	70	true	Play
sunny	80	90	true	Don't Play
sunny	85	85	false	Don't Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
overcast	72	90	true	Play
overcast	83	78	false	Play
overcast	64	65	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play
rain	65	70	true	Don't Play
rain	75	80	false	Play
rain	68	80	false	Play
rain	70	96	false	Play

Table 1: A small training data set [Qui93]

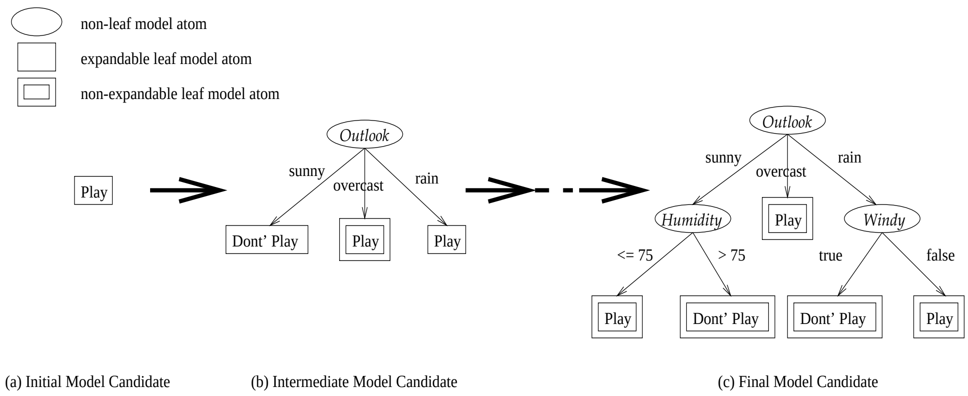

Figure 1: Search process

The model candidate generator uses model constraints to provide controls and boundaries to the search space. Model constraints consist of acceptability constraints, expandability constraints, a traversal strategy and a data–entropy–calculation function. These constraints are specified based on several properties of the model candidate and model atom, and are defined later in this section. The model candidate generator first checks to see whether the current model candidate satisfies the acceptability constraints. If the current model candidate satisfies the acceptability constraints, then the search process ends and the current model candidate is returned as the final model candidate. Otherwise, the model candidate generator selects an expandable leaf model atom to expand according to the traversal strategy.

The model candidate generator checks whether the model atom is expandable according to the expandability constraints. If the model atom is not expandable, the model candidate generator marks the model atom as non–expandable. Otherwise, the model candidate generator marks the model atom as non–leaf and then generates child model atoms. The model candidate generator considers all possible attributes that can be used to generate child model atoms. Using a data–entropy–calculation function, the model candidate generator evaluates the entropy gain for all the possible child model atoms resulting from the choice of different data attributes. One data attribute that maximizes the entropy gain is chosen for the expansion. One model atom is created for each possible value of the attribute chosen, and the training data set is also split accordingly and assigned to each corresponding child model atom. All the generated model atoms are initialized as leaf and expandable.

An **acceptability constraint predicate** specifies when a model candidate is acceptable and thus allows the search process to stop. Some examples of the acceptability predicate are:

A1) total number of expandable leaf model atoms = 0 (default)

A2) overall model candidate error rate on the training data $\leq$ acceptable error rate

A3) total number of model atoms in the model candidate $\geq$ maximum allowable tree size

A4) other predicates that are logical combinations of the above predicates

The acceptability predicate A1 is used in $C4.5$ and $\mathcal{CDP}$. The predicate A2 can be used to terminate the search process early if the overall error rate is acceptable. Note that the actual error rate of the model may be different on the test data set. The predicate A3 limits the decision tree size with the maximum allowable number of model atoms. Predicates A2 and A3 prevent the the decision tree from becoming too large and force the search process to stop early. Smaller decision trees maybe desirable for two reasons. First, the creation of each model atom requires a non–trivial amount of computation, as all the data set contained in the model atom may have to be visited to compute the entropy of different attributes. Second, if the decision tree is too large, then it can over–generalize [BEHW87]; i.e., it may give very good performance on the training data set, but very poor performances on the test data sets. In fact, for some algorithms, e.g. $C4.5$, MDL-based pruning ([MRA95]), it is desirable to prune the final model candidate generated by removing some model atoms from the model candidate. This pruning technique has been shown to be effective in remedying the problem of over–generalization in the decision trees.

An **expandability constraint predicate** specifies whether a leaf model atom is expandable or not. Some examples of the expandability predicate are:

E1) local error rate (i.e., the error rate on the training data set contained in the leaf model atom) $>$ 0.0 (default)

E2) entropy gain $>$ 0.0 (default)

E3) $(\frac{depth-1}{maxlength})^2 <$ local error rate, where $maxlength$ is user provided constant and $depth$ is the depth of the model atom in the decision tree

E4) $(\frac{adj\_depth-1}{maxlength})^2 <$ local error rate, where $maxlength$ is a user–provided constant and adj_depth is adjusted depth (see Section 2.2 for details)

E5) other predicates on the combination of model atom attributes

Algorithm	Accept. Predicate	Traversal Strategy	Exp. Predicate	Post–proc essing
$C4.5$	A1	T1	E1,E2	none
$C4.5P$	A1	T1	E1,E2	error-based pruning
BF	A1,A2	T3	E1,E2	none
$\mathcal{CDP}$	A1	T2	E3	none
$\mathcal{CDP}+$	A1	T2	E4	error-based pruning

Table 2: Comparison of different classification learning algorithms

$C4.5$ uses the expandability predicate E1 and E2. $\mathcal{CDP}$ uses the predicate E2 and E3.

The **traversal strategy** ranks expandable leaf model atoms based on the model atom attributes. Some of the examples are:

T1) increasing order of depth (Breadth–First)

T2) decreasing order of depth (Depth–First)

T3) orders based on other model atom attributes (Best–First)

$C4.5$ uses Depth–First strategy and $\mathcal{CDP}$ uses Breadth–First strategy.

The **data–entropy–calculation function** provides a mechanism for calculating the entropy of training cases with respect to classes. Some examples of the entropy–calculation function are based on information theory [Qui93], and other user–provided functions such as the Gini function [WK91].

2.2 New Algorithms for Classification-Rule Learning

We show how to arrive at an algorithm, BF, based on the Best–First search idea. For acceptability criteria, in addition to the default predicate A1, we also include the acceptability predicate A2. The predicate A2 allows a user to specify an overall acceptable error rate for the decision tree. The traversal strategy chosen is Best–First, where the expandable leaf model atoms are ranked according to the decreasing order of the number of misclassified training cases. This traversal strategy will expand a model atom that has the most misclassified training cases, thus has the largest potential for reducing the overall error rate by expanding one extra node. For the expandability predicate, default predicates E1 and E2 are used. For the data–entropy–calculation function, a widely used function based on information theory is chosen for the algorithm.

We can also construct a new algorithm by modifying existing algorithms and combining useful features of different algorithms. We propose a modification of $\mathcal{CDP}$ and call this new algorithm $\mathcal{CDP}+$. $\mathcal{CDP}$ uses the expandability predicate E3 which fixes an acceptable error rate for the model atom statically based on its depth. In $\mathcal{CDP}+$ we modify the expandability predicate E3 as follows to obtain the new predicate E4. We want our new algorithm to generate a deeper subtree at the model atom if the size of the training data set at the model atom is larger than the expected average. We set $adj\_depth = -log_b(\frac{t}{T})$, where b is the estimated branching factor of the decision tree, t is the size of training data set belonging to the model atom, and T is the size of the whole training data set.

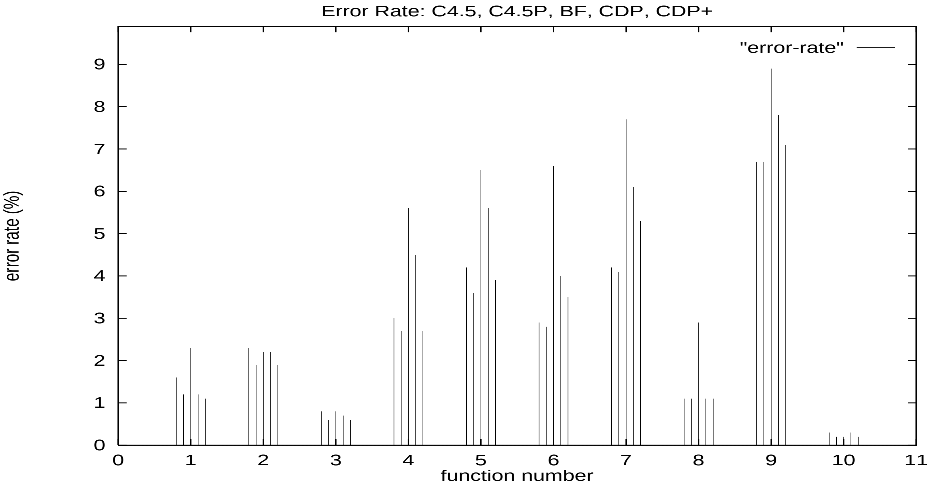

Figure 2: Comparison of classification error for 10 data-sets. Five bars are shown for each problem. They represent C4.5, C4.5P, BF, CDP and CDP+ from left to right.

Table 2 compares different learning algorithms based on 3 different criteria of model constraints and post–processing. Data–entropy–calculation function is omitted, since all the algorithms use a function based on information theory.

3 Experimental Evaluation of the New Algorithms

We compare the new BF and $\mathcal{CDP}+$ algorithms with the $C4.5$ and the $\mathcal{CDP}$ algorithms to demonstrate the usefulness of the search framework. We used the $C4.5$ algorithm provided with [Qui93]. We present two sets of result for $C4.5$: one in which the resulting classification tree has been pruned as described in [Qui93] ($C4.5P$) and the other in which the resulting tree is not pruned ($C4.5$). We do not have a copy of the $\mathcal{CDP}$ algorithm, and thus we implemented the algorithm based on the description in [AIS93]. $\mathcal{CDP}$ uses a parameter *maxlength* for the adaptive precision threshold. We have chosen 10 for the parameter, which is the choice in [AIS93]. We also used the same value for $\mathcal{CDP}+$. In the BF algorithm, we chose a 6% overall error rate as the acceptable error rate. We also perform error–based pruning of $\mathcal{CDP}+$ algorithm using the method used in $C4.5P$.

We have followed the data generation methodology of [AIS93] very closely. Data records were randomly generated and assigned class labels using one of the ten classification functions described in [AIS93]. For each classification function, we used a training set size of 2500 tuples and a test set size of 10000 tuples. We have performed experiments on 10 different sets of data and report the mean values of the results.

The experimental results of various classification algorithms are shown in Figure 2, Figure 3, and Figure 4. Figure 2 presents the classification errors of the different algorithms for each of the 10 classification functions. Figure 3 presents the number of nodes generated by the different algorithms. Figure 4 compares the size of the final decision trees of the different algorithms. The final tree size for all algorithms except $C4.5P$ and $\mathcal{CDP}+$ is the same as the number of nodes generated.

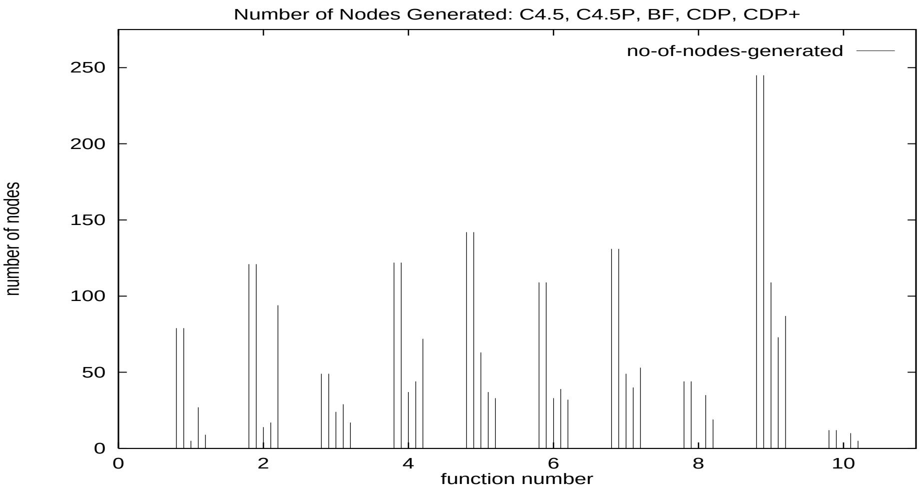

Figure 3: Comparison of nodes generated for 10 data-sets. Five bars are shown for each problem. They represent C4.5, C4.5P, BF, CDP and CDP+ from left to right.

Our results show that $\mathcal{CDP}$ generates considerably fewer nodes than $C4.5$, but its error rate is often worse than that of $C4.5$.[1] Furthermore, the error rate of $\mathcal{CDP}$ is consistently and significantly worse than that of $C4.5P$ for all the functions. BF produced decision trees that approximately met the user's acceptable error rate (6%) on the testing data set for most functions except function 7 and 9. For all functions, the error rate for BF was 6% or less for the training data sets, although the error rates on test data sets are sometimes higher. In general, BF with a 6% acceptable error rate generated a smaller number of nodes than $\mathcal{CDP}$, but it has a higher error rate on the test data compared to $\mathcal{CDP}$. We set a 6% acceptable error rate for all 10 functions in this experiment. However, after we adjusted the error rate of BF for individual functions, we were able to produce decision trees with a lower error rate that is comparable to $\mathcal{CDP}$.[2] $\mathcal{CDP}+$ consistently outperformed $\mathcal{CDP}$ in error rate but the number of nodes generated by the two algorithms are roughly similar (except in the case of function 2 for which the $\mathcal{CDP}+$ generates many more nodes). It can be also seen that the accuracy of $\mathcal{CDP}+$ is slightly worse than $C4.5P$, but $\mathcal{CDP}+$ generates a considerably smaller number of nodes than $C4.5P$. Even the final tree size of $C4.5P$ is considerably larger than that of $\mathcal{CDP}+$.

The experimental result shows that $\mathcal{CDP}+$ is the best overall algorithm for the 10 functions, and the data sets we generated, when we consider all the performance metrics together. $C4.5P$ is the winner if classification accuracy alone is considered. For function 8, even though BF has a three times greater error rate than the other algorithms, it produced decision trees with 10 to 40 times fewer nodes. Depending on user requirements, this decision tree may be preferable over those found by other algorithms due to its exceptionally small size. Different data sets require different algorithms for the best results, and diverse user requirements put different constraints on the final decision tree. These experiments show that our framework can help find an optimal algorithm for some given data set and user requirements.

[1] Note that in [AIS93], the error rate for $\mathcal{CDP}$ was somewhat better relative to $ID3$. This could be due to the fact that $C4.5$ is an improved version of the $ID3$ package used in the [AIS93] experiments.

[2] The results are not reported here.

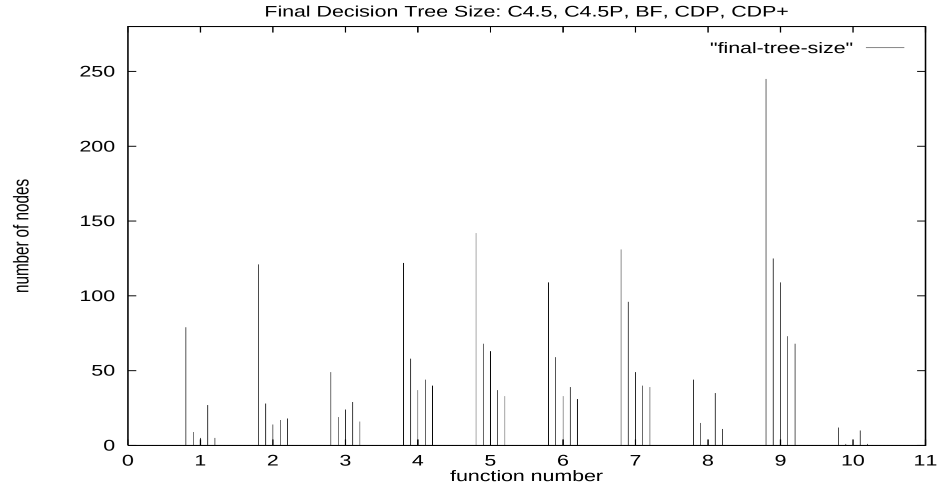

Figure 4: Comparison of final decision tree size for 10 data-sets. Five bars are shown for each problem. They represent C4.5, C4.5P, BF, CDP and CDP+ from left to right.

4 Conclusions and Future Work

We provide a search–based framework that allows users to formulate a variety of classification algorithms, to control the computational efficiency of the mining process and to specify the quality of the final decision tree. We have developed two new algorithms, namely BF and $\mathcal{CDP}+$, from the search framework and evaluated their performance with 10 datasets from the literature. BF uses best–first ordering to expand the frontier nodes of the decision trees instead of the conventional depth–first or breadth–first criteria. $\mathcal{CDP}+$ dynamically adjusts the depth pruning criteria for a node, which is static in the case of $\mathcal{CDP}$ based on depth of the node in the tree. Experimental results show that BF has a new capability to guide the construction of decision trees based on the overall error–rate criteria. In addition, $\mathcal{CDP}+$ often outperforms several decision–tree learning algorithms in error rate and number of nodes generated. This illustrates the promise of the search–based framework in providing a mechanism for user interaction in data–mining process and a formal mechanism to address accuracy, efficiency, robustness and other related issues.

Future work includes the exploration of search frameworks for algorithms discovering other patterns such as associations [AS94, HS95], clustering rules [Fis95] and temporal sequences [MTV95]. Such a framework can be quite useful in a more general environment for visual data-mining in which the user can visualize the intermediate results of the algorithm, and guide the algorithm dynamically by adjusting the model constraints. In [GHK$^+$96] we present our preliminary work on such a visual data-mining system.

Acknowledgments

We would like to thank the data–mining group at the University of Minnesota, Dept. of Computer Science, for its comments and criticisms of our work. This has helped improve the technical content of this paper, as well as the clarity of our presentation.

References

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engg.*, 5(6):914–925, December 1993.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*, pages 487–499, Santiago, Chile, 1994.

[BEHW87] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, 1987.

[Fis95] D. Fisher. Optimization and simplification of hierarchical clusterings. In *Proc. of the First Int'l Conference on Knowledge Discovery and Data Mining*, pages 118–123, Montreal, Quebec, 1995.

[GHK+96] M. Ganesh, E.H. Han, V. Kumar, S. Shekhar, and J. Srivastava. Visual data mining: Framework and algorithm development. Technical Report TR-96-021, Department of Computer Science, University of Minnesota, Minneapolis, 1996.

[HS95] M. A. W. Houtsma and A. N. Swami. Set-oriented mining for association rules in relational databases. In *Proc. of the 11th Int'l Conf. on Data Engg.*, pages 25–33, Taipei, Taiwan, 1995.

[KK88] L. N. Kanal and Vipin Kumar, editors. *Search in Artificial Intelligence.* Springer-Verlag, New York, NY, 1988.

[Lan96] P. Langley. *Elements of Machine Learning.* Morgan-Kaufman, San Francisco, CA, 1996.

[MRA95] M. Mehta, J. Rissanen, and R. Agrawal. Mdl-based decision tree pruning. In *Proc. of the First Int'l Conference on Knowledge Discovery and Data Mining*, pages 216–221, Montreal, Quebec, 1995.

[MTV95] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proc. of the First Int'l Conference on Knowledge Discovery and Data Mining*, pages 210–215, Montreal, Quebec, 1995.

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

[SAD+93] M. Stonebraker, R. Agrawal, U. Dayal, E. J. Neuhold, and A. Reuter. Dbms research at a crossroads: The vienna update. In *Proc. of the 19th VLDB Conference*, pages 688–692, Dublin, Ireland, 1993.

[WK91] S. M. Weiss and C. A. Kulikowski. *Computer Systems That Learn.* Morgan-Kaufman, San Mateo, CA, 1991.

The following appendix is for the convenience of the reviewers and is NOT intended to be included in the conference proceedings.

Appendix

A: Data Attributes

Our data set consists of records with the attributes described in table 3. Attributes **elevel, car** and **zipcode** are discrete attributes, while all the other attributes are continuous. The data records are divided into classes based on classification functions of varying complexity. These functions are explained in detail in Appendix B.

Attribute	Description	Value
salary	salary	uniformly distributed from 20000 to 150000
commission	commission	salary $\geq$ 75000 $\Rightarrow$ 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 the **zipcode**
hyears	years house owned	uniformly distributed from 1 to 30
loan	total loan amount	uniformly distributed from 0 to 500000

Table 3: Description of Attributes [AIS93]

B: Classification Functions

For our experimental evaluation we have selected the classification functions used in [AIS93]. These functions are described below. Each record was classified as either Group A or Group B. We describe only the conditions for a record to be classified as Group A; all others are classified as Group B. The functions below use a C language like syntax for the sequential conditional expressions P ? Q : R. This is equivalent to the logical expression $(P \wedge Q) \vee (\neg P \wedge R)$.

Function 1

$$(40 > age \geq 60)$$

Function 2

$$((age < 40) \wedge (50K \leq salary \leq 100K))\vee$$
$$((40 \leq age < 60) \wedge (75K \leq salary \leq 125K))\vee$$
$$((age \geq 60) \wedge (25K \leq salary \leq 75K))$$

Function 3

$$((age < 40) \wedge (elevel \in [0 \cdots 1]))\vee$$
$$((40 \leq age < 60) \wedge (elevel \in [1 \cdots 3]))\vee$$
$$((age \geq 60) \wedge (elevel \in [2 \cdots 4]))$$

Function 4

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

Function 5

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

Function 6

$((age < 40) \wedge (50K \leq (salary + commission) \leq 100K))\vee$
$((40 \leq age < 60) \wedge (75K \leq (salary + commission) \leq 125K))\vee$
$((age \geq 60) \wedge (25K \leq (salary + commission) \leq 75K))$

Function 7

$disposable > 0$ where,
$disposable = (0.67 * (salary + commission) - 0.2 * loan - 20K)$

Function 8

$disposable > 0$ where,
$disposable = (0.67 * (salary + commission) - 5000 * elevel - 20K)$

Function 9

$disposable > 0$ where,
$disposable = (0.67 * (salary + commission) - 5000 * elevel - 0.2 * loan - 10K)$

Function 10

$disposable > 0$ where,
$disposable = (0.67 * (salary + commission) - 5000 * elevel + 0.2 * loan - 10K)$
$equity = ((hyears < 20)?(equity = 0) : (equity = 0.1 * hvalue * (hyears - 20)))$

Contents

List of Figures