Multi-relational decision tree algorithm -

implementation and experiments

by

Anna Atramentov

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Vasant Honavar, Major Professor
Drena Leigh Dobbs
Yan-Bin Jia

Iowa State University

Ames, Iowa

2003

Graduate College
Iowa State University

This is to certify that the master's thesis of

Anna Atramentov

has met the thesis requirements of Iowa State University

Major Professor

For the Major Program

TABLE OF CONTENTS

CHAPTER 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 (46) 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 (2; 26; 14; 29; 31; 16; 25; 45; 12; 9; 20; 23).

Knobbe et al. (26) 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 (36) developed a multi-relational decision tree learning algorithm (MRDTL). Experiments reported by Leiva (36) 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:

(a) Slow running time: MRDTL (like other algorithms based on the multi-relational data mining framework proposed by Knobbe et al. (26)) 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 (6) 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 (6), 70%

of CLASS, 50% of COMPLEX and 50% of MOTIF attribute values are missing. Leiva's implementation of MRDTL (36) 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%.

The slow running time of MRDTL makes it hard to incorporate into it, many of the standard approaches to handling missing attribute values that have been explored in the machine learning literature (48). Against this background, this thesis describes MRDTL-2 which attempts to overcome these limitations of Leiva's implementation of MRDTL:

(a) MRDTL-2 includes techniques for significantly speeding up, often by a couple of orders of magnitude, 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 (6), PKDD 2001 (7) and the widely studied Mutagenesis data set (http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/mutagenesis.html) 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 (31)

The thesis is organized as follows: Chapter 2 overviews the relevant literature in the relational data mining field, Chapter 3 reviews the multi-relational data-mining framework; Chapter 4 describes our implementation of MRDTL-2, a multi-relational decision tree learning algorithm; Chapter 5 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; Chapter 6 concludes with a brief summary and discussion of the main results and an outline of some directions for further research.

CHAPTER 2. RELATIONAL DATA MINING APPROACHES

2.1 Inductive Logic Programming

ILP is a broad field which evolved from initial focus on developing algorithms for the synthesis of logic programs from examples and background knowledge (i.e. knowledge acquisition for some domain) to more recent considerations of classification, regression, clustering, and association analysis (12). Due to its flexible and expressive way of representing background knowledge and examples, the field considers not only the single-table representation of the data but also the multiple-table representation. The main paradigm in ILP is the use of Induction and Logic Programming. Induction is one of the main techniques used in several Machine Learning algorithms to produce models that generalize beyond specific instances. Logic Programming is the programming paradigm that uses first order logic to represent relations between objects, Prolog, as the main representative language and implements deductive reasoning.

A sample ILP problem is considered below.

Let $E+ = \{\mathrm{daughter(Mary, Ann)}, \mathrm{daughter(Eve, Tom)}\}$ be the positive examples of the relation *daughter*;

$E- = \{\mathrm{daughter(Tom, Ann)}, \mathrm{daughter(Eve, Ann)}\}$ be the negative examples of the same relation; and

$B = \{\mathrm{mother(Ann, Mary)}, \mathrm{mother(Ann, Tom)}, \mathrm{father(Tom, Eve)}, \mathrm{father(Tom, Ian)},$

$\mathrm{female(Ann)}, \mathrm{female(Mary)}, \mathrm{female(Eve)}, \mathrm{male(Pat)}, \mathrm{male(Tom)},$

$\mathrm{parent(X, Y)} \leftarrow \mathrm{mother(X, Y)}, \mathrm{parent(X, Y)} \leftarrow \mathrm{father(X, Y)}\}$ be the background knowledge.

In this example the relations *daughter, mother, father, female, male,* and *parent* have the common meaning. If the goal of this ILP problem is to learn the concept *daughter*, then for predictive ILP a solution could be the following synthesized clause:

$\mathrm{daughter(X, Y)} \leftarrow \mathrm{female(X), parent(Y, X)}$

or a set of definite clauses:

$\mathrm{daughter(X, Y)} \leftarrow \mathrm{female(X), mother(Y, X)}$

$\mathrm{daughter(X, Y)} \leftarrow \mathrm{female(X), father(Y, X)}$

The deduction process is usually based on the use of sound rules of inference. Inductive inference involves unsound conjectures based on statistical support from data (39), which makes it challenging to produce well established ILP engines similar to those existing in deductive logic programming with Prolog.

Among other learning approaches the ILP has been one of the first and most expanded ones. Its use in relational data mining has been limited, though, due to the differences in input specification and language bias in different ILP engines (28). In order to deal with the logic formalism and to unify the different input specifications of different ILP engines, (28) propose the use of Unified Modeling Language (UML) as the common specification language for a large range of ILP engines. The idea is to replace the logic-based formalism to specify the language bias with a language that is easy to use, can be used by a range of ILP systems, and can represent the complexity of the problem in a clear and simple way. In this way, even the non-expert users will be able to model their problems and use a wide range of ILP engines, choosing the one that best fits their needs.

Another difficulty with the adaptation of the ILP techniques for relational learning is their limited use of relational database capabilities. The use of relational database engines increases the efficiency of the ILP systems. In (3), three ways of connecting ILP and relational databases are presented, which are briefly described below:

(a) The simplest approach is to pre-process the relational data into Prolog syntax (2).

(b) The second approach makes the next step in linking a Prolog systems with the relational databases. A relational database is given together with the Prolog knowledge base. Each time a literal $p(a, b)$ has to be evaluated, where the predicate p corresponds to a relation P in the relational database, Prolog has to open a connection with the database and make corresponding query to determine whether the tuple (a, b) is in P.

(c) The last solution exploits the close relationships between first order logic and relational databases (a predicate is a relation between its arguments which are, in turn, attributes in relational databases) and between logical clauses and relational database queries. Therefore, entire logical clauses (not only a literal at a time) can be translated into SQL statements and submitted to the database to obtain necessary statistics. Systems implementing this solution already exist. One of the such systems is RDT/DB (37; 3), which couples the ILP system RDT (24) with the Oracle database system.

ILP engines can be placed within one of the two main logical approaches: learning from entailment and learning from interpretations. Learning from entailment approach is also called explanatory ILP (2). Most of ILP systems are learning from entailments (e.g. RDT (24), Progol (39), FOIL (49), SRT (30), Fors (23)). Learning from interpretations is also called descriptive ILP. Examples of the ILP engines which are based on this setting are Claudien, ICL, and Tilde (11). It was shown in (2) how learning from interpretations evolved from a descriptive setting to a predictive one. Additional approaches to ILP can be found in (10). The difference between the two ILP approaches are in the way they represent the examples, the background knowledge and the way the final hypothesis is induced. In learning from interpretations each example e is represented by a separate Prolog program encoding its specific properties as sets of interpretations. The background knowledge B is given in the form of another Prolog program. The learning from entailment paradigm represents all the data examples together with the background knowledge as a simple Prolog program. A hypothesis in the learning from interpretations setting is a set of clauses such that only each positive example together with the background knowledge makes each clause in the set true. Let E^+ and E^- be the sets of all positive and negative examples respectively, a hypothesis H has to be found such that the following constraints hold

$\forall e \in E^+ : H$ is true in $M(e \wedge B)$

$\forall e \in E^- : H$ is false in $M(e \wedge B)$

where $M(e \wedge B)$ is the minimal Herbrand model of $e \wedge B$.

In the learning from entailment framework the problem is to find H such that the following entailment constraints hold

$\forall e \in E^+ : H\ H \wedge B$ entails e

$\forall e \in E^+ : H\ H \wedge B$ doesn't entail e

Within learning from entailment there are two principal approaches to the induction process: inverse deduction and a generalization of top-down induction techniques to the first-order case. An example of the system using inverse deduction technique is Progol (39). It uses inverse entailment which is a generalization and enhancement of the inverse deduction. FOIL system (49) is one of the first ILP systems using top-down induction. Most commonly the ILP systems that learn from interpretations use top-down induction of hypothesis. Although it has been shown that learning from interpretations reduces to learning from entailment (10), which in turn reduces to learning from satisfiability (learning from partial interpretations, where missing values are considered), an increased interest has been focused on learning from interpretations in several recent ILP systems (2; 11). The important reason for that is that the attribute value representations are a special case of this setting. Many attribute value techniques exploit the independence of examples. The learning from interpretations setting also assumes this independence through the separation of information between examples. Therefore, the attribute value learning algorithms can be upgraded in a more trivial way to the learning from interpretations setting than to the learning from entailment. This has been shown by the systems Claudien, ICL and Tilde (11). The learning from interpretations setting has also inspired the shifting from a pure logical search space to the one consisting exclusively of database queries in relational algebra or SQL-like language in (3; 26; 27).

2.2 First Order Extensions of Bayesian Networks

Probabilistic models, most specifically Bayesian Networks (BNs) (44), differ from the Inductive Logic Programming approaches by specifying a probability distribution over a fixed set of random variables that could be the attributes in an attribute-value setting. Thus, BNs are a probabilistic extension of propositional logic (25). One of the most important advantages of these models is their capability of reasoning under uncertainty. When applied to relational data, probabilistic models have the same limitations as propositional logic.

Several approaches for combining first order logic and Bayesian Networks have been proposed in

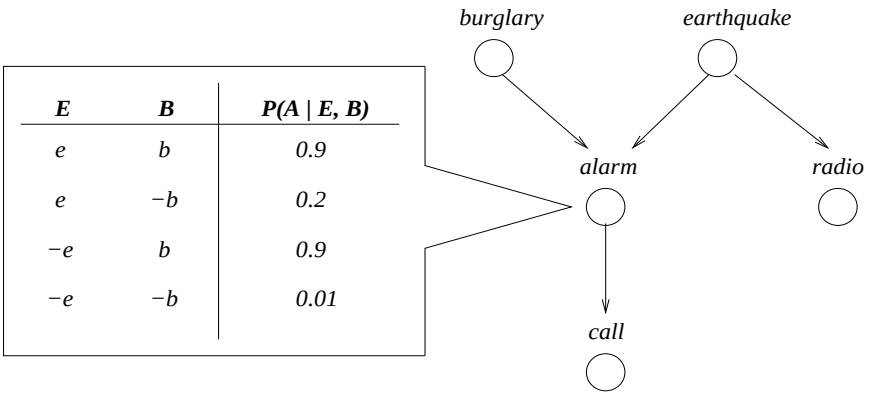

Figure 2.1 Example of the Bayesian Network

the literature. Different solutions to this problem appeared in different fields. The most representative ones are Probabilistic Logic Programs (PLPs) (43) from logic programming; Relational Bayesian Networks (RBNs) (20) from model theory, and Probabilistic Relational Models (PRMs) (29). In spite of their different backgrounds, they all seem to share the commonalities represented by Bayesian Logic Programs (BLPs) as shown in (25). This approach is a simplification and reformulation of PLPs.

Bayesian networks consist of two components: qualitative and quantitative. The qualitative part is represented by a directed acyclic graph (DAG) where each node represents a domain variable (usually called random variables), and each arc represents a probabilistic dependency between two variables. The probabilistic dependency is the quantitative part represented through a conditional probability distribution for each node. An example of a Bayesian network taken from (15) is shown in Figure 2.1.

A BN is a compact representation of probability distributions via conditional independence. A BN can compute the conditional probability for each value of a node given that values have been assigned to the other nodes. In this sense, BN classifiers can compute the conditional probability of a class value given the values of the other attributes. From the example in Figure 2.1, where the conditional probability table for alarm node is given, we can see that the alarm starting depends on the values of burglary and earthquake. For a more detailed description of BNs refer to (21).

There are several motivations to the use of BNs for relational data mining. The main advantage of using BNs is that they can naturally represent relationship among attributes and this is clearly needed in relational domains. At the same time, they can help decision making process under uncertainty.

The structural representation of BNs makes the relationships between attributes easy to understand

and can be modified by domain experts in order to obtain better predictive models or models with certain characteristics. One of the drawbacks of BNs is the fact that generally the domain of each of the random variables has to be finite. Thus, some discretisation algorithm must be used before learning the network and its parameters, which sometimes results in loosing important information.

2.2.1 Relational Bayesian Networks (20)

In RBNs the nodes are predicate symbols corresponding to every relation r of some vocabulary S, the values of these random variables are the possible interpretations of the predicates over some finite domain. The semantic of a RBN is a probability distribution over the relations of S.

The qualitative component of RBNs is represented by Probability Formulae (PFs) that employ combination functions as well as equality constraints. The PFs allow nested combination functions and not only specify the CPT but also the dependency structure.

2.2.2 Probabilistic Relational Models (16)

PRMs are an extension of BN to relational domain where the nodes of the network are the attributes of a relational database. The edges represent probabilistic dependences between attributes. The dependences between attributes are either true or false; they do not have certain probability of occurrence as in RBN. The semantic of a PRM is a probability distribution over the attributes and the relations that hold between them (16). Like traditional BNs, PRMs consist of the two components: the structure of the network and the parameters (conditional probability distribution) associated with it. For learning PRMs, (14; 15) extend the techniques for learning BNs to learn from structured data. The goal is to learn the two components of PRMs. The parameter estimation is easier task than learning the structure. The key function used to estimate the parameters in PRM is the likelihood function (i.e. the probability of the data given the model). The higher the value of this function, the better the model predicts the data. For structure learning task three components need to be defined: the hypothesis space (the structures that must be considered by the algorithm); a scoring function to evaluate each hypothesis relative to the data; and the search algorithm. The search algorithm is usually a heuristic, for example the greedy hill-climbing search.

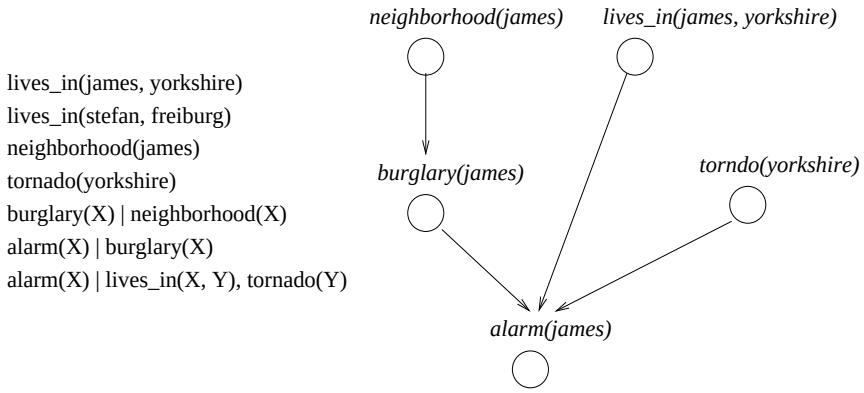

Figure 2.2 Example of the Bayesian Network

2.2.3 Bayesian Logic Programs

Bayesian Logic Programs were introduced in (25) as a simplification and reformulation of PLPs, but also as a common framework for the previously mentioned approaches.

In BLPs, the qualitative part is represented by a set of Bayesian definite clauses. The difference between these clauses and classical clauses is that each ground atom in a BLP represents a random variable (25). The set of random variables corresponds to the least Herbrand model of this logical program, i.e. the set of all ground atoms that are logically entailed by it. The parents of a certain random variable n are all facts directly influencing n. An example is given in Figure 2.2. For a determined query, it is not necessary to compute the least Herbrand model of the complete logic program; we need to consider only those variables that influence the query result. A query on BLP asks for the probabilities of the clause of taking any of the values in its domain. The query $?-alarm(james)$ asks for the probabilities of $alarm(james) = true$ and $alarm(james) = false$ (Figure 2.2).

The quantitative part of a BLP is represented by conditional probability tables and combining rules stored for each node or literal. A query can be answered using any Bayesian net inference engine.

Figure 2.2 shows how a Bayesian network can be obtained from a logical program. An example of a Bayesian logic program obtained from a Bayesian network is shown below.

$$burglary.$$

$$earthquake.$$

$$alarm:-burglary, earthquake.$$

$$radio : -earthquake.$$

$$call : -alarm.$$

This program corresponds to the Bayesian Network from Figure 2.1.

Algorithms for learning BLPs (the qualitative and the quantitative parts) have been proposed in (25) but have not been implemented yet. The authors of PRMs suggest that algorithms for learning their models can be adapted to BLPs too (16).

2.2.4 Combining First-Order Logic and Probability Theory

Other approaches for combining first-order logic and probability theory (not only Bayesian networks) have been proposed, one of the most interesting is Stochastic Logic Programs (SLPs) (40; 8; 41). Stochastic Logic Programs (SLPs) are a generalization of stochastic context-free grammars (32) and Markov Networks (or undirected Bayes Networks) (8). An SLP consists of a set of labeled clauses $p : C$ where $p \in [0, 1]$ is a first-order range-restricted definite clause (every variable in the head of C is the body too) and for each predicate symbol q the probability labels for all clauses with q in the head sum to 1. An SLP induces a probability distribution over the possible ground facts for a given predicate.

An algorithm for learning SLPs described in (41) consists of the two steps (as in Bayesian networks):

1. Construction of the logic program (LP) with uniform probability labels for each clause and near maximal posterior probability with respect to the examples, and

2. Parameter estimation to increase the posterior probability with respect to the examples.

For the step 1 Progol4.5 was used, implying that the task of learning the structure of an SLP is based on the ILP approaches. Step 2 uses a maximum likelihood approach, as most Bayesian approaches do, which is based on maximizing the posterior probability of the program.

2.3 Multi-Relational Data Mining

The idea of shifting from a pure logical search space to a search space consisting of database queries was initially proposed in (3). The implementation of this algorithm was shown in (36). That algorithm is given for relational algebra, although a translation to a database query language such as SQL is straightforward.

That work was motivated by the fact that most of the ILP systems were with a deductive database such as a Prolog database, rather than relational database. Therefore, systems that exploit relational database capabilities directly were needed.

A multi-relational data mining (MRDM) framework (26) was proposed later by the same group. It borrows techniques from ILP but the search space consists of SQL queries. This approach is the main focus of this thesis and discussed in details in the next chapter.

2.4 Other Recent Approaches and Extensions

When relational learning was introduced in the first chapter, the second way of transforming a multiple relations database into a single relation to which propositional learning algorithms can be applied to was the creation of new attributes in a central relation that summarizes or aggregate information from other tables in the relational database. Variants of this method have been used in systems such as LINUS (33) and DINUS (34). These systems are examples of a class of ILP methods that transform restricted ILP problems into attribute-value form through the technique called propositionalization and solve the transformed problem with a propositional learning algorithm. These two systems have some limitations that do not allow them to tackle some problems that can be present in relational databases such as non-determinate relationships (a record in one table can have more than one corresponding records in another table).

More recently, (31) introduced a new transformation-based ILP learner that can deal with non-determinate relationships, non-constrained literals and it is oriented to the business domain where the relational databases are often structurally simple, but large in size. This approach is called Rely on Aggregation, Sometimes (RELAGGS) and has been applied with significant success to learning problems from the business domain (31) and the biology domain (6). This feature construction method shows

that if good attributes are constructed by the propositionalization algorithm, a feature-based learner can outperform relational learning techniques.

Continuing in the same trend, (42) proposed the use of a simple propositional Bayesian classifier in an iterative way for relational domains. The approach keeps the relational structure and the objects are flattened when is required by the algorithm. Inferences made about an object can assist inferences about related objects; therefore, inferences made with high confidence about some objects are fed back into the database and used for posterior inferences about possible related objects. Although simple Bayesian classifier is an attribute-value learner, keeping the relational structure of the data helps to perform the feedback procedure into related objects.

In the framework of Probabilistic Relational Models, (45) proposes an extension to this language in order to make PRMs capable of cumulative learning. A cumulative learning agent learns and reasons as it interacts with the world. Bayesian networks provide a natural framework for this kind of learning. The dynamic technique proposed in (45) modifies the learned model to represent the reality in a more fair way along the time.

Approaches for mining structural data in form of graph have been proposed in (19; 17). In this framework, objects in the data correspond to vertices in the graph, and relationships between objects correspond to directed or undirected edges in the graph. The Subdue system looks for patterns embedded in graphs. Once a pattern (substructure) is found, it is added to the graph in order to simplify it by replacing instances of the substructure with the substructure itself.

CHAPTER 3. MULTI-RELATIONAL DATA MINING FRAMEWORK

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

Definition. *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 3.1. 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., 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.

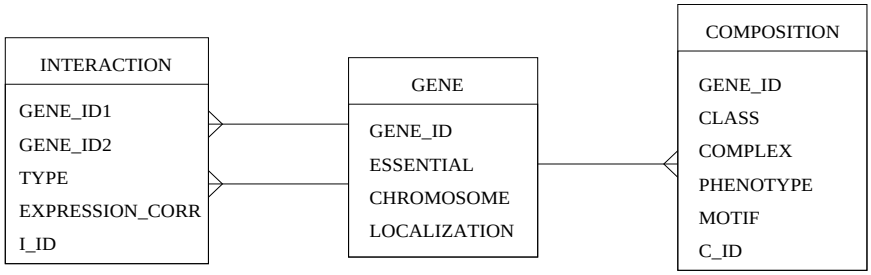

Figure 3.1 Example database

3.2 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” (26).

Definition. *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.* (26)

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, subpatterns 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.

3.3 Selection Graphs

Multi-relational patterns are expressed in a graphical language of selection graphs (27).

Definition. *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*

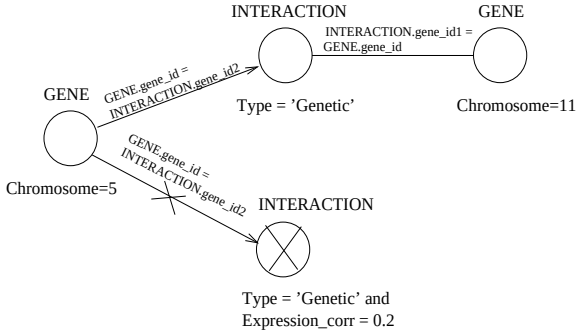

Figure 3.2 Selection graph corresponding to those GENEs which belong to chromosome number 5, that have at least one interaction of type 'Genetic' with a corresponding gene on chromosome number 11 but for which none of the interaction records are of the type 'Genetic' and expression_corr value 0.2.

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 3.1 is shown in Figure 3.2. 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 presence or absence of a cross in the node, representing the value *open* and *closed*, respectively. The value for e, in turn, is indicated by the presence (*present* value) or absence (*absent* 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 joint 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 3.2 is GENE.GENE_ID = INTERACTION.GENE_ID2.

```
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
```

Figure 3.3 Translation of selection graph into SQL query

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*.

Knobbe et al. (27) introduce the algorithm (Figure 3.3) for translating a selection graph into SQL query that returns the records in the target table covered by this selection graph, where 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 reseting 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

```
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)
```

Figure 3.4 SQL query corresponding to the selection graph in Figure 3.2

the table p in relation $j.a$. Using this procedure the graph in Figure 3.2 translates to the SQL statement shown in Figure 3.4.

3.4 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 (27) are given below. We will illustrate all the refinements on the selection graph from Figure 3.2. Note that a refinement can only be applied to the *open, 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 3.2 positive condition expression_corr=0.5 applied to the node INTERACTION results in the selection graph shown on Figure 3.5 a.

(b) Add negative condition (Figure 3.5 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.

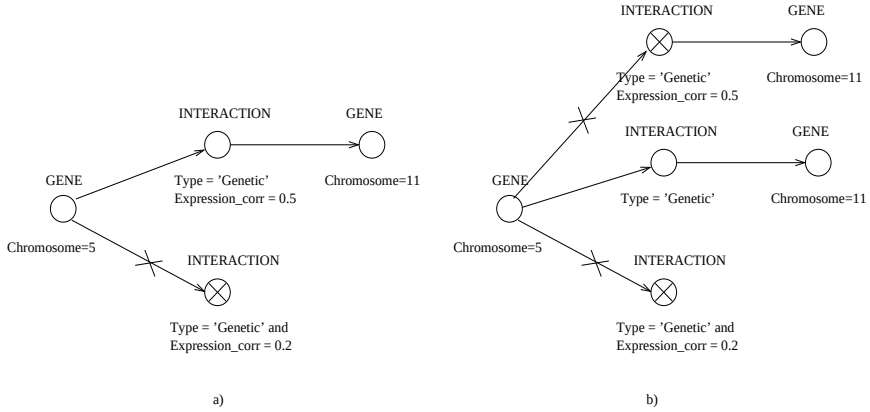

Figure 3.5 Complement refinements for adding condition expression_corr=0.5 to the node INTERACTION in the selection graph from Figure 3.2: a) positive condition, b) negative condition

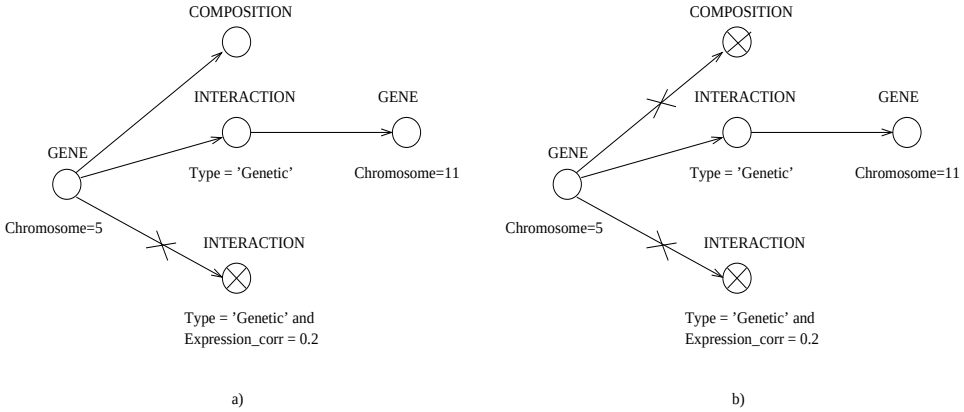

Figure 3.6 Complement refinements for adding edge from GENE node to COMPOSITION node to selection graph from Figure 3.2: a) adding present edge and open node, b) adding absent edge and closed node

(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 3.2 adding edge from GENE node to COMPOSITION node results in the selection graph shown in (Figure 3.6 a).

(d) Add absent edge and closed node (Figure 3.6 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 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.

CHAPTER 4. MDRTL-2: AN EFIICIENT MULTI-RELATIONAL DECISION TREE LEARNING ALGORITHM

4.1 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 which we describe below is based on MRDTL proposed by Leiva (36) which in turn is based on the algorithm described by (27) and the logical decision tree induction algorithm called TILDE proposed by (2). 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 4.1.

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 Chapter 3 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 doesn't result in further split of the

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)
```

Figure 4.1 MRDTL algorithm

data, multi-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 (48). Splits based on numerical attributes are handled using a technique similar to that of C4.5 algorithm (48) with modifications proposed in (13), (47).

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 4.2 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:

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

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_command_list

Figure 4.2 SQL query that returns counts of the type $count(c_i, R_{v_j}(S))$ for each of the possible values $c_i \in$ DOM(T_0.target_attribute) and $v_j \in$ DOM($X.A$).

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. The goal is to calculate entropies associated with the split based on these two 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 4.3 returns a list of the desired counts: $count(c_i, R_e(S))$ for each possible value $c_i \in$ DOM(T_0.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 $e.a$

Figure 4.3 SQL query returning counts of the type $count(c_i, R_e(S))$ for each possible value $c_i \in$ DOM(T_0.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 4.2, 4.3) needed to examine the candidate refinements of the corresponding selection graph. Consequently, the straightforward implementation of MRDTL as described in (36) 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 4.2 and 4.3 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 4.2 and 4.3 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 4.4 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 + ')'
```

Figure 4.4 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 4.5, and for the "add edge" refinements as shown in Figure 4.6.

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

Figure 4.5 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

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 $e.a$

Figure 4.6 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

It is easy to see that now the number of tables that needs to be joined is not more than 3, whereas the number of tables needed to be joined in Figures 4.2 and 4.3 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 algorithm from Figure 4.4 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 4.7.

The simple modifications described above make MRDTL-2 significantly faster to execute compared to MRDTL. This is confirmed by experimental results presented in Chapter 5.

4.2 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:

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 closed 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 + ')'
```

Figure 4.7 Algorithm for generating SQL query corresponding to sufficient table $I_{R(S)}$

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

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.

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.

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

4.3 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. Instances that do not match the selection graphs associated with any of the leaf nodes in the tree are assigned unknown label and are counted as incorrectly classified when evaluating the accuracy of the tree on test data.

CHAPTER 5. EXPERIMENTAL RESULTS

Our experiments focused on three data sets - the mutagenesis database which has been widely used in Inductive Logic Programming (ILP) research (http://www.mlnet.org/), the data for prediction of protein/gene localization and function from KDD Cup 2001 (http://www.cs.wisc.edu/ dpage/kddcup2001/) and the data for predicting thrombosis from PKDD 2001 Discovery Challenge (http://www.uncc.edu/knowledgediscovery). We compared the results we obtained using MRDTL-2 algorithm with those reported in the literature for the same datasets.

5.1 Mutagenesis Data Set

This database, available from the Machine Learning Network (http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/mutagenesis.html), is one of the most widely used databases in ILP research. The original format of this database is Prolog syntax. In order to use this dataset with MRDTL algorithm we needed to translate it to the relational format. The entity-relation diagram for the part of the Mutagenesis database used in this work is shown on Figure 5.1. 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. The regression friendly subset consists of 4893 atoms and 5243 bonds. There are 1001 atoms and 1066 bonds in the regression unfriendly subset.

The mutagenesis database contains descriptions of molecules. The characteristic to be predicted is their mutagenic activity (ability to cause DNA to mutate) represented by attribute label in molecule table. This problem comes from the field of organic chemistry and the compounds analyzed are nitroaromatics. These compounds occur in automobile exhaust fumes and sometimes are intermediates in the synthesis of thousands of industrial compounds. High mutagenic level has been found to be

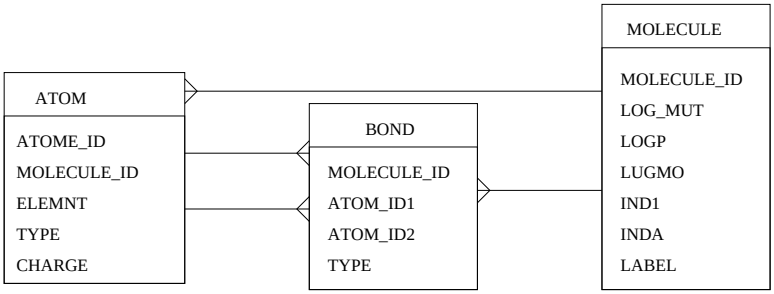

Figure 5.1 Schema of the mutagenesis database

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

Table 5.1 Experimental results for mutagenesis data

carcinogenic.

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 crossvalidation are shown in the Table 5.1.

There are five levels of background knowledge, B_0, B_1, B_2, B_3, b_4 for this dataset recognized in the literature (50). Higher levels provide richer descriptions of the instances. 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 crossvalidation are shown in the Table 5.1.

5.2 KDD Cup 2001 Data Set

We also considered the two tasks from the KDD Cup 2001 data mining competition (6): prediction of gene/protein function and localization.

We normalized the data given in the task which resulted in the schema shown in Table 3.1. The resulting database consists of 3 tables, GENE, INTERACTION and COMPOSITION, where GENE

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

Table 5.2 Experimental results for gene/protein localization 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 5.3 Experimental results for gene/protein function prediction task

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 5.2 and 5.3, and not using it (considering each missing values as a special value ”missing”).

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 5.3 summarizes the results we got for predicting the function of the protein.

5.3 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, ANA_PATTERN and THROMBOSIS) which all have a

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

Table 5.4 Experimental results for Thrombosis data set

dataset	MRDTL accuracy	best reported accuracy	reference
mutagenesis	87.5 %	86 %	(50)
localization	76.11 %	72.1 %	(6)
function	91.44 %	93.6 %	(6)
thrombosis	98.1 %	99.28 %	(7)

Table 5.5 Comparison of MRDTL-2 performance with the best-known reported results

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 crossvalidation are shown in Table 5.4.

5.4 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.

CHAPTER 6. 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 (46). 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 (2; 26; 14; 29; 31; 16; 25; 45; 12; 9; 20; 23). Learning classifiers from relational databases has also been a focus of KDD Cup Competition (held in conjunction with the ACM SIGMOD Conference on Knowledge Discovery and Data Mining (KDD 2001)) and the PKDD 2001 Discovery Challenge (held in conjunction with the Pacific Conference on Knowledge Discovery and Data Mining). Against this background, this thesis describes the design and implementation of MRDTL-2 - an algorithm for learning decision tree classifiers from relational databases. MRDTL-2 is based on the framework for multi-relational data mining originally proposed by Knobbe et al (26). MRDTL-2 extends an MRDTL, an algorithm for learning decision tree classifiers described by Leiva (36). 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 reduce the running time of the algorithm by one to two orders of magnitude, thereby making it possible to apply multi-relational decision tree learning algorithms on far more complex data mining tasks than was previously possible. 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. (26) which rely on the use of selection graphs for specifying relational database queries needed to extract the statistics needed constructing predictive models.

(b) Inability to handle missing attribute values: MRDTL-2 includes a simple and computationally efficient technique which uses Naive Bayes classifiers for 'filling in' missing attribute values. For multi-relational databases encountered in many real-world applications of data mining, it is not uncommon to have one or more missing attribute values in a significant fraction of the data set. Hence, MRDTL-2 significantly enhances the applicability of multi-relational decision tree learning algorithms to such classification tasks.

Our experiments with several classification tasks drawn from KDD Cup 2001 (6) 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 5.5)

(c) compares favorably with feature-based learners that are based on clever propositionalization methods (31)

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 (based on the recently developed propositional decision tree counterparts of such algorithms (52))

(d) Development of variants of MRDTL that can learn from heterogeneous, distributed, autonomous data sources based on recently developed techniques for distributed learning (5; 4) and ontology-based data integration (46)

(e) More extensive experimental evaluation of MRDTL-2 on real-world data sets, including data sets that are encountered in computational biology applications such as protein function prediction (51) and prediction of protein-protein interactions.

(f) Incorporation of more sophisticated methods for evaluation of MRDTL-2, based on recent work of Jensen et al. (22)

ACKNOWLEDGEMENTS

I would like to express my thanks to those who helped me with the various aspects of conducting this research. First, to Dr. Vasant Honavar for his guidance, help and support throughout this research; also to my colleges at the Artificial Intelligence Research Laboratory at Iowa State University for the helpful discussions; and to my committee members, Dr. Drena Dobbs and Dr. Yan-Bin Jia, for their help in completing this work.

I am very grateful to my family and friends who always support me and believe in me.

This research was supported in part by an Information Technology Research (ITR) grant from the National Science Foundation (NSF IIS 0219699) to Vasant Honavar and a research assistantship funded by the Iowa State University Graduate College.

BIBLIOGRAPHY

[1] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H, and Verkamo, A. I.: Fast discovery of association rules. In U. Fayyad, G. Piatesky-Shapiro, R. Smyth, and R. Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, MIT Press, (1996)

[2] Hendrik Blockeel: Top-down induction of first order logical decision trees. Department of Computer Science, Katholieke Universiteit Leuven (1998)

[3] Blockeel, H., and De Raedt, L.: Relational Knowledge Discovery in Databases. In: Proceedings of the 6th International Workshop on Inductive Logic Programming, volume 1314 of Lecture Notes in Artificial Intelligence, Springer- Verlag (1997)

[4] 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.

[5] Caragea, D. and 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)

[6] Cheng, J. and Krogel, M. and Sese, J. and Hatzis C. and Morishita, S. and 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)

[7] Coursac,I. - Duteil,N. - Lucas,N.: pKDD 2001 Discovery Challenge - Medical Domain. In: PKDD Discovery Challenge 2001, vol. 3, issue 2 (2002)

[8] Cussens, J. : Loglinear models for first-order probabilistic reasoning. In Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence, Kaufmann (1999)

[9] 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)

[10] De Raedt, L.: Logical settings for concept-learning. . In: Artificial Intelligence, volume 95, issue 1 (1997)

[11] De Raedt, L., Blockeel, H., Dehaspe, L., and Van Laer, W.: Three companions for data mining in first order logic. In: Dzeroski, S. and Lavrac, N. editors. Relational Data Mining. Springer-Verlag (2001)

[12] Dzeroski, S. and Lavrac, N: Relational Data Mining. Springer-Verlag (2001)

[13] Fayyad, U. M. and Irani, K. B: On the handling of continuous-valued attributes in decision tree generation. In: Machine Learning, vol.8 (1992)

[14] Friedman, N. and Getoor, L. and Koller, D. and and Pfeffer: Learning probabilistic relational models. In: Proceedings of the 6th International Joint Conference on Artificial Intelligence (1999)

[15] Friedman, N., and Koller, D.: Tutorial: Learning Bayesian Networks from Data. Neural Information Processing Systems: Natural and Synthetic (NIPS 2001) http://www-2.cs.cmu.edu/Web/Groups/NIPS/NIPS2001/nips2001.html (date accessed: 07/10/2003)

[16] Getoor, L.: Multi-relational data mining using probabilistic relational models: research summary. In: Proceedings of the First Workshop in Multi-relational Data Mining (2001)

[17] Gonzalez, J., Jonyer, I., Holder, L. B., and Cook, D. J.: Efficient Mining of Graph-Based Data. In: Proceedings of AAAI 2000 Workshop on Learning Statistical Models from Relational Data, AAAI Press (2000)

[18] Grzymala-Busse, J. W., and Hu, M.: A comparison of several approaches to missing attribute values in data mining. In: Proceedings of the Second International Conference on Rough Sets

and Current Trends in Computing, RSCTC 2000, volume 2005 of Lecture Notes in Computer Science, Springer (2001)

[19] Holder, L. B., and Cook, D. J.: Graph-based data mining. In: IEEE Intelligent Systems 15 (2000)

[20] Jaeger, M: Relational Bayesian networks. In: Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-1997)

[21] Jensen, F. V. : Bayesian Networks and Decision Graphs. Springer-Verlag (2001)

[22] 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)

[23] Karalic and Bratko: First order regression. In: Machine Learning 26, vol. 1997

[24] Kietz, J-U. and Wrobel, S.: Controlling the complexity of learning in logic through syntactic and task-oriented models. In: S. Muggleton, editor, Inductive logic programming, Academic Press (1992)

[25] 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)

[26] Knobbe, J. and Blockeel, H. and Siebes, A. and Van der Wallen D.: Multi-relational Data Mining. In: Proceedings of Benelearn (1999)

[27] Knobbe, J. and Blockeel, H. and 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

[28] Knobbe, A. J., Siebes, A., Blockeel, H., and Van der Wallen, D. : Multi-relational data mining using UML for ILP. In: Proceedings of the First Workshop in Multi-relational Data Mining, 2001.

[29] Koller, D: Probabilistic Relational Models. In: Proceedings of 9th International Workshop on Inductive Logic Programming (ILP-99)

[30] Kramer, S.: Structural regression trees. In: Proceedings of the 13th National Conference on Artificial Intelligence (1996)

[31] 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)

[32] Lari, K. and Young, S. J.: The estimation of stochastic context-free grammars using the inside-outside algorithm. Computer Speech and Language, volume 4 (1990)

[33] Lavrac, N., Dzeroski, S., and Grobelnik, M.: Learning nonrecursive definitions of relations with LINUS. In Proceedings of the 5th European Working Session on Learning, Springer (1991)

[34] Lavrac, N. and Dzeroski, S.: Techniques and Applications. Ellis Horwood (1994)

[35] Lavrac, N.: Introduction to Inductive Logic Programming. Computer Science course ComS70301: Learning from Structured Data, University of Bristol, Department of Computer Science. http://www.cs.bris.ac.uk/Teaching/Resources/COMSM0301/index.html (date accessed: 07/10/2003)

[36] Hector Ariel Leiva: A multi-relational decision tree learning algorithm. M.S. thesis. Department of Computer Science. Iowa State University (2002)

[37] Lindner, G., and Morik, K.: Coupling a relational learning algorithm with a database system. In: Workshop Notes of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases (1995)

[38] Mitchell, T.: Machine Learning. McGraw Hill, 1997

[39] Muggleton, S. H.: Inverse entailment and progol. New Generation Computing, 1995

[40] Muggleton, S. H.: Stochastic Logic Programs. In Proceedings of the 5th International Workshop on Inductive Logic Programming (1996)

[41] Muggleton, S. H.: Learning Stochastic Logic Programs. In Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data, AAAI Press (2000)

[42] Neville, J., and Jensen, D.: Iterative Classification in Relational Data. AAAI 2000 Workshop on Learning Statistical Models from Relational Data (2000)

[43] Ngo, L., and Haddawy, P.: Answering queries form contextsensitive probabilistic knowledge bases. Theoretical Computer Science (1997)

[44] Pearl, J.: Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, 2nd edition (1991)

[45] Pfeffer, A.: A Bayesian Language for Cumulative Learning. In: Proceedings of AAAI 2000 Workshop on Learning Statistical Models from Relational Data, AAAI Press (2000)

[46] 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)

[47] Quinlan, R.: Improved Use of Continuous Attributes in C4.5. In: Journal of Artificial Intelligence Research, vol.4 (1996)

[48] Quinlan, R.: C4.5: Programs for Machine Learning. In: San Mateo: Morgan Kaufmann (1993)

[49] Quinlan, R.: FOIL: A midterm report. In: Proceedings of the 6th European Conference on Machine Learning, volume 667 of Lecture Notes in Artificial Intelligence, Springer-Verlag (1993)

[50] 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.

[51] 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.

[52] 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.