An Efficient Multi-relational Naïve Bayesian Classifier Based on Semantic Relationship Graph

Hongyan Liu[1]
Department of Management Science and Engineering
Tsinghua University
Beijing, China, 100084
hyanliu@uiuc.edu

Xiaoxin Yin
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801
Xyin1@uiuc.edu

Jiawei Han
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801
hanj@uiuc.edu

ABSTRACT
Classification is one of the most popular data mining tasks with a wide range of applications, and lots of algorithms have been proposed to build accurate and scalable classifiers. Most of these algorithms only take a single table as input, whereas in the real world most data are stored in multiple tables and managed by relational database systems. As transferring data from multiple tables into a single one usually causes many problems, development of multi-relational classification algorithms becomes important and attracts many researchers' interests. Existing works about extending Naïve Bayes to deal with multi-relational data either have to transform data stored in tables to main-memory Prolog facts, or limit the search space to only a small subset of real world applications. In this work, we aim at solving these problems and building an efficient, accurate Naïve Bayesian classifier to deal with data in multiple tables directly. We propose an algorithm named *Graph-NB*, which upgrades Naïve Bayesian classifier to deal with multiple tables directly. In order to take advantage of linkage relationships among tables, and treat different tables linked to the target table differently, a semantic relationship graph is developed to describe the relationship and to avoid unnecessary joins. Furthermore, to improve accuracy, a pruning strategy is given to simplify the graph to avoid examining too many weakly linked tables. Experimental study on both real-world and synthetic databases shows its high efficiency and good accuracy.

Categories and Subject Descriptors
H.2.8 [**Database Application**]: Data Mining

General Terms
Algorithm, Management, Performance, Design, Experimentation.

Keywords
Classification, Naïve Bayes, Data Mining

This paper appears in the Proceedings of the Fourth International Workshop on Multi-Relational Data Mining (MRDM-2005), August 21, 2005, Chicago. The proceedings were edited by Saso Dzeroski and Hendrik Blockeel. The paper is published here with the permission of the authors, who retain the copyright of this material.

[1] This work was done while the author was visiting University of Illinois at Urbana-Champaign

1. INTRODUCTION
Most real-world structured data are stored in multiple relations. Multi-relational data mining aims at discovering interesting knowledge directly from multiple tables without joining data of multiple tables into a single table explicitly. It has attracted the interests of many researchers, and has been successfully applied in many application areas, such as marketing, sales, finance, fraud detection, and natural sciences.

Classification is one of most popular tasks in data mining, and therefore one has witnessed numerous studies targeting on solving multi-relational classification problems. Generally, there are basically two major directions. One is to use conventional classification algorithms to deal with multi-relational data. In order to do that, various methods are proposed to convert multiple relations into one by flattening and feature construction, which is known as propositionalization. Most of the propositionalization methods are heuristic, and the representation change is incomplete. So information is lost in the process [1]. The other direction is to update existing classification algorithms to deal with multi-relational data directly [2, 3, 4, 5, 6, 7, 8, 15, 9, 23]. Most approaches of this direction are either inductive logic programming (ILP) approaches or probabilistic relational learning approaches. The above approaches have shown their high classification accuracy in multi-relational environments. Unfortunately, due to the complex structure of multi-relational data, it is a time-consuming task to explore the hypothesis space in relational databases for useful attributes or relational structural neighbors, which makes these algorithms quite expensive in terms of runtime.

Naïve Bayesian [16, 17, 18] is a good classification method. It is simple to train, easy to understand, and performs pretty well for real applications. In this paper, we extend Naïve Bayes classification to deal with multi-relational data directly. There are already some works [19, 12, 13, 23, 7] about it. We have the following observations about them. First, most of such works use the ILP-based method. They are usually two-step based methods. In the first step, a set of first-order features or rules are extracted, and then in the second step, these features or rules are combined to compute probabilities according to Naïve Bayesian formula. Although these methods can deal with multi-relational data, most of them cannot do it directly. They usually work on a set of main-memory Prolog facts, while in real application data are

stored in tables in database. Transforming tuples in tables to ground facts has many disadvantages [23]. Second, some methods can deal with data in tables directly, but they limit the search space by only allowing one type of join paths. For example, while linking tables to target table, only tables along primary-key to foreign-key path are considered [23]. Tables in a foreign-key to primary-key path or foreign-key and primary-key mixed path are neglected. This constraint limits their application to real world situation. Third, most of these methods consider all tables linked to the target table equally important. In real applications, databases usually have complicated schemas. A database may contain many tables, some of which may be almost irrelevant to the classification task, and could not improve the classification accuracy. Therefore, how to deal with data in tables directly, how to describe the relationship between tables, how to treat tables differently are some major challenges to make a good multi-relational classifier.

In order to solve these problems, we propose an algorithm called *Graph_NB* to build an accurate classifier efficiently. We make the following contributions in this paper.

- We upgrade conventional Naïve Bayes method to deal with multi-relational classification task. We call this method *multi-relational Naïve Bayes*. With this method, data stored in multiple tables can be mined directly, and no transformation is needed.
- In order to guide the search within the hypothesis space (called *relation space*) in relational database, we define a *semantic relationship graph* (SRG) to describe relationship between tables, and also to avoid unnecessary joins between tables. We will give the formal definition of this graph in the next section. SRG not only offers a method to navigate among all tables, but also provides a way to express the semantic relationship between tables. Also, it makes capable to treat the tables linked to target table differently according to the relationship presented in the graph.
- In order to ignore those weakly linked tables, instead of controlling the rule length by user-defined parameters, we propose a pruning strategy named "cutting off" strategy to optimize the semantic relationship graph so that we can stop further futile mining to prevent introducing weakly linked tables. Experimental results show that this pruning strategy is effective for improving accuracy.
- We implement algorithm *Graph_NB* for training multi-relations and algorithm *Classify_NB* for classifying unseen objects. Comprehensive experimental study shows that it achieves good accuracy and high efficiency.

The rest of the paper is organized as follows. Section 2 presents the definition and case study of semantic relationship graph. Section 3 describes how to extend Naïve Bayes to deal with multiple relations directly. The algorithm and pruning strategy are introduced in section 4, and experiments results are presented in section 5. Finally, related work is discussed in section 6 and we conclude this study in section 7.

2. SEMANTIC RELATIONSHIP GRAPH

For a classification task in a multi-relational database, there is usually one table containing the class label attribute. We call this table as *target table*, and call the class label attribute as *target attribute*. Apart from the target table, there are usually many other tables linked to the target table directly or indirectly through arbitrarily long chains of joins. In order to represent this kind of relationship between tables, we propose to use a graph, which is called a semantic relationship graph.

Definition 2.1 (**Semantic Relationship Graph**) Semantic Relationship Graph is a directed acyclic graph SRG (*V*, *E*, *W*), where *V* is a set of vertices, each of which corresponding to a table in the database. *E* is a set of directed edges, and an edge (*v*, *w*) means table *w* can be linked to table *v* by directly joining these two tables. W is a set of attributes, each of which links two tables. We call this kind of attribute link attribute.

Each edge of the semantic relationship graph represents one of the following two relationships between tables *v* and *w*:

(1) *Primary-key to foreign-key relationship*, indicating that table *w* contains foreign-key referring to primary-key in table *v*.

(2) *Foreign-key to primary-key relationship*, indicating that table *v* contains foreign-key referring to primary-key in table *w*.

The reason we define a directed graph instead of undirected graph is that we need to start from the target table and link other tables with the target table step by step. Suppose table *t* is the target table, and there is an edge pointing to *v* from *t* and an edge pointing to *w* from *v*. After joining table *v* with *t*, and join *w* with *v*, usually we do not need to join *w* and *v* by same link attribute in the same path again. For example, suppose table *student* contains information of each student, table *professor* contains information of each professor. Between these two tables, there is a semantic relationship by meaning of supervision, and assume there is no other relations between them in the context of the application. In this case, we can join table *professor* with table *student.* If they are only involved in one path, then this kind of join only needs to be done once. If they are involved in more than one path but with the same semantic relationship, then this join could be done many times. With directed graph, we can prevent unnecessary iteration for one specific relationship in one path.

When there are more than one relationship between two tables or there is a relationship between tuples of one table (i,e, there is a self-join on that table), there might be a cycle in the relationship graph. In this case, to prevent the iteration to go on too deep, we can limit the number of iterations, and transform the cyclic graph into acyclic graph. We argue that this kind of links become weaker as more iterations are done. Taking table *student* and *professor* as example, suppose there is another relationship between them by meaning of friend, which means that a student is linked to a professor if the student is a friend of the professor. Now there is a cycle between these two tables. If two is the maximum number of iterations, then we can make two additional tables named *student1* and *professor1* to duplicate these two tables, and link them by directed edge.

We can also relax the constraints of semantic relationship graph by allowing the existence of cycle. If so, in order to avoid the iteration doing too many times, we can also set a parameter to control the iteration times. In the following sections, we only regard SRG as an acyclic graph.

Semantic relationship graph is similar to ER diagram, which is usually used to model data during conceptual design of database [9]. Some commercial database systems also provide graphic interface to define the foreign-key and primary-key relationships between tables. Semantic relationship graph can be automatically generated from these diagrams. Following are two cases about semantic relationship graph.

2.1 Case Study: Financial Database

Figure 1 shows the foreign-key primary-key relationship diagram of a financial database from PKDD CUP99. Among these tables, table *loan* is the target table, and attribute *status* is the target attribute. Basically, each edge between these tables is a primary key and foreign key relationship. For example, attribute *account_id* is the primary key of table *Account*, and it is a foreign key of target table *loan*. By this kind of relationship, we can easily join them together. Based on this relationship diagram, each edge can lead to many iterative join operations, which is usually not very meaningful in semantics. So we believe semantic relationship graph could be a better choice to guide the join operations between tables. Figure 2 shows a Semantic Relationship Graph of this database.

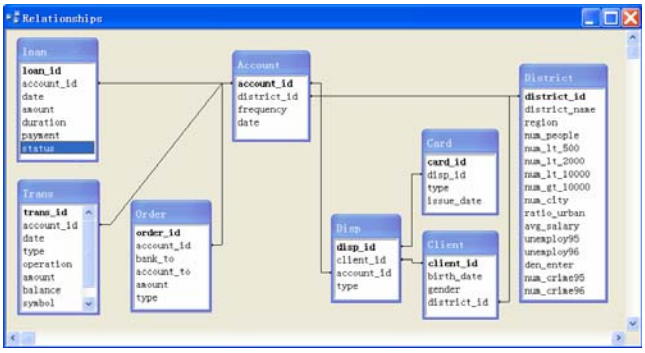

Fig. 1. Relationship of the financial DB from PKDD CUP 99

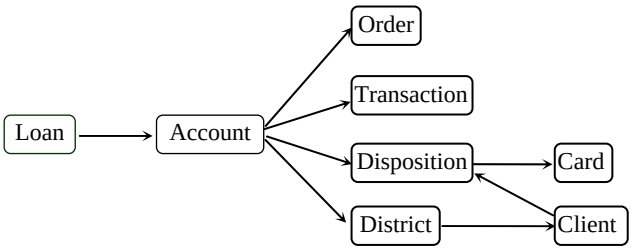

Fig. 2. Semantic Relationship Graph for financial database

In Figure 2, each vertex is a table. Tables are linked together by directed edges, and the attribute names used to join the two tables are neglected for simplicity. For example, attribute *account_id* can be used to join table *Account* with table *Loan*. Here we assume these two tables use the same attribute name to represent the link attribute. In real life applications, sometimes attributes having the same semantic meaning may have different names in different tables. This problem can be easily solved by doing some mappings.

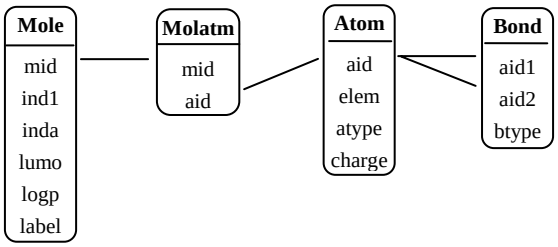

Fig. 3. Relationship of Mutagenesis database

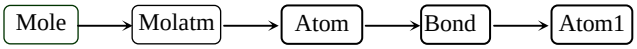

Fig. 4. Semantic Relationship Graph for Mutagenesis Database

Note that between two tables, there may be several edges, each of them having a different link attribute with different semantic meaning. Also, for a specific application, one could create several different semantic relationship graphs. By doing this, one can easily put one's domain knowledge into the graph. For example, if according to the domain knowledge, table *Client* should be linked to target table by table *Disposition* other than *District*, one can link *Client* to *Disposition* instead of linking *District* to *Client*.

2.2 Case Study: Mutagenesis Database

Figure 3 shows the foreign-key primary-key relationship diagram of a Mutagenesis database, which is a frequently used ILP benchmark. There are four tables in this database. Table *Mole* is the target table, and attribute *label* is the target attribute.

One can see from Figure 3 that there is a cycle between table *Atom* and *Bond*. In order to create a semantic relationship graph for this database, we can do decyclization by making duplications for table *Atom* and *Bond*. In Figure 4, it is done by duplication of *Atom* once. However, several times of duplication is also allowed. This kind of duplication can be done virtually and automatically by doing some mapping. As mentioned earlier, this kind of cycle can also be dealt with by using a parameter to limit the times of iteration.

3. Multi-relational Naïve Bayes

Naïve Bayes is a statistical classification method. It is simple but usually performs well even compared with many complicated methods. Originally it only deals with the data in one single table. We extend it to make it capable of dealing with multiple tables directly.

For a single table learning task, given an object x which can be described by a set of attribute values: $x = (x_1, x_2, \ldots, x_n)$, we assign it the most probable class label $c_i \in C$ as follows:

$$c_{MAP} = \operatorname{argmax}_{c_i \in C} P(c_i \mid x) = \operatorname{argmax}_{c_i \in C} \prod_{j=1}^{n} P(c_i \mid x_1, x_2, \ldots, x_n) \quad (1)$$

According to Bayes rule and the assumption that the values of different attributes are independent given the class label, we can get the following Naïve Bayesian formula:

$$c_{MAP} = \text{argmax}_{c_i \in C} P(x_1, x_2, \ldots, x_n \mid c_i)\, P(c_i) = \text{argmax}_{c_i \in C} \prod_{j=1}^{n} P(x_j \mid c_i) P(c_i) \quad (2)$$

For a multi-relational learning task, in order to deal with multiple tables by Naïve Bayes, there are two types of methods. One is converting multiple tables into one table, and the other is dealing with each table separately. In this paper, we prefer the latter. For the former method, a simple method is to join all tables together, which will be shown in Example 1. Now we need to extend the formula 2 to deal with multiple tables.

Suppose t is the target table, and s is another table that can be joined with table t. Table t has n attributes and table s has m attributes. For a tuple x in table t: $x=(x_1, x_2, \ldots, x_n)$, there are l tuples in table s which can be joined with x. These l tuples are $(y_{k1}, y_{k2}, .y_{kl})$, where each tuple y_{ki} is represented by m values: $y_{ki} = (y_{ki1}, y_{ki2}, \ldots, y_{kim})$. Then, the class label of tuple x can be assigned according to the following formula:

$$c_{MAP} = \text{argmax}_{c_i \in C} P(c_i \mid x) = \text{argmax}_{c_i \in C} \prod_{j=1}^{n} P(c_i \mid x_1, \ldots, x_n, y_{k1}, \ldots, y_{kim}, \ldots, y_{k1}, \ldots, y_{km}) \quad (3)$$

Besides the original assumption about the independence of attribute values of different attributes, we further assume that given the class label, each tuple $(y_{k1}, y_{k2}, \ldots, y_{km})$ from table s that can join with t is independent. Then, formula (3) becomes formula (4).

$$c_{MAP} = \text{argmax}_{c_i \in C} P(x, y_{k1}, \ldots, y_{ki} \mid c_i)\, P(c_i) = \text{argmax}_{c_i \in C} \prod_{j=1}^{n} P(x_j \mid c_i) \prod_{q=k1}^{kl} \prod_{j=1}^{m} P(y_{qj} \mid c_i) P(c_i) \quad (4)$$

$$\frac{\prod_{j=1}^{n} P(x_j \mid c_i) \prod_{q=k1}^{kl} \prod_{j=1}^{m} P(y_{qj} \mid c_i) P(c_i)}{\sum_{ci \in C} \prod_{j=1}^{n} P(x_j \mid c_i) \prod_{q=k1}^{kl} \prod_{j=1}^{m} P(y_{qj} \mid c_i) P(c_i)} \quad (5)$$

Note that in formula (4), although each computation result ($\prod_{j=1}^{n} P(x_j \mid c_i) \prod_{q=k1}^{kl} \prod_{j=1}^{m} P(y_{qj} \mid c_i) P(c_i)$) corresponding to each class label C_i is not the probability of C_i, and they do not sum to one, by comparing them we can know which class label is more likely for current tuple x. We also can make them sum to 1 by normalizing them using formula (5). Although the number of tuples linked to each target tuple is not fixed, whenever there is a tuple linked to it, the probability of each attribute value of this tuple with respect to each class label will be considered. In the end, we use the ratio of likelihood of different classes to decide the class label of each target tuple. Thus it does matter if different target tuples are joinable with different numbers of tuples in other relations. When the number of linked tuple becomes big, the ratio may become very small, and this can be easily solved by doing logarithm conversion or by multiplying a positive number.

Example 1 (Naïve Bayes and Multi-relational Naïve Bayes) Tables 1, 2 and 3 are three tables named *Researcher*, *University* and *Paper* respectively, among which *Researcher* is the target table. Attribute *status* in *Researcher* is the target attribute. The primary key and foreign key relationship is clear in this context. Table 4 is the universal relation obtained by joining these three tables together.

Table 1. Researcher

R#	sex	age	U#	status
r1	F	48	u1	Y
r2	M	51	u2	N
r3	M	62	u3	Y
r4	F	36	u4	N

Table 2. University

U#	rank	history
u1	2	⩾100
u2	2	⩾100
u3	1	<100
u4	2	<100

Table 3. Paper

P#	type	level	R#
p1	conference	1	r1
p2	conference	2	r2
p3	conference	3	r3
p4	journal	1	r1
p5	journal	1	r4
p6	journal	2	r2
p7	conference	2	

Table 4. Universal Relation

R#	sex	age	P#	type	level	U#	rank	history	status
r1	F	48	p1	conference	1	u1	2	⩾100	Y
r2	M	51	p2	conference	2	u2	2	⩾100	N
r3	M	62	p3	conference	3	u3	1	<100	Y
r1	F	48	p4	journal	1	u1	2	⩾100	Y
r4	F	36	p5	journal	1	u4	2	<100	N
r2	M	51	p6	journal	2	u2	2	⩾100	N

Now suppose there is an unseen example u = (*r5, F, 30, u2, p7*), we can classify it by two methods. The first is taking Table 4 as training table, then, we can compute probabilities with respect to *P(Y|u)* and *P(N|u)* as follows:

$$\prod_{j=1}^{n} P(x_j|Y)\prod_{q=k1}^{kl}\prod_{j=1}^{m} P(y_{qj}|Y)P(Y) = \frac{2}{3}\times\frac{2}{3}\times\frac{2}{3}\times\frac{2}{3}\times\frac{2}{3}\times\frac{2}{4}\times\frac{3}{6}$$

$$\prod_{j=1}^{n} P(x_j|N)\prod_{q=k1}^{kl}\prod_{j=1}^{m} P(y_{qj}|N)P(N) = \frac{1}{3}\times\frac{1}{3}\times\frac{3}{3}\times\frac{2}{3}\times\frac{2}{3}\times\frac{1}{2}\times\frac{3}{6}$$

Note that in the above computation, the first six figures are probabilities regarding to six attributes, sex, age, rank, history, type and level. As the first one is greater than the second, its class label will be assigned Y.

The second method is to take Tables 1, 2 and 3 as training data individually, and compute the probabilities according to formula (4) as follows.

$$\prod_{j=1}^{n} P(x_j|Y)\prod_{q=k1}^{kl}\prod_{j=1}^{m} P(y_{qj}|Y)P(Y) = \frac{1}{2}\times\frac{1}{2}\times\frac{1}{2}\times\frac{1}{2}\times\frac{2}{3}\times\frac{2}{4}\times\frac{2}{4}$$

$$\prod_{j=1}^{n} P(x_j|N)\prod_{q=k1}^{kl}\prod_{j=1}^{m} P(y_{qj}|N)P(N) = \frac{1}{2}\times\frac{1}{2}\times\frac{2}{2}\times\frac{1}{2}\times\frac{2}{3}\times\frac{2}{4}\times\frac{2}{4}$$

The class label information of these tables can be obtained by tuple ID propogation [14]. In this case *u* should be classified as class label *N*. Now a question arises about which of the two different answers is correct. In the first method, tuples in table *Researcher* and *University* are blown up after joining with table *Paper*. Originally there are only four researchers in table *Researcher*. But putting all of tables together, there seems to have six different researchers. As a result, the probability of each attribute value of tables *Researcher* and *University* cannot reflect the real statistical information. Because skewed information is used to compute the final probability of an example belonging to a class label, we believe that the answer could not be as correct as the second method.

In sum, from the above example one can see that simply joining all tables together could distort semantics in classification, and thus lead to wrong classification result.

4. Algorithms

With the help of semantic relationship graph and the extended Naïve Bayesian formula shown in formula (4), we can implement the multi-relational Naïve Bayes algorithm easily. In order to speed up the join operation, we adopt tuple ID propagation method [14] to do virtual join along each path of the semantic relationship graph. However, is it necessary to take all tables in the semantic graph into account? Our experimental results indicate that only part of these tables contribute to the improvement of classification accuracy. Therefore, we need to decide which part of the semantic relationship graph needs to be considered in order to achieve better classification accuracy. To do that, we design a pruning strategy that is called "cutting off".

Just as the meaning of the phrase "cutting off", we want to cut the semantic relationship graph into two parts just like cutting it by a knife. The part that contains the target table will be the optimized semantic relationship graph, and will be used to classify test dataset or unseen objects. The other part will be discarded. The challenge is that how to cut it. Our method is simple. Since we want to get the best classification accuracy, we can use training dataset to help make the decision. While navigating within the semantic relationship graph, whenever a table is linked to the target table directly or indirectly, we could classify each training tuple of the target table using the probability information of each attribute value for each class label in each table corresponding to each vertex currently already processed, and get an accuracy result.

For example, in Figure 2, after we collect information for each attribute value in table *Loan*, we can classify every tuple in *Loan* by this probability information, and get a classification accuracy result, say $accuracy_1$. Then after joining table *Account* with *Loan*, we can use the probability information of both table *Account* and *Loan* to classify each tuple in *Loan*, and get another accuracy result, say $accuracy_2$. In the end, we could choose the most accurate result among all of accuracy results, and cut off the graph at that point. For example, if $accuracy_2$is the best one, then only table *Account* and *Loan* and the edge between them will remain in the graph, and other tables and edges will be discarded.

Integrated with the "cutting off" pruning strategy, a multi-relational Naïve Bayes training algorithm called *Graph-NB* is designed and outlined in Figure 5.

Algorithm Graph-NB

Input: A training target table *tt*, a set *T* of other tables, and the semantic relationship graph *G*

Output: An optimized semantic relationship graph *OG*, and a set *P* of attribute value probabilities corresponding to each class label.

Method:

1. *maxAccuracy= CollectAndClassify*(tt)
2. Initialize *edgeNo*=0, *maxEdgeNo*=0;
3. **for each** out edge of target table *tt* do
   edgeNo++;
   NextEdge(*tt*)
4. Cut off all of edges with number$\geqslant$*maxEdgeNo*, and output remaining part of *G* as *OG*.

Subroutine *NextEdge*(*parentTable pt*)

Method:

1. let *t*=table pointed to by edge with number *edgeNo*;
2. *propagate(pt, t)*;
3. *accuracy=CollectAndClassify*(t);
4. **if** (*accuray> maxAccuracy)* **then**
   maxAccuracy=accuracy;
   maxEdgeNo=edgeNo;
5. **for each** out edge of current table *t* **do**
   edgeNo++;
   NextEdge(t);

Fig. 5. Training algorithm Graph-NB

Figure 5 shows the main processing steps of algorithm *Graph-NB*. First of all, Function *collectandClassify* is called to scan target table *tt*, collect attribute value count information for each class label, and use this information to classify each tuple of target

table *tt*. The overall classification accuracy is returned to variable *maxAccuracy*. Then in step 2, global variable *edgeNo* and *maxEdgeNo* are initialized, where *edgeNo* is a number used to record the processing order of all of the edges in graph G, and *maxEdgeNo* is one of these numbers with the maximum classification accuracy. After that, table pointed to by each out edge of target table *tt* is processed by calling the main subroutine *NextEdge*. Finally, with best accuracy among all the edges in the graph, we can cut off all of edges with number greater than *maxEdgeNo*, and the remaining part of the graph will be used to classify test or unseen objects. In the mean time, probability of attribute value in every table pointed to by each edge included in the SRG will also be returned.

In subroutine *NextEdge*, for the table *t* pointed to by current edge with number *edgeNo*, function *propogate is called to p*ropagate tuple ID from parent table *pt* to table *t*. Then Function *collectandClassify* is used to collect probability information and classify each tuple of target table. If the current accuracy is better than the existing best one, value of *maxAccuracy* and *maxEdgeNo* are changed to reflect this. After that, in step 5, each out edge of current table *t* is dealt with by recursive call of *NextEdge*.

Algorithm *Classify-NB*
Input: a test target table *st*, a set *T* of other tables, and the optimized semantic relationship graph *OG*
Output: class label for each tuple of table *st*
Method:
1. *computeProb(st);*
2. **for each** table *t* pointed to by each out edge of target table s*t* do
Classify(*st*,*t*)
3. output class label of each tuple in table *st*.

Subroutine *Classify*(*parentTable pt, currentTable t*)
Method:
1. *propagate(pt, t)*;
2. *computeProb(t)*;
3. **for each** table *ct* pointed to by each out edge of table *t* **do**
Classify(*t*, *ct*)

Fig. 6. Test algorithm Classify-NB

Based on the optimized semantic relationship graph, and the probability information returned by the training algorithm *Graph-NB*, any tuple of the test table or unseen object can be classified by algorithm *Classify-NB*, which is illustrated in Figure 6. The main procedure of this algorithm is similar to *Graph-NB*, so we will not explain it in detail.

5. Experimental Study

We conduct comprehensive experiments on both real and synthetic databases to compare the performance of multi-relational Naïve Bayes with those of other multi-relational classifiers: *CrossMine*[14], *FOIL*[10], 1BC[12], 1BC2[13], and Mr-SBC[19]. For *CrossMine*, We use *CrossMine* with sampling [25] and set the same values for parameters as in [14], that is, *MIN FOIL GAIN* = 2.5. *MAX RULE LENGTH* = 6. *NEG POS RATIO* = 1. *MAX NUM NEGATIVE* =600.

All experiments were performed on an IBM R40 laptop with Pentium 4 2.2Ghz CPU and 512MB RAM, running windows 2000 Professional. All runtimes for algorithm *Graph-NB* and *FOIL* include both computation time and I/O time, and runtimes for *CrossMine* only include computation time.

For both real world databases and synthetic databases, each table is stored in file to simulate the table in database. These files do not need to reside in memory. In the following subsections, we study the accuracy, efficiency and scalability of these algorithms. Besides, we also show the effectiveness of the "cutting off" pruning strategy.

5.1 Real World Databases

As illustrated in the case study in Section 2, we performed the first part of experiments on two real world databases. Accuracy is the result of ten-fold cross-validation.

5.1.1 Mutagenesis Database

This database is widely used in the area of ILP. It contains 4 tables and totals to 15218 tuples. The target table contains 188 elements of Mutagenesis. SRG showed in Figure 4 is used for this database. In order to compare the performance of Graph-NB with other methods for which experimental results are available in the literature, we use three levels of background knowledge of this dataset. Table 5 shows these three sets of background knowledge used in our experiments.

Table 5. Background knowledge for Mutagenesis databases

Background	Description
BK_0	For each compound, it obtains *atoms*, *bonds*, *bond types*, *atom types*, and *partial charges on atoms*.
BK_1	Consists of definitions in BK_0 plus attributes *indl* and *inda* in the mole table.
BK_2	Attributes *logp* and *lumo* are added to the mole table used in BK_1.

Table 6. Performance on BK_0

Algorithm	Accuracy	Runtime (second)
Graph-NB with pruning	77%	1.2
Graph-NB without pruning	77.5%	1.1
CrossMine	68.8%	2.5
FOIL	61%	4950
TILDE	75%	41
1BC	80.3%	---
1BC2	72.9%	---
Mr-SBC	76.5%	36

Table 7. Performance on BK_1

Algorithm	Accuracy	Runtime (second)
Graph-NB with pruning	84.1%	1.1
Graph-NB without pruning	77%	1.1
CrossMine	88.2%	1.6
FOIL	61%	9138
TILDE	79%	170
1BC	---	---
1BC2	---	---
Mr-SBC	81%	42

Table 8. Performance on BK_2

Algorithm	Accuracy	Runtime (second)
Graph-NB with pruning	86.2%	1.1
Graph-NB without pruning	78.1%	1.1
CrossMine	88.8%	1.4
FOIL	61%	0.5
TILDE	85%	142
1BC	87.2%	---
1BC2	72.9%	---
Mr-SBC	89.9%	48

The performances of the three algorithms for these two databases are shown in Tables 6, 7 and 8. In these tables, results for *FOIL*, *TILDE* are taken from [11]. Results for *1BC* are taken from [24]. Results for *1BC2* are taken from [13]. Results for *Mr-SBC* are taken from [23]. The average accuracy for these three datasets is shown in Table 9.

Tables 6 to 8 show that Graph-NB has the best performance in terms of runtime. Since this database is rather small, the difference of runtime for some algorithms is not very big. As shown in following experiments, the difference becomes large as the database become large.

Table 9. Average accuracy for Mutagenesis databases

Algorithm	Accuracy
Graph-NB with pruning	82.3%
Graph-NB without pruning	77.5%
CrossMine	81.9%
FOIL	68.3%
TILDE	79.7%
Mr-SBC	82.4%

As for the accuracy, Tables 6 to 8 indicate that *Graph-NB* achieves comparable accuracy with existing algorithms. For the average accuracy shown in Table 9, the accuracy of *Graph-NB* is only 0.1% lower than the highest accuracy among the five algorithms. The accuracy results also show the effectiveness of the "cutting off" strategy. It is effective for two of three datasets, and it increases the average accuracy by almost 5%.

Table 10. Performance on Financial database

Algorithm	Accuracy	Runtime (second)
Graph-NB with pruning	85.25%	1.9
Graph-NB without pruning	79.25%	1.9
CrossMine	87.25%	13.4
FOIL	71.5%	3479.3
TILDE	81.3%	2429

5.1.2 Financial Database

This database is the financial database used in PKDD CUP 1999. We did the same modification as that in [14]. In addition, we discretize the continuous attributes into 10 intervals for each attribute. Finally, this database has eight tables and 75982 tuples totally. The target table Loan contains 324 positive tuples and 76 negative tuples. We use the *SRG* showed in Figure 2 for this database.

In Table 10, the results for *TILDE* are taken from [14].

The performances of four algorithms for this database are shown in Table 10. Accuracy is the result of ten-fold cross-validation.

From Table 10 we can see that *Graph-NB* is about one order of magnitude faster than *CrossMine*, and several orders of magnitude faster than *FOIL* and *TILDE*. This is because *Graph-NB* usually does not need to scan each table as many times as *CrossMine*, *FOIL* and *TILDE*. Each table in each path of the semantic relationship graph only needs to be processed once, which takes much less time than searching every table for best predicates over and over. For accuracy, *Graph-NB* achieves better accuracy than *FOIL* and *TILDE*, and slightly lower accuracy than *CrossMine*.

From Tables 6 to 10, it is easy to see that the "cutting off" pruning strategy is effective in terms of accuracy improvement. Therefore, we will only report the result of *Graph-NB* with pruning in the following experimental study.

5.2 Synthetic Databases

In order to further test the efficiency of *Graph-NB*, we use synthetic database. The database is generated by a generator [14], which takes parameters shown in Table 11.

Keeping other parameters constant, we vary three parameters when generating each database, which are the number of relations, the expected number of tuples in each relation, and the expected number of foreign keys in each relation. We use *Rx.Ty.Fz* to represent a database with *x* relations, *y* expected tuples in each relation, and *z* expected foreign keys in each relation. For each database generated, we take the first 90% of tuples of target table as training data, and the remaining 10% of tuples of target table as test data. Accuracy is obtained on the test data.

Table 11. Parameters of Database Generator

Name	Description	Def.
|R|	# relations	x
T_{min}	Min # tuples in each relation	50
T	Expected # tuples in each relation	y
A_{min}	Min # attributes in each relation	2
A	Expected # attributes in each relation	5
V_{min}	Min # values in each relation	2
V	Expected # values in each relation	10
F_{min}	Min # foreign-keys in each relation	2
F	Expected # foreign-keys in each relation	z
|r|	# rules	10
L_{min}	Min # complex predicates in each rule	2
L_{max}	Max # complex predicates in each rule	6
f_A	Prob. of a predicate on active relation	0.25

In each of the following experiments, the runtime and accuracy of *Graph-NB* and *CrossMine* are compared. For *FOIL*, it usually takes about ten hours per database to train, which means that it is almost five orders of magnitude slower than *Graph-NB*. Therefore, we do not report the results of *Foil* here.

To evaluate the scalability with respect to the number of relations, we generate three databases with 10, 15 and 20 relations respectively. For all of these databases, the expected number of tuples in each relation is 1000 and the expected number of foreign keys in each relation is 1. The runtime and accuracy are shown in Figures 7 and 8 respectively. The runtime in these and following figures is in logarithmic scale.

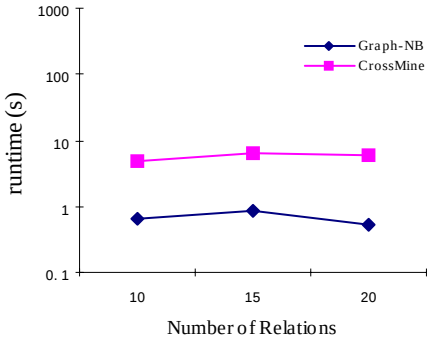

Fig. 7. Runtime on R*.T1000.F1

Figure 7 indicates that *Graph-NB* is much faster than *CrossMine*, and Figure 8 shows that the accuracy of *Graph-NB* is sometimes equal to *CrossMine* and sometimes a bit lower than *CrossMine*.

To evaluate the scalability of these algorithms with respect to the number of tuples per relation, we generate four databases with 1000, 3000, 6000 and 10000 expected tuples per relation respectively. For all of these databases, the number of relations is 10 and the expected number of foreign keys per relation is 1. The runtime and accuracy are shown in Figures 9 and 10 respectively.

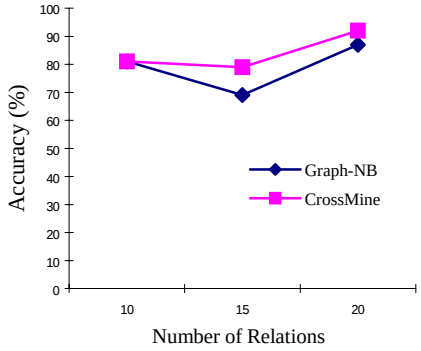

Fig. 8. Accuracy on R*.T1000.F1

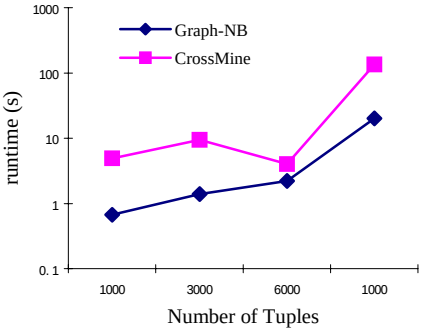

Fig. 9. Rutime on R10.T*.F1

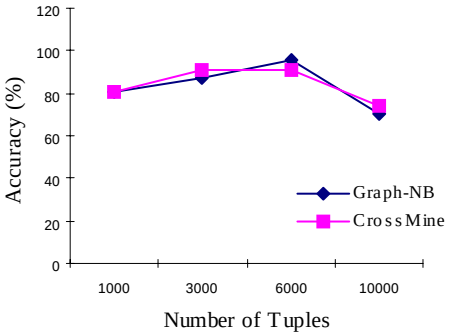

Fig. 10. Accuracy on R10.T*.F1

To evaluate the scalability of these algorithms with respect to the number of tuples per relation, we generate four databases with 1000, 3000, 6000 and 10000 expected tuples per relation respectively. For all of these databases, the number of relations is 10 and the expected number of foreign keys per relation is 1. The runtime and accuracy are shown in Figure 9 and 10 respectively.

Figure 9 indicates that *Graph-NB* is much faster than *CrossMine* for most databases. It also shows that as the number of tuples increases the runtime also increases for algorithms *Graph-NB*. Figure 10 shows that the accuracy of *Graph-NB* is sometimes a bit higher than *CrossMine* and sometimes a bit lower.

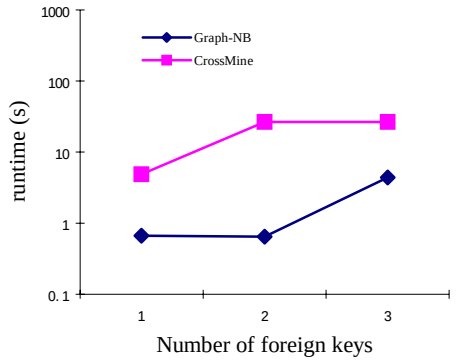

Fig. 11. Runtime on R10.T1000.F*

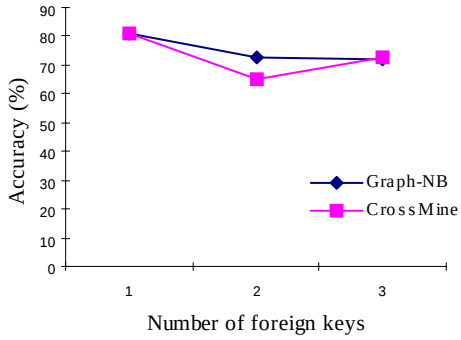

Fig. 12. Accuracy on R10.T1000.F*

To evaluate the scalability of these algorithms with respect to the number of foreign keys per relation, three databases with 1, 2, and 3 expected foreign keys per relation respectively are generated. For all of these databases, the number of relations is 10 and the expected number of tuples is 1000 per relation. The runtime and accuracy are shown in Figure 11 and 12 respectively.

We can see from Figure 11 that *Graph-NB* is about one order of magnitude faster than *CrossMine*. It also shows that as the number of foreign key increases the runtime also increases. This is because as number of foreign keys increases the number of edges increases, so more time needed to deal with them. Figure 12 shows that the accuracy of *Graph-NB* is generally a bit greater than *CrossMine.*

In general, these above experimental results tell us that algorithms *Graph-NB* is highly efficient, and also has comparable accuracy compared with *CrossMine*.

6. Related Work

We have seen a lot of work on multi-relational classification, especially in the community of Inductive Logic Programming (ILP). But few of them focus on both classification accuracy and efficiency. *CrossMine* [14] is an algorithm proposed to enhance the efficiency of multi-relational classification. It uses the idea of tuple ID propagation to do join operation virtually between tables. By avoiding the cost of physical joins, *CrossMine* achieves better scalability and accuracy compared to ILP algorithm *FOIL* (first-order inductive learner) and decision tree based algorithms *TILDE* [11]. In our work, we also adopt this method to link non-target tables with target table efficiently. Multi-relational Naïve Bayes classifier is first studied in the ILP area. Pompe and Kononenko [19] proposed a two-step method. In the first step the ILP-R system [20] is used to learn a set of first-order rules. And then in the second step, rules are analyzed. When classifying an unseen example, these rules are combined according to Naïve Bayesian formula. *1BC* is also a first-order Bayesian classifier for multi-relational data [12]. It mainly focuses on decomposing related tuples with a target tuple into set of items and attribute values. This is done by first finding a set of first-order conditions, and then using these conditions as attributes in a classical Naïve Bayesian classifier. *1BC* is implemented in the context of the first-order descriptive learner *Tertius* [21]. *1BC2* [13] follows *1BC*'s work and uses complex list and set-valued estimators to model lists and sets of attribute values. All of these three first-order naïve Bayesian classifiers take ground fact, which is a flattened and function-free Prolog representation, as input. As there are disadvantages to work on a set of Prolog facts, a system named *Mr-SBC* [23] is proposed to extend naïve Bayes classification method to multi-relation setting. *Mr-SBC* implements a new algorithm based on integrating first-order classification rules with naïve Bayesian classification so that the computation of probabilities of shared literals can be separated from the computation of probabilities for the remaining literals. However, while searching first-order rules, only tables in a *foreign key path* which is defined in [23] can be considered, and other join paths are neglected. In real applications, there are several other link paths between tables. Taking only *foreign key paths* into account limits its applications to real world situations very much. Another recent work is Relation Bayesian Classifier [7], which studies the performance of four estimators to estimate the conditional probability of a set of attribute values. It also studies the effects of common characteristics of relational data such as concentrated linkage on bias and variance. However, this work deals with multi-relational data by decomposing related objects down to attribute level, and each target object is considered as a subgraph with the target object in the center surrounded by all related non-target objects. These non-target objects are treated equally in classification. In our work, we use semantic relationship graph to describe different relationships between tables, and use "cutting off" pruning strategy to treat tables differently. In addition, our work deals with tables instead of Prolog facts directly.

7. Conclusions

It has been shown that considering not only information in the target table but also that in other related tables can improve classification accuracy. But there are several challenges for developing an efficient and accurate multi-relational classification algorithm. One is how to define the relationship between all tables in a database so that both semantic relationship and existing domain knowledge can be easily exploited, and in the meantime, unnecessary joins could be avoided. Another challenge is that how far we should go into the relationship between these tables so that weakly linked tables will not be considered. In this paper, we propose an algorithm named *Graph-NB* to tackle these problems. First, traditional Naïve Bayes is extended to deal with data stored

in tables directly. Second, a directed acyclic graph named semantic relationship graph is defined to describe the relationship between tables. Third, a pruning strategy is proposed to simplify the semantic relationship graph so that weakly linked tables will be ignored. Comprehensive experimental study on both real world databases and synthetic databases shows that it is scalable with respect to the number of relations, the expected number of tuples and the expected number of foreign keys. In the meantime, it also achieves good accuracy in general.

8. ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant No. 70471006 and 70321001, and by the U.S. National Science Foundation NSF IIS-02-09199 and IIS-03-08215.

9. REFERENCES

[1] Kramer, S., N. Lavrac and P. Flach. Propositionalization approaches to relational data mining. In S. Dzeroski and N. Lavrac, eds. *Relational Data Mining*. pp 262-291, Springer-Verlag, 2001.

[2] Lavrac N. and Dzeroski S. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[3] Muggleton S. Inductive Logic Programming. Academic Press, New York, NY, 1992.

[4] Friedman N., Getoor L., Koller D., and Pfeffer A. Learning Probabilistic Relational Models. In *Sixteenth International Joint Conference on Artificial Intelligence* (IJCAI), 1999.

[5] Taskar B., Segal E., and Koller D. Probabilistic Classification and Clustering in Relational Data. In *17th International Joint Conference on Artificial Intelligence*, pages 870–878, 2001.

[6] Emde W. and Wettschereck D. Relational Instance-Based Learning. In L. Saitta, editor, *Proceedings 13th International Conference on Machine Learning*, pages 122–130. Morgan Kaufmann, 1996.

[7] Neville J., Jensen D., Gallagher B., and Fairgrieve R. Simple Estimators for Relational Bayesian Classifiers. *Int. Conf. on Data Mining (ICDM'03)*, Melbourne, FL, Nov. 2003.

[8] Neville J., Jensen D., Friedland L., and Hay M.. *Learning Relational Probability Trees*. Technical Report 02-55, Department of Computer Science, University of Massachusetts Amherst, 2002. Revised version February 2003.

[9] Garcia-Molina H., J. Ullman D., and Widom J. *Database Systems: The Complete Book*. Prentice Hall, 2002.

[10] Quinlan J. R. and Cameron-Jones R. M. FOIL: A midterm report. In *Proc. 1993 European Conf. Machine Learning*, Vienna, Austria, 1993.

[11] Blockeel H., De Raedt L., and Ramon J. Top-down induction of logical decision trees. In *Proc. 1998 Int. Conf. Machine Learning (ICML'98)*, Madison, WI, Aug. 1998.

[12] Flach, P. and N. Lachiche. 1BC: A first-order Bayesian classifier. *Proceedings of the 9th International Workshop on Inductive Logic Programming*, pp. 92--103, 1999.

[13] Lachiche, N. and P. Flach. 1BC2: a true first-order Bayesian Classifier. *Proceedings of the 12th International Conference on Inductive Logic Programming*, 2002.

[14] Yin X., Han J., Yang J., and Yu P. S. "CrossMine: Efficient Classification across Multiple Database Relations", *Proc. 2004 Int. Conf. on Data Engineering (ICDE'04)*, Boston, MA, March 2004.

[15] Macskassy S. and Provost F. A Simple Relational Classifier. *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2nd workshop on Multi-relational Data Mining*. Washington, DC, 2003.

[16] Duda R. and Hart P. *Pattern Classification and Scene Analysis*. New York: John Wiley & Sons, 1973.

[17] Join G. H. *Enhancements to the Data Mining Process*. Ph.D. Thesis, computer Science Dept, Stanford University, 1997.

[18] Domingos P. and Pazzani M. Beyond Independence: Conditions for the Optmality of the Simple Bayesian Classifier. In Proc. *13th Intl. Conf. Machine Learning*, p105-112, 1996.

[19] Pompe U. and Kononenko I. Naive Bayesian classifier within ILP-R. In L. De Raedt, editor, *Proc. of the 5th Int. Workshop on Inductive Logic Programming*, pages 417--436. Dept. of Computer Science, Katholieke Universiteit Leuven, 1995.

[20] Pompe U., Kononenko I. Linear space induction in first order logic with relief. In R. Kruse, R. Viertl. & G. Della Riccia (Eds.), *CISM Lecture Notes.* Udine Italy, 1994.

[21] Flach P.A. and Lachiche N. A First-order approach to unsupervised learning. Submitted, 1999.

[22] 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 ComputingLaboratory, Oxford, 1999.

[23] Ceci M., Appice A., and Malerba D. Mr-SBC: a Multi-Relational Naive Bayes Classifier, in N. Lavrac, D. Gamberger, L. Todorovski & H. Blockeel (Eds.), *Knowledge Discovery in Databases PKDD 2003*, Lecture Notes in Artificial Intelligence, 2838, 95-106, Springer, Berlin, Germany.

[24] Flach P. and Lachiche N. First-order Bayesian Classification with 1BC. Submitted. Downladable from http://hydria.u-strasbg.fr/~lachiche/1BC.ps.gz

[25] Yin X. CrossMine software. http://www-sal.cs.uiuc.edu/~hanj/pubs/software.htm.