Towards Data Mining Operators in Database Systems: Algebra and Implementation

Ingolf Geist[1] und Kai-Uwe Sattler[2]

[1] Department of Computer Science, University of Magdeburg
P.O.Box 4120, D-39016 Magdeburg, Germany
geist@iti.cs.uni-magdeburg.de
[2] Department of Computer Science, TU Dresden
D-01602 Dresden, Germany
k.sattler@computer.org

Abstract. The KDD process is a non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. This process comprises several steps which are invoked and parametrized in an interactive and iterative manner. A uniform framework for different kinds of patterns and operators is needed to support KDD efficiently and in an integrated way. Furthermore, because of large data sets it is necessary to scale up mining algorithms in order to achieve fast user support. One task of scaling data mining algorithms is the integration of KDD operators in database management systems.

Two aspects of supporting KDD are addressed in this paper. First, a uniform framework is proposed that is based on constraint database concepts as well as interestingness values of patterns. Different operators are defined uniformly in this model. Second, DBMS-coupled implementations of selected operators for decision tree mining are discussed.

1 Introduction

Nowadays, it is possible to store a huge amount of data in data warehouses or similar data storages. However, these data are only useful if the analysts can discover new information in form of patterns or rules. This problem is addressed by techniques from the area of *Knowledge Discovery in Databases*. The KDD process is the *non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data* [9]. This process comprises several steps which are invoked and parametrized in an interactive and iterative manner. The steps (Fig. 1) can be divided into pre-processing of data, data mining and post-processing of the data. A uniform framework is needed to support an integrated and therefore efficient view to the process for the user. This framework has to provide a unifying description of models and patterns. Further, general operators have to be defined in the framework. The operators in a KDD process can be distinguished into three groups: data operators, which implement data preparation and transformation steps, operators creating mining models or patterns as well as getting the extension of patterns and operators manipulating and merging mining models and patterns. Thus, it is possible to define a uniform model which consists of two types of objects: data and KDD-objects as defined in [13, 15].

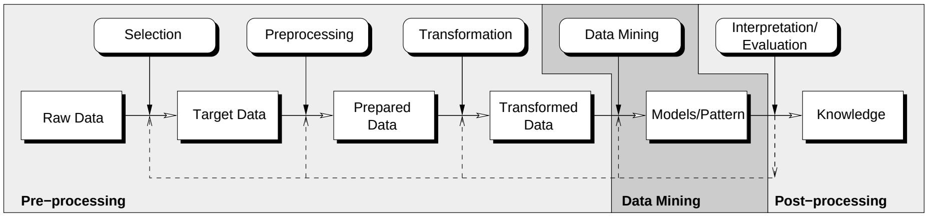


Fig. 1. KDD process

Supporting these operators for larger data sets requires a tight coupling between KDD systems and database systems. This is motivated not only by the fact that todays mostly main-memory-based data mining techniques suffer from the drawback of limited scalability. If the data set does not fit into the main memory the performance decreases significantly. In contrast, tightly coupled and SQL-aware KDD techniques could utilize advanced features available in modern DBMS, such as management of Gigabyte-sized data sets, parallelization, filtering and aggregation and in this way improve the scalability. Another motivation for building such systems is the ability of ad-hoc mining, i.e., allowing to mine arbitrary query results instead of only base data. Finally, treating mining operators as query operators allows the uniform optimization of mining queries, e.g. by exploiting basic heuristics like pushing down selections.

The contribution of this paper is the definition of operators of the groups data mining and post-processing. In addition, we discuss possible, tightly-coupled DBMS-centric implementations of operators in this framework.

The remainder of this paper is organized as following. A KDD algebra based on concepts of constraint algebra and interestingness is proposed in Section 2. Different implementations of the proposed operators are discussed in Section 3. The following Section 4 discusses existing works on KDD frameworks and integrating data mining techniques into database systems. Finally, Section 5 concludes the paper and gives an outlook to the future work.

2 A KDD Algebra

A uniform view on different operators in KDD is an important point in supporting KDD processes in a database management system. Such a KDD framework requires a comprehensive data model and a sufficient set of operators supporting different kinds of pattern and rules as well as operations.

The KDD process deals with two kinds of objects: data and “KDD objects” [13]. KDD objects can be described as *models* and *patterns*. On the one hand a model describes the whole input training data, for instance a decision tree or a set of clusters. On the other hand a pattern describes either a local characteristic in a data set (e.g. a cluster) or is a local sub-model (e.g. a node in a decision tree). Both ideas are not strictly divided and can interchange [12, p. 165ff.].

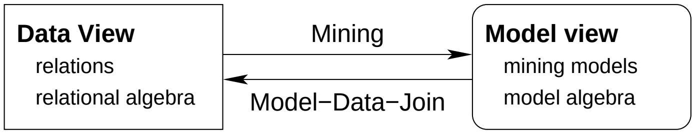


Fig. 2. Data and Model View

An algebra for a KDD process have to support the *data view* (the data objects) and the *model view* (patterns and models). Thus, operators in each of the views have to be provided as well as operators for linking both views together. Figure 2 gives an overview about this idea.
The remaining part of this section gives an introduction to the proposed data model and its algebra as well as an example for the usage of the framework. A more detailed discussion of the framework ideas is given in [10].

2.1 Pattern and Model Representation

The ideas for a pattern and model representation is based on three concepts: data subset description, interestingness function and labels. In [15] it was shown that patterns always describe a certain subset in the training data, for instance the nodes in a decision tree subdivide the training set into axis-parallel hyper-rectangles. Thus, concepts of constraint databases [16, 18] can be used to describe these subsets or patterns. The subsets are specified by extended generalized tuples [4] in the proposed data model. An extended generalized tuple is in our case a formula consisting of *linear inequality constraints*[1] that are connected by $\{\wedge, \vee\}$. The interpretation of a set of extended generalized tuples – a generalized relation – is a nested relation. That means, for each tuple a set of data points is described, and the generalized relation is a set of data sets. Different set operations, e.g. set selection and set difference, can be defined based on these assumptions [4].

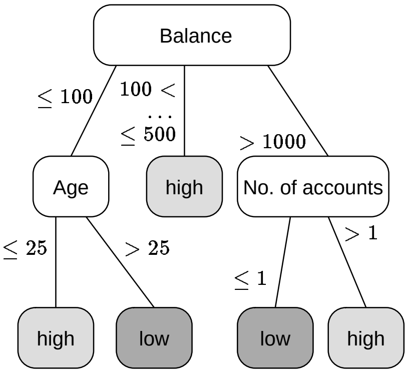
The second concept of pattern specification is the *interestingness function*. An interestingness function is user defined and interprets the meaning of a pattern according to the training data. The goal of the function is the description of the importance of a pattern to a user. An interestingness function I maps a pattern according to the training data into the interval $[0, 1]$, where 0 stands for low interest and 1 for high interest[2]. Formally, $I : (c, r) \rightarrow [0, 1]$, where c is a extended constraint tuple over $\mathbf{R}$ and r a database instance of $\mathbf{R}$.

Label attributes form the third and last concept of pattern description. These attributes model further information about the pattern, for instance, a hierarchy between the patterns, class labels or cluster id's. Label attributes are simple relational attributes.

The *schema* of a model can be defined as $M = \{C_R, I, L_1, \dots, L_l\}$ with C_R is a constraint relation over a set of relational attributes $\mathbf{R}$, I is an interestingness function

[1] For a set of attributes $\{A_1, \dots, A_n\}$ a linear inequality constraint is defined as $a_1 A_1 + \dots + a_n A_n \leq k$, with k is constant.

[2] Many known interestingness values can be normalized into this interval.



C_{CUST}	$Entropy$	ID	P_ID	Class
$true$	.940	0	null	null
$B \leq 100$	.970	1	0	null
$B > 100 \wedge B \leq 500$	0	2	0	high
$B > 1000$	.970	3	0	null
$B < 100 \wedge A \leq 25$	0	4	1	high
$B < 100 \wedge A > 25$	0	5	1	low
$B > 1000 \wedge NA \leq 1$	0	6	2	high
$B > 1000 \wedge NA > 1$	0	7	2	low

Fig. 3. Decision Tree as tree and model

and $L_1, \ldots, L_l$ are relational label attributes. Assuming $\mathcal{D} = dom(C_R) \cup [0,1] \cup dom(L_1) \cup \ldots \cup dom(L_l)$, a *pattern* is a partial function defined $p : M \to \mathcal{D}$. Thereby is $p(C_R) \in dom(C_R)$, $p(I) \in [0,1]$ and $p(L_i) \in dom(L_i)$, for $i = 1, \ldots, l$. A *model* is a set of patterns that belong together and is specified by a unique name.

An example shall describe a decision tree modeled with the help of these definitions. Figure 3 illustrates on the left side a decision tree that classifies bank customer according there migration risk. Assuming the training data set is specified by the schema $CUST$, the right side of Figure 3 illustrates the resulting patterns. The interestingness function is the Entropy. The label attribute $Class$ specifies the assigned class $\{high, low\}$. The hierarchy of the tree is described with help of the attributes ID and P_ID.

2.2 Model View Operators and View Transition Operators

After the specification of the model view objects, this section describes model view operators as well as the operators from data view into model view and from model view to data view, respectively.

The set of operators is closed within the model view, that means the result of an operator is again a model. Thus, different operators can be easily combined. A set of the operators has to be sufficient, especially it has to support many needs of the analyst in the data mining and post-processing steps. The operators for pre-processing are described elsewhere, e.g. [26]. Following kinds of operators are supported by the proposed framework:

- extracting patterns of special interest,
- projecting patterns (shrinking the feature set),
- comparing of models (according patterns and interestingness),
- merging of models (combining two models) and
- renaming of attributes.

Furthermore, two view transition operators are defined – the mining operator and the model-data-join operator. The operators are based on concepts of constraint databases (see [4, 16]), relational algebra operators and distributed data mining [17, 22].

Extracting interesting patterns The *selection* operator is used to extract a set of patterns that are useful to the user. The operator supports constraint selection, interestingness value selection and label selection. The constraint selection uses the set selection operator [4], which extracts all constraint tuple that satisfy spatial conditions, like contain, intersect, disjunct or overlap. Assuming a model with a schema $M = \{C_R, I, \mathbf{L}\}$ with $\mathbf{L}$ is a set of label attributes $L_1, \ldots, L_l$ and a region $P = 10 \leq A_1 \leq 65 \wedge 25 \leq A_2 \leq 70$ with $A_i \in \mathbf{R},\ i = 1, 2$, then the following operation extracts all patterns, whose data sets intersect with the query region: $sigma_{(p \cap R)}(m) = \{p :\ p \in m, ext(p(C_R)) \cap ext(P) \neq \emptyset\}^3$, with $ext(p(C_R))$ describes the extension of a constraint tuple $p(C_R)$.

The interestingness and label selection are conventional relational selections. The interestingness values selection can be taken into account during the mining operation (for instance, restriction of the candidate set during the frequent item set extraction).

Projection Besides the restriction of the number of patterns, it is also possible to restrict the supported feature or input attribute set in a model. The *projection* operator provides this functionality. The operator is used to project the input attribute set or the label attribute set.

The former projection – input attribute projection – allows only these patterns in the result model, whose input attribute set is a subset of the projected attribute set. For instance, all frequent item sets with attributes not specified in the projection are removed from the model. Possibly, the remaining patterns have to be re-computed and new interestingness and label values are assigned. The projection can be included into the mining algorithm by projecting the input data without duplicate removal. Assuming a relation r with attributes $\mathbf{R}$ and an attribute set $X \subset R$, the projection of a model m over $\mathbf{R}$ is defined as: $\pi_X(m) = \{p : p \in m, \alpha(p(C_R)) \subseteq X, p(I) = I(p(C_R), \pi_X(r)\}^4$. The schema of the result model is $M = \{C_X, I, \mathbf{L}\}$.

The label projection is the straightforward relational projection on the set of the label attributes. The result is a model with all patterns with the projected labels.

Comparing Models—Intersection and Difference Another task for an analyst is the comparison of models. For instance, he/she wants to know the difference between the patterns (frequent item sets) of this week and the week one year ago. In this case the operations *intersection* and *difference* are necessary. The difference is defined using the set difference of extended generalized relations. Assume two models m_1 and m_2 and the schemas $M_1 = M_2$. The difference is defined as: $m_1 - m_2 = \{p : p \in m_1, ext(p(C_R)) \neq ext(p'(C_R)), \forall p' \in m_2\}$. Furthermore, we can also restrict the other model values. The intersection can be defined as $m_1 \cap m_2 = m_1 - (m_1 - m_2)$.

Another operator is the *renaming* operator which is defined in a straightforward way by using constraint and relation renaming operators.

[3] The second condition is $\subseteq$. See [4] for more details.

[4] $\alpha(p(C_R))$ returns the set of input attributes used in $p(C_R)$.

Merging Models – Join and Union The operators for merging models are used to create a new model from the information of two input models. The new model represents the data sets represented by the input models. Implementations for these operators are algorithms from distributed data mining or meta learning. The idea of these operators is based on the possibility of further adding scalability to mining algorithm by partitioning the input data [23]. Possible algorithms are for instance [17, 22]. These algorithms recompute and restructure the input models.

There are two cases for combining models: two model with equal or different input attribute sets, respectively. The first case—equal input attribute sets—is supported by the *union* operator. It uses two models m_1 and m_2 with schema $M_1 = \{C_R, I, \mathbf{L_1}\}$ and $M_2 = \{C_R, I, \mathbf{L_2}\}$ with $L_1 \cap L_2 \neq \emptyset$. The resulting model has the schema $\{C_R, I, \mathbf{L_1} \cup \mathbf{L_2}\}$ and the patterns of it are calculated by an appropriate algorithm.

The *join* operator is used to merge models with different input attribute sets, e.g. $R_1 \cap R_2 \neq \emptyset$, but the interestingness function and the label attributes have to be equal. The resulting model is defined over the union of the input attributes.

Model–Data–Transition Operators It is necessary to define operators that build a link between the two views to complete the specification of an entire KDD process. In the proposed system the *mining* operator (μ) performs the transition from data view to model view. The operator *model-data-join* ($\bowtie_{md}$) is the vice versa link and computes the extension of a pattern according to a data set.

The *Mining* operator invokes a data mining algorithm to extract from a relation a set of pattern. The operator creates a model. The kind of patterns as well as the interestingness function and label attributes are defined by parameters of the mining operator. Other parameters as for instance thresholds or number of clusters will be specified by additional selection operators. The input of the *mining* operator is a relation r over an attribute set $\mathbf{R}$. The result of $\mu_{dt,Entropy,class}(r)$ is a decision tree m with the schema $M = \{C_R, Entropy, class\}$. Model m represents a decision tree extracted from r.

Model-Data-Join computes the extension of patterns according to an input data set. Thus, the operator takes a model m and a r as inputs. The operator is restricted by $\mathbf{R} \cap \alpha(m(C_R)) \neq \emptyset$[5], with $\mathbf{R}$ is the schema of r and $\alpha(m(C_R))$ the set of attributes, on which the tree is defined. The result of $m \bowtie_{md} r$ is a relation $r_1 = \{t : \ t' \in r, p \in m, t' \in ext(p(C_R)), t = t' \times p(\mathbf{L})\}$. The schema of r_1 is $R_1 = R \cup \mathbf{L}$. Note that a tuple t can be in different pattern extensions.

Example Assume the example from Figure 3. Now, the analyst wants to select all patterns with “high” risk label and extract all tuples from a relation $CUST$ which belong to these patterns. The corresponding algebra expression is:

$$\pi_{CUST \cup \{Class\}}((\sigma_{Class='high'}(m)) \bowtie_{md} r).$$

The result is a relation of tuples which belong to the high risk class and have their Class labels assigned.

[5] Note that $\alpha(m(C_R))$ returns is union of the supported input attribute set of all patterns in the model.

The second task is to perform a complete prediction join. For this task the analyst has to select all patterns with an assigned label (the value of the $Class$ is not null) and compute the model-data-join with a new data set.

3 Implementation Issues

For a realization of the introduced framework two important issues have to be considered:

- Mining model have to be represented in a way that allows to apply operations on them, whereas for representing data we can rely on the native database structures, e.g. relations.
- Operations for manipulating models have to be implemented, either by providing dedicated operators or building them on top of existing database operations. In any case, these operators should be usable as part of queries in order to enable ad-hoc querying of models as well as ad-hoc mining.

In addition to these issues some kind of language support is necessary to exploit the functionality to the user. This could be realized for example by providing a dedicated data mining language as described in [11] or a set of user-defined types and functions as proposed in [27]. However, this is out of the focus of this paper. In principle, mapping these approaches to our algebra operations is rather straightforward.

Finally, because both the representation of models and the implementations of generic operators depend strongly on the actual class of the data mining technique in the following we focus on decision tree classification.

3.1 Model Representation

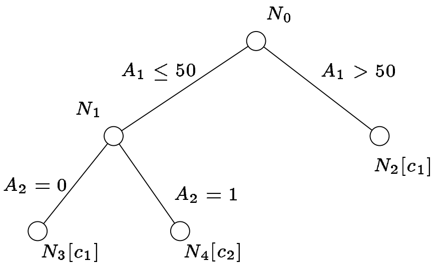
In principle, there are several possible approaches for representing a decision tree in a relational database.

A first approach is based on the Predictive Model Markup Language (PMML) [1]. Here, a decision tree is stored in an XML representation as a BLOB value in a table column. This simplifies the exchange of models but requires parsing the XML document before any operation on the tree.

An alternative solution is to store the tree in a flat table structure as illustrated in Fig. 3, where the constraint attributes are stored as strings. This approach allows to apply basic operations such as projection and selection but requires still an interpretation of the constraint attributes used for representing the splitting criteria.

Because of the restrictions of the these approaches with respect to applying operations on the tree, we have chosen a third solution for our experiments. Fig. 4 illustrates the main ideas of this structure.

Each tuple in the table represents an edge of the tree from node *parent* to node *node*. Each edge is associated with a condition, where the attribute name is stored in the field *attrib* and the domain for the split is represented by the values of the fields *minval* and *maxval*. The field *class* holds the label of the most frequent class in the partition associated with the node *node* of the edge together with the probability *prob* of the class occurrence.



(a) Tree representation

node	*parent*	*attrib*	*minval*	*maxval*	*class*	*prob*
N_0	$\perp$	$\perp$	$\perp$	$\perp$	c_1	0.70
N_1	N_0	A_1	MIN	50	c_1	0.86
N_2	N_0	A_1	50	MAX	c_2	0.66
N_3	N_1	A_2	-1	0	c_1	0.98
N_4	N_1	A_2	0	1	c_2	0.60

(b) Table representation

Fig. 4. Representation of a decision tree

3.2 Mining Operator

The purpose of the mining operator μ is to induce a (predictive) model based on the input data. For classification that means for example to derive a decision tree. Classification has been studied extensively over the years ans several algorithms for constructing decision trees were proposed, e.g. ID3, C4.5, SLIQ, SPRINT and PUBLIC. Most algorithms follow a greedy approach consisting of two phases. In the *tree-growing* phase the algorithm starts with the whole data set at the root node. The data set is partitioned according to a splitting criterion into subsets. This procedure is repeated recursively for each subset until each subset contains only members belonging to the same class or is sufficiently small. In the second phase – the *tree-pruning* phase – the full grown tree is cut back to prevent over-fitting and to improve the accuracy of the tree.

The mining operator for decision tree classifiers can be implemented by using one of the above mentioned algorithms and integrated with the DBMS by implementing it as a user-defined function. However, as most data mining techniques decision tree classification is very time-consuming particularly for larger data sets. Thus, in this form the mining operator is currently not very well-suited for ad-hoc mining. Database primitives as described e.g. in [25] are a first step towards improving performance and scalability for database-centered mining, but real online mining requires further improvements.

3.3 Model-Data-Join and Prediction Join

Predictive data mining techniques such as classification allow to use the induced model to predict the class of new data records, where the class values is missing. For this prediction join operation the model has to be interpreted by following a path in the tree and evaluating for each node the associated split criteria.

Assuming a model representation as shown in Fig. 4 this operation can be implemented by the following algorithm:

```
**procedure** PREDICTIONJOIN (source table $S$, model table $M$)
  **foreach** tuple $t_S = (a_1, \dots, a_m) \in S$
    execute query $q(t_S)$
    fetch tuple $t_M = (n, p, c, prob)$
    $node := n$
    $class := c$
    $finished$ :=**false**
    **do**
      **do**
        fetch tuple $t_M = (n, p, class, prob)$
        **if** tuple not found
          produce result tuple $(a_1, \dots, a_m, c)$
          $finished$ :=**true**
      **while** $p \neq node$
      $node := n$
      $class := c$
    **while** $\neg finished$
```

For each tuple $t_S = (a_1, \dots, a_m)$ of the source relation the nodes are selected, whose condition is fulfilled by the attribute values of the given tuple. This is performed by the following query $q(t_S)$:

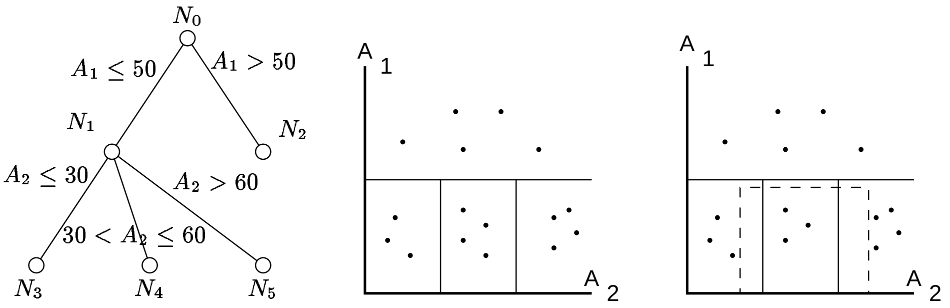
```
select *
from Model
where (aname='$A_1$' and minval < $A_1$ and maxval >= $A_1$) or
      (aname='$A_2$' and minval < $A_2$ and maxval >= $A_2$) or
      ...
      (aname='$A_m$' and minval < $A_m$ and maxval >= $A_m$)
```

These candidate nodes are ordered by their node-id. Next, the candidate nodes are processed in this order as follows: Starting with the root node the next node with a parent-id equal to current node-id is obtained until no further node can be found. In this case, the class and probability values are assigned to the active source tuple.

3.4 Selection

The selection operator is used by the analyst to extract interesing patterns from his/her point of view. The selection is divided into spatial selection—the selection of content—as well as the selection by interestingness and class label.

Spatial Selection The spatial conditions intersection and containment are considered, that select all patterns that intersect with or contain a certain query region, repectively. The first insight is that the patterns describe axis-parallel hyper-rectangles. Furthemore,



(a) Two-Dimensional Decision tree

(b) Data Space Visualization

(c) Data Space Visualization with query

Fig. 5. Decision tree and data space

a parent node contains all its children nodes and the subsets of the children of one parent are not overlapping. These properties of the subsets follow directly from the features of a decision tree. Figure 5 illustrates the data space partitions (Fig. 5(b)) of a decision tree (Fig. 5(a)). Here, the subset defined by node N_1 covers the subsets of N_2, N_3 and N_4.

Assume the query rectangle[6] $Q = 0 \leq A_1 \leq 40 \wedge 35 \leq A_2 \leq 60$. Now, the following query selects all patterns from a model that contain Q: $\sigma_{Q \subseteq p(C_R)}(m)$. The computation over the model representation described in section 3.1 consists of different steps. First, the following query filters all nodes that contain the query region in their splitting dimension:

```
select *
from model
where attrib = 'A1' and minval < 0 and maxval >= 40 or
      attrib = 'A2' and minval < 35 and maxval >= 60
```

The second step removes all patterns, whose parent is not in the result set. The algorithm is similar to the prediction join, but it returns a subset of the model. It starts with the first node besides the root node, which is always included in the result set. The algorithm is supported by the facts, that the root node represents the whole result set as well as the nodes are ordered by their ID. The example query is illustrated in Figure 5(c) and the result is set of patterns represented by the node N_1.

We assume now that the query is $\sigma_{Q \cap p(C_R)}(m)$, that means select all pattern that intersect q. First the following query has to be executed:

```
select *
from model
where attrib = 'A1' and ((minval >= 0 and minval < 40) or
```

[6] We can always compute the bounding box of query region and use the bounding box for querying.

```
                   (maxval >= 0 and maxval < 40)) or
attrib = ’A2’ and ((minval >= 35 and minval < 60) or
                   (maxval >= 35 and maxval < 60))
```

The second step is performed in the same manner as the containment selection. Additionally, a clipping algorithm [3, p. 288 ff.] is used, if the query region is not a rectangle.

If a dimension is not defined in the query we have to remove all nodes that are restricted in this dimension in the case of containment. This is already done in the proposed query. In the second case – intersection – these nodes have to be included, so we have to modify the `where`-clause.

Besides these implementation ideas, the usage of a different representation and a multi-dimensional index, such as a R-tree, is an alternative strategy.

Interestingness and Labels The selection of interestingness and label is performed in a straightforward way by applying a relational selection over the $class$ or $prob$ attribute. Obviously, the combination of both kinds of selection is possible by adding additional `where` conditions to the queries above.

3.5 Projection

The projection is a partial new computation of the decision tree. The goal of the projection is the removal of an attribute from the pattern descriptions. This operator requires a partial recomputation of the patterns in the case of decision trees.

The first step is a query which computes all nodes that contain not projected attributes and removes these nodes and their children. Subsequently, uthe remaining nodes are used to compute the subtrees from the database. This computation uses the primitives `FilterPartition` and `ComputeNodeStatistics` as described in [25] und utilizes the proposed index structure on the data.

A real value is the operator if the analyst uses it in connection with the mine operator. The analyst specifies the sequence of mining and model projection in the query $\pi_{A_{i_1},\ldots,A_{i_j}}(\mu_{dt,Entropy,class}(r))$. Now, the mining system can decide to first project the input data (without duplicate elimination) and then create the decision tree.

4 Related Work

In the recent years many algorithms for different data mining tasks were developed. Overviews of different techniques are for instance [19] and [8]. Association rules were introduced in [2].

Different frameworks were proposed to support the KDD process in a uniform manner. The 3W model and algebra in [15] is close to our model. It uses regions, dimensions and hierarchies to define a uniform framework and operators. The model consists of three worlds: The intensional world contains the description of data mining models, the extensional world holds the extensions of regions of the intensional world and the data world consists of the raw data. Furthermore, operators for moving in and out of the worlds and for the intensional world are defined. The interestingness value is not addressed, which is explicitly contained in our model.

There exist several frameworks for association rule mining, which provide operators for mining and post-processing association rules. Thereby, data mining is defined as querying a possible infinite set of association rules in a inductive database [13]. The query operators support rule constraints as well as the selections on support and confidence values [5]. These frameworks are limited to association rule mining.

Different data mining query languages were introduced to provide an integrated data mining environment [14, 11, 20, 27]. These languages try to create a descriptive interface to data mining algorithms and so the integration with database management systems.

Many work has been done in scaling up data mining algorithms. An overview is given in [23]. One task of adding scalability to DM algorithms is their integration with database management systems [6]. Different kinds of approaches were proposed: the implementation of the DM algorithms in SQL, e.g. the EM algorithm [21] or usage of user defined functions, e.g. for association rules [24]. User-defined aggregates for decision tree classification were discussed in [28]. Another task is the optimization of special operators, for instance the prediction join. First ideas in this area are given in [7].

5 Conclusions and Outlook

Nowadays, huge data sets require efficient and scalable KDD and data mining techniques. The explorative and iterative KDD process makes a strong user interaction necessary. Ideas towards a solution of these issues were presented in this paper. First, a uniform framework was proposed, that is based on constraint database concepts and interestingness values. The model consists of data objects and KDD objects. Several operators on KDD objects were proposed in the paper supporting an efficient and integrated view to the KDD process.

The second contribution was the discussion of the implementation of different operators for decision trees in a DBMS. The tight coupling of mining algorithms and DBMS is an important research issue in scaling up data mining algorithms. The implementation can be carried out with help of extension mechanisms of DBMS. Several primitives are used to implement data intensive parts of the algorithms.

Because of the differences between different kinds of patterns it is necessary to implement the operators in a polymorphic way. That means, additional meta data is necessary. This is one future research direction. Furthermore, we have to implement the operators also for different kinds of mining techniques for instance clustering and association rules. These implementations should be specified in a database centric way by identifying different primitives.

Another issue is the development of optimization rules for the global, uniform model as well as the mapping between the algebra operators and an appropriate data mining query language.

References

1. PMML 1.1 – Predictive Model Markup Language. http://www.dmg.org/html/pmml_v1_1.html, March 2001.
2. Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining Association Rules between Sets of Items in Large Databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*, pages 207–216. ACM Press, 1993.
3. Edward Angel. *Interactive Computer Graphics – A top-down approach with OpenGL^{TM}*. Addison Wesley Longman, Inc., Reading, Massachusetts, 2 edition, 2000.
4. Alberto Belussi, Elisa Bertino, and Barbara Catania. An Extended Algebra for Constraint Databases. *TKDE*, 10(5):686–705, 1998.
5. Jean-Francois Boulicaut, Mika Klemettinen, and Heikki Mannila. Modeling KDD Processes within the Inductive Database Framework. In *Proc. Data Warehousing and Knowledge Discovery, DaWak99,Florenz*, pages 293–302. Springer-Verlag, 1999.
6. Surajit Chaudhuri. Data Mining and Database Systems: Where is the Intersection? *Data Engineering Bulletin*, 21(1):4–8, 1998.
7. Surjit Chaudhuri, Vivek Narasayya, and Sunita Sarawagi. Efficient Evaluation of Queries with Mining Predicates. In *ICDE 2002*, pages 529 – 540, 2002.
8. Daniel Fasulo. An Analysis of Recent Work on Clustering Algorithms. Technical Report 01-03-02, University of Washington, April 1999.
9. Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From Data Mining to Knowledge Discovery: An Overview. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 1, pages 1–34. AAAI Press/ The MIT Press, 1996.
10. Ingolf Geist. A Framework for Data Mining and KDD. In *Applied Computing 2002. Proceedings of the 2002 ACM Symposium on Applied Computing, Madrid, Spain, March 11-14, 2002*, pages 508 – 513. ACM, 2002.
11. J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A Data Mining Query Language for Relational Databases. In *SIGMOD'96 Workshop. on Research Issues on Data Mining and Knowledge Discovery (DMKD'96), Montreal, Canada*, June 1996.
12. David Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. A Bradford Book, The MIT Press, Cambridge, Massachusetts, London, England, 2001.
13. T. Imielinski and H. Mannila. A Database Perspective on Knowledge Discovery. *Communications of ACM*, 39:58–64, 1996.
14. Tomasz Imielinski and Aashu Virmani. MSQL: A Query Language for Database Mining. *Data Mining and Knowledge Discovery*, 3(4):373–408, December 1999.
15. Theodore Johnson, Laks V. S. Lakshmanan, and Raymond T. Ng. The 3W Model and Algebra for Unified Data Mining. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 21–32. Morgan Kaufmann, 2000.
16. Paris C. Kanellakis, Gabrial M. Kuper, and Peter Revesz. Constraint Query Languages. *JCSS*, 51(1):26–52, August 1995.
17. Hillol Kargupta, Byung-Hoon Park, Daryl Hershberger, and Erik Johnson. Collective Data Mining: A New Perspective toward Distributed Data Mining. In Hillol Kargupta and Philip Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*, chapter 5, pages 131–178. MIT/AAAI Press, 2000.
18. Gabriel Kuper, Leonid Libkin, and Jan Paradaens, editors. *Constraint Databases*. Spring-Verlag, Berlin Heidelberg, 2000.

19. Tjen-Sien Lim and Yu-Shan Shih. A Comparision of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms. *Machine Learning*, 40(3):203–229, September 2000.
20. Amir Netz, Surajit Chaudhuri, Usama M. Fayyad, and Jeff Bernhardt. Integrating Data Mining with SQL Databases: OLE DB for Data Mining. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany.*, pages 379–387. IEEE Computer Society, 2001.
21. Carlos Ordonez and Paul Cereghini. SQLEM: Fast Clustering in SQL using the EM Algorithm. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, volume 29, pages 559–570. ACM, 2000.
22. Andreas L. Prodromidis, Philip K. Chan, and Salvatore J. Stolfo. Meta-Learning in Distributed Data Mining Systems: Issues and Approaches. In Hillol Kargupta and Philip Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*, chapter 3, pages 79–112. MIT/AAAI Press, 2000.
23. Foster J. Provost and Venkateswarlu Kolluri. A Survey of Methods for Scaling Up Inductive Algorithms. *Data Mining and Knowledge Discovery*, 3(2):131 – 169, Juni 1999.
24. Sunita Sarawagi, Shiby Thomas, and Rakesh Agrawal. Integrating Mining with Relational Database Systems: Alternatives and Implications. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 343–354. ACM Press, 1998.
25. K. Sattler and O. Dunemann. SQL Database Primitives for Decision Tree Classifiers. In *Proc. of the 10th ACM CIKM Int. Conf. on Information and Knowledge Management, November 5–10, 2001, Atlanta, Georgia, USA*, 2001.
26. E. Schallehn, K. Sattler, and G. Saake. Extensible and similarity-based grouping for data integration *Poster paper*. In *8th Int. Conf. on Data Engineering (ICDE), San Jose, CA*, 2002. To appear.
27. Friedemann Schwenkreis. *New Working Draft of SQL/MM Part 6: Data Mining based on BHX008 and BHX033-BHX039*. International Organization for Standardization (ISO), May 2000.
28. Haixun Wang and Carlo Zaniolo. User-Defined Aggregates for Datamining. In Kyuseok Shim and Ramakrishnan Srikant, editors, *1999 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Philadelphia, USA, May 30, 1999*, 1999.