# A Multi-relational Decision Tree Learning Algorithm - Implementation and Experiments

Anna Atramentov, Hector Leiva, and Vasant Honavar

Artificial Intelligence Research Laboratory
Computer Science Department
226 Atanasoff Hall, Iowa State University
Ames, IA 50011-1040, USA
{anjuta,aleiva,honavar}@cs.iastate.edu

**Abstract.** We describe an efficient implementation (MRDTL-2) of the Multi-relational decision tree learning (MRDTL) algorithm [23] which in turn was based on a proposal by Knobbe et al. [19] We describe some simple techniques for speeding up the calculation of sufficient statistics for decision trees and related hypothesis classes from multi-relational data. Because missing values are fairly common in many real-world applications of data mining, our implementation also includes some simple techniques for dealing with missing values. We describe results of experiments with several real-world data sets from the KDD Cup 2001 data mining competition and PKDD 2001 discovery challenge. Results of our experiments indicate that MRDTL is competitive with the state-of-the-art algorithms for learning classifiers from relational databases.

## 1 Introduction

Recent advances in high throughput data acquisition, digital storage, and communications technologies have made it possible to gather very large amounts of data in many scientific and commercial domains. Much of this data resides in relational databases. Even when the data repository is not a relational database, it is often convenient to view heterogeneous data sources as if they were a collection of relations [28] for the purpose of extracting and organizing information from multiple sources. Thus, the task of learning from relational data has begun to receive significant attention in the literature [1,19,11,21,22,12,18,26,9,8,14,16].

Knobbe et al. [19] outlined a general framework for multi-relational data mining which exploits structured query language (SQL) to gather the information needed for constructing classifiers (e.g., decision trees) from multi-relational data. Based on this framework, Leiva [23] developed a multi-relational decision tree learning algorithm (MRDTL). Experiments reported by Leiva [23] have shown that decision trees constructed using MRDTL have accuracies that are comparable to that obtained using other algorithms on several multi-relational data sets. However, MRDTL has two significant limitations from the standpoint of multi-relational data mining from large, real-world data sets:

T. Horváth and A. Yamamoto (Eds.): ILP 2003, LNAI 2835, pp. 38–56, 2003.

(a) Slow running time: MRDTL (like other algorithms based on the multi-relational data mining framework proposed by Knobbe et al. [19]) uses *selection graphs* to query the relevant databases to obtain the statistics needed for constructing the classifier. Our experiments with MRDTL on data from KDD Cup 2001 [5] showed that the execution of queries encoded by such selection graphs is a major bottleneck in terms of the running time of the algorithm.
(b) Inability to handle missing attribute values: In multi-relational databases encountered in many real-world applications of data mining, a significant fraction of the data have one or more missing attribute values. For example, in the case of gene localization task from KDD Cup 2001 [5], 70% of CLASS, 50% of COMPLEX and 50% of MOTIF attribute values are missing. Leiva's implementation of MRDTL [23] treats each missing value as a special value ("missing") and does not include any statistically well-founded techniques for dealing with missing values. Consequently, the accuracy of decision trees constructed using MRDTL is far from satisfactory on classification tasks in which missing attribute values are commonplace. For example, the accuracy of MRDTL on the gene localization task was approximately 50%.

Against this background, this paper describes MRDTL-2 which attempts to overcome these limitations of Leiva's implementation of MRDTL:

(a) MRDTL-2 includes techniques for significantly speeding up some of the most time consuming components of multi-relational data mining algorithms like MRDTL that rely on the use of selection graphs.
(b) MRDTL-2 includes a simple and computationally efficient technique which uses Naive Bayes classifiers for 'filling in' missing attribute values.

These enhancements enable us to apply multi-relational decision tree learning algorithms to significantly more complex classification tasks involving larger data sets and larger percentage of missing attribute values than was feasible in the case of MRDTL. Our experiments with several classification tasks drawn from KDD Cup 2001 [5], PKDD 2001 [7] and the widely studied Mutagenesis data set [25] show that MRDTL-2

(a) significantly outperforms MRDTL in terms of running time
(b) yields results that are comparable to the best reported results obtained using multi-relational data mining algorithms
(c) compares favorably with feature-based learners that are based on clever propositionalization methods [22]

The rest of the paper is organized as follows: Section 2 reviews the multi-relational data-mining framework; Section 3 describes our implementation of MRDTL-2, a multi-relational decision tree learning algorithm; Section 4 describes the results of our experiments with MRDTL-2 on several representative multi-relational data mining tasks and compares them with the results of other approaches available in the literature; Section 5 concludes with a brief summary and discussion of the main results and an outline of some directions for further research.

## 2 Multi-relational Data Mining

### 2.1 Relational Databases

A relational database consists of a set of tables $D = \{X_1, X_2, ...X_n\}$, and a set of associations between pairs of tables. In each table a row represents description of one record. A column represents values of some attribute for the records in the table. An attribute $A$ from table $X$ is denoted by $X.A$.

**Definition 1.** *The* domain *of the attribute $X.A$ is denoted as $DOM(X.A)$ and is defined as the set of all different values that the records from table $X$ have in the column of attribute $A$.*

Associations between tables are defined through primary and foreign key attributes.

**Definition 2.** *A* primary key *attribute of table $X$, denoted as $X.ID$, has a unique value for each row in this table.*

**Definition 3.** *A* foreign key *attribute in table $Y$ referencing table $X$, denoted as $Y.X\_ID$, takes values from $DOM(X.ID)$.*

An example of a relational database is shown in Figure 1. There are three tables and three associations between tables. The primary keys of the tables GENE, COMPOSITION, and INTERACTION are: GENE_ID, C_ID, and I_ID, respectively. Each COMPOSITION record references some GENE record through the foreign key COMPOSITION.GENE_ID, and each INTERACTION record references two GENE records through the foreign keys INTERACTION.GENE_ID1 and INTERACTION.GENE_ID2. In this setting the attribute of interest (e.g.,

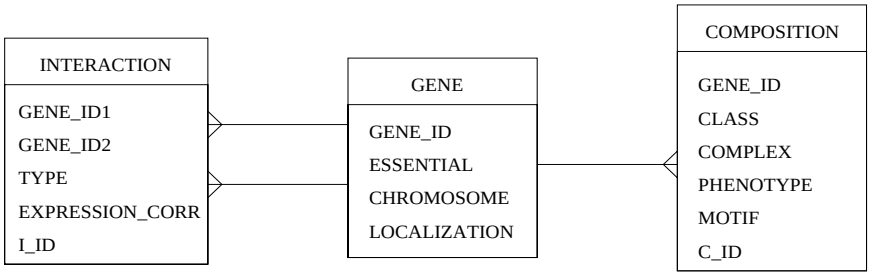


**Fig. 1.** Example database

class label) is called *target attribute*, and the table in which this attribute is stored is called *target table* and is denoted by $T_0$. Each record in $T_0$ corresponds to a single object. Additional information about an object is stored in other tables of the database that can be looked up by following the associations between tables.

### 2.2 Multi-relational Data Mining Framework

Multi-relational data mining framework is based on the search for interesting patterns in the relational database, where multi-relational patterns can be viewed as "pieces of substructure encountered in the structure of the objects of interest" [19].

**Definition 4.** ([19]) *A multi-relational object is* covered *by a multi-relational pattern iff the substructure described by the multi-relational pattern, in terms of both attribute-value conditions and structural conditions, occurs at least once in the multi-relational object.*

Multi-relational patterns also can be viewed as subsets of the objects from the database having some property. The most interesting subsets are chosen according to some measure (i.e. information gain for classification task), which guides the search in the space of all patterns. The search for interesting patterns usually proceeds by a top-down induction. For each interesting pattern, sub-patterns are obtained with the help of refinement operator, which can be seen as further division of the set of objects covered by initial pattern. Top-down induction of interesting pattern proceeds recursively applying such refinement operators to the best patterns. Multi-relational pattern language is defined in terms of *selection graphs* and *refinements* which are described in the following sections.

### 2.3 Selection Graphs

Multi-relational patterns are expressed in a graphical language of selection graphs [20].

**Definition 5.** ([19]) *A* selection graph $S$ *is a directed graph* $S = (N, E)$. $N$ *represents the set of nodes in* $S$ *in the form of tuples* $(X, C, s, f)$, *where* $X$ *is a table from* $D$, $C$ *is the set of conditions on attributes in* $X$ *(for example,* $X.color$ = *'red' or* $X.salary > 5{,}000$*),* $s$ *is a flag with possible values* open *and* closed, *and* $f$ *is a flag with possible values* front *and* back. $E$ *represents edges in* $S$ *in the form of tuples* $(p, q, a, e)$, *where* $p$ *and* $q$ *are nodes and* $a$ *is a relation between* $p$ *and* $q$ *in the data model (for example,* $X.ID = Y.X\_ID$*), and* $e$ *is a flag with possible values* present *and* absent. *The selection graph should contain at least one node* $n_0$ *that corresponds to the target table* $T_0$.

An example of the selection graph for the data model from Figure 1 is shown in Figure 2. This selection graph corresponds to those GENE(s) that belong to chromosome number 5, that have at least one INTERACTION record of type 'Genetic' with a corresponding GENE on chromosome number 11, but for which none of the INTERACTION records have type value 'Genetic' and expression_corr value 0.2. In this example the target table is GENE, and within GENE the target attribute is LOCALIZATION.

In graphical representation of a selection graph, the value of $s$ is represented by the absence or presence of a cross in the node, representing values *open*

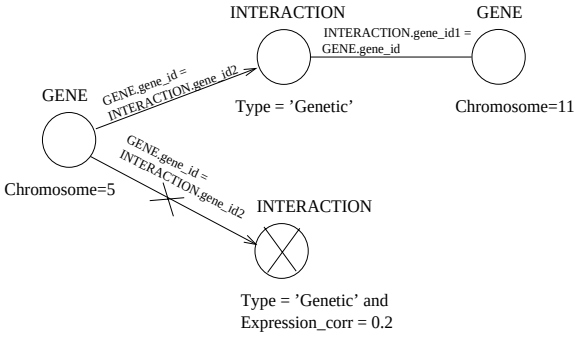


**Fig. 2.** Selection graph corresponding to those GENEs which belong to chromosome number 5, that have at least one interaction of type 'Genetic' with a corresponding gene on chromosome number 11 but for which none of the interaction records are of the type 'Genetic' and expression_corr value 0.2

---

TRANSLATE($S$, key)
*Input* Selection graph $S$, key (primary or foreign) in the target node of the selection graph $S$
*Output* SQL query for objects covered by selection graph $S$

```
1  table_list := ''
2  condition_list := ''
3  join_list := ''
4  for each node i in S do
5      if (i.s = 'open' and i.f = 'front')
6          table_list.add(i.table_name + 'T' + i)
7          for each condition c in i do
8              condition_list.add(c)
9  for each edge j in S do
10     if (j.e = 'present')
11         if (j.q.s = 'open' and j.q.f = 'front')
12             join_list.add(j.a)
13     else join_list.add(j.p + '.' + j.p.primary_key + ' not in ' +
           TRANSLATE( subgraph(S, j.q), j.q.key)
15 return 'select distinct' + 'T0.' + key +
       ' from ' + table_list +
       ' where ' + join_list + ' and ' + condition_list
```

---

**Fig. 3.** Translation of selection graph into SQL query

and *closed*, respectively. The value for $e$, in turn, is indicated by the presence (*absent* value) or absence (*present* value) of a cross on the corresponding arrow representing the edge. An edge between nodes $p$ and $q$ chooses the records in the database that match the join condition, $a$, between the tables which is defined by the relation between the primary key in $p$ and a foreign key in $q$, or the other way around. For example, the join condition, $a$, between table GENE

and INTERACTION in selection graph from Figure 2 is GENE.GENE_ID = INTERACTION.GENE_ID2.

A *present* edge between tables $p$ and $q$ combined with a list of conditions, $q.C$ and $p.C$, selects those objects that match the list of conditions, $q.C$ and $p.C$, and belong to the join between $p$ and $q$, specified by join condition, $e.a$. On the other hand, an *absent* edge between tables $p$ and $q$ combined with a list of conditions, $q.C$ and $p.C$, selects those objects that match condition $p.C$ but do not satisfy the following: match $q.C$ and belong to the join between tables at the same time. Flag $f$ is set to *front* for those nodes that on their path to $n_0$ have no closed edges. For all the other nodes flag $f$ is set to *back*.

```
select  distinct T0.gene_id
from    GENE T0, INTERACTION T1, GENE T2
where   T0.gene_id = T1.gene_id2 and T1.gene_id2 = T2.gene_id
        and T0.chromosome = 5 and T1.type = 'Genetic' and T2.chromosome = 11
        and T0.gene_id not in ( select T0.gene_id2
                                from  INTERACTION T0
                                where T0.type = 'Genetic' and T0.expression_corr = 0.2)
```

**Fig. 4.** SQL query corresponding to the selection graph in Figure 2

Knobbe et al. [20] introduce the algorithm (Figure 3) for translating a selection graph into SQL query. This algorithm returns the records in the target table covered by this selection graph. The subgraph($S$, $j.q$) procedure returns the subgraph of the selection graph $S$ starting with the node $q$ as the target node, which label $s$ is reset to *open*, removing the part of the graph that was connected to this node with the edge $j$ and resetting all the values of flag $f$ at the resulting selection graph by definition of $f$. Notation $j.q$.key means the name of the attribute (primary or foreign key) in the table $q$ that is associated with the table $p$ in relation $j.a$. Using this procedure the graph in Figure 2 translates to the SQL statement shown in Figure 4.

### 2.4 Refinements of Selection Graphs

Multi-relational data mining algorithms search for and successively refine interesting patterns and select promising ones based on some impurity measure (e.g. information gain). The set of refinements introduced by [20] are given below. We will illustrate all the refinements on the selection graph from Figure 2. Labels $s$ and $f$ help identify the nodes that needs to be refined. Note that a refinement can only be applied to the *open* and *front* nodes in the selection graph $S$.

(a) *Add positive condition.* This refinement simply adds a condition $c$ to the set of conditions $C$ in the node that is being refined in the selection graph $S$ without actually changing the structure of $S$. For the selection graph from Figure 2 positive condition expression_corr=0.5 applied to the node INTERACTION results in the selection graph shown on Figure 5 a.

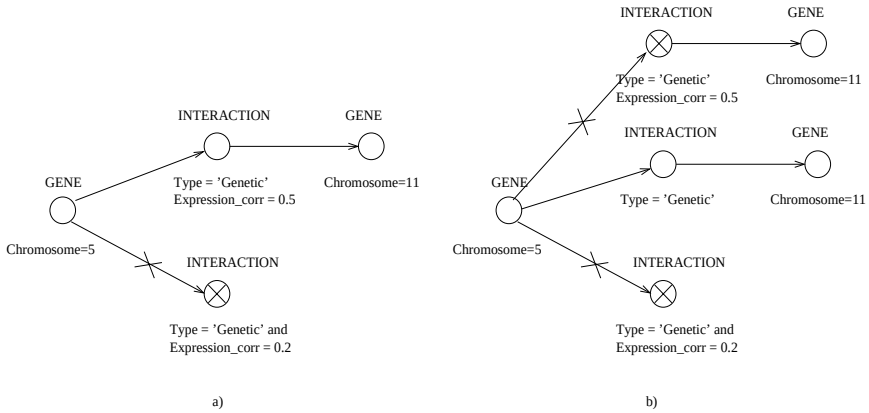


**Fig. 5.** Complement refinements for adding condition expression_corr=0.5 to the node INTERACTION in the selection graph from Figure 2: a) positive condition, b) negative condition

(b) *Add negative condition* (Figure 5 b). If the node which is refined is not $n_0$, the corresponding refinement will introduce a new absent edge from the parent of the selection node in question. The condition list of the selection node is copied to the new closed node and extended by the new condition. This node gets the copies of the children of the selection graph in question and open edges to those children are added. If the node which is refined represents the target table, the condition is simply negated and added to the current list of conditions for this node. This refinement is the complement of the "add positive condition refinement", in the sense that it covers those objects from the original selection graph which were not covered by corresponding "add positive condition" refinement.

(c) *Add present edge and open node.* This refinement introduces a present edge together with its corresponding table to the selection graph $S$. For the selection graph from Figure 2 adding edge from GENE node to COMPOSITION node results in the selection graph shown in (Figure 6 a).

(d) *Add absent edge and closed node* (Figure 6 b). This refinement introduces an absent edge together with its corresponding table to the selection graph $S$. This refinement is complement to the "add present edge and open node", in the sense that it covers those objects from the original selection graph which were not covered by "add present edge and open node" refinement.

It is important to note that only through the "add edge" refinements the exploration of all the tables in the database is carried out. We can consider "add condition" refinement on some attribute from some table only after the edge to that table has been added to the selection graph. This raises the question as to what happens if the values of the attributes in some table are important for the task but the edge to this table can never be added, i.e. adding edge doesn't result in further split of the data covered by the refined selection graph. Look ahead

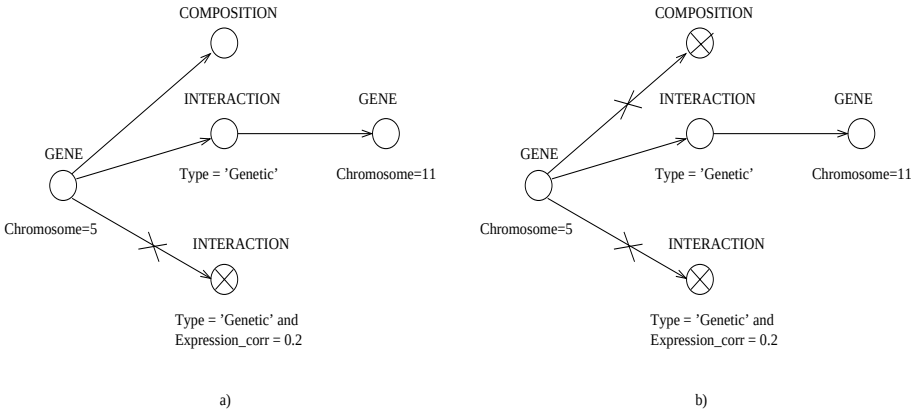


**Fig. 6.** Complement refinements for adding edge from GENE node to COMPOSITION node to selection graph from Figure 2: a) adding present edge and open node, b) adding absent edge and closed node

refinements, which are a sequence of several refinements, are used for dealing with this situation. In the case when some refinement does not split the data covered by the selection graph, the next set of refinements is also considered as refinements of the original selection graph.

## 3 MDRTL-2: An Efficient Multi-relational Decision Tree Learning Algorithm

### 3.1 Decision Tree Construction

Multi-relational decision tree learning algorithm constructs a decision tree whose nodes are multi-relational patterns i.e., selection graphs. MRDTL-2 that we describe below is based on MRDTL proposed by Leiva [23] which in turn is based on the algorithm described by [20] and the logical decision tree induction algorithm called TILDE proposed by [1]. TILDE uses first order logic clauses to represent decisions (nodes) in the tree, when data are represented in first order logic rather than a collection of records in a relational database. MRDTL deals with records in relational databases, similarly to the TILDE's approach. Essentially, MRDTL adds selection graphs as the nodes to the tree through a process of successive refinement until some termination criterion is met (e.g., correct classification of instances in the training set).

The choice of refinement to be added at each step is guided by a suitable impurity measure (e.g., information gain). MRDTL starts with the selection graph containing a single node at the root of the tree, which represents the set of all objects of interest in the relational database. This node corresponds to the target table $T_0$. The pseudocode for MRDTL is shown in Figure 7.

```
TREE_INDUCTION(D, S)
Input Database D, selection graph S
Output The root of the tree, T
1 R := optimal_refinement(S)
2 if stopping_criteria(S)
3     return leaf
4 else
6     T_left := TREE_INDUCTION(D, R(S))
8     T_right := TREE_INDUCTION(D, R̄(S))
9     return node(T_left, T_right, R)
```

**Fig. 7.** MRDTL algorithm

The function optimal_refinement($S$) considers every possible refinement that can be made to the current selection graph $S$ and selects the (locally) optimal refinement (i.e., one that maximizes information gain). Here, $R(S)$ denotes the selection graph resulting from applying the refinement $R$ to the selection graph $S$. $\overline{R}(S)$ denotes the application of the complement of $R$ to the selection graph $S$. Our implementation of MRDTL considers the refinements described in the Section 2 as well as the look ahead refinements. The program automatically determines from the relational schema of the current database when look ahead might be needed. When adding an edge does not result in further split of the data, two-step refinements of the original selection graph are considered. Each candidate refinement is evaluated in terms of the split of the data induced by the refinement with respect to the target attribute, as in the case of the propositional version of the decision tree learning algorithm [30]. Splits based on numerical attributes are handled using a technique similar to that of C4.5 algorithm [30] with modifications proposed in [10,29].

Our implementation of MRDTL uses SQL operations to obtain the counts needed for calculating information gain associated with the refinements. First we show the calculation of the information gain associated with "add condition" refinements. Let $X$ be the table associated with one of the nodes in the current selection graph $S$ and $X.A$ be the attribute to be refined, and $R_{v_j}(S)$ and $\overline{R}_{v_j}(S)$ be the "add condition" $X.A = v_j$ refinement and the complement of it respectively. The goal is to calculate entropies associated with the split based on these two refinements. This requires the following counts: $count(c_i, R_{v_j}(S))$ and $count(c_i, \overline{R}_{v_j}(S))$, where $count(c_i, S)$ is the number of objects covered with selection graph $S$ which have classification attribute $T_0$.target_attribute $= c_i \in$ DOM($T_0$.target_attribute). The result of the SQL query shown in Figure 8 returns a list of the necessary counts: $count(c_i, R_{v_j}(S))$ for each possible values $c_i \in$ DOM($T_0$.target_attribute) and $v_j \in$ DOM($X.A$).

The rest of the counts needed for the computation of the information gain can be obtained from the formula:

| | |
|---|---|
| **select** | $T_0$.target_attribute, $X.A$, count(distinct $T_0$.id) |
| **from** | TRANSLATE(S).get_table_list |
| **where** | TRANSLATE(S).get_join_list and TRANSLATE(S).get_condition_list |

**Fig. 8.** SQL query that returns counts of the type $count(c_i, R_{v_j}(S))$ for each of the possible values $c_i \in \text{DOM}(T_0.\text{target\_attribute})$ and $v_j \in \text{DOM}(X.A)$

$$count(c_i, \overline{R}_{v_j}(S)) = count(c_i, S) - count(c_i, R_{v_j}(S))$$

Consider the SQL queries needed for calculating information gain associated with "add edge" refinements. Let $X$ be the table associated with one of the nodes in the current selection graph $S$ and $e$ be the edge to be added from table $X$ to table $Y$, and $R_e(S)$ and $\overline{R}_e(S)$ be the "add edge" $e$ refinement and its complement respectively. In order to calculate the entropies associated with the split based on these refinements we need to gather the following counts: $count(c_i, R_e(S))$ and $count(c_i, \overline{R}_e(S))$. The result of the SQL query shown in Figure 9 returns a list of the desired counts: $count(c_i, R_e(S))$ for each possible value $c_i \in \text{DOM}(T_0.\text{target\_attribute})$.

| | |
|---|---|
| **select** | $T_0$.target_attribute, count(distinct $T_0$.id) |
| **from** | TRANSLATE(S).get_table_list, $Y$ |
| **where** | TRANSLATE(S).get_join_list and TRANSLATE(S).get_command_list and $e.a$ |

**Fig. 9.** SQL query returning counts of the type $count(c_i, R_e(S))$ for each possible value $c_i \in \text{DOM}(T_0.\text{target\_attribute})$

The rest of the counts needed for the computation of the information gain can be obtained from the formula:

$$count(c_i, \overline{R}_e(S)) = count(c_i, S) - count(c_i, R_e(S))$$

The straightforward implementation of the algorithm based on the description given so far suffers from an efficiency problem which makes its application to complex real-world data sets infeasible in practice. As one gets further down in the decision tree the selection graph at the corresponding node grows. Thus, as more and more nodes are added to the decision tree the longer it takes to execute the corresponding SQL queries (Figures 8, 9) needed to examine the candidate refinements of the corresponding selection graph. Consequently, the straightforward implementation of MRDTL as described in [23] is too slow to be useful in practice.

MRDTL-2 is a more efficient version of MRDTL. It exploits the fact that some of the results of computations that were carried out in the course of adding nodes at higher levels in the decision tree can be reused at lower levels in the

course of refining the tree. Note that the queries from Figures 8 and 9 unnecessarily repeats work done earlier by retrieving instances covered by the selection graph whenever refining an existing selection graph. This query can be significantly simplified by storing the instances covered by the selection graph from previous iteration in a table to avoid retrieving them from the database. Thus, refining an existing selection graph reduces to finding a (locally) optimal split of the relevant set of instances.

Now we proceed to show how the calculation of the SQL queries in Figures 8 and 9 is carried out in MRDTL-2. In each iteration of the algorithm, we store the primary keys from all open, front nodes of the selection graph for every object covered by it together with its classification value. This can be viewed as storing the 'skeletons' of the objects covered by the selection graph, because it stores no other attribute information about records except for their primary keys. The SQL query for generating such a table for the selection graph $S$ is shown in Figure 10. We call the resulting table of primary keys the *sufficient table* for $S$ and denote it by $I_S$.

```
SUF_TABLE(S)
Input Selection graph S
Output SQL query for creating sufficient table I_S
1  table_list, condition_list, join_list := extract_from(TRANSLATE(S))
2 primary_key_list := 'T_0.target_attribute'
3 for each node i in S do
4       if (i.s = 'open' and i.f = 'front')
5             primary_key_list .add(i.ID)
6 return 'create table I_S as ( select ' + primary_key_list +
                              ' from ' + table_list +
                              ' where ' + join_list + ' and ' + condition_list + ')'
```

**Fig. 10.** Algorithm for generating SQL query corresponding to the sufficient table $I_S$ of the selection graph $S$

Given a sufficient table $I_S$, we can obtain the counts needed for the calculation of the entropy for the "add condition" refinements as shown in Figure 11, and for the "add edge" refinements as shown in Figure 12.

It is easy to see that now the number of tables that need to be joined is not more than 3, whereas the number of tables needed to be joined in Figures 8 and 9 grows with the size of the selection graph. It is this growth that was responsible for the significant performance deterioration of MRDTL as nodes get added to the decision tree. It is important to note that it is inefficient to use the algorithm from Figure 10 in each iteration, since again the size of the query would increase with the growth of the selection graph. It is possible to create the sufficient table for the refined selection graph using only the information about refinement and sufficient table of the original selection graph as shown in Figure 13.

| | |
|---|---|
| **select** | $I_S.T_0$_target_attribute, $X.A$, count(distinct $I_S.T_0$_id) |
| **from** | $I_S$, $X$ |
| **where** | $I_S.X$_ID = $X$.ID |

**Fig. 11.** SQL query which returns the counts needed for calculating entropy for the splits based on "add condition" refinements to the node $X$ for the attribute $X.A$ using sufficient table $I_S$

The simple modifications described above make MRDTL-2 significantly faster to execute compared to MRDTL. This is confirmed by experimental results presented in section 4.

### 3.2 Handling Missing Values

The current implementation of MRDTL-2 incorporates a simple approach to dealing with missing attribute values in the data. A Naive Bayes model for each attribute in a table is built based on the other attributes (excluding the class attribute). Missing attribute values are 'filled in' with the most likely value predicted by the Naive Bayes predictor for the corresponding attribute. Thus, for each record, $r$, from table $X$ we replace its missing value for the attribute $X.A$ with the following value:

$$v_{NB} = \underset{v_j \in \mathrm{DOM}(X.A)}{\operatorname{argmax}} \quad P(v_j) \prod_{\forall X_l \in D} \prod_{\substack{\forall X_l.A_i, A_i \neq A, \\ A_i \neq T_0.\text{target\_attribute}}} \prod_{\substack{\forall r_n \in X_l, r_n \\ \text{associatedwith} r}} P(X_l.A_i | v_j)$$

Here the first product is taken over the tables in the training database; The second product is taken over all the attributes in that table, except for the target attribute, $T_0$.target_attribute, and the attribute which is being predicted, namely, $X.A$; The third product is taken over all the records in the table $X_l$ which are connected to the record $r$ through the associations between the tables $X$ and $X_l$. In the case of one-to-many relation between the tables $X$ and $X_l$, one record from table $X$ may have several corresponding records in the table $X_l$. The value $P(X_l.A_i|v_j)$ is defined as the probability that some random element from table $X$ has at least one corresponding record from table $X_l$.

Once the tables in the database are preprocessed in this manner, MRDTL-2 proceeds to build a decision tree from the resulting tables that contain no missing attribute values.

| | |
|---|---|
| **select** | $I_S.T_0$_target_attribute, count(distinct $I_S.T_0$_id) |
| **from** | $I_S$, $X$, $Y$ |
| **where** | $I_S.X$_ID = $X$.ID and $e.a$ |

**Fig. 12.** SQL query which returns counts needed for calculating entropy for the splits based on "add edge" $e$ refinements from the node $X$ to the node $Y$ using sufficient table $I_S$

```
REFINEMENT_SUF_TABLE(I_S, R)
Input Sufficient table I_S for selection graph S, refinement R
Output SQL query for sufficient table for R(S)
1  table_list := I_S
2  condition_list := ''
3  join_list := ''
4  primary_key_list := primary_keys(I_S)
5  if R == add positive condition, c, in table T_i
6       table_list += T_i
7       condition_list += T_i.c
8       join_list += T_i.ID+' ='+I_S.T_i_ID
9  else if R == add negative condition, c, in table T_i
10      condition_list += T_0.ID + 'is not in ( select distinct' + I_S.T_0_ID +
                        ' from' + I_S, T_i +
                        ' where' + T_i.c + 'and' + T_i.ID+' =' +I_S.T_i_ID+')'
12 else if R = add present edge, e, from T_i to T_j
13      table_list += T_i+',' +T_j
14      join_list += T_i.ID+' =' +I_S.T_i_ID+' and ' + e.a
15      primary_key_list += T_j.ID
16 else if R == add absent edge, e from T_i to T_j
17      condition_list += T_0.ID + 'is not in ( select distinct' + I_S.T_0_ID +
        ' from ' + I_S+',' +T_i+',' +T_j +
        ' where ' + T_i.ID+' =' +I_S.T_i_ID+' and ' + e.a+')'
19 return 'create table I_R as ( select ' + primary_key_list +
                            ' from ' + table_list +
                            ' where ' + join_list + ' and ' + condition_list + ')'
```

**Fig. 13.** Algorithm for generating SQL query corresponding to sufficient table $I_{R(S)}$

In the future it would be interesting to investigate more sophisticated techniques for dealing with missing values.

### 3.3 Using the Decision Tree for Classification

Before classifying an instance, any missing attribute values are filled in by preprocessing the tables using the method described above on the database of instances to be classified.

The decision tree produced by MRDTL-2, as in the case of MRDTL, can be viewed as a set of SQL queries associated with the selection graphs that correspond to the leaves of the decision tree. Each selection graph (query) has a class label associated with it. If the corresponding node is not a pure node, (i.e., it does not unambiguously classify the training instances that match the query), the label associated with the node is based on the classification of the majority of training instances that match the corresponding selection graph in our implementation. (Alternatively, we could use probabilistic assignment of

labels based on the distribution of class labels among the training instances that match the corresponding selection graph). The complementary nature of the different branches of a decision tree ensures that a given instance will not be assigned conflicting labels. It is also worth noting that it is not necessary to traverse the entire tree in order to classify a new instance; all the constraints on a certain path are stored in the selection graph associated with the corresponding leaf node.

## 4 Experimental Results

Our experiments focused on three data sets - the mutagenesis database which has been widely used in Inductive Logic Programming (ILP) research [25], the data for prediction of protein/gene localization and function from KDD Cup 2001 [17] and the data for predicting thrombosis from PKDD 2001 Discovery Challenge [27]. We compared the results we obtained using MRDTL-2 algorithm with those reported in the literature for the same datasets.

### 4.1 Mutagenesis Data Set

The entity-relation diagram for the part of the Mutagenesis database [25] we used in our experiments is shown in Figure 14. The data set consists of 230 molecules divided into two subsets: 188 molecules for which linear regression yields good results and 42 molecules that are regression-unfriendly. This database contains descriptions of molecules and the characteristic to be predicted is their mutagenic activity (ability to cause DNA to mutate) represented by attribute label in molecule table.

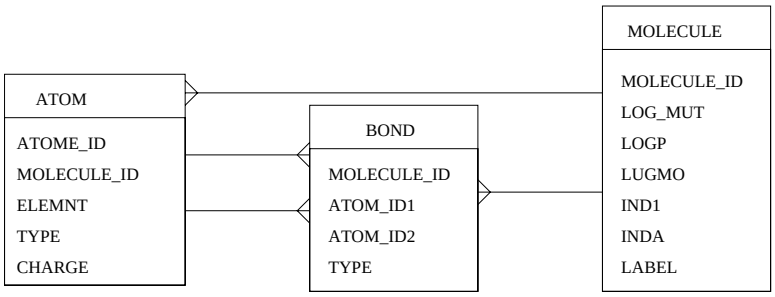


**Fig. 14.** Schema of the mutagenesis database

This dataset comes with different levels of background knowledge $B_0$, $B_1$, $B_2$, and $B_3$. In our experiments we chose to use the background knowledge $B_2$ and regression friendly subset of the dataset in order to compare the performance of MRDTL-2 with other methods for which experimental results are available in the literature. The results averaged with ten-fold cross-validation are shown in the Table 1.

**Table 1.** Experimental results for mutagenesis data

| | accuracy | time with MRDTL-2 | time with MRDTL |
|---|---|---|---|
| mutagenesis | 87.5 % | 28.45 secs | 52.155 secs |

**Table 2.** Experimental results for gene/protein localization prediction task

| localization | accuracy | time with MRDTL-2 | time with MRDTL |
|---|---|---|---|
| accuracy with *mvh* | 76.11 % | 202.9 secs | 1256.387 secs |
| accuracy without *mvh* | 50.14 % | 550.76 secs | 2257.206 secs |

### 4.2 KDD Cup 2001 Data Set

We also considered the two tasks from the KDD Cup 2001 data mining competition [5]: prediction of gene/protein function and localization.

We normalized the data given in the task which resulted in the schema shown in Table 1. The resulting database consists of 3 tables, GENE, INTERACTION and COMPOSITION, where GENE table contains 862 entries in the training set and 381 in the testing set. Gene localization and function tasks present significant challenges because many attribute values in the data set are missing. We have conducted experiments both using technique for handling missing values, denoted *mvh* in the tables 2 and 3, and not using it (considering each missing values as a special value "missing").

Table 2 summarizes the results we obtained for predicting GENE.localization value on this set.

In the case gene/protein function prediction, instances often have several class labels, since a protein may have several functions. MRDTL-2, like its propositional counterpart C4.5, assumes that each instance can be assigned to only one of several non-overlapping classes. To deal with multivalued class attributes, we transformed the problem into one of separately predicting membership in each possible class. i.e. for each possible function label we predicted whether the protein has this function or not. The overall accuracy was obtained from the formula:

$$\frac{(\text{true\_positive} + \text{true\_negative})}{(\text{true\_positive} + \text{true\_negative} + \text{false\_positive} + \text{false\_negative})}$$

for all binary predictions. Table 3 summarizes the results we got for predicting the function of the protein.

### 4.3 PKDD 2001 Discovery Challenge Data Set

The Thrombosis Data from the PKDD 2001 Discovery Challenge Data Set [7] consists of seven tables. PATIENT_INFO contains 1239 records about patients. For our experiments we used 4 other tables (DIAGNOSIS, ANTIBODY_EXAM,

**Table 3.** Experimental results for gene/protein function prediction task

| function | accuracy | time with MRDTL-2 | time with MRDTL |
|---|---|---|---|
| accuracy with *mvh* | 91.44 % | 151.19 secs | 307.82 secs |
| accuracy without *mvh* | 88.56 % | 61.29 secs | 118.41 secs |

**Table 4.** Experimental results for Thrombosis data set

| | accuracy | time with MRDTL-2 | time with MRDTL |
|---|---|---|---|
| thrombosis | 98.1 % | 127.75 secs | 198.22 secs |

**Table 5.** Comparison of MRDTL-2 performance with the best-known reported results

| dataset | MRDTL accuracy | best reported accuracy | reference |
|---|---|---|---|
| mutagenesis | 87.5 % | 86 % | [31] |
| localization | 76.11 % | 72.1 % | [5] |
| function | 91.44 % | 93.6 % | [5] |
| thrombosis | 98.1 % | 99.28 % | [7] |

ANA_PATTERN and THROMBOSIS) which all have a foreign key to the PATIENT_INFO table. There are no other relations between the tables in this dataset. The task is to predict the degree of thrombosis attribute from ANTIBODY_EXAM table. The results we obtained with 5:2 cross-validation are shown in Table 4. The cross-validation was done by partitioning the set of all records in the ANTIBODY_EXAM table and their corresponding records from other tables into training and test sets.

### 4.4 Comparative Evaluation

The results of the comparison of MRDTL-2 performance with the best-known reported results for the datasets we described above are shown in the Table 5.

## 5 Summary and Discussion

Advances in data acquisition, digital communication, and storage technologies have made it possible to gather and store large volumes data in digital form. A large fraction of this data resides in relational databases. Even when the data repository is not a relational database, it is possible to extract information from heterogeneous, autonomous, distributed data sources using domain specific ontologies [28]. The result of such data integration is in the form of relational tables. Effective use of such data in data-driven scientific discovery and decision-making calls for sophisticated algorithms for knowledge acquisition from relational databases or multi-relational data mining algorithms is an

important problem in multi-relational data mining which has begun to receive significant attention in the literature [1,19,11,21,22,12,18,26,9,8,14,16,2,6,13,24]. Learning classifiers from relational databases has also been a focus of KDD Cup 2001 Competition and the PKDD 2001 Discovery Challenge. Against this background, this paper describes the design and implementation of MRDTL-2 - an algorithm for learning decision tree classifiers from relational databases, which is based on the framework for multi-relational data mining originally proposed by Knobbe et al. [19]. MRDTL-2 extends an MRDTL, an algorithm for learning decision tree classifiers described by Leiva [23]. MRDTL-2 includes enhancements that overcome two significant limitations of MRDTL:

(a) Slow running time: MRDTL-2 incorporates methods for speeding up MRDTL. Experiments using several data sets from KDD Cup 2001 and PKDD 2001 Discovery Challenge show that the proposed methods can significantly reduce the running time of the algorithm, thereby making it possible to apply multi-relational decision tree learning algorithms on far more complex data mining tasks. The proposed methods are potentially applicable to a broad class of multi-relational data mining algorithms based on the framework proposed by Knobbe et al. [19].
(b) Inability to handle missing attribute values: MRDTL-2 includes a simple and computationally efficient technique using Naive Bayes classifiers for 'filling in' missing attribute values, which significantly enhances the applicability of multi-relational decision tree learning algorithms to the real-world classification tasks.

Our experiments with several classification tasks drawn from KDD Cup 2001 [5] and PKDD 2001 Discovery Challenge [7] and the widely studied Mutagenesis data set show that MRDTL-2

(a) significantly outperforms MRDTL in terms of running time
(b) yields results that are comparable to the best reported results obtained using multi-relational data mining algorithms (Table 5)
(c) compares favorably with feature-based learners that are based on clever propositionalization methods [22]

Work in progress is aimed at:

(a) Incorporation of more sophisticated methods for handling missing attribute values into MRDTL-2
(b) Incorporation of sophisticated pruning methods or complexity regularization techniques into MRDTL-2 to minimize overfitting and improve generalization
(c) Development of ontology-guided multi-relational decision tree learning algorithms to generate classifiers at multiple levels of abstraction [33]
(d) Development of variants of MRDTL that can learn from heterogeneous, distributed, autonomous data [4,3,28]
(e) More extensive experimental evaluation of MRDTL-2 on real-world data sets.

(f) Incorporation of more sophisticated methods for evaluation of MRDTL-2 [15].

**Acknowledgments**

This work is supported in part by an Information Technology Research (ITR) grant (0219699) from the National Science Foundation, and a Biological Information Science and Technology Initiative (BISTI) award (GM066387) from the National Institutes of Health. The paper has benefited from discussions with Doina Caragea and Adrian Silvescu of the Iowa State University Artificial Intelligence Research Laboratory.

## References

1. Hendrik Blockeel: Top-down induction of first order logical decision trees. Department of Computer Science, Katholieke Universiteit Leuven (1998)
2. Blockeel, H., and De Raedt, L.: Relational Knowledge Discovery in Databases. In: Proceedings of the sixth internal workshop of Inductive Logic Programming, volume 1312 of Lecture Notes in Artificial Intelligence, 199-212, Springer-Verlag (1996)
3. Caragea, D., Silvescu, A., and Honavar, V. (ISDA 2003)Decision Tree Induction from Distributed, Heterogeneous, Autonomous Data Sources. In: Proceedings of the Conference on Intelligent Systems Design and Applications. In press.
4. Caragea, D., Silvescu, A. and Honavar, V.: Invited Chapter. Toward a Theoretical Framework for Analysis and Synthesis of Agents That Learn from Distributed Dynamic Data SourcesTechnical Report. In: Emerging Neural Architectures Based on Neuroscience, Berlin: Springer-Verlag (2001)
5. Cheng, J., Krogel, M., Sese, J., Hatzis C., Morishita, S., Hayashi, H. and Page, D.: KDD Cup 2001 Report. In: ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) Explorations, vol. 3, issue 2 (2002)
6. Santos Costa et al.: Query Transformations for Improving the Efficiency of ILP Systems. In: Journal of Machine Learning Research (2002)
7. Coursac, I., Duteil, N., Lucas, N.: pKDD 2001 Discovery Challenge - Medical Domain. In: PKDD Discovery Challenge 2001, vol. 3, issue 2 (2002)
8. L. Dehaspe and L. De Raedt: Mining Association Rules in Multiple Relations. In: Proceedings of the 7th International Workshop on Inductive Logic Programming, vol. 1297, p. 125-132 (1997)
9. Dzeroski, S., and Lavrac, N.: Relational Data Mining. Springer-Verlag (2001)
10. Fayyad, U.M., and Irani, K.B: On the handling of continuous-valued attributes in decision tree generation. In: Machine Learning, vol.8 (1992)
11. Friedman, N., Getoor, L., Koller, D., and Pfeffer: Learning probabilistic relational models. In: Proceedings of the 6th International Joint Conference on Artificial Intelligence (1999)
12. Getoor, L.: Multi-relational data mining using probabilistic relational models: research summary. In: Proceedings of the First Workshop in Multi-relational Data Mining (2001)
13. Masaki Ito and Hayato Ohwada: Efficient Database Access for Implementing a Scalable ILP engine. In: Work-In-Progress Report of the Eleventh International Conference on Inductive Logic Programming (2001)

14. Jaeger, M.: Relational Bayesian networks. In: Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-1997)
15. Jensen, D., and J. Neville: Autocorrelation and Linkage Cause Bias in Evaluation of Relational Learners. In: Proceedings of the 12th International Conference on Inductive Logic Programming (ILP 2002)
16. Karalic and Bratko: First order regression. In: Machine Learning 26, vol. 1997
17. The KDD Cup 2001 dataset: `http://www.cs.wisc.edu/$\sim$dpage/kddcup2001/`
18. Kersting, K., and De Raedt, L.: Bayesian Logic Programs. In: Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming (2000)
19. Knobbe, J., Blockeel, H., Siebes, A., and Van der Wallen, D.: Multi-relational Data Mining. In: Proceedings of Benelearn (1999)
20. Knobbe, J., Blockeel, H., Siebes, A., and Van der Wallen, D.: Multi-relational decision tree induction. In: Proceedings of the 3rdEuropean Conference on Principles and Practice of Knowledge Discovery in Databases, PKDD-99 (1999)
21. Koller, D.: Probabilistic Relational Models. In: Proceedings of 9th International Workshop on Inductive Logic Programming (ILP-99)
22. Krogel, M., and Wrobel, S.: Transformation-Based Learning Using Multirelational Aggregation. In: Proceedings of the 11th International Conference on Inductive Logic Programming, vol. 2157 (2001)
23. Hector Ariel Leiva: A multi-relational decision tree learning algorithm. M.S. thesis. Deparment of Computer Science. Iowa State University (2002)
24. Morik, K., and Brockhausen, P.: A multistrategy approach to relational discovery in databases. In: Machine Learning, 27(3), 287-312 (1997)
25. The mutagenesis dataset: `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/mutagenesis.html`
26. Pfeffer, A.: A Bayesian Language for Cumulative Learning. In: Proceedings of AAAI 2000 Workshop on Learning Statistical Models from Relational Data, AAAI Press (2000)
27. The PKDD 2001 Discovery Challenge dataset: `http://www.uncc.edu/knowledgediscovery`
28. Jaime Reinoso-Castillo: Ontology-driven information extraction and integration from Heterogeneous Distributed Autonomous Data Sources. M.S. Thesis. Department of Computer Science. Iowa State University (2002)
29. Quinlan, R.: Improved Use of Continuous Attributes in C4.5. In: Journal of Artificial Intelligence Research, vol.4 (1996)
30. Quinlan, R.: C4.5: Programs for Machine Learning. In: San Mateo: Morgan Kaufmann (1993)
31. Srinivasan, A., King, R.D., and Muggleton, S.: The role of background knowledge: using a problem from chemistry to examine the performance of an ILP program. Technical Report PRG-TR-08-99, Oxford University Computing Laboratory, Oxford, 1999.
32. Wang, X., Schroeder, D., Dobbs, D., and Honavar, V. (2003). Data-Driven Discovery of Rules for Protein Function Classification Based on Sequence Motifs. Information Sciences. In press.
33. Zhang, J., and Honavar, V. (2003). Learning Decision Tree Classifiers from Attribute-Value Taxonomies and Partially Specified Data. In: Proceedings of the International Conference on Machine Learning. Washington, DC. In press.