# Efficient Evaluation of Queries with Mining Predicates

Surajit Chaudhuri
Microsoft Corp.

Vivek Narasayya
Microsoft Corp.

Sunita Sarawagi
IIT Bombay

## 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 may be able to significantly reduce the cost of evaluating such queries.*

## 1. Introduction

Progress in database technology has made massive data warehouses of business data ubiquitous [9]. There is increasing commercial interest in mining the information in such warehouses. Specifically, 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 could have either been built natively or imported, e.g., 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 where the application of prediction models is integrated with other SQL 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.

In this paper, we propose a technique to exploit the knowledge of the mining model's content to optimize the evaluation of 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 observe that if "baseball fans" represent a very small fraction of the set of customers who visited MSNBC last week, then it is attractive to exploit the above mining predicate for query optimization, i.e., leverage its low selectivity for access path selection. Unfortunately, it is a challenging task to exploit the mining predicate for query optimization. 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 over attribute values as in a WHERE clause of a SQL query. 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. The challenge then is to be able to derive such upper envelopes efficiently that are also "reasonably tight".

We concentrate on predictive mining models that when applied to an instance $\vec{x}$ predict a single discrete class $c$ as the outcome. Most classification and clustering models fall in this category. A second class of models whose prediction is real-valued is deferred to a future topic. In our setting a predictive model $M$ assigns one of $K$ distinct classes $c_1, \dots c_K$ to an input tuple $\vec{x}$ consisting of $n$ attributes. For every possible class $c$ that the model $M$ generates, its upper envelope is a predicate of the form $M_c(\vec{x})$ such that the instance $\vec{x}$ has class $c$ only if it satisfies the predicate $M_c(\vec{x})$, but not necessarily vice-versa. As stated above, we require that $M_c(\vec{x})$ is a simple filter expression over attributes of $\vec{x}$ i.e., it is an AND/OR expression as in the WHERE clause of a simple selection query in SQL. 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 useful mining models with low overhead. In this paper, we discuss in detail how for a wide set of mining models we can do exactly that by looking only at the content of the mining model. 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.

**Outline:** The rest of the paper is organized as follows. In Section 2 we review the 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 the 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 method in improving the efficiency of queries with mining predicates. We discuss related work in Section 6 and conclude with Section 7.

# 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 an example of creating 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 to insert 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. Of course, the data needs to match the schema of the mining model. Since the model does not actually contain data details, a prediction join is different from a traditional equi-join on tables. The following example illustrates prediction join. In this example, the value for "Risk" is not known. Joining rows in the Customers table to the model and selecting the "Risk" column from the model returns a predicted "Risk" for each of the customers.

```
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
  ON M.Gender = D.Gender
  and M.Age = D.Age
  and M.Purchases = t.SP
  Where M.Risk = "low"
```

When a model is joined with a table, predicted values are obtained for each case that "matches" the model (i.e., cases for which the model is capable of making a prediction). The WHERE clause associated with the prediction join 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 [22]. The programming interface for IM Scoring is based on the ISO draft recommendations for accessing data mining functionality through SQL [1]. Trained mining models in flat file, XML or PMML format can be imported into the database. Imported models are stored as named entities in system catalog tables based on their type. 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 into a database, 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_level = 'HIGH-RISK'.

**Summary** In this section, we saw two example queries with mining predicates. In existing systems these predicates would typically be applied as a filter during the last stage of query processing. Our goal is to extract from the mining model, additional enveloping predicates corresponding to the original mining predicate. For example, in the second query, by consulting the mining model we might find that whenever Risk = 'low', age $< 25$. If there is an index on the age attribute, this new predicate could be used to provide a better access path and retrieve only a subset of the rows. In the next section we describe how to extract these enveloping predicates for different mining models.

# 3. Deriving Upper Envelopes for Mining Predicates

In this section, we present algorithms for deriving the upper envelopes for three popular discrete-output models.

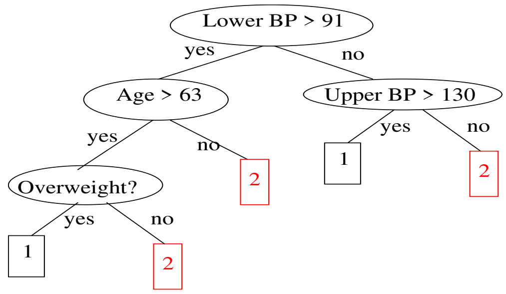



Figure 1. Example of a decision tree

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 shown 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 enveloping predicate. An upper envelope of a class $c$ is said to be *exact* if it is a tight bound on all instances predicted to be $c$, that is, if it includes all points belonging to $c$ and no point that belong 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 engine might spend a lot of time in evaluating these otherwise redundant predicates. We revisit these issues in Sections 4.2 and 5.2.3.

## 3.1. Decision trees

In a decision tree [32] 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. In extracting the upper envelope for a class $c$, for each leaf predicting $c$, we derive a conjunct by ANDing the test conditions on the path from the root to the leaf. The final envelope is the disjunct (OR) of these conjuncts. Clearly, these envelopes are *exact*.

For the example in Figure 1 the upper envelope of class $c_1$ is "((lower BP $>$ 91) and (age $>$ 63) 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 [29, 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 and 3.2.3 respectively. Finally, we present a proof of correctness (that can be skipped on first reading) in Section 3.2.4.

### 3.2.1. Primer on Naive Bayes classifiers

We present a brief primer on Naive Bayes classifier first [29]. Bayesian classifiers perform a probabilistic modeling of each class. Let $\vec{x}$ be an instance for which the classifier needs to predict one of $K$ classes $c_1, c_2, \ldots c_K$. The predicted class 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 prior probability of class $c_k$ and $\Pr(\vec{x}|c_k)$ is the probability of vector $\vec{x}$ 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 $x_1 \ldots x_n$ 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 $m_{1d} \ldots m_{n_d d}$ 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 $\Pr(x_d = m_{ld}|c_k)$.

**Example** An example of a naive-bayes classifier is shown in Table 1 for $K = 3, n = 2, n_0 = 4, n_1 = 3$. The triplet along the column and row margins show the trained $\Pr(x_d = m_{jd}|c_k)$ values for each of the three classes. The top-margin shows the class priors. Given these parameters, the predicted class for each of the 12 possible distinct instance $\vec{x}$ (found using Equation 1) is shown in the internal cells.

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. The parameters of this distribution are estimated using the training data. In this paper we will describe the algorithm assuming that all attributes are discrete. Continuous attributes, if any, are assumed to be discretized before invoking the naive-bayes algorithm. This algorithm can be extended to handle continuous attributes but we do not go into its details in this paper.

### 3.2.2. A bottom-up algorithm

Consider the region defined in $n$ dimensions[1] where a given class $c_k$ is the winner. In general, the region could be of any arbitrary shape and in most cases will not even be connected. Our goal is to cover *all* combinations where class $c_k$ is the winner with a collection of rectangles.

A naive solution is to explicitly enumerate each possible member combination in $n$ dimensions; giving rise to $\prod_{d=1}^{n} n_d$ combinations where $n_d$ is the size of the domain of dimension $d$. For each combination calculate the winning class. Then use an appropriate covering algorithm [2, 33, 19] to find the upper cover of all cells of class $c_k$ with a small number of rectangles. This solution turns out to be impractically slow since the enumeration phase gets bogged down by the exponential blowup of the number of combinations. For example, in Table 3 the fourth column (marked "combinations") shows the number of possible combinations for twenty different data sets. 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 the explicit enumeration of the naive solution through appropriately constructed bounds.

### 3.2.3. 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. It starts by assuming that the entire region belongs to class $c_k$. It then estimates an upper and lower bound on the probabilities for each class within the region. The upper bound is obtained by taking a product of the maximum probability values for each dimension for each class and similarly the

[1] We use the terms *dimension* and *attribute* interchangeably.

| $d_1 \downarrow$ | | $p(c_1) = 0.33, p(c_2) = 0.5, p(c_3) = 0.17$ | | | |
|---|---|---|---|---|---|
| $m_{01}$ | .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 with $K = 3$ 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 three values in each cell along the dimension denote the probability values $\Pr(m_{ld}|c_k)$ for each of the three classes. The internal cells are proportional to the $\Pr(c_k|\vec{x})$ values for an instance $\vec{x}$ with the corresponding values along the two dimensions. The winning class (the position with the largest of the three values) is shown within brackets.

lower bound is obtained by multiplying the minimum probability values. Let maxProb$(c_j)$ and minProb$(c_j)$ denote the maximum and minimum probability of class $c_j$ respectively. Thus,

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

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

In Figure 2(a) we show the maxProb and minProb 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 $\Pr(c_2) = 0.5, \min_{l \in 0..3} \Pr(m_{l0}|c_2) = 0.1, \min_{l \in 0..2} \Pr(m_{l1}|c_2) = 0.1$.

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

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

Later in Section 3.2.4 we sketch a proof of why these bounds are correct and also show how to improve them further.

When the third condition applies, we split the region into smaller regions, re-evaluate the upper and lower bounds in each region and recursively apply the above tests until all regions satisfy the first two terminating condition or the algorithm has made a maximum number of splits (an input parameter of the algorithm).

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 and we need to split the region further.

**Split:** We next present the method used for splitting a region. Regions are split by partitioning the values along a dimension. We cycle through the 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

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

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

This provides us a tighter bound of the maxProb and minProb values specialized for a given member of dimension $d$. We use these revised bounds to further shrink or split the region as follows:

First, we trim the region along both sides of each dimension so as to remove regions that are sure not to belong to $c_k$ by testing the MUST-LOSE condition above on the revised bounds. 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).

If a region cannot be shrunk further, we split it. 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 [29] 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

| | | | | | |
|---|---|---|---|---|---|
| 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 $d_1 = 2$* | *(c) Shrinking $d_1$ to [0..1]* | *(d) Spliting $d_0$ into [0..1] and [2..3]* | *(e) Spliting $d_0$ into [0..1] and [2..3]* |

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 values respectively the minProb and maxProb of each of the three classes. The fourth line is the status of the region with respect to class $c_1$.

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.

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.

**Reordering the discrete attributes:** We can improve the efficiency of the algorithm by clustering the members of each discrete dimension so as to bring together the cells that most likely belong to class $c_k$. For each dimension $d$, the members are ordered in decreasing order of the value of $\Pr(m_{ld}|c_k) - \max_{j\neq k}(\Pr(m_{ld}|c_j))$. When the number of classes is 2, this reordering ensures that the region covered by class $c_k$ is a connected stair-case like structure starting from the origin of this reordered space. For the general case, the regions belonging to different classes could be disconnected in spite of the reordering.

### 3.2.4. Proof of correctness

*This section contains a 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 to keep the proof brief. However, it is easy to see how these can be handled. Our implementation of the algorithm can handle ties.

**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. In other words,

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

implies

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

That is, (3) $\Rightarrow$ (4). The reverse implication does not hold, i.e., (4) $\not\Rightarrow$ (3).

PROOF. Let $f(v,j)$ denote $\Pr(c_j)\prod_{d=1}^{n}\Pr(v_d|c_j)$. Clearly, if $\min_v(f(v,k) > \max_{j\neq k}\max_v(f(v,j))$ then every $f(v,k) > f(v',j)$ in the other classes irrespective of whether $v$ and $v'$ are the same or not. 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) $\Rightarrow$ (4).

We next give an example to show that (4) $\not\Rightarrow$ (3). Let $K = 2, d = 1, n_0 = 2, \Pr(c_1) = \Pr(c_2) = 0.5$, the probability values for the first member $m_{00}$ be (0.2, 0.1) respectively for the two classes and for the second member $m_{10}$ be (0.4, 0.3). Here class 1 dominates class 2 for all values but condition (3) does not. ■

**Lemma 3.2** If a region satisfies the MUST-LOSE condition maxProb$(c_k) < \max_{j\neq k}$ minProb$(c_j)$ then for every possible cell $v$ in the region the probability of class $c_k$ is less than the probability of at least some other class.

PROOF. The proof is similar to the MUST-WIN situation in lemma 3.1. ■

**Lemma 3.3** Replacing $\Pr(v_d|c_j)$ with $\Pr'(v_d|c_j) = \frac{\Pr(v_d|c_j)}{\max_{i\neq j}\Pr(v_d|c_i)}$ in inequalities 3 of lemma 3.1 maintains the correctness of the implication. That is, condition 4 is implied by

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

Similar results hold for the MUST-LOSE condition of lemma 3.2.

PROOF. By reducing all class probabilities of a member $v$ by the same factor, the relative ordering of the classes is not changed. Therefore, using the result from lemma 3.1, we can claim that (5) ⇒ (4). ■

An improved exact bound for the case when the number of classes is 2 is presented next.

**Lemma 3.4** When the number of classes is $K = 2$, the MUST-WIN and the MUST-LOSE bounds are exact with the above modified probabilities. That is, condition 4 is equivalent to condition 5. Similar results hold for the MUST-LOSE bound.

PROOF. First consider the MUST-WIN case. If $\forall v$ a class $c_1$ dominates the other class $c_2$, then

$$l\forall v\left(\frac{\Pr(c_1)\prod_{d=1}^{n}\Pr(v_d|c_1)}{\Pr(c_2)\prod_{d=1}^{n}\Pr(v_d|c_2)} > 1\right) \quad (6)$$

$$\Rightarrow \min_{v}\left(\frac{\Pr(c_1)\prod_{d=1}^{n}\Pr(v_d|c_1)}{\Pr(c_2)\prod_{d=1}^{n}\Pr(v_d|c_2)}\right) > 1 \quad (7)$$

$$\Rightarrow \Pr(c_1)\min_{v}\left(\prod_{d=1}^{n}\frac{\Pr(v_d|c_1)}{\Pr(v_d|c_2)}\right) > \Pr(c_2) \quad (8)$$

$$\Rightarrow \Pr(c_1)\left(\prod_{d=1}^{n}\min_{v_d}\frac{\Pr(v_d|c_1)}{\Pr(v_d|c_2)}\right) > \Pr(c_2) \quad (9)$$

$$\Rightarrow \Pr(c_1)\prod_{d=1}^{n}\min_{v_d}\Pr'(v_d|c_1) > \Pr(c_2) \quad (10)$$

$$\Rightarrow \text{minProb}(c_1) > \text{maxProb}(c_2) \ \text{ since } \Pr'(v_d|c_2) = 1 \quad (11)$$

Combining the above proof of (4) ⇒ (5) for $K = 2$ with lemma 3.3 we get that for $K = 2$, (5) ⇔ (4).

Similarly, we can prove for the MUST-LOSE condition by reversing the direction of the inequality and replacing $\max$ for $\min$. ■

When $K > 2$, (4) $\not\Rightarrow$ (5) as we see in this simple example where $K = 3, d = 1, n_0 = 2, p(c_1) = p(c_2) = p(c_3) = \frac{1}{3}$, the probabilities for the three classes of the first member $m_{00}$ are $(.5, .2, .7)$ respectively and for $m_{10}$ are $(.5, .7, .2)$. Class $c_1$ is the loser for both members but the MUST-LOSE condition is not satisfied with the revised $\Pr'$ values since $\text{minProb}'(c_1) = \frac{1}{3}\max(\frac{.5}{.7}, \frac{.5}{.7})$ is not smaller than $\max(\frac{1}{3}\min(\frac{.2}{.7}, \frac{.7}{.7}), \frac{1}{3}\min(\frac{.7}{.7}, \frac{.2}{.7})) = \frac{1}{3}\frac{.2}{.7}$

In view of these proofs, in our implementation we use the revised bounds in lemma 3.3 in checking for the MUST-WIN and MUST-LOSE conditions.

### 3.3. Clustering

Clustering [24, 23] has several variations arising out of differences in the objective function, algorithms used for finding the cluster, representation of the clusters and criteria used for assigning instances to clusters. Many of these variations are relevant only when finding the clusters whereas we are interested in the post-clustering phase where the output clusters in conjunction with a suitable membership criteria is used to assign new instances to existing clusters. We concentrate on disjoint 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 deferred to a future topic. Partitional clusters 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. One of the most common cases is when the 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 $i$th 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 are model-based [27, 13]. 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. Assuming a weighted Euclidean measure and 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 $K$ values and choose the group with the largest sum.

For several model-based clusters the situation is similar. Each group $k$ is associated with a mixing parameter called $\tau_k$ ($\sum_{k=1}^{K} \tau_k = 1$) in addition to the parameters $\theta_k$ 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 [33, 19, 2]. We defer further investigation of this problem to a future topic.

# 4. Optimizing Mining Queries

In this section we discuss how the upper envelopes found in the previous section can be used for optimizing queries. In the paper so far we have been giving examples of mining predicates of the form "Prediction_column = class_label". However, the idea of exploiting upper envelopes is not restricted to only predicates of this simple form. We first show (in Section 4.1) how a wider class of mining predicates may be optimized using the upper envelopes. 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 $\langle c_1, .., c_k\rangle$, where $c_1, ..c_k$ 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 following atomic mining predicates: (M.Prediction_column = $c_1$),..(M.Prediction_column = $c_k$)

**Join predicates between a predicted column and a data column** Consider predicates of the form M.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 which predicted age is of the same category as the actual age"[2]. Such queries can occur, for example, in cross-validation tasks. If no other predicates occur in the query, then it is not possible to derive an upper envelope for such a predicate as the content of the Data_column is not known during optimization of the query. However, if the query contains additional predicates on Data_columns that indirectly limits the possible domain values Prediction_column can assume, then we can apply the optimization of the IN predicates discussed in the previous paragraph. 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 $\langle$ 'old', 'middle-aged'$\rangle$ for which we can add the upper-enveloping predicates as discussed above. These upper envelopes might restrict a third indexed attribute, say salary, which can be used to optimize the data access cost. Thus, for this class of join predicates, we rely on a traditional optimizer's ability to add derived predicates via transitivity.

**Join predicates between two predicted columns** Another significant class of join predicates are of the form M1.Prediction_column1 = M2.Prediction_column2. Such predicates select predictions on which two models M1 and M2 concur in their predicted class labels. An example of such a query is "Find all customers who visited microsoft.com last week who are predicted to be web developers by both $SAS\_customer\_model$ and $SPSS\_customer\_model$" where $SAS\_customer\_model$ and $SPSS\_Customer\_model$ represent two different mining models. In order to optimize this query using upper envelopes, we assume that the class labels for the prediction columns can be enumerated during optimization by examining the metadata associated with the model. Let the class labels that are common to these two prediction columns 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 = $c_i$). Let $M1_{c_i}$ and $M2_{c_i}$ represent the upper envelopes corresponding to selection predicates $(M1.Prediction\_column1 = c_i)$ and $(M2.Prediction\_column1 = c_i)$ respectively. Then the upper envelope for the predicates $(M1.Prediction\_column1 = M2.Prediction\_column2 = c_i)$ is $M12_{c_i} = M1_{c_i} \wedge M2_{c_i}$. Thus, the upper envelope for the entire predicate is: $\bigvee_i M12_{c_i}$. 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

[2]We should consider age as a discretized attribute with the domain consisting of ''young", "middle-aged', "senior".

guaranteed to return no answers. These observations can be leveraged during the optimization process to improve efficiency.

### 4.2. Key Steps in Optimization of Mining Predicates

The following are key steps in optimizing queries involving mining predicates:

- Use traditional transitivity rules to derive selection predicates on mining columns
- Derive upper envelopes for each selection predicate involving mining columns
- For equi-join among mining columns (the last case discussed above), generate the upper envelope by enumerating the possible set of class labels.
- Generate a query expression by combining the given query with upper envelopes.
- Optimize and execute the generated query.

There are two implicit assumptions that underly the above strategy. First, adding upper envelopes do not increase the cost of execution. Since we assume that every upper envelope consists of AND/OR expression of simple predicates only, the cost of evaluation of such predicates can indeed be ignored, keeping with traditional database cost modeling convention. But, we do rely on a smart optimizer which is not misguided by the presence of a complex boolean predicate, i.e., an optimizer whose selectivity computations and access path selections are robust for complex boolean expressions. Next, our evaluation strategy is non-traditional in the sense that the preprocessing strategy needs to read the content of the mining model to generate the upper envelopes. In contrast, traditional optimization is driven only by the metadata and not by the content of the data. Our approach is justified since mining models evolve slowly, much like schema. But, it is important that we need to invalidate an execution plan (if cached or persisted) in case the underlying mining model changes since the computation of the upper envelopes depend on the structure of the mining model.

## 5. Experiments

In this section, we present the results of experiments to evaluate the effectiveness of upper-envelope predicates generated by algorithms presented in Section3. Our experiments focus on three important aspects: (i) Impact of upper-envelope predicates on the physical plan and running time of queries. We study this in Section 5.2.1 and Section 5.2.2 respectively. (ii) Degree of tightness of the approximation (Section 5.2.3). (iii) Time taken to generate upper-envelope predicates (Section 5.2.4).

### 5.1. Experimental Setup

**Mining Models:** We have implemented the algorithms presented in Section 3 for the following mining models:

| Data Set | Test size in millions | Training size | # of classes | # of clusters |
|---|---|---|---|---|
| Anneal-U | 1.83 | 598 | 6 | 6 |
| Australian | 1.41 | 460 | 2 | 5 |
| Balance-Scale | 1.28 | 416 | 3 | 5 |
| Breast | 1.40 | 457 | 2 | 5 |
| Chess | 1.63 | 2130 | 2 | 5 |
| CRX | 1.36 | 472 | 2 | 5 |
| Diabetes | 1.57 | 512 | 2 | 5 |
| German | 1.02 | 666 | 2 | 5 |
| Hypothyroid | 1.78 | 1339 | 2 | 5 |
| Led7 | 1.64 | 200 | 10 | 10 |
| Letter | 1.28 | 15000 | 26 | 26 |
| Mushroom | 1.04 | 5416 | 2 | 5 |
| Pairty5+5 | 1.04 | 100 | 2 | 5 |
| Pima | 1.57 | 512 | 2 | 5 |
| Shuttle | 1.85 | 43500 | 7 | 7 |
| Soybean-large | 1.40 | 455 | 19 | 19 |
| Tic-tac-toe | 1.96 | 638 | 2 | 5 |
| Vehicle | 1.73 | 564 | 4 | 5 |
| Waveform40 | 1.28 | 300 | 3 | 5 |
| Kdd-cup-99 | 4.72 | 100000 | 23 | 23 |

Table 2. Summary of Data Sets used in experiments

Decision Tree, Clustering, Nave Bayes Classifier. We generated Decision Tree and Clustering mining models using Microsoft Analysis Server that ships with Microsoft SQL Server 2000. The Analysis server implements a model-based clustering algorithm and the output structure of the cluster model is similar to the Naive Bayes model. For generating Naive Bayes mining models we used the Discrete Naive bayes inducer packaged with the MLC++ machine learning library [25].

**Data Sets:** We report numbers on 20 data sets consisting of 19 UCI [6] data sets and the 1999 KDDcup data set available at [4]. Table 2 summarizes the following characteristics of each data set: (i) Size of training data set for all mining model, (ii) Size of test data set, (iii) Number of classes for Decision Tree and Naive Bayes mining models, and number of clusters for Clustering mining model. 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. Each test data set was stored in a Microsoft SQL Server database.

**Implementation:** We now briefly describe the implementation issues involved in generating upper-envelope predicates. We note that 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 predicates, and submit the rewritten query to the database engine. However, we briefly comment below on some implementation aspects when generation of upper-envelope predicates is integrated with the database engine. In our implementation, the database is stored in Microsoft SQL Server 2000 and the mining models are stored in Microsoft Analysis Server.

We can currently generate upper-envelope predicates for the class of queries that include a mining predicate of the form "Mining-Column = Class". For a given mining model, we query Analysis Server using the MINING_MODEL_CONTENT schema rowset defined in the OLE DB for Data Mining [15] interface, and extract from the mining model object, information required for generation of upper-bound predicates for each class of the mining model. For example, for a Decision-Tree mining model, we extract information stored in each leaf node of the tree about the class and its associated predicate (i.e., the path from the root node to that leaf node). Using this information, we can generate the upper-envelope predicate for any class of the mining model. We note that the above step needs to be carried out only once per mining model, and repeated only if the mining model is updated.

When executing a mining query, we first identify the mining model object(s) referenced in the mining query. Next, we detect if there are any predicates in the mining query for which we can include an upper-envelope predicate. If so, we rewrite the query to include the corresponding upper-envelope predicate and submit the rewritten query to the database engine for execution.

An alternative implementation is to integrate generation of upper-envelope predicates into the database engine. In this case too, the upper-envelope predicates for each class of a mining model can be precomputed (possibly during creation of the mining model or as described above) and stored in system catalogs as part of an expanded database schema. During optimization of the mining query, the query optimizer can look up the catalogs to perform the appropriate query modification step of including upper-envelope predicates.

**Evaluation Methodology:** To evaluate the effectiveness of the upper-envelope predicates, we adopt the following methodology for each (data set, mining model) combination. For each class/cluster, we first generated 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/clusters for that (data set, mining model) combination. To generate an appropriate physical design for this workload, we invoke the Index Tuning Wizard tool [11, 3] that ships with Microsoft SQL Server 2000 by passing it the above workload file as input. We then implement the index recommendations produced by the Index Tuning Wizard on the test database. Once the recommendations are implemented, we execute the work-

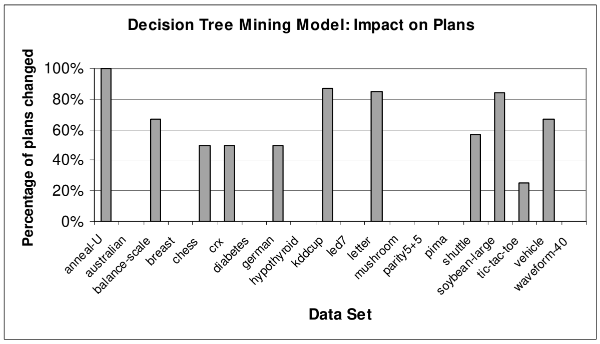



Figure 3. Impact of upper-envelope predicates on physical plan for Decision tree model

load on the database and record the plan and running time of each query in the workload. We compare this plan and running time with a query that performs a full scan of the table, i.e., "SELECT * FROM T". We record running times for both "cold" runs as well as when the cache was warmed. Since the results for cold runs and warm cache runs are similar, we report numbers for cold runs only. All numbers reported are averages over three runs. We record a plan for the query with upper-envelope predicates as having *changed* compared to the full scan query if either of the following is true: (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).

## 5.2. Results

### 5.2.1. Impact of Upper-Envelope Predicates on Plan

In this experiment, 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 in the workload whether the plan chosen by the query optimizer changed compared to the query without upper-envelope predicates. The table below shows the percentage of queries for which the plan changed over all data sets and mining models.

| Decision Tree | Clustering | Naive Bayes |
|---|---|---|
| 62.6% | 67.7% | 67.4% |

As we can see from this table, for all 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 examining the results for each data set. Figures 3, 4, and 5 show these numbers for the Decision Tree, Clustering and Naive Bayes mining models respectively. From this experiment we make a couple of observations. First, we see that upper-envelope predicates have greater impact on the plan for data sets where the number of classes/clusters is relatively large (e.g.,

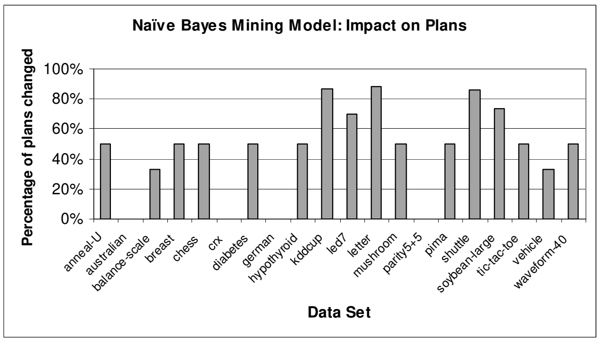



Figure 4. Impact of upper-envelope predicates on physical plan for Naive Bayes model

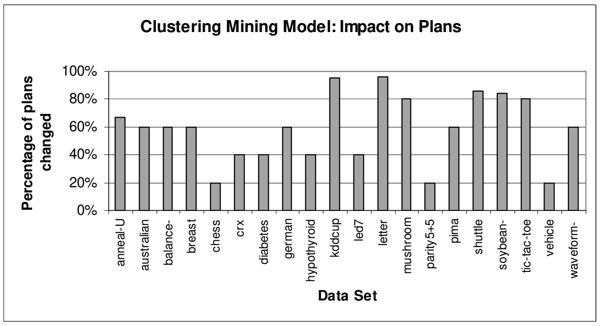



Figure 5. Impact of upper-envelope predicates on physical plan for Clustering model

kddcup, letter, shuttle, soybean-large etc.), and less impact for data sets where number of classes/clusters is small (e.g., australian, breast, diabetes, mushroom, parity etc.). This is due to the fact that when the number of classes/clusters 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 indexes/data to answer the query. Second, we noticed that the upper-envelope predicates generated by our algorithms 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.

### 5.2.2. Impact of Upper-Envelope Predicates on Running Time

In this experiment we evaluate the impact of upper-envelope predicates on the running time of queries *for which the plan changed*. We first report the average reduction in running time over queries whose plan changed for each mining model, compared to a full scan of the data.

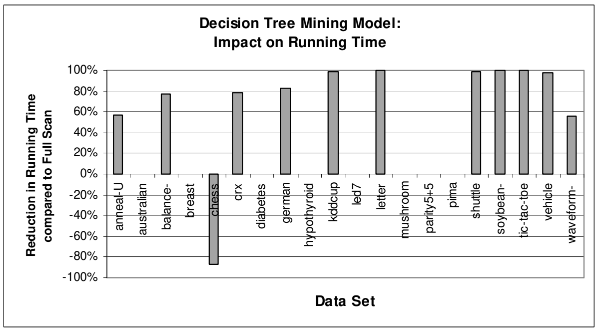



Figure 6. Impact of upper-envelope predicates on Running Time for Decision Tree mining model

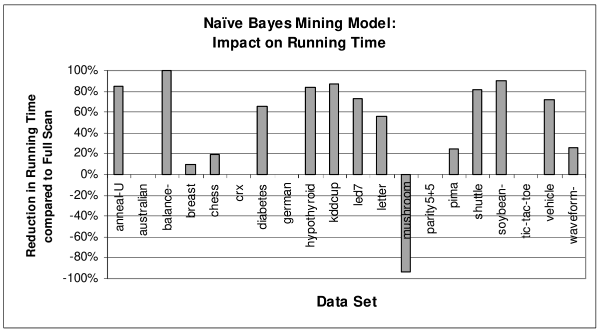



Figure 7. Impact of upper-envelope predicates on Running Time for Naive Bayes mining model

| Decision Tree | Clustering | Naive Bayes |
|---|---|---|
| 72.2% | 69.3% | 56.3% |

The reduction in running time we report here is a pessimistic bound on the actual reduction because our comparison is with a “select *” query without 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 change.

We now examine the percentage reduction in running time of queries for each data set and mining model. Figures 6, 7, and 8 show these numbers for the Decision Tree, Clustering and Naive Bayes mining models respectively. We observe from these charts that for most data sets there is a significant reduction in running time for all three mining models. This confirms our intuition that introduction of upper-envelope predicates not only affects the plan of the query, but in most cases has the desirable impact of reducing running time. We also notice that for a few data sets, the running time *increases* significantly. This is due to the fact that in those cases, the increased CPU cost of evaluating the upper-envelope

predicates outweighs the benefit of reduced I/O obtained by indexed access. However, we also note that in all cases where the running time increases, we could detect this occurrence by comparing the optimizer- estimated cost

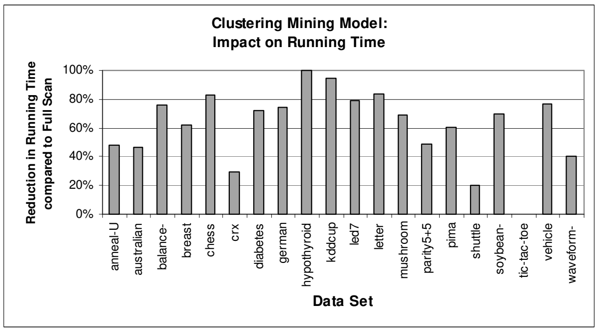
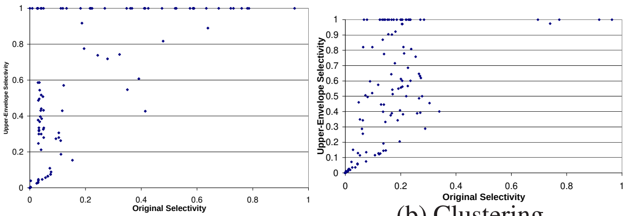



Figure 8. Impact of upper-envelope predicates on Running Time for Clustering mining model




(a) Naive Bayes (b) Clustering

Figure 9. Tightness of approximation

of the query with the upper-envelope predicate with the full scan query. Thus, if our algorithms for generating upper-envelope predicates are integrated with the query optimizer, we would not incur an increase in running time due to upper-envelope predicates, since the optimizer would choose the plan with the lower cost.

### 5.2.3. Tightness of Approximation

In our next experiment, we compare the tightness of approximation of the upper-envelope predicates for the Clustering and Naive Bayes mining models. We compare the selectivity of the original class/cluster with the selectivity of the corresponding query containing the upper-envelope predicate. For Decision-Tree mining models, since the upper-envelope predicates are exact, this comparison is not necessary. Figures 9(a),(b) show for the Naive Bayes and Clustering mining models respectively, a scatter plot of the original selectivity of each class vs. the selectivity of the corresponding upper-envelope predicate for all classes in all data sets. Each point in the scatter-plot corresponds to one class of a data set.

As we see from the figures, a significant fraction of the upper-envelope predicates either have selectivities close to the original selectivity or have selectivity small enough that use of one or more indexes (particularly clustered indexes) for answering the predicate is attractive. Another observation is that most cases where the algorithm is not able to find a suitable predicate (selectivity = 1) corresponds to where the original selectivity is greater than 10%. We found

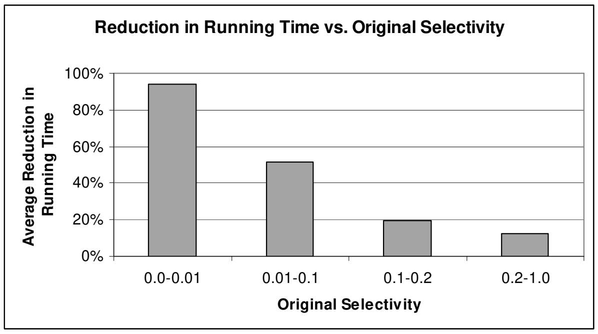



Figure 10. Running Time improvement vs. Original Selectivity: All mining models and data sets

that out of the queries whose original selectivity is less than 10%, our algorithms were able to find a suitable predicate that resulted in reduced running time in 91% of the cases. On the other hand, for queries whose original selectivity exceeded 10%, our algorithms were successful in finding predicates that reduced the running time in only about 30% of the cases. Finally, Figure 10 shows the average reduction in running time as a function of the original selectivity of the class over all classes of all mining models and data sets. We see that the reduction in running time due to inclusion of upper-envelope predicates is most significant when the selectivity is below 10%. We note that the low reduction in running time for higher selectivities is not a reflection of the effectiveness of our algorithm. This is because when a predicate's selectivity is high (e.g., above 10%) the optimizer would rarely select an index for reducing access cost, particularly when the index is non-clustered. Thus, for high selectivity classes, adding upper-envelope predicates is rarely useful, even if we could find *exact* predicates.

### 5.2.4. Time to Generate Upper-Envelope Predicate

The derived predicates are found one-time during the training phase of each model. Therefore, the execution speed for this phase is not critical as long as it is within the same range as the training time of the model. A second observation about these algorithms is that the running time is independent of the size of the training data — all algorithms operate only on the parameters of the models after the model training phase is completed. In Table 3 we show the running time in seconds of our top-down algorithm for the twenty data sets alongside their number of dimensions ($n$), number of classes ($K$), number of possible combinations of discrete values and number of classes. For comparison, in the last column we present the time to enumerate the class of each possible combination — an operation that would be needed in a bottom-up algorithm. We observe that even for a medium sized data-space like in the case of the "german" data set with 1.9008e+08 combinations enumerating the in-

| Data Set | $n$ | $K$ | Number of Combinations | Top-down Time (sec) | Bottom-up Time (sec) |
|---|---|---|---|---|---|
| anneal-U | 38 | 6 | 4.75525e+16 | 3.961 | |
| australian | 14 | 2 | 870912 | 0.256 | 8.64 |
| balance-scale | 4 | 3 | 16 | 0.014 | 0.013 |
| breast | 10 | 2 | 17496 | 0.642 | 1.574 |
| chess | 36 | 2 | 1.03079e+11 | 0.505 | |
| crx | 15 | 2 | 4.64486e+06 | 0.282 | 53.963 |
| diabetes | 8 | 2 | 96 | 0.023 | 0.022 |
| german | 20 | 2 | 1.9008e+08 | 0.365 | 2481.602 |
| hypothyroid | 25 | 2 | 2.51658e+07 | 0.331 | 587.130 |
| led7 | 7 | 10 | 128 | 0.103 | 0.123 |
| letter | 16 | 26 | 1.52354e+14 | 63.30 | |
| mushroom | 22 | 2 | 1.40429e+15 | 0.535 | |
| parity5+5 | 10 | 2 | 1024 | 0.042 | 0.5 |
| pima | 8 | 2 | 32 | 0.019 | 0.025 |
| shuttle | 9 | 7 | 3.21593e+10 | 3.449 | |
| soybean-large | 35 | 19 | 1.24825e+15 | 11.50 | |
| tic-tac-toe | 9 | 2 | 19683 | 0.175 | 0.34 |
| vehicle | 18 | 4 | 1.54829e+09 | 1.223 | 47700 |
| waveform-40 | 40 | 3 | 3.02331e+07 | 0.992 | 1227 |
| kdd-cup-99 | 41 | 23 | 2.38883e+34 | 0.048 | |

Table 3. Time to find the upper bound predicates for Naive-bayes predicates

stances takes more than 45 minutes whereas the top-down algorithm terminates in less than a second. The "shuttle" data set and the all the others with number of combinations greater than 1.0e+10 did not terminate after more than 30 hours!

These experiments show that our top-down algorithm for extracting upper envelopes for Naive Bayes and clustering models is both (1) tight in cases where the original selectivity is small enough that using an index make sense in the first place and (2) fast even for high dimensional datasets like the KDD-cup dataset with 41 dimensions.

## 6. Related Work

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 [35]. 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*. Our paper represents the first work in that direction where mining predicates are exploited to derive additional predicates which can significantly impact choice of access paths by the optimizer and hence could reduce cost of execution.

Our work can be viewed as part of the broader field of semantic query optimization which has a long history of work in relational databases as well as in logic programming and artificial intelligence. 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 semantics we exploit, which is that of 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 [10, 8] but these techniques of derivation do not apply for deriving upper envelopes for mining predicates.

It is important to distinguish our approach to optimization with some of the recent work in the database literature on optimization of user-defined functions and predicates [20, 12, 36] where the evaluation of user-defined predicates have an associated cost of evaluation and selectivity associated with it. These techniques attempt to reduce the number of invocations of the user-defined predicates while leveraging its selectivity. Such techniques may be applicable to reduce the cost of evaluation of mining predicates although in some situations mining predicates are cheap to evaluate (e.g., decision tree classifier). We have instead looked at an orthogonal aspect where the structure of the mining model helps derive predicates that limit the amount of data that needs to be scanned. Note that our technique of deriving upper envelopes could result in a boolean filter expression with AND/OR. We rely on the optimizer's ability to efficiently exploit such predicates. The paper by Mohan et al. [30] discusses how an optimizer can in fact optimize such expressions.

Our work falls in the broader area of integration of data mining and database systems and there are many pieces of complementary work in that arena. Sarawagi et al. [34] discuss how to construct mining models using SQL and its extensions. The work by Meo et al. [28] is an example of specialized extensions to SQL to provide support for association rules. Another body of work is related to the emerging commercial standards and APIs for integrating mining models in different applications and with existing data sources. With the growing importance of XML as the emerging standard for data exchange, Predictive Model Markup Language (PMML) has been proposed to enable portability and model sharing by expressing model description in XML[31]. More recently, CRISP-DM [7] (`http://www.crisp-dm.org/`) has been proposed as a standard for Data Mining process model. We have already discussed Microsoft's OLE-DB for DM [15] in Sections 1 and 2.

Our algorithm for extracting predicates from mining models is related to the problem of rule extraction from hard to interpret models like Neural networks [26, 16]. However the goals of the two problems differ significantly. In the former case predicates are required to be strict upper envelopes of all points belonging to a class whereas in the later case the rules for a class are approximations of the original classification function. In addition, the algorithm for rule learning as proposed in [26] 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.

Another related problem is the coverage problem that has been addressed in several different contexts, including, covering a set of points with the smallest number of rectangles [33, 19, 2], covering a collection of clauses with simpler terms in logic minimization problems [21] and constructing clusters with rectilinear boundaries [5]. All of these assume that the points are already enumerated in the $n$-dimensional space. This is not a feasible option in our case as we showed in the results of Table 3. Furthermore, the first two problems require an exact cover of smallest size whereas we want an upper-envelope. Finally, most of these approaches assume a small number of dimensions (two or three) and do not scale to higher dimensions.

## 7. Concluding Remarks

Querying and mining represent two aspects of data analysis and support for extending database queries with mining predicates have begun to appear. In this paper, we have proposed a novel approach to optimizing queries with mining predicates by exploiting the selectivities of such predicates. This has been accomplished by deriving a predicate (upper envelope) that exploits the structure of the mining models. Such a predicate is a traditional database filter condition that SQL systems can exploit for access path selection, and hence for generating an efficient execution plan. Through extensive experiments on Microsoft SQL Server 2000, we demonstrate the significant impact upper envelopes can have on performance of queries with mining predicates.

This is the first work we know of that attempts to optimize queries with mining predicates. While in this paper we have shown how to extract upper-enveloping predicates for some of the popular discrete mining models, there is abundant opportunity for future work including extending the technique to other mining models and prediction problems.

## 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] 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.

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

[5] 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.

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

[7] P. Chapman, R. Kerber, J. Clinton, T. Khabaza, T. Reinartz, and R. Wirth. The CRISP-DM process model. Discussion Paper - Technical Report. `http://www.crisp-dm.org/`, 1999.

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

[10] 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.

[11] 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.

[12] 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.

[13] P. Cheeseman and J. Stutz. Bayesian classification (autoclass): Theory and results. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 6, pages 153–180. AAAI/MIT Press, 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] D. Franzblau. Performance guarantees on a sweep-line heuristic for covering rectilinear polygons with rectangles. *SIAM J. Disc. Math*, 2:307–321, 3 (1989).

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

[21] 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.

[22] IBM, `http://www.ibmlink.ibm.com/usalets&parms=H_201-069`. *IBM Intelligent Miner Scoring, Administration and Programming for DB2 Version 7.1*, March 2001.

[23] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

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

[25] 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.

[26] 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.

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

[28] 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.

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

[30] 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.

[31] D. Organization. PMML 1.0 – Predictive Model Markup Languange, a public specification. `http://www.dmg.org/public/techreports/pmml-1_0.html`, 1999.

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

[33] 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.

[34] 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.

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

[36] 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.