Using Bayesian Classifiers to Combine Rules

Jesse Davis, Vítor Santos Costa, Irene M. Ong,
David Page and Inês Dutra

Department of Biostatistics and Medical Informatics
University of Madison-Wisconsin
{jdavis, vitor, ong, page, dutra}@biostat.wisc.edu

Abstract. One of the most popular techniques for multi-relational data mining is Inductive Logic Programming (ILP). Given a set of positive and negative examples, an ILP system ideally finds a logical description of the underlying data model that discriminates the positive examples from the negative examples. However, in multi-relational data mining, one often has to deal with erroneous and missing information. ILP systems can still be useful by generating rules that captures the main relationships in the system. An important question is how to combine these rules to form an accurate classifier. An interesting approach to this problem is to use Bayes Net based classifiers. We compare Naïve Bayes, Tree Augmented Naïve Bayes (TAN) and the Sparse Candidate algorithm to a voting classifier. We also show that a full classifier can be implemented as a CLP($\mathcal{BN}$) program [14], giving some insight on how to pursue further improvements.

1 Introduction

The last few years have seen a surge of interest in multi-relational data mining, with applications in areas as diverse as bioinformatics and link discovery. One of the most popular techniques for multi-relational data mining is Inductive Logic Programming (ILP). Given a set of positive and negative examples, an ILP system ideally finds a logical description of the underlying data model that differentiates between the positive and negative examples. ILP systems confer the advantages of a solid mathematical foundation and the ability to generate understandable explanations.

As ILP systems are being applied to tasks of increasing difficulty, issues such as large search spaces and erroneous or missing data have become more relevant. Ultimately, ILP systems can only expect to search a relatively modest number of clauses, usually on the order of millions. Evaluating increasingly complex clauses may not be the solution. As clauses grow larger, they become more vulnerable to the following errors: a query will fail because of missing data, a query will encounter an erroneous database item, and a clause will give correct answers simply by chance.

Our work relates to a sizeable application in the field of link discovery. More precisely, our concern involves finding aliases in a relational domain [15] where the data is subject to high levels of corruption. As a result, we cannot hope that the learned rules will generally model the entire dataset. In these cases, ILP can at best generate rules that describe fragments of the underlying model. Our hope is that such rules will allow us to observe the central relationships within the data.

An important question is how to combine the partial rules to obtain a useful classifier. We have two major constraints in our domain. First, we expect the number of positives to grow linearly with the number of individuals in the domain. In contrast, the number of negatives increases with the number of pairs, and therefore grows quadratically. Consequently, any approach should be robust to false positives. Furthermore, flexibility is also an important consideration as we ultimately want to be able to weigh precision versus recall through some measure of confidence, ideally in the form of a probability. Secondly, we expect to use the system for different datasets: our method should not be prone to overfitting and it should be easy to parameterize for datasets with different observabilities and error rates.

The previous discussion suggests probabilistic-based classifiers as a good approach to our problem. We explore three different Bayes net based approaches to this problem. Each ILP learned rule is represented as a random variable in the network. The simplicity and robustness of the Naïve Bayes classifier make it a good candidate for combining the learned rules [12]. Unfortunately, Naïve Bayes assumes independence between features and our rules may be quite interdependent and perhaps even share literals. A natural extension is to use TAN [6] classifiers as they offer an efficient way to capture dependencies between rules. Additionally, we explore using the Sparse Candidate algorithm [7] for learning the structure of a full Bayes net. An alternative approach we consider is to group our rules as an ensemble [3] and use voting, which has had excellent results in practice. We will evaluate the relative merits of these approaches.

The paper is organized as follows. We first discuss the problem in more detail. Then, we explain the voting and Bayesian based approaches to rule combination. Next, we present the main applications and discuss our results. We follow this by demonstrating how we can represent Bayesian classifiers as a logic program with probabilities, using CLP($\mathcal{BN}$). Finally, we end with related work and our conclusions.

2 Using ILP

From a logic perspective, the ILP problem can be defined as follows. Let E^+ be the set of positive examples, E^- be the set of negative examples, $E = E^+ \wedge E^-$, and B be the background knowledge. In general, B and E can be arbitrary logic programs. The aim of an ILP system is to find a set of hypotheses (also referred to as a theory) H, in the form of a logic program, such that all positive examples and none of the negative examples are covered by the program.

In practice, learning processes generate relatively simple clauses which only cover a limited subset of E^+. Moreover, such clauses often cover some examples in E^-. One possible reason for the presence of these errors is that these examples may have been misclassified. A second reason is that approximated theories can never be as strict as the ground truth: if our clause is only a subclause of the actual explanation, it is possible that the clause will cover a few other incorrect examples. We also have to address implementational difficulties: for most cases we can only search effectively for relatively simple explanations (clauses). Therefore, we assume that clauses represent fragments of the ground-truth and that the learning process can capture different "features" of the ground truth. Clauses have some distribution, which is likely to be non-uniform, over the interesting aspects of the ground-truth theory. Even if we do not capture all features of the ground truth, we can still learn interesting and relevant clauses.

Given a set H of clauses learned in an incomplete world we can combine them to obtain a better classifier. One possible approach to combine clauses would be to assume that each clause is an explanation, and form a disjunction over the clauses. Although this approach has the merit of simplicity, and should work well for cases where we are close to the ground truth, it does have two serious issues we need to consider:

- We are interested in applications where the number of false instances dominates. Unfortunately, the disjunction of clauses maximizes the number of false positives.
- We expect the classifier to make mistakes, so ideally we would like to know the degree of confidence we have in a classification.

Our problem is not novel, and several approaches come to mind. We shall focus on two such approaches here. The idea of exploiting different aspects of an underlying classifier suggests ensemble-based techniques. Previous work on applying ensemble methods to ILP [4] suggests that exploring the variability in the seed is sufficient for generating diverse classifiers. We thus decided to use a simple approach where we use the ILP engine to generate clauses and then use voting to group them together. A second alternative is to consider each clause as a feature of an underlying classifier. We want to know which features are most important. Several possibilities exist and we focus on Bayesian networks, as they provide us with an estimated probability for each different outcome.

3 Combining Rules

3.1 Voting

It is well known that ILP systems that learn clauses using seeds are exploiting different areas of the search space, in a manner analogous to ensemble methods. In this vein, recent ILP work has exploited several techniques for ensemble generation, such as bagging or bootstrapping [4] and different forms of boosting [5,13,9,10]. Bagging is a popular ensemble method that consists of generating different training sets where each set contains a sample, with replacement, of the original dataset. Hypotheses are learned from each dataset, and combined through a voting method. Alternatively, in boosting each classifier is built depending on the errors made by previous classifiers. Each new rule thus depends on the performance of the previous one.

Previous work on applying bagging to ILP [4] suggests that exploring variability from using different seed examples can be sufficient for generating diverse classifiers. We shall follow a similar approach: we use hypotheses generated from different runs of the ILP system, and combine them through unweighted voting. With this method, we consider an example to be positive depending on the number of clauses that are satisfied for that example. The number of clauses we need to satisfy to classify the example as positive is a variable threshold parameter. One major advantage of using a voting method is that we can obtain different values of precision and recall by varying the voting threshold. Thus, although a voting method does not give an estimate of the probability for each classification, it does provide an excellent baseline to compare with Bayesian-based methods.

3.2 Bayesian Networks

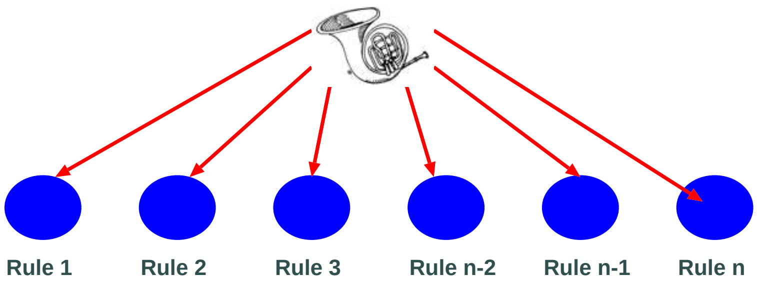

Fig. 1. A Naïve Bayes Net.

We expect every learned clause to be related to a clause in the "true" theory. Hence, we would also expect that the way each learned clause classifies an example is somehow dependent on the example's true classification. This suggests a simple approach where we represent the outcome for each clause as a random variable, whose value depends on the example's classification. The Naïve Bayes approach is shown in Figure 1 [12]. Advantages of this approach are that it is straightforward to understand as well as easy and fast to train.

The major drawback with Naïve Bayes is that it makes the assumption that the clauses are independent given the class value. Often, we expect clauses to be strongly related. Learning a full Bayes Net is an NP-complete problem, so in this work, we experimented with Tree Augmented Naïve Bayes (TAN) [6] networks. Figure 2 shows an example of a TAN network. TAN models allow for more complex network structures than Naïve Bayes. The model was proposed by Geiger in 1992 [8] and it extends work done by Chow and Liu [2]. Friedman,

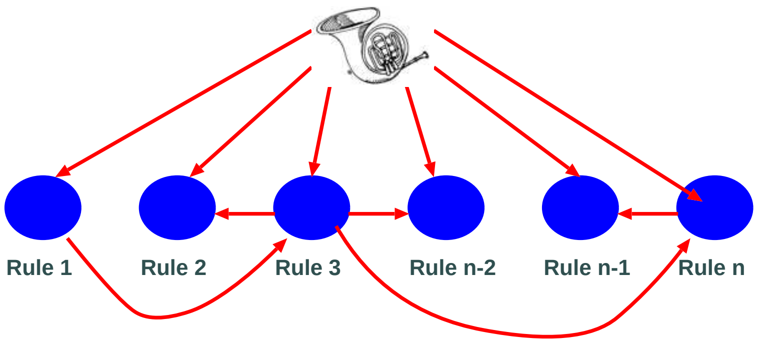

Fig. 2. A TAN Bayes Net.

Geiger and Goldszmidt [6] evaluated the algorithm on its viability for classification tasks. The TAN model, while retaining the basic structure of Naïve Bayes, also permits each attribute to have at most one other parent, allowing the model to capture dependencies between attributes. To decide which arcs to include in the 'augmented' network, the algorithm makes a complete graph between all the non-class attributes, where the weight of each edge is given as the conditional mutual information between those two attributes. A maximum weight spanning tree is constructed over this graph, and the edges that appear in the spanning tree are added to the network. Geiger proved that the TAN model can be constructed in polynomial time with a guarantee that the model maximizes the Log Likelihood of the network structure given the dataset.

The problem arises of whether different Bayes networks could do better. We report on some preliminary work using the Sparse Candidate Algorithm [7]. The Sparse Candidate algorithm tries to speed up learning a full Bayesian Network by limiting the search space of possible networks. The central premise is that time is wasted in the search process by evaluating edges between attributes that are not highly related. The algorithm retains standard search techniques, such as greedy hill climbing, but uses mutual information to limit the number of possible parents for each attribute to small 'candidate' set. The algorithm works in two phases. In the first phase, the candidate set of parents is picked for each attribute. The candidate set must include all current parents of a node. The second step involves performing the actual search. These two steps are repeated either for a set number of times or until the score of the network converges.

4 Results

This section presents our results and analysis of the performance of several applications. For each application we show precision versus recall curves for the four methods: Naïve Bayes, TAN, Sparse Candidate and voting. All our experiments were performed using Srinivasan's Aleph ILP system [16] running on the Yap

Prolog system. We used our own software for Naïve Bayes and TAN. For the Sparse Candidate Algorithm we used the LearnBayes program provided by Nir Friedman and Gal Elidan. For this algorithm we set the number of candidate parents to be five and we used the Bayesian Information Criterion as the scoring function. All results are obtained using five fold cross-validation.

Our main experiment was performed on synthetic datasets developed by Information Extraction & Transport, Inc. within the EAGLE Project [15,11]. The datasets are generated by simulating an artificial world with large numbers of relationships between agents. The data focuses on *individuals* which may have capabilities, belong to groups, and participate in a wide range of *events*. In our case, given that some individuals may be known through different identifiers (e.g., through two different phone numbers), we were interested in recognizing whether two identifiers refer to the same individual.

All datasets were generated by the same simulator, but with different parameters for observability (how much information is available as evidence), corruption, and clutter (irrelevant information that is similar to the information being sought). Five datasets were provided for training, and six for evaluation. All the datasets include detailed data on a few individuals, including aliases for some individuals. Depending on the dataset, the data may or may not have been corrupted.

Our methodology was as follows. First, we used the five training datasets to generate rules, using the ILP system Aleph. Using the rules learned from the training set, we selected the ones with best accuracy and combined them with domain expert knowledge to provide new feedback to the training phase. Using the final set of learned rules, we converted each of the evaluation datasets into a set of propositional feature vectors, such that each rule appeared as an attribute in the feature vector. Each rule served as a boolean attribute, which received a value of one if the rule matched the example and zero otherwise. For each of the six test datasets, we performed five fold cross validation. The network structure and parameters were learned on four of the folds, while the accuracy was tested on the remaining fold. For each dataset, we fixed the ratio of negative examples to positive examples at seventy to one. This is an arbitrary ratio since the full datasets are exceedingly large, and the ground truth files were only recently released.

The precision/recall (P/R) curves for the different datasets are seen in Figures 3 through 8. On each curve, we included 95% confidence intervals on the precision score for select levels of recall. The curves were obtained by averaging the precision and recall values for fixed thresholds. The precision recall curve for the TAN algorithm dominates the curves for Naïve Bayes and voting on all six of the datasets. For each dataset, there are several places where TAN yields at least a 20 percentage point increase in precision, for the same level of recall, over both Naïve Bayes and voting. On two of the six datasets, Naïve Bayes beats voting, while on the remaining four they have comparable performance. One reason for TAN's dominance compared to Naïve Bayes is the presence of rules which are simply refinements of other rules. The TAN model is able to capture some

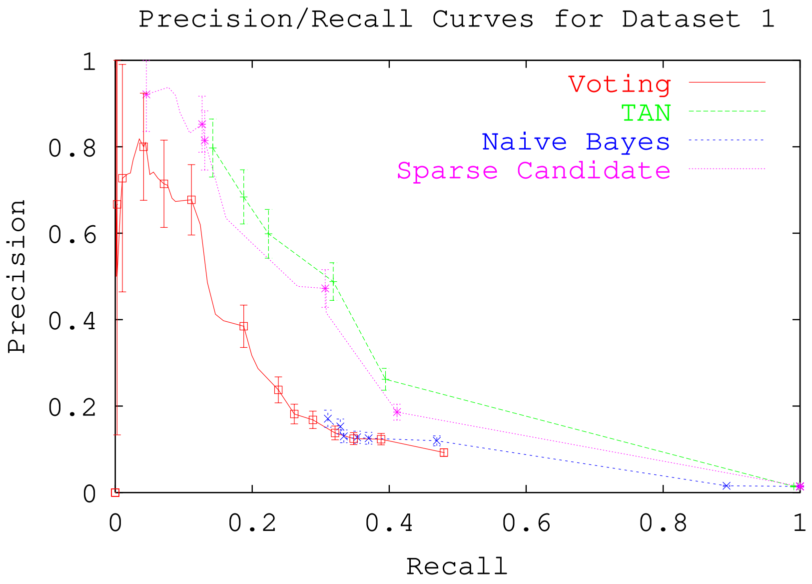

Fig. 3. P/R for Dataset 1

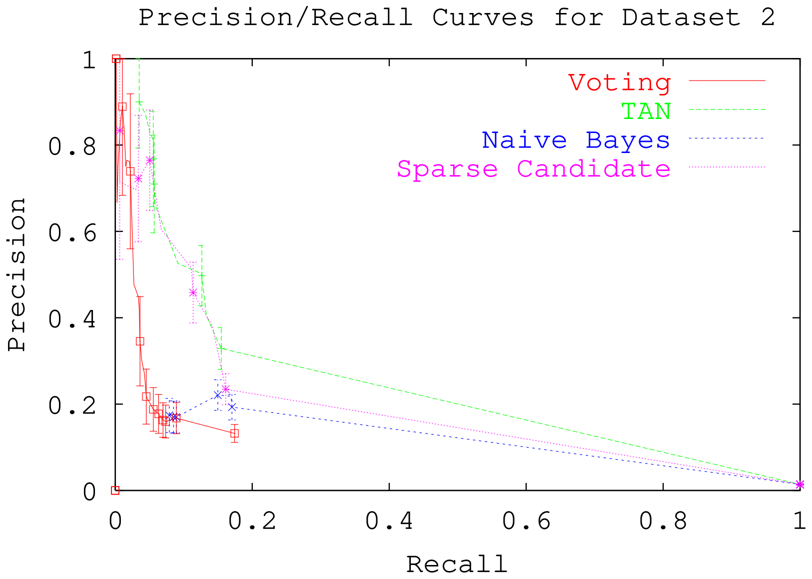

Fig. 4. P/R for Dataset 2

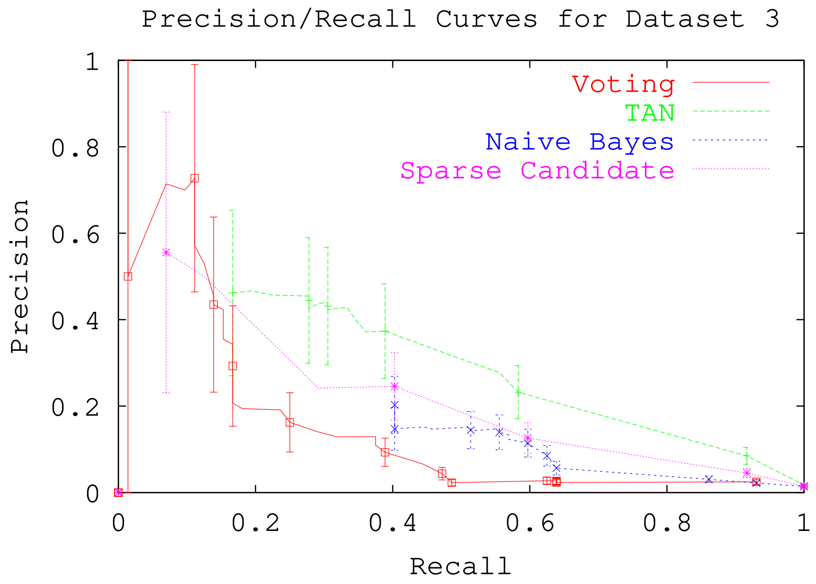

Fig. 5. P/R for Dataset 3

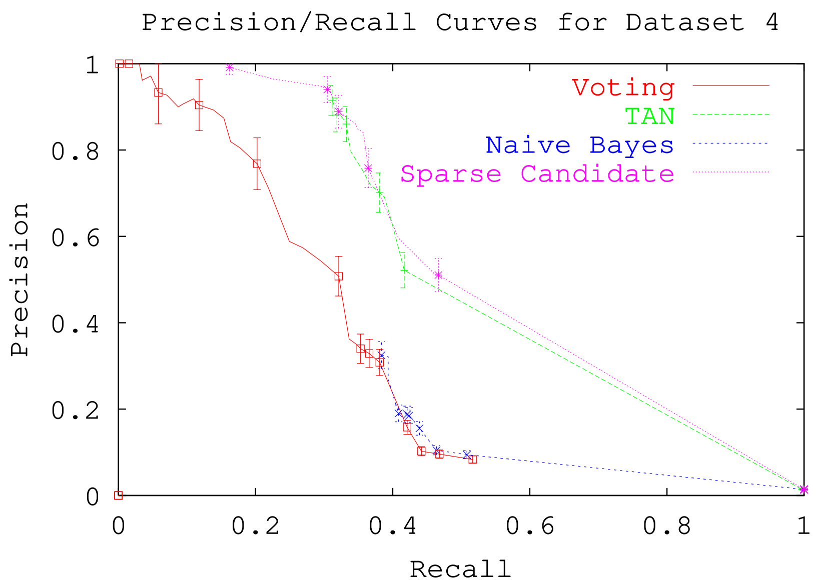

Fig. 6. P/R for Dataset 4

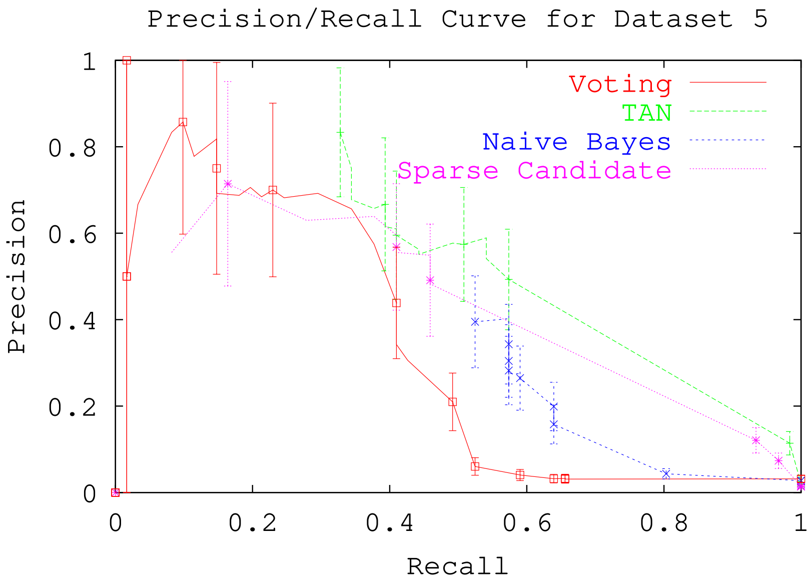

Fig. 7. P/R for Dataset 5

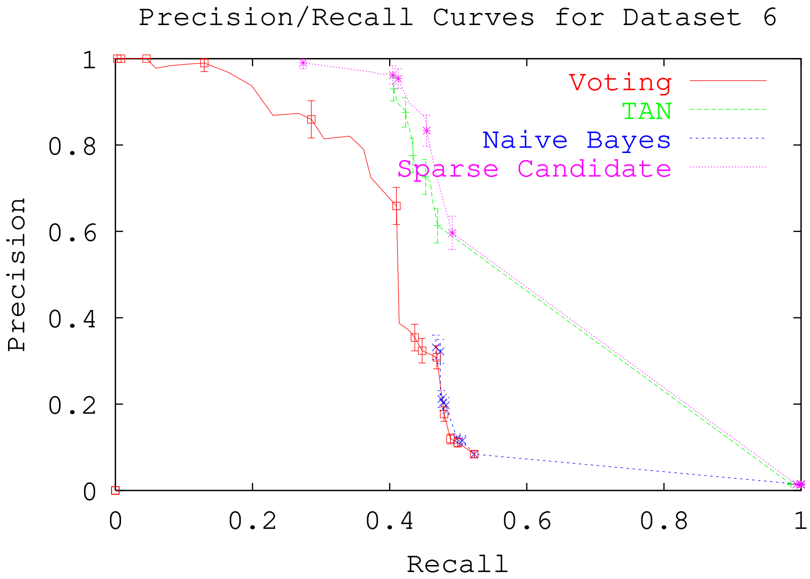

Fig. 8. P/R for Dataset 6

of these interdependencies, whereas Naïve Bayes explicitly assumes that these dependencies do not exist. Naïve Bayes' independence assumption accounts for the similar performance compared to voting on several of the datasets. TAN and the Sparse Candidate algorithm had similar precision recall curves. The package we used for the Sparse Candidate algorithm only allows for building generative models. TAN is a discriminative model, so it emphasizes differentiating between positive and negative examples. An important follow-up experiment would be to adapt the Sparse Candidate algorithm to use discriminative scoring functions.

In situations with imprecise rules and a preponderance of negative examples, such as these link discovery domains, Bayesian models and especially TAN provide an advantage. One area where both TAN and Naïve Bayes excel is in handling imprecise rules. The Bayes nets effectively weight the precision of each rule either individually or based on the outcome of another rule in the case of TAN. The Bayesian nets further combine these probabilities to make a prediction of the final classification, allowing them to discount the influence of spurious rules in the classification process. Ensemble voting does not have this flexibility and consequently lacks robustness to imprecise rules. Another area where TAN provides an advantage is when multiple imprecise rules provide significant overlapping coverage on positive examples and a low level of overlapping coverage on negative examples. The TAN network can model this scenario and weed out the false positives. One potential disadvantage to the Bayesian approach is that it could be overly cautious about classifying something as a positive. The high number of negative examples relative to the number of positive examples, and the corresponding concern of a high false positive rate, helps mitigate this potential problem. In fact, at similar levels of recall, TAN has a lower false positive rate than voting.

5 The CLP($\mathcal{BN}$) Representation

Using Bayesian classifiers to join the rules means that we will have two distinct classifiers using very different technology: a logic program (a set of rules), and a Bayes net. Some further insight may be obtained by using formalisms that combine logic and probabilities, such as CLP($\mathcal{BN}$).

CLP($\mathcal{BN}$) is based on the observation that in Datalog, missing values are represented by Skolem constants; more generally, in logic programming missing values, or existentially-quantified variables, are represented by terms built from Skolem functors. CLP($\mathcal{BN}$) represents such terms with unknown values as *constraints*. Constraints are kept in a separate *store* and can be updated as execution proceeds (ie, if we receive new evidence on a variable). Unifying a term with a constrained variable invokes a specialized *solver*. The solver is also activated before presenting the answer to a query. Syntactically, constraints are represented as terms of the form $\{C = Skolem\ with\ CPT\}$, where C is the logical variable, $Skolem$ identifies the skolem function, and CPT gives the parameters for the probability distribution.

First, we show how the Naïve Bayes net classifier can be built using CLP($\mathcal{BN}$). The value taken by the classifier is a random variable that may take the value `t` or `f` with some prior probability:

```
classifier(C) :-
       { C = classifier with p([f,t],[0.25,0.75]) }.
```

Each rule I's score V is known to depend on the classifier only:

```
rule(I,V) :-
       classifier(C),
       rule_cpt(I,P1,P2,P3,P4),
       { V = rule(I) with p([f,t],[P1,P2,P3,P4],[C]) }.
```

Rule I's score is V, which is either `f` or `t`. The value of V depends on the value of the classifier, C, according to the conditional probability table `[P1,P2,P3,P4]`. Our implementation stores the tables for each rule in a database:

```
rule_cpt(1,0.91,0.66,0.09,0.34).
rule_cpt(2,0.98,0.87,0.02,0.13).
rule_cpt(3,0.99,0.79,0.01,0.21).
rule_cpt(4,0.99,0.87,0.01,0.13).
.....
```

This fully describes the Bayes net. To actually evaluate a rule we just need to introduce the evidence given by the different rules:

```
nbayes(A,B,C) :-
        all_evidence(0,39,A,B),
        classifier(C).

all_evidence(N,N,_,_).
all_evidence(I0,N,A,B) :-
        I0 < N, I is I0+1,
        rule_evidence(I,A,B),
        all_evidence(I,N,A,B).

rule_evidence(I,A,B) :- equals(I,A,B), !, rule(I,t).
rule_evidence(I,A,B) :- rule(I,f).
```

The predicate `nbayes/3` receives a pair of individuals `A` and `B`, adds evidence from all rules, and then asks for the new probability distribution on the classifier,`C`. The predicate `all_evidence` recursively considers evidence from every rule. The predicate `rule_evidence/3` calls rule `I` on the pair `A` and `B`. If the rule succeeds, evidence from rule `I` is `t`, otherwise it adds evidence `f`.

A TAN network only differs in that a rule node may have two parents, the classifier `C` and some other node `J`. This is described in the following clause:

```
rule(I,V) :-
       rule_cpt(I,J,P1,P2,P3,P4,P5,P6,P7,P8),
       classifier(C),
       rule(J,V1),
       { V = rule(I) with p([f,t],[P1,P2,P3,P4,P5,P6,P7,P8],[C,V1]) }.
```

More complex networks can be described in a similar fashion.

CLP($\mathcal{BN}$) offers two main advantages. First, we can offer interactive access to the full classifier. Second, we gain some insight since our task now involves learning a single CLP($\mathcal{BN}$) program, where each newly induced rule will result in recomputing the probability parameters currently in the database.

6 Relationship to Other Work

Our present work fits into the popular category of using ILP for feature construction. Such work treats ILP-constructed rules as Boolean features, re-represents each example as a feature vector, and then uses a feature-vector learner to produce a final classifier. To our knowledge, the work closest to ours is by Kononenko and Pompe [12], who were the fist to apply Naïve Bayes to combine clauses. Other work in this category was by Srinivasan and King [17], for the task of predicting biological activities of molecules from their atom-and-bond structures. Some other research, especially on propositionalization of First Order Logic (FOL) [1], have been developed that convert the training sets to propositions and then apply feature vector techniques to the learning phase. This is similar to what we do; however, we first learn from FOL and then learn the network structure and parameters using the feature vectors obtained with the FOL training, resulting in much smaller feature vectors than in propositionalization.

Our paper contributes three novel points to this category of work. First, it highlights the relationship between this category of work and ensembles in ILP, because when the feature-vector learner is Naïve Bayes the learned model can be considered a weighted vote of the rules. Second, it shows that when the features are ILP-learned rules, the independence assumption in Naïve Bayes may be violated badly enough to yield a high false positive rate. This false positive rate can be brought down by permitting strong dependencies to be explicitly noted, through learning a tree-augmented Naïve Bayes net (TAN). Third, the present paper provides some early experimental evidence suggesting that a more computationally expensive full Bayes net learning algorithm may not provide added benefit in performance.

7 Conclusions

One often has to deal with erroneous and missing information in multi-relational data mining. We compare how four different approaches for combining rules learned by an ILP system perform for an application where data is subject to corruption and unobservability. We were particularly interested in Bayesian methods because they associate a probability with each prediction, which can be thought of as the classifier's confidence in the final classification.

In our application, we obtained the best precision/recall results using a TAN network to combine rules. Precision was a major concern to us due to the high ratio of negative examples to positive examples. TAN had better precision than Naïve Bayes because it is more robust at handling high redundancy between clauses. TAN also outperformed voting in this application. Initial results for the sparse candidate algorithm show a significant increase in computation time, but no significant improvements in precision/recall.

In future work we plan to experiment with different applications and with full Bayesian networks trained using a discriminative scoring function. We also plan to further continue work based on the observation that we learn a single CLP($\mathcal{BN}$) network: this suggests that the two learning phases could be better integrated.

8 Acknowledgments

Support for this research was partially provided by U.S. Air Force grant F30602-01-2-0571. We would also like to thank the referees for their insightful comments.

References

1. E. Alphonse and C. Rouveirol. Lazy propositionalisation for relational learning. In H. W., editor, *14th European Conference on Artificial Intelligence, (ECAI'00) Berlin, Allemagne*, pages 256–260. IOS Press, 2000.
2. C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependece trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968.
3. T. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2000.
4. I. Dutra, D. Page, V. Santos Costa, and J. Shavlik. An empirical evaluation of bagging in inductive logic programming. In S. Matwin and C. Sammut, editors, *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *Lecture Notes in Artificial Intelligence*, pages 48–65. Springer-Verlag, 2003.
5. Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 148–156. Morgan Kaufman, 1996.
6. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian networks classifiers. *Machine Learning*, 29:131–163, 1997.
7. N. Friedman, I. Nachman, and D. Pe'er. Learning bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 206–215, San Francisco, CA, 1999. Morgan Kaufmann Publishers.
8. D. Geiger. An entropy-based learning algorithm of bayesian conditional trees. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighth Conference (UAI-1992)*, pages 92–97, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

9. S. Hoche and S. Wrobel. Relational learning using constrained confidence-rated boosting. In C. Rouveirol and M. Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 51–64. Springer-Verlag, September 2001.
10. S. Hoche and S. Wrobel. A comparative evaluation of feature set evolution strategies for multirelational boosting. In T. Horváth and A. Yamamoto, editors, *Proceedings of the 13th International Conference on Inductive Logic Programming*, volume 2835 of *Lecture Notes in Artificial Intelligence*, pages 180–196. Springer-Verlag, 2003.
11. J. M. Kubica, A. Moore, and J. Schneider. Tractable group detection on large link data sets. In *The Third IEEE International Conference on Data Mining*, pages 573–576. IEEE Computer Society, November 2003.
12. U. Pompe and I. Kononenko. Naive Bayesian classifier within ILP-R. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 417–436. Department of Computer Science, Katholieke Universiteit Leuven, 1995.
13. J. R. Quinlan. Boosting first-order learning. *Algorithmic Learning Theory, 7th International Workshop, Lecture Notes in Computer Science*, 1160:143–155, 1996.
14. V. Santos Costa, D. Page, M. Qazi, and J. Cussens. CLP($\mathcal{BN}$): Constraint Logic Programming for Probabilistic Knowledge. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI03)*, pages 517–524, Acapulco, Mexico, August 2003.
15. R. C. Schrag. EAGLE Y2.5 Performance Evaluation Laboratory (PE Lab) Documentation Version 1.5. Internal report, Information Extraction & Transport Inc., April 2004.
16. A. Srinivasan. *The Aleph Manual*, 2001.
17. A. Srinivasan and R. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. In S. Muggleton, editor, *Proceedings of the Sixth Inductive Logic Programming Workshop*, LNAI 1314, pages 89–104, Berlin, 1997. Springer-Verlag.