Efficient Evaluation of Queries with Mining Predicates

Surajit Chaudhuri
Microsoft Corp.
surajitc@microsoft.com

Vivek Narasayya
Microsoft Corp.
viveknar@microsoft.com

Sunita Sarawagi
IIT Bombay
sunita@it.iitb.ac.in

Abstract

Modern relational database systems are beginning to support ad hoc queries on mining models. In this paper, we explore novel techniques for optimizing queries that apply mining models to relational data. For such queries, we use the internal structure of the mining model to automatically derive traditional database predicates. We present algorithms for deriving such predicates for some popular discrete mining models: decision trees, naive Bayes, and clustering. Our experiments on Microsoft SQL Server 2000 demonstrate that these derived predicates can significantly reduce the cost of evaluating such queries.

1. Introduction

Progress in database technology has made massive warehouses of business data ubiquitous [10]. There is increasing commercial interest in mining the information in such warehouses. Data mining is used to extract predictive models from data that can be used for a variety of business tasks. For example, based on a customer's profile information, a model can be used for predicting if a customer is likely to buy sports items. The result of such a prediction can be leveraged in the context of many applications, e.g., a mail campaign or an on-line targeted advertisement.

Recently, several database vendors have made it possible to apply predictive models on relational data using SQL extensions. The predictive models can either be built natively or imported, using PMML or other interchange format. This enables us to express queries containing *mining predicates* such as: "Find customers who visited the MSNBC site last week and who are *predicted* to belong to the category of baseball fans". The focus of this paper is to optimize queries containing such mining predicates. To the best of our knowledge, this is the first study of its kind. The techniques described in this paper are general and do not depend on the specific nature of the integration of databases and data mining.

We propose a technique that exploits knowledge of the mining model's content to optimize queries with mining predicates. Today's systems would evaluate the above query by first selecting the customers who visited the MSNBC site, then applying the mining model (treated as black-box) on the selected rows, and filtering the subset that are predicted to be "baseball fans". In contrast, we wish to exploit the mining predicate for better access path selection, particularly if "baseball fans" represent a very small fraction of MSNBC visitors. The main challenge in exploiting mining predicates is that each mining model has its own specific method of predicting classes as a function of the input attributes, and some of these methods are too complex to be directly usable by traditional database engines.

We present a general framework in which, given a mining predicate, a model-specific algorithm can be used to infer a simpler derived predicate expression. The derived predicate expression is constrained to be a propositional expression consisting of simple selection predicates on attribute values. Such a derived predicate, which we call an *upper envelope* of the mining predicate, can then be exploited for access path selection like any other traditional database predicate.

We concentrate on predictive mining models that when applied to a tuple $\vec{x}$ predict one of K discrete classes $c_1, \ldots c_K$. Most classification and clustering models fall in this category. For every possible class c that the model M predicts, its upper envelope is a predicate of the form $M_c(\vec{x})$ such that the tuple $\vec{x}$ has class c only if it satisfies the predicate $M_c(\vec{x})$, but not necessarily vice-versa. We require that $M_c(\vec{x})$ is a propositional predicate expression consisting of simple selection conditions on attributes of $\vec{x}$. Such upper envelopes can be added to the query to generate a semantically equivalent query that would result in the same set of answers over any database. Since $M_c(\vec{x})$ is a predicate on the attributes of $\vec{x}$, it has the potential of better exploiting index structures and improving the efficiency of the query.

The effectiveness of such semantic optimization depends on two criteria. First, we must demonstrate that the upper-envelope predicates can be derived for a wide set of commonly used mining models. Second, we need to show that the addition of these upper-envelope predicates can have a significant impact on the execution time for queries with mining predicates. In turn, this requires that our derivation of upper envelopes to be "tight" and the original mining predicate to be selective so that they are effective in influencing the access path selection. Our extensive experiments on Microsoft SQL Server provide strong evidence of the promise of such semantic optimization. Moreover, our experiments demonstrate that little overhead is incurred during optimization for using such upper envelopes.

Outline: The rest of the paper is organized as follows. In Section 2 we review existing support for mining predicates in SQL queries in two commercially available relational database engines: Microsoft SQL Server's Analysis Server and IBM DB2's Intelligent Miner Scoring facility. In Section 3 we present algorithms for deriving such predicates for three popular discrete mining models: decision trees, naive Bayes classifiers and clustering. In Section 4 we discuss the operational issues of using upper envelopes to optimize queries with mining predicates. In Section 5 we report the results of our experimental study to evaluate the effectiveness of our technique in improving the efficiency of queries with mining predicates. We discuss related work in Section 6.

2. Expressing Mining Queries in Existing Systems

In this section, we describe some of the possible approaches to expressing database queries with mining predicates. We emphasize that our techniques are general in the sense that they do not depend on the specific nature of such integration of databases and data mining.

2.1. Extract and Mine

The traditional way of integrating mining with querying is to pose a traditional database query to a relational backend. The mining model is subsequently applied in the client/middleware on the result of the database query. Thus, for the example in the introduction, the mining query will be evaluated in the following phases: (a) Execute a SQL query at the database server to obtain all the customers who visited MSNBC last week (b) For each customer fetched into the client/middleware, apply the mining model to determine if the customer is predicted to be a "baseball fan".

2.2. Microsoft Analysis Server

In the Microsoft Analysis Server product (part of SQL Server 2000) mining models are explicitly recognized as first-class table-like objects. Creation of a mining model corresponds to schematic definition of a mining model. The following example shows creation of a mining model that predicts risk level of customers based on source columns gender, purchases and age using Decision trees.

```
CREATE MINING MODEL Risk_Class      // Name of Model
(
Customer_ID LONG KEY,               // source column
Gender TEXT DISCRETE,               // source column
Risk TEXT DISCRETE PREDICT,         // prediction column
Purchases DOUBLE DISCRETIZED(),     // source column
Age DOUBLE DISCRETIZED,             // source column
)
USING [Decision_Trees_101]          // Mining Algorithm
```

The model is trained using the INSERT INTO statement that inserts training data into the model (not discussed due to lack of space), Predictions are obtained from a model M on a dataset D using a prediction join [15] between D and M. A prediction join is different from a traditional equi-join on tables since the model does not actually contain data details. The following example illustrates prediction join.

```
SELECT D.Customer_ID, M.Risk
FROM [Risk_Class] M
PREDICTION JOIN
(SELECT Customer_ID, Gender, Age, sum(Purchases) as SP
  FROM Customers D Group BY Customer_ID, Gender, Age ) as D
  ON M.Gender = D.Gender
  and M.Age = D.Age
  and M.Purchases = t.SP
  Where M.Risk = "low"
```

In this example, the value of "Risk" for each customer is not known. Joining rows in the Customers table to the model M returns a predicted "Risk" for each customer. The WHERE clause specifies which predicted values should be extracted and returned in the result set of the query. Specifically, the above example has the mining predicate Risk = "low".

2.3. IBM DB2

IBM's Intelligent Miner (IM) Scoring product integrates the model application functionality of IBM Intelligent Miner for Data with the DB2 Universal Database [21] [1]. Trained mining models in flat file, XML or PMML format can be imported into the database. We show an example of importing a classification model for predicting the risk level of a customer into a database using a UDF called IDMMX.DM_impClasFile().

```
  INSERT INTO IDMMX.ClassifModels values ('Risk_Class',
     IDMMX.DM_impClasFile('/tmp/myclassifier.x'))
```

Once the model is loaded, it can be applied to compatible records in the database by invoking another set of User Defined Functions (UDFs). An example of applying the above classification mining model ("Risk_Class") on a data table called Customers is shown below.

```
SELECT Customer_ID, Risk
FROM (
  SELECT Customer_ID, IDMMX.DM_getPredClass(
    IDMMX.DM_applyClasModel(c.model,
        IDMMX.DM_applData(IDMMX.DM_applData('AGE',s.age),
         'PURCHASE',s.purchase))) as Risk
  FROM ClassifModels c, Customer_list s
  WHERE c.modelname='Risk_Class' and s.salary<40000
) WHERE Risk = 'low'
```

The UDF IDMMX.DM_applData is used to map the fields s.salary and s.age of the Customer_list table into the corresponding fields of the model for use during prediction. The UDF applyClasModel() applies the model on the mapped data and returns a composite result object that has along with the predicted class other associated statistics like confidence of prediction. A second UDF IDMMX.DM_getPredClass extracts the predicted class from this result object. The mining predicate in this query is: Risk = 'low'.

3. Deriving Upper Envelopes for Mining Predicates

We present algorithms for deriving upper envelopes for three popular mining models. We focus on mining models that produce a discrete class as output. The class of models whose prediction is real-valued is a topic of our future work. For some models like decision trees and rule-based classifiers, derivation of such predicates is straightforward as we show in Section 3.1. The process is more involved for naive Bayes classifiers and clustering as we show in Sections 3.2 and Sections 3.3 respectively.

In deriving these upper envelopes two conflicting issues that arise are the *tightness* and *complexity* of the upper envelope predicate. An upper envelope of a class c is said to be *exact* if it includes all points belonging to c and no point belonging to any other class. In most cases, where the model is complex we need to settle for looser bounds because both the complexity of the enveloping predicate and the running time for deriving the upper envelope might get intolerable. Complex predicates are also ineffective in improving the efficiency of the query because the DBMS might spend a lot of time in evaluating these otherwise redundant predicates. We revisit these issues in Sections 4.2.

3.1. Decision trees

In a decision tree [29] the internal nodes define a simple test on one of the attributes and the leaf-level nodes define a class label. An example of a decision tree is shown in Figure 1. The class label of a new instance is determined by evaluating the test conditions at the nodes and based on the outcome following one of the branches until a leaf node is reached. The label of the leaf is the predicted class of the instance. We extract the upper envelope for a class c, by ANDing the test conditions on the path from the root to each leaf of the class and ORing them together. Clearly, this envelope is *exact*. For the example in Figure 1 the upper envelope of class c_1 is "((lower BP $>$ 91) AND (age $>$ 63)

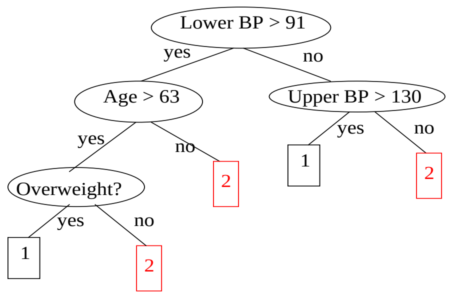

Figure 1. Example of a decision tree

AND (overweight)) OR ((lowerBP $\leq$ 91) AND (upper BP $>$ 130))". Similarly, of class c_2 is "((lower BP $>$ 91) AND (age $\leq$ 63)) OR ((lower BP $>$ 91) AND (age $>$ 63) AND (not overweight)) OR ((lowerBP $\leq$ 91) AND (upper BP $\leq$ 130))".

Extraction of upper envelopes for rule-based classifiers [27, 14] is similarly straightforward. A rule-based learner consists of a set of if-then rules where the body of the rule consists of conditions on the data attributes and the head (the part after "then") is one of the k class-labels. The upper envelope of each class c is just the disjunction of the body of all rules where c is the head. Unlike for decision trees, the envelope may not be exact because some rule learners allow rules of different classes to overlap. Therefore, an input instance might fire off two rules, each of which predicts a different class. Typically, a resolution procedure based on the weights or sequential order of rules is used to resolve conflict in such cases. It may be possible to tighten the envelope in such cases by exploiting the knowledge of the resolution procedure.

3.2. naive Bayes Classifiers

Extracting the upper envelopes for naive Bayes classifiers is considerably more difficult than for decision trees. We first present a primer on naive Bayes classifiers in Section 3.2.1. Then we present two algorithms for finding upper envelopes in Sections 3.2.2. Finally, we present a proof of correctness in Section 3.2.3.

3.2.1. Primer on naive Bayes classifiers

Bayesian classifiers [27] perform a probabilistic modeling of each class. Let $\vec{x}$ be an instance for which the classifier needs to predict one of K classes $c_1, c_2, \dots c_K$. The predicted class $C(\vec{x})$ of $\vec{x}$ is calculated as

$$C(\vec{x}) = \text{argmax}_k \Pr(c_k|\vec{x}) = \text{argmax}_k \frac{\Pr(\vec{x}|c_k)\Pr(c_k)}{\Pr(\vec{x})}$$

where $\Pr(c_k)$ is the probability of class c_k and $\Pr(\vec{x}|c_k)$ is the probability of $\vec{x}$ in class c_k. The denominator $\Pr(\vec{x})$ is the same for all classes and can be ignored in the selection of the winning class.

Let n be the number of attributes in the input data. Naive Bayes classifiers assume that the attributes $x_1, \dots, x_n$ of $\vec{x}$ are independent of each other given the class. Thus, the above formula becomes:

$$C(\vec{x}) = \text{argmax}_k(\prod_{d=1}^{n} \Pr(x_d|c_k)\Pr(c_k)) \quad (1)$$

$$= \text{argmax}_k(\sum_{d=1}^{n} \log\Pr(x_d|c_k) + \log\Pr(c_k)) \quad (2)$$

Ties are resolved by choosing the class which has the higher prior probability $\Pr(c_k)$.

The probabilities $\Pr(x_d|c_k)$ and $\Pr(c_k)$ are estimated using training data. For a discrete attribute d, let $m_{1d} \dots m_{n_d d}$ denote the n_d members of the domain of d. For each member m_{ld}, during the training phase we learn a set of K values corresponding to the probability $\Pr(x_d = m_{ld}|c_k)$. Continuous attributes are either discretized using a preprocessing step (see [17] for a discussion of various discretization methods) or modeled using a single continuous probability density function, the most common being the Gaussian distribution. In this paper we will describe the algorithm assuming that all attributes are discretized.

Example An example of a naive Bayes classifier is shown in Table 1 for $K = 3$ classes, $n = 2$ dimensions, first dimension d_0 having $n_0 = 4$ members and the second dimension d_1 having $n_1 = 3$ members. The triplet along the column margin show the trained $\Pr(m_{j1}|c_k)$ values for each of the three classes for dimension d_1. The row margin shows the corresponding values for dimension d_0. For example, the first triplet in the column margin (.01, .7, .05) stands for $(\Pr(m_{01}|c_1), \Pr(m_{01}|c_2), \Pr(m_{01}|c_3))$ respectively. The top-margin shows the class priors. Given these parameters, the predicted class for each of the 12 possible distinct instances $\vec{x}$ (found using Equation 1) is shown in the internal cells. For example, the value 0.001 for the top-leftmost cell denotes $\Pr(\vec{x}|c_1)$ where $\vec{x} = (m_{00}, m_{01})$.

3.2.2. Finding the upper envelope of a class

We next present algorithms for finding the upper envelope to cover all regions in the n dimensional attribute space where the naive Bayes classifier will predict a given class c_k. For example, the upper envelope for class c_2 in the example of Figure 1 is ($d_0 \in \{m_{20}, m_{30}\}$ AND $d_1 \in \{m_{01}, m_{11}\}$) OR ($d_1 = m_{01}$). We will express this envelope as two regions described by their boundaries as $(d_0 : [2..3], d_1 : [0..1]) \vee (d_1 : [0..0])$.

A simple way to find such envelopes is to enumerate for each combination in this n dimensional space the predicted class as we have done for the example above. We can then cover all combinations where class c_k is the winner with a collection of contiguous regions using any of the known multidimensional covering algorithms [2, 30]. Each region will contribute one disjunct to the upper envelope. This is in fact a generic algorithm applicable to any classification algorithm, not simply naive Bayes. Unfortunately, it is impractically slow to enumerate all $\prod_{d=1}^{n} n_d$ (n_d is the size of the domain of dimension d) member combinations. A medium sized data set in our experiments took more than 24 hours for just enumerating the combinations. We next present a top-down algorithm that avoids this exponential enumeration.

A top-down algorithm The algorithm proceeds in a top-down manner recursively narrowing down the region belonging to the given class c_k for which we want to find the upper envelope. The main intuition behind this algorithm is to exploit efficiently computable upper bounds and lower bounds on the probabilities of classes to quickly establish the winning and losing classes in a region consisting of several combinations.

The algorithms starts by assuming that the entire region belongs to class c_k. It then estimates an upper bound $\text{maxProb}(c_j)$ and lower bound $\text{minProb}(c_j)$ on the probabilities of each class c_j as follows:

$$\text{maxProb}(c_j) = \Pr(c_j)\prod_{d=1}^{n} \max_{l \in 1 \dots n_d} \Pr(m_{ld}|c_j)$$

$$\text{minProb}(c_j) = \Pr(c_j)\prod_{d=1}^{n} \min_{l \in 1 \dots n_d} \Pr(m_{ld}|c_j)$$

Computation of these bounds requires time only linear in the number of members along each dimension. In Figure 2(a) we show the minProb (second row) and maxProb (third row) values for the region shown in Figure 1. For example, in the figure the minProb value of 0.0005 for class c_2 is obtained by multiplying the three values $\Pr(c_2) = 0.5, \min_{l \in 0..3}\Pr(m_{l0}|c_2) = \min(0.1, 0.1, 0.4, 0.4) = 0.1, \min_{l \in 0..2}\Pr(m_{l1}|c_2) = \min(0.7, 0.29, 0.01) = 0.01$.

Using these bounds we partially reason about the class of the region to distinguish amongst one of these three outcomes.

1. MUST-WIN: *All points in the region belong to class* c_k. This is true if the minimum probability of class c_k ($\text{minProb}(c_k)$) is greater than the maximum probability ($\text{maxProb}(c_j)$) values of all classes c_j.
2. MUST-LOSE: *No points in the region belong to class* c_k. This is true if there exists a class c_j for which $\text{maxProb}(c_k) < \text{minProb}(c_j)$. In this case class c_j will win over class c_k at all points in this region.
3. AMBIGUOUS: Neither of the previous two conditions apply, i.e., possibly a subset of points in the region belong to the class.

In Section 3.2.3 we sketch a proof of why these bounds are correct and also show how to improve them further.

When the status of a region is AMBIGUOUS, we need to first shrink the region and then split it into smaller regions, re-evaluate the upper and lower bounds in each region and recursively apply the above tests until all regions either satisfy one of the first two terminating conditions or the al-

	$d_1 \downarrow$	$p(c_1) = 0.33, p(c_2) = 0.5, p(c_3) = 0.17$			
m_{01}	.01, .7, .05	.001, .03, .0005 (c_2)	.001, .03, .0005 (c_2)	.0002, .1, .004 (c_2)	.0002, .1, .004 (c_2)
m_{11}	.5, .29, .05	.07, .01, .0005 (c_1)	.07, .01, .0005 (c_1)	.009, .06, .004 (c_2)	.009, .06, .004 (c_2)
m_{21}	.49, .1, .9	.07, .0005, .009 (c_1)	.07, .0005, .009 (c_1)	.009, .002, .07 (c_3)	.009, .002, .07 (c_3)
	$d_0 \rightarrow$	.4, .1, .05	.4, .1, .05	.05, .4, .4	.05, .4, .4
		m_{00}	m_{10}	m_{20}	m_{30}

Table 1. Example of a naive-Bayes classifier. Refer the Example paragraph of Section 3.2.1 for a description.

Region: d_0, d_1	[0..3], [0..2]	[0..3], [2..2]	[0..3], [0..1]	[0..1], [0..1]	[2..3], [0..1]
MinProb:	.0002, .0005, .0005	.0002, .03, .0005	.009, .0005, .0005	.07, .0005, .0005	.009, .002, .004
MaxProb:	.07, .1, .07	.0014, .1, .004	.07, .06, .07	.07, .01, .009	.009, .06, .07
Status:	AMBIGUOUS	MUST-LOSE	AMBIGUOUS	MUST-WIN	AMBIGUOUS
	(a) Starting region	*(b) Tighter bounds with member m_{21} of d_1*	*(c) Shrinking d_1 to [0..1]*	*(d) 1st child on splitting d_0 into [0..1] and [2..3]*	*(e) 2nd child*

Figure 2. First three steps of finding predicates for class c_1 of the classifier in Figure 1 showing a shrinkage step along dimension 1 followed by a split along dimension 0. In each box, the first line identifies the boundary of the region, the second and third lines show respectively the minProb and maxProb values of each of the three classes. The fourth line is the status of the region with respect to class c_1.

gorithm has made a maximum number of splits (an input parameter of the algorithm). A sketch of the algorithm appears below.

Algorithm 1 UpperEnvelope(c_k)

```
1:  $T$: Tree initialized with the entire region as root;
2:  while number of tree nodes expanded $<$ Threshold do
3:      $r$= an unvisited leaf of $T$;
4:      $r$.status = Compute using $c_k$ and maxProb, minProb
        values of $r$;
5:      if $r$.status = MUST-WIN then mark $r$ as visited;
6:      if $r$.status = MUST-LOSE then remove $r$ from $T$;
7:      if $r$.status = AMBIGUOUS then
8:          Shrink $r$ along all possible dimensions;
9:          Split $r$ into $r_1$ and $r_2$;
10:         Add $r_1$ and $r_2$ to $T$ as children of $r$;
11:     end if
12: end while
13: Sweep $T$ bottom-up merging all contiguous leaves;
14: Upper_Envelope($c_k$) = disjunct over all leaves of $T$.
```

Shrink: We cycle through all dimensions and for each dimension d evaluate for each of its member m_{ld} the maxProb(c_j, d, m_{ld}) and minProb(c_j, d, m_{ld}) value as

$$\text{maxProb}(c_j, d, m_{ld}) = \Pr(c_j)\Pr(m_{ld}|c_j)\prod_{e \neq d}\max_r \Pr(m_{re}|c_j)$$

$$\text{minProb}(c_j, d, m_{ld}) = \Pr(c_j)\Pr(m_{ld}|c_j)\prod_{e \neq d}\min_r \Pr(m_{re}|c_j)$$

We use these revised tighter bounds to further shrink the region where possible. We test the MUST-LOSE condition above on the revised bounds and remove any members of an unordered dimension that satisfy this condition. For ordered dimensions, we only remove members from the two ends to maintain contiguity.

In Figure 2(a), from the minProb and maxProb values of the starting region $[0..3], [0..2]$ we find that for class c_1 neither of the MUST-WIN or MUST-LOSE situation hold. Hence the situation is AMBIGUOUS for c_1 and we attempt to shrink this region. In Figure 2(b) we show the revised bounds for the last member m_{21} of dimension 1. This leads to a MUST-LOSE situation for class c_1 because in the region maxProb for class c_1 is smaller than minProb for class c_2. The new maxProb and minProb values in the shrunk region are shown in Figure 2(c). The shrunk region is again in an AMBIGUOUS state and we attempt to split it next.

Split: Regions are split by partitioning the values along a dimension. In evaluating the best split, we want to avoid methods that require explicit enumeration of the class of each combination. In performing the split our goal is to separate out (as best as possible) the regions which belong to class c_k from the ones which do not belong to c_k. For this, we rely on the well-known entropy function [27] for quantifying the skewness in the probability distribution of class c_k along each dimension. The details of the split are exactly as in the case of binary splits during decision tree construction. We evaluate the entropy function for split along each member of each dimension and choose the split which has the lowest average entropy in the two sub-regions. The only difference is that we do not have explicit counts of each class, instead we rely on the probability values of the members on each side of the splitting dimension.

Continuing with our example, in Figure 2(d) and (e) we show the two regions obtained by splitting dimension d_0 into [0..1] and [2..3]. The first sub-region shown in Figure 2(d) leads to a MUST-WIN situation and gives one disjunct for the upper envelope of class c_1. The second region is still in an AMBIGUOUS situation – however a second round of shrinkage along dimension d_1 on the region leads to an empty region and the top-down process terminates.

Merging regions: Once the above top-down split process terminates, we merge all regions that do not satisfy the MUST-LOSE condition. During the course of the above partitioning algorithm we maintain the tree structure of the split so that whenever all children of a node belong to the same class, they can be trivially merged together. This is followed by another iterative search for pairs of non-sibling regions that can be merged. The output is a set of non-overlapping regions that totally subsume all combinations belonging to a class.

Complexity The above top-down algorithm has a complexity of $O(tnmK)$ where t is the threshold that controls the depth of the tree to which we expand and $m = \max_{d=1}^{n}(n_d)$ is the maximum length of a dimension. Contrast this with the exponential complexity $K\prod_{d=1}^{n} n_d$ of just the enumeration step of the naive algorithm.

3.2.3. Formal Results

This section contains a sketch of the proof of correctness of the top-down algorithm and can be skipped on first reading.

The main concern about the correctness of the above algorithm arises from the use of the maxProb and minProb bounds in determining the two MUST-WIN and MUST-LOSE conditions. We sketch a proof of why these bounds are correct and also present a set of improved bounds for the special case of two classes. In this proof we do not explicitly discuss the case where there is a tie in the $\Pr(c_k|\vec{x})$ values of two classes.

Lemma 3.1 If a region satisfies the MUST-WIN condition minProb$(c_k) > \max_{j\neq k}$ maxProb(c_j) then for every possible cell v in the region the probability of class c_k is greater than the probability of every other class. Let $p_j(m_{ld})$ denote $\Pr(m_{ld}|c_j)$. We wish to prove that

$$\Pr(c_k)\prod_{d=1}^{n}\min_{l} p_k(m_{ld}) > \max_{j\neq k}\Pr(c_j)\prod_{d=1}^{n}\max_{l} p_j(m_{ld}) \qquad (3)$$

implies

$$\forall v\left(\Pr(c_k)\prod_{d=1}^{n} p_k(v_d) > \max_{j\neq k}\Pr(c_j)\prod_{d=1}^{n} p_j(v_d)\right) \qquad (4)$$

That is, (3) ⇒ (4). Similar results hold for the MUST-LOSE condition.

PROOF. Let $f(v,j)$ denote $\Pr(c_j)\prod_{d=1}^{n}\Pr(v_d|c_j)$. If $\min_v f(v,k) > \max_{j\neq k}\max_v(f(v,j))$ then $f(v,k) > f(v',j)$ for all values v' and all classes $j \neq k$. Also $\min_v(\Pr(c_k)\prod_{d=1}^{n}\Pr(v_d|c_k)) = \Pr(c_k)\prod_{d=1}^{n}\min_{v_d}\Pr(v_d|c_j)$ because all the terms within the product are non-negative. Similarly, moving the $\max()$ beyond the $\prod$ leaves the result unchanged. Thus, (3) ⇒ (4).

■

We next present a lemma that will help us get *exact* bounds for the case when the number of classes $K = 2$.

Lemma 3.2 When the number of classes $K = 2$, the MUST-WIN and the MUST-LOSE bounds are exact when the probability values $\Pr(v_d|c_j)$ in condition 3 of Lemma 3.1 are replaced with $\Pr'(v_d|c_j) = \frac{\Pr(v_d|c_j)}{\max_{i\neq k}\Pr(v_d|c_i)}$. Let $p'_j(m_{ld})$ denote $\Pr'(m_{ld}|c_j)$. We wish to prove that, when $K = 2$ condition 4 *is equivalent to*

$$\Pr(c_k)\prod_{d=1}^{n}\min_{l} p'_k(m_{ld}) > \max_{j\neq k}\Pr(c_j)\prod_{d=1}^{n}\max_{l} p'_j(m_{ld}) \qquad (5)$$

Similar results hold for the MUST-LOSE condition.

PROOF. Omitted due to lack of space. ■

3.3. Clustering

Clustering models [22] are of three broad kinds: partitional, hierarchical and fuzzy. We concentrate on partitional clusters where the output is a set of k clusters and each point is assigned to exactly one of these k clusters. Hierarchical and fuzzy clusters are a subject of our ongoing work. Partitional clustering methods can be further subdivided based on the membership criteria used for assigning new instances to clusters. We consider three variants: centroid-based, model-based and boundary-based (commonly arising in density-based clusters).

In the popular centroid-based method each cluster is associated with a single point called the centroid that is most representative of the cluster. An appropriate distance measure on the input attributes is used to measure the distance between the cluster centroid and the instance. A common distance function is Euclidean or weighted Euclidean. The instance is assigned to the cluster with the closest centroid. This partitions the data space into K disjoint partitions where the i-th partition contains all points that are closer to the ith centroid than to any other centroid. A cluster's partition could take arbitrary shapes depending on the distance function, the number of clusters and the number of dimensions. Our goal is to provide an upper envelope on the boundary of each partition using a small number of hyper-rectangles.

A second class of clustering methods is model-based [25]. Model-based clustering assumes that data is generated from a mixture of underlying distributions in which each distribution represents a group or a cluster.

We show that both distance based and model-based clusters can be expressed exactly as naive Bayes classifiers for the purposes of finding the upper envelopes. Consider distance-based clustering first. Let $c_1, c_2 \ldots c_K$ be the K clusters, n be the number of attributes or dimensions of an instance $\vec{x}$ and $(c_{1k} \ldots c_{nk})$ be the centroid of the k-th cluster. Assume a weighted Euclidean distance measure. Let $(w_{1k} \ldots w_{nk})$ denote the weight values. Then, a point $\vec{x}$ is assigned to a cluster as follows:

$$\text{cluster of } \vec{x} = \text{argmax}_k \sum_{d=1}^{n} w_{dk}(x_d - c_{dk})^2$$

This is similar in structure to Equation 2 with the prior term missing. In both cases, for each component of $\vec{x}$, we have a set of K values corresponding to the K different clusters/classes. We sum over these n values along each dimension and choose of these K sums the class with the largest sum.

For several model-based clusters the situation is similar. Each group k is associated with a mixing parameter called τ_k ($\sum_{k=1}^{K} \tau_k = 1$) in addition to the parameters θ_k of the distribution function of that group. Thus, an instance will be assigned to the cluster with the largest value of

$$\text{cluster of } \vec{x} = \text{argmax}_k(\tau_k f_k(\vec{x}|\theta_k))$$

When the distribution function f_k treats each dimension independently, for example, mixtures of Gaussians with the covariance entries zero, we can again express the above expression in the same form as Equation 2.

Boundary-based clusters [18] explicitly define the boundary of a region within which a point needs to lie in order to belong to a cluster. Deriving upper envelopes is equivalent to covering a geometric region with a small number of rectangles. This is a classical problem in computation geometry for which several approximate algorithms exist [30, 2]. Further investigation of this problem is part of our future work.

4. Optimizing Mining Queries

So far we have considered examples of mining predicates of the form "Prediction_column = class_label". In Section 4.1, we show a wider class of mining predicates that may be optimized using upper envelopes for mining predicates of the above form. Then in Section 4.2 we discuss the key steps needed in enabling such optimization in a traditional relational database engine.

4.1. Types of mining predicates

We discuss three additional types of mining predicates that can be optimized using the derived per-class upper envelopes.

IN predicates: A simple generalization is mining predicates of the form: M.Prediction_column IN $(c_1, \ldots, c_l)$, where $c_1, \ldots, c_l$ are a subset of the possible class labels on M.Prediction_column. An example of such a query is to identify customers who a data mining model predicts to be either baseball fans or football fans. For such a mining predicate, the upper envelope is a disjunction of the upper envelopes corresponding to each of the atomic mining predicates. Thus, if M_{c_i} denotes the predicate (M.$Prediction\_column = c_i$), we can express the overall disjunct as: $\bigvee_{i=1}^{l} M_{c_i}$

Join predicates between two predicted columns: Another form of join predicates is M1.Prediction_column1 = M2.Prediction_column2. Such predicates select instances on which two models M1 and M2 concur in their predicted class labels. An example of such a query is "Find all microsoft.com visitors who are predicted to be web developers by two mining models $SAS\_customer\_model$ and $SPSS\_customer\_model$". In order to optimize this query using upper envelopes, we assume that the class labels for each of the mining models can be enumerated during optimization by examining the metadata associated with the mining models. In typical mining models we expect the number of classes to be quite small. Let the class labels that are common to these two mining models be $\{c_1, c_2, .., c_k\}$. Then, the above join predicate, is equivalent to this disjunction: $\bigvee_{i=1}^{k}$(M1.Prediction_column1 = M2.Prediction_column2 = c_i). Adopting the notation of the previous paragraph, this can be expressed as: $\bigvee_i (M1_{c_i} \wedge M2_{c_i})$. Note that if M1 and M2 are identical models, then the resulting upper envelope results in a tautology. Conversely, if M1 and M2 are contradictory, then the upper envelope evaluates to false and the query is guaranteed to return no answers. These observations can be leveraged during the optimization process to improve efficiency.

Join predicates between a predicted column and a data column: Consider predicates of the form M1.Prediction_column = T.Data_column that check if the prediction of a mining model matches that of a database column. An example of this type of predicate is: "Find all customers for whom predicted age is of the same category as the actual age"[1]. Such queries can occur, for example, in cross-validation tasks. Evaluation of the above query seems to require scanning the entire table. Fortunately, like in the previous paragraph, we can use the approach of enumerating the set of possible class labels. Once again, such an approach is feasible since in most mining models we expect the number of class to be small. If the set of classes are $\{c_1, c_2, .., c_k\}$, then, we can derive an implied predicate $\bigvee_i (M1_{c_i} \wedge T.Data\_column = c_i)$. This transforms the query to a disjunct or a union of queries. More importantly, we now have the option of leveraging the content of the mining model for access path selection. For exam-

[1] In this example, we consider age as a discretized attribute with the domain consisting of three categories: ''young", "middle-aged', "senior".

ple, for the i-th disjunct, the optimizer can potentially consider either the predicate $T.Data\_column = c_i$ or a predicate in $M1_{c_i}$ for access path selection. Of course, the final plan depends on other alternatives considered by the optimizer (including sequential scan) but our rewriting opens the door for additional alternatives. In addition to the above technique, the traditional approach of exploiting transitivity of predicates in the WHERE clause can also be effective. For example, if the query contains additional predicates on T.Data_columns that indirectly limits the possible domain values M1.Prediction_column can assume, then we can apply the optimization of the IN predicates discussed earlier in this section. For example, if the query were "Find all customers for which predicted age is the same as the actual age and the actual age is either old or middle-aged" then, via transitivity of the predicate, we get a predicate M.Prediction_column IN ('old', 'middle-aged') for which we can add the upper-enveloping predicates as discussed in the earlier paragraphs.

4.2. Key Steps in Optimization of Mining Predicates

The framework for optimizing queries with mining predicates has two key parts. First, during training of the mining models, upper envelopes for mining predicates of the form Model.Prediction_column = class_label have to be precomputed using the algorithms described in Section 3. Precomputation of such "atomic" upper envelopes reduces overhead during query optimization. Second, during query optimization we optimize queries with mining predicates using the following key steps:

1. Apply traditional normalization and transitivity rules to the given query to derive an equivalent query to be used for the following steps.
2. For each mining predicate f in the query, do the following. Assume that the mining predicate f references a mining model m_f:
   (a) Look up the information on class labels of m_f from the database, if needed.
   (b) Depending on the type of the mining predicate, derive an additional upper envelope u_f using the techniques described in Section 4.1. Computation of such an upper envelope requires looking up "atomic" upper envelopes computed during training (see earlier in this subsection).
   (c) Replace m_f with $m_f \wedge u_f$.
3. Apply normalization and transitivity rules to derive an equivalent query. If new mining predicates are inferred, return to step 2, else return.

Our experiments demonstrate that the additional work during training to derive "atomic" upper envelopes as well as step 2(b) during query optimization add little additional overhead in themselves. However, our strategy for optimization relies on the following assumptions about query optimization and evaluation.

Complexity of upper envelopes does not impact execution cost: We assume the following: (a) The evaluation of upper envelopes do not add to the cost of the query. This is consistent with traditional assumptions made in database optimization since every upper envelope consists of AND/OR expression of simple predicates. (b) The optimizer is well-behaved and is not misguided by the introduction of additional complex boolean predicates due to upper envelopes. We rely on optimizers whose selectivity computations and access path selections are robust for complex boolean expressions. Although we make the above two assumptions for simplicity, they rarely hold in all situations. Failure to satisfy condition (a) can be dealt by more careful rewriting. For example, if none of the predicates in the upper envelope is chosen for access path, the upper envelope can be removed at the end of the optimization. In general, we need to retain only a subset of relevant upper envelope for evaluation as filter conditions. We omit these details due to lack of space. Unfortunately, handling violation of condition (b) is more challenging, yet happens routinely. Today's query optimizers often degenerate to sequential scan when presented with a complex AND/OR expression. This would negate any benefits of upper envelopes as the latter typically consist of several disjuncts over conjuncts of atomic predicates on the data columns. Despite past work (e.g., [28]), handling complex filter conditions remains a core challenge for SQL query optimizers. This remains an area of our active research in the context of query optimization. However, for the time being, we rely on thresholding of the number of disjuncts (see Section 3.2) and simplification based on selectivity estimates to limit the complexity so that commercial optimizers are able to exploit upper envelopes. We omit detailed discussion due to lack of space.

Accessing content of mining models during query optimization should be enabled: Our strategies for deriving upper envelopes (as described in Section 4.1) requires access to content of the mining models (e.g., class labels) *during* optimization. Such information is different from the traditional statistical information about tables because the correctness of our optimization is impacted if the mining model is changed. In such cases, we need to invalidate an execution plan (if cached or persisted) in case it had exploited upper envelopes. Nonetheless, our approach of leveraging the content of mining models is justified because mining models evolve slowly and the size of a typical mining model is relatively small compared to data size. Therefore, optimization time is not severely impacted for accessing the content of a mining model.

Data Set	Test size in millions	Training size	# of classes	# of clusters
Anneal-U	1.83	598	6	6
Balance-Scale	1.28	416	3	5
Chess	1.63	2130	2	5
Diabetes	1.57	512	2	5
Hypothyroid	1.78	1339	2	5
Letter	1.28	15000	26	26
Pairty5+5	1.04	100	2	5
Shuttle	1.85	43500	7	7
Vehicle	1.73	564	4	5
Kdd-cup-99	4.72	100000	23	23

Table 2. Summary of Data Sets used in experiments

5. Experiments

In this section, we present results of experiments to evaluate the effectiveness of upper envelope predicates generated by algorithms presented in Section 3. Our experiments focussed on three important aspects: (i) Impact of upper envelope predicates on the running time and physical plan of queries. We study this in Section 5.2.1. (ii) Degree of tightness of the approximation, studied in Section 5.2.2. (iii) Time taken to generate upper envelope predicates. The significant outcome of the last experiment was that in almost all data sets the time to precompute the upper envelope predicate for each class (see Section 4.2) was a negligible fraction of the model training time. Likewise, the time to look up "atomic" upper envelope predicates was insignificant compared to the time for optimizing the query. We do not present further details of this experiment due to lack of space.

5.1. Experimental Setup

Mining Models: We have implemented the algorithms presented in Section 3 for the decision tree, naive Bayes and clustering mining models. We generated decision tree and clustering mining models using Microsoft Analysis Server that ships with Microsoft SQL Server 2000. For generating naive Bayes mining models we used the discrete naive Bayes inducer packaged with the MLC++ machine learning library [23].

Data Sets: We report numbers on 10 data sets consisting of 9 UCI [7] data sets and the 1999 KDDcup data set available at [5]. Table 2 summarizes various characteristics of each data set. We generated the test data set (for the UCI data sets) by repeatedly doubling all available data until the total number of rows in the data set exceeded 1 million rows. This way, the data distribution of each column (and hence selectivity of predicates on the column) in the test data set is the same as in the training data set. All data sets were stored in Microsoft SQL Server databases.

Implementation: When executing a mining query, we first identify the mining model object(s) referenced in the mining query and identify mining predicates for which generation of upper envelopes may be possible. In our current implementation, generation of upper envelope predicates is not integrated with the database engine; rather we rewrite the mining query externally to include the upper envelope predicates, and submit the rewritten query to the database engine. The upper envelopes are generated during training time by referring the MINING_MODEL_CONTENT schema rowset defined in the OLE DB for Data Mining [15] interface.

Evaluation Methodology: For each class (or cluster), we first generate the query with the upper envelope predicate for that class. Thus, if T is the table containing the test data, and $\langle p \rangle$ is the upper envelope predicate, we generate the query "SELECT * FROM T WHERE $\langle p \rangle$". We create a workload file containing all queries for the (data set, mining model) combination. Thus, the number of queries in the workload file is equal to the number of classes (or clusters) for that (data set, mining model) combination. To generate an appropriate physical design for this workload, we invoke the Index Tuning Wizard tool [12, 4] that ships with Microsoft SQL Server 2000 by passing it the above workload file as input, and implement the index recommendations proposed by the tool. We then execute the workload on the database and record the plan and running time of each query in the workload. We compare this with a query that performs a full scan of the table, i.e., "SELECT * FROM T".

Although in practice, mining queries may also contain other predicates, the above comparison with a "SELECT *" query is reasonable since our goal is to determine if addition of upper envelopes can reduce running time in a significant number of cases (due to indexed access path selection). Whether upper envelope predicates are indeed chosen over other predicates for indexing will of course depend on other predicates and their relative selectivity. Finally, a design that stores the class label with each tuple (e.g., as an additional column) in the base relation is not acceptable since (a) It does not scale well with the number of mining models and (b) In many cases, mining queries are issued not over the base relations but on queries (or views) over possibly multiple base relations Note that such precomputation of the class label may however be appropriate in limited cases (e.g., in materialized views).

5.2. Results

5.2.1. Impact of Upper Envelope Predicates on Running Time and Plan

We first evaluate the impact of upper envelope predicates on the running time of all queries for all mining models. The following table shows the average reduction in running time over all queries for each type of mining model, compared to a full scan of the data. We note that the reduction

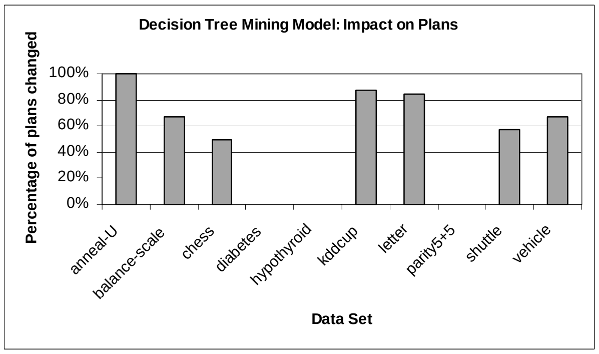

Figure 3. Impact of upper envelope predicates on physical plan for decision tree model

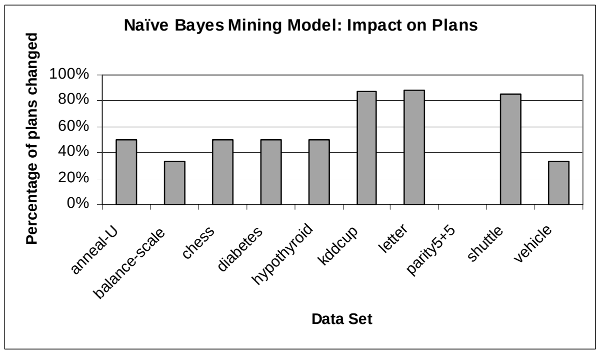

Figure 4. Impact of upper envelope predicates on physical plan for naive Bayes model

in running time we report here is in comparison to a "SELECT *", which does not include the time for actually invoking the mining model on the columns. If the application of mining models is time consuming, then we can expect to see an even greater percentage reduction.

Decision Tree	Naive Bayes	Clustering
73.7%	63.5%	79.0%

To further analyze the reason for the reduced running time, we measured the impact of the upper envelope predicates on the physical plan chosen by the query optimizer. For a given data set and mining model, we recorded for each query whether the plan chosen by the query optimizer changed compared to the query without upper envelope predicates. A plan is said to have *changed* if either: (a) The query optimizer chose one or more indexes to answer the query. (b) The query optimizer decided to use a "Constant Scan" operator since upper envelope predicate was NULL (i.e., it does not need to reference the data at all to answer the query). The table below shows the percentage of queries for which the plan changed over all data sets and mining models.

Decision Tree	Naive Bayes	Clustering
72.7%	75.3%	76.6%

As we can see from this table, for all types of mining models, a significant fraction of the queries had their physical plans altered as a result of introducing upper envelope predicates.

We now analyze these results further by drilling-down into the results for each data set. Figures 3, 4 and 5 show these numbers for the decision tree, naive Bayes and clustering mining models respectively. We observe that upper envelope predicates have greater impact on the plan for data sets where the number of classes is relatively large (e.g., kddcup, letter, shuttle etc.), and less impact for data sets where number of classes is small (e.g., Diabetes, Parity etc.). This is due to the fact that when the number of classes is large, there are typically more classes with small selectivity for which the query optimizer picks an index to answer the query. In fact, in some cases, the selectivity is 0, i.e., the upper envelope predicate is NULL. In such cases, the optimizer does not need to access any data to answer the query. A more detailed analysis of the average reduction in running time as a function of the selectivity (both original and upper envelope) of the class/cluster over all classes and clusters of all mining models and data sets is shown in Figure 6. We see that the reduction in running time is most significant when the selectivity is below 10%. Also, a comparison of the bars for original and upper envelope selectivities shows that the low reduction in running time for higher selectivities is not a reflection of the effectiveness of our algorithm. Rather, when a predicate's selectivity is high (e.g., above 10%) the optimizer rarely selects indexes, particularly non-clustered indexes. Thus, for high selectivity classes, adding upper envelope predicates is rarely useful, even if we could find *exact* predicates.

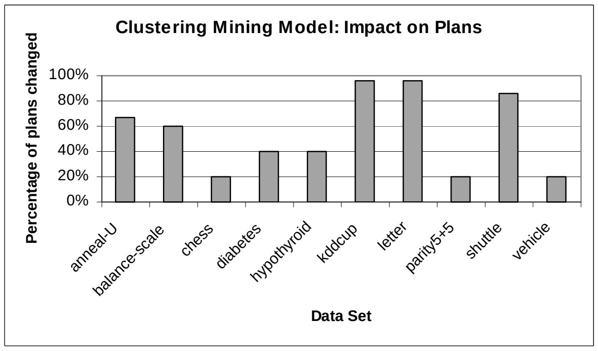

Figure 5. Impact of upper envelope predicates on physical plan for clustering model

Finally, we noticed that in many cases, the upper envelope predicates generated by our algorithms for these data sets are relatively simple, i.e., consisting of few disjuncts. This increases the likelihood that the query optimizer can use an index lookup to answer the query. Overall, this experiment confirms our intuition that inclusion of upper envelope predicates significantly impacts the plan, and hence running times of queries with mining predicates.

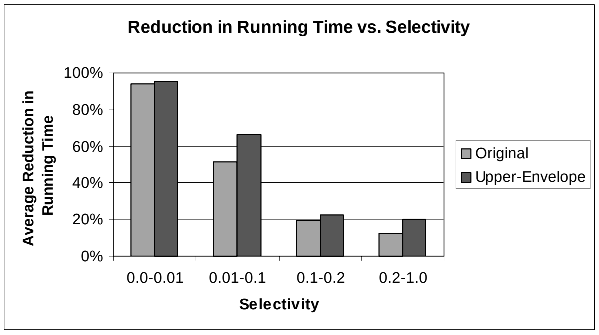

Figure 6. Running Time improvement vs. Original Selectivity: All mining models and data sets

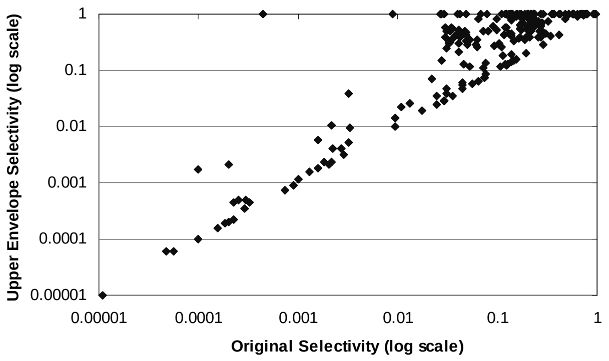

Figure 7. Tightness of approximation: naive Bayes and clustering

5.2.2. Tightness of Approximation

In this experiment, we compare the tightness of approximation of the upper envelopes for naive Bayes and clustering. For decision-trees, since the upper envelopes are exact, this comparison is not necessary. Figure 7 shows for all classes in all data sets for the naive Bayes and clustering mining models, a scatter plot of the original selectivity of each class vs. the selectivity of the corresponding upper envelope predicate (on a log scale). Each point in the scatter-plot corresponds to one class of a data set.

As we see from the figure, a significant fraction of the upper envelope predicates either have selectivities close to the original selectivity or have selectivity small enough that use of indexes for answering the predicate is attractive. Most cases where the algorithm failed to find a tight upper envelope correspond to cases where the original selectivity is large to start with. In such cases, the upper envelope predicates are unlikely to be useful for improving access paths even if they were exact.

6. Related Work

Our work falls in the broader area of integration of data mining and database systems and there are several pieces of related work in that area. The case for building the infrastructure for supporting mining on not only stored results but also on the result of an arbitrary database query was made in [9]. Agrawal et al [3] looked at the problem of generating decision tree classifiers such that the predicate could easily be pushed into the SQL query. However, they do not discuss how the method will work for other mining models. Other complementary areas of work include construction of mining models using SQL [31] and defining language extensions and application programming interfaces for integrating mining with relation DBMS [26], [1] and [15]. More recently, database systems, such as Microsoft Analysis Server or IBM DB2 have enabled specification of such queries. However, none of these systems exploit mining predicates for *optimization* in a general setting. Our paper represents the first work in that direction.

Our work can be viewed as part of the broader field of semantic query optimization. Early work in database systems recognized value of query modification (e.g., in INGRES) whereby a semantically implied predicate, perhaps derived from integrity constraints, is added to make evaluation of the query more efficient. Our technique follows the same approach but our novelty is in the specific information we exploit - the internal structure of the mining model to derive upper envelopes. To the best of our knowledge, this has not been attempted before. There has been past study of upper envelopes that represent approximation to given recursive queries [11, 8] but these do not apply to mining predicates.

Recently, there has been work on optimization of user-defined functions and predicates [19, 13, 32]. Mining predicates can certainly be viewed as user-defined predicates Thus, it is an interesting research question whether our idea of deriving implied database predicates based on content of mining models can be effectively applied to other examples of user-defined predicates as well.

The problem of rule extraction from hard to interpret models like Neural networks [24, 16] bears resemblance, but differs from our problem in that the extracted rules need to approximate the classification function but are *not* required to be implied predicates (upper envelopes). Moreover, the algorithm for rule learning as proposed in [24] requires an enumeration of the discretized input space, similar to our first-cut bottom up algorithm. Such an approach has been shown to be infeasible in our case.

The *coverage* problem has been addressed in several different contexts, including, covering a set of points with the smallest number of rectangles [30, 2], covering a collection of clauses with simpler terms in logic minimization problems [20] and constructing clusters with rectilinear boundaries [6]. Despite the apparent similarities, the coverage problems differ in a few important aspects. First, they assume that the points are already enumerated in the n-dimensional space. This is not a feasible option in our case. Next, the first two problems require an exact cover of smallest size whereas we only need an upper envelope. Finally, most of these approaches assume a small number of dimensions (two or three) and do not scale to higher dimensions.

References

[1] SQL multimedia and application packages part 6: Data mining, ISO draft recommendations, 1999.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. ACM SIGMOD International Conf. on Management of Data*, Seattle, USA, June 1998.

[3] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proc. of the VLDB Conference*, pages 560–573, Vancouver, British Columbia, Canada, August 1992.

[4] S. Agrawal, S. Chaudhuri, L. Kollar, and V. Narasayya. Index tuning wizard for Microsoft SQL Server 2000. White paper. `http://msdn.microsoft.com/library/techart/itwforsql.htm`, 2000.

[5] S. D. Bay. The UCI KDD archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science, 1999.

[6] M. Berger and I. Regoutsos. An algorithm for point clustering and grid generation. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5):1278–86, 1991.

[7] C. Blake and C. Merz. UCI repository of machine learning databases, 1998.

[8] S. Chaudhuri. Finding nonrecursive envelopes for datalog predicates. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC*, pages 135–146, 1993.

[9] S.Chaudhuri. Data mining and database systems: Where is the intersection? In *Bulletin of the Technical Committee on Data Engineering*, volume 21, Mar 1998.

[10] S. Chaudhuri and U. Dayal. An overview of data warehouse and OLAP technology. *ACM SIGMOD Record*, March 1997.

[11] S. Chaudhuri and P. G. Kolaitis. Can datalog be approximated? In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 24-26, 1994, Minneapolis, Minnesota*, pages 86–96, 1994.

[12] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for microsoft sql server. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 146–155, 1997.

[13] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 87–98, 1996.

[14] W. W. Cohen. Fast effective rule induction. In *Proc. 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[15] M. Corporation. OLE DB for data mining. `http://www.microsoft.com/data/oledb`.

[16] M. W. Craven and J. W. Shavlik. Using neural networks for data mining. In *Future Generation Computer Systems*, 1997.

[17] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proc. 12th International Conference on Machine Learning*, pages 194–202. Morgan Kaufmann, 1995.

[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, August 1996.

[19] J. M. Hellerstein and M. Stonebraker. Predicate migration: Optimizing queries with expensive predicates. In *SIGMOD Conference*, pages 267–276, 1993.

[20] S. J. Hong. MINI: A heuristic algorithm for two-level logic minimization. In R. Newton, editor, *Selected Papers on Logic Synthesis for Integrated Circuit Design*. IEEE Press, 1987.

[21] IBM. *IBM Intelligent Miner Scoring, Administration and Programming for DB2 Version 7.1*, March 2001.

[22] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, 1990.

[23] R. Kohavi, D. Sommerfield, and J. Dougherty. Data mining using MLC++: A machine learning library in C++. In *Tools with Artificial Intelligence*, pages 234–245. IEEE Computer Society Press, available from `http://www.sgi.com/tech/mlc/`, 1996.

[24] H. Lu, R. Setiono, and H. Lui. Neurorule: A connectionist approach to data mining. In *Proc. of the Twenty first Int'l conf. on Very Large Databases (VLDB)*, Zurich, Switzerland, Sep 1995.

[25] G. McLachlan and K. Basford. Mixture models: Inference and applications to clustering, 1988.

[26] R. Meo, G. Psaila, and S. Ceri. An extension to sql for mining association rules. *Data Mining and Knowledge Discovery*, 2(2):195–224, 1998.

[27] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[28] C. Mohan, D. Haderle, Y. Wang, and J. Cheng. Single table access using multiple indexes: optimization, execution, and concurrency control techniques. In *Proc. International Conference on Extending Database Technology*, pages 29–43, 1990.

[29] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1993.

[30] R. A. Reckhow and J. Culberson. Covering simple orthogonal polygon with a minimum number of orthogonally convex polygons. In *Proc. of the ACM 3rd Annual Computational Geometry Conference*, pages 268–277, 1987.

[31] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with databases: alternatives and implications. In *Proc. ACM SIGMOD International Conf. on Management of Data*, Seattle, USA, June 1998.

[32] G. M. Wolfgang Scheufele. Efficient dynamic programming algorithms for ordering expensive joins and selections. In *Proc. of the 6th Int'l Conference on Extending Database Technology (EDBT)*, Valencia, Spain, 1998.