Boosting Descriptive ILP for Predictive Learning

Ning Jiang and Simon Colton

Department of Computing, Imperial College, London
{nj2, sgc}@doc.ic.ac.uk

1 Introduction

Inductive Logic Programming has been very successful in application to multi-relational predictive tasks. Sophisticated predictive ILP systems, such as Progol and FOIL, can achieve high predictive accuracy, while the learning results remain understandable. Although boosting [1] is an established method to promote predictive accuracy of weak algorithms, there have been relatively few efforts to apply it to ILP systems. Some studies include [2], which applied AdaBoost to the FFOIL ILP system, and [3], in which MOLFEA, a domain-specific ILP system, was used as the base learner for AdaBoost. While these studies showed that predictive accuracy of ILP can often be increased by boosting, there is still much room for improvement.

In particular, the run-time performance of ILP systems becomes an issue because AdaBoost has to invoke them many times to produce base classifiers. This prevents boosting from running more iterations to achieve higher predictive accuracy. Also, base classifiers from these ILP systems tend to be fairly accurate, which causes boosting to converge quickly, hence it is liable to overfitting, particularly on noisy datasets. Moreover, boosting needs to apply a weighting over training examples when the base algorithm is invoked. As ILP systems are usually not able to handle weighted examples, resampling is adopted, in which low weighted examples may be lost. To attempt to overcome these weaknesses, we have investigated the use of boosting with descriptive ILP systems, which generate first-order classification rules from training data in a class-blind manner. Using two methods to convert clauses from a descriptive ILP method to classifiers, we present the results of this approach for three bioinformatics datasets.

2 Boosting Descriptive ILP

In contrast to predictive ILP systems which learn a target concept from labelled examples, a descriptive ILP system requires no class labels, and produces many first-order classification rules to characterize given examples. Descriptive ILP systems include $\mathcal{C}$LAUDIEN [4] and HR [5]. Without any limitation, descriptive ILP systems generate an excessive number of classification rules. To avoid this problem, descriptive ILP systems typically search only a specific type of rule specified by an explicit language bias, and some additional constraints involving accuracy and coverage may be applied.

Given $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{+1, -1\}$
Initialize $d_1(x_i) = 1/m$ for each example x_i
Generate candidate base classifiers Γ from descriptive ILP
For $t = 1, \ldots, T$:
- select $h_t(x) \in \Gamma$ to minimize $\epsilon_t = \sum_i h_t(x_i) y_i d_t(x_i)$
- let $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
- update $d_{t+1}(x_i) = d_t(x_i) \exp(-\alpha_t y_i h_t(x_i))$

Output the final classifier: $\mathrm{H}(x) = \mathrm{sign}\left(\sum_t \alpha_t h_t(x)\right)$

Fig. 1. The boosting descriptive ILP algorithm

For predictive learning tasks, we convert first-order classification rules from descriptive ILP into binary classifiers according to their evaluation for each example, as described below. These classifiers are then used as candidate base classifiers for boosting, as portrayed in figure 1. An intuitive means to convert a classification rule into a classifier is based on its truth-value for each example. This is also the means adopted in other attempts to combine ILP with propositional learning systems [2, 3]. In this case, supposing that $R(X)$ is a classification rule, then the corresponding classifier is defined as:

$$f(x_i) = \begin{cases} +1 & \text{if } R(x_i) \text{ is true} \\ -1 & \text{otherwise} \end{cases} \tag{1}$$

where $R(x_i)$ is the substitution gained by replacing X with specific example x_i.

Instead of this simple method, we propose a different rule conversion based on the number of ground instantiations of the classification rule. The classifier of $R(X)$ is defined as:

$$f(x_i, \beta) = \begin{cases} +1 & \text{if } \mathit{ins}(R(x_i)) \leq \beta \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

where β is a non-negative integer and $\mathit{ins}(R)$ is a function to calculate the number of ground instantiations of a clause R with respect to the background knowledge.

3 Application to Bioinformatics Datasets

We performed experiments with three bioinformatics datasets: mutagenicity [6], DSSTox [7] and carcinogenicity [8]. In each case, the goal is to predict the activity of a biochemical characteristic of molecules from the structures and physical properties of them. Although there are additional background theories in these datasets, the language of classification rules was restricted to simple conjunctions of *atom* and *bond* literals which describe a fragment of molecule structure including 1 or 2 atoms. We used a novel descriptive ILP method, the details of which are presented in the full version of this paper.

After receiving all classification rules from descriptive ILP, classification rules are turned into classifiers using either formula (1) or (2). In the case of using

Dataset	$f(x_i)$		$f(x_i, \beta)$	
	base	boosted	base	boosted
Mutagenicity	0.6646	0.7655	0.8406	0.9041
DSSTox	0.6301	0.6614	0.6787	0.7309
Carcinogenicity	0.5385	0.5128	0.5128	0.4102

Table 1. Accuracy of the proposed algorithm on the three datasets, using the either $f(x_i)$ and $f(x_i, \beta)$ rule conversion method as defined in formula (1) and (2).

(2), $ins(R(x_i))$ of a classification rule $R(x)$ can be understood as the number of occurrences of the fragment $R(x)$ in a molecule x_i. We employed 10-fold cross validation to estimate generalization performance of the methods. The number of loops in AdaBoost was chosen to minimize the error for a validation set.

Table 1 shows the results for our two different rule conversion methods. For both cases, the highest test accuracy over all base classifiers is compared against the test accuracy for the boosted classifier. We see that for mutagenicity and DSSTox, boosting increased the test accuracy and the second rule conversion performed better than the first. While our method caused serious overfitting on the carcinogenicity dataset, accurate classifiers were observed in different stages of AdaBoost. A more sophisticated validation method may help choose a proper parameter for boosting.

Comparing with standard ILP systems, the boosted approach is competitive for the mutagenicity and DSSTox datasets, which is encouraging. We provide a detailed comparison of results in the full version of this paper. Future work in choosing a proper parameter for boosting is necessary to prevent the method from overfitting. We also intend to experiment with richer background knowledge and alternative language biases to attempt to improve the boosting approach.

References

1. Schapire, R.: The boosting approach to machine learning: An overview. In: MSRI Workshop on Nonlinear Estimation and Classification, 2001.
2. Quinlan, J.R.: Boosting first-order learning. In: Algorithmic Learning Theory, 7th International Workshop, ALT '96. Volume 1160., Springer 143–155, 1996.
3. Kramer, S.: Demand-driven construction of structural features in ILP. Lecture Notes in Computer Science **2157** 132–141, 2001.
4. Deraedt, L., Dehaspe, L.: Clausal discovery. Machine Learning **26** 99–146, 1997.
5. Colton, S., Muggleton, S.: Mathematical applications of Inductive Logic Programming. Machine Learning (2006 - online first).
6. Srinivasan, A., Muggleton, S., Sternberg, M.J.E., King, R.D.: Theories for mutagenicity: A study in first-order and feature-based induction. Artificial Intelligence **85**(1-2) 277–299, 1996.
7. Muggleton, S.H., Lodhi, H., Amini, A., Sternberg, M.J.E.: Support Vector Inductive Logic Programming. Innovations in Machine Learning, 2006.
8. Srinivasan, A., King, R.D., Muggleton, S., Sternberg, M.J.E.: Carcinogenesis predictions using ILP. In: Proceedings of the 7th International Workshop on Inductive Logic Programming, 273–287, 1997.