Boosting Descriptive ILP for Predictive Learning in Bioinformatics

Ning Jiang and Simon Colton

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

Abstract. Boosting is an established propositional learning method to promote the predictive accuracy of weak learning algorithms, and has achieved much empirical success. However, there have been relatively few efforts to apply boosting to Inductive Logic Programming (ILP) approaches. We investigate the use of boosting descriptive ILP systems, by proposing a novel algorithm for generating classification rules which searches using a hybrid language bias/production rule approach, and a new method for converting first-order classification rules to binary classifiers, which increases the predictive accuracy of the boosted classifiers. We demonstrate that our boosted approach is competitive with normal ILP systems in experiments with bioinformatics datasets.

1 Introduction

Inductive Logic Programming (ILP) has been very successful in application to relational predictive tasks. Sophisticated predictive ILP systems, such as Progol [1] and FOIL [2], can achieve high predictive accuracy, while the learning results remain understandable. To achieve higher predictive accuracy, there have been attempts to combine ILP with propositional learning algorithms, such as Support Vector Machines [3]. While the predictive accuracy of such systems can be better than ILP systems, the learning results can be less understandable due to the complex representations employed.

Boosting [4] is an established method to increase the predictive accuracy of other learning algorithms, which are known as base learners. The result of boosting is a weighted sum of the predictions of the classifiers received from the base learner, and therefore can be easily understood. Although boosting has many advantageous characteristics, there have been relatively few efforts to apply it to ILP systems. Some studies include [5], which applied AdaBoost [4] to the FFOIL ILP system, and [6], in which MOLFEA, a domain-specific ILP system, was used as the base learner for AdaBoost. While these studies showed that the 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 generated by 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 learner is invoked, and it expects that the base learner can minimise the weighted training error instead of the normal one. 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. In order to control the generation of classification rules, we have introduced a novel descriptive ILP system that employs a declarative language bias which in turn enables a new method to convert classification rules to binary classifiers. We present the results of this approach for four bioinformatics datasets, and show that our method is competitive with state of the art ILP systems.

This paper is structured as follows. Section 2 gives a brief introduction to descriptive ILP and boosting algorithms. An overview of our boosted descriptive ILP approach is given at the beginning of section 3, followed by the details of the language bias, rule conversion and boosting steps. The benefits of combining boosting with ILP is also explained. Our experiments with bioinformatics datasets are described in section 4, and we describe some directions for further work in section 5.

2 Background

2.1 Boosting

Boosting is a machine learning algorithm that attempts to increase the predictive accuracy of a weak learning algorithm (known as a base learner) by aggregating multiple classifiers from it (known as base classifiers). Early studies of boosting were motivated by Kearns and Valiant's research on the PAC learning model [7]. The most widely used boosting algorithm, AdaBoost, was introduced by Freund and Schapire [4]. AdaBoost is simple to implement, and has many favourable characteristics. In particular, while the learning algorithm is understood as a stepwise optimisation [8] in training accuracy, its generalisation error is efficiently bounded by margins independent of the number of base classifiers [9, 10]. The effectiveness of AdaBoost at minimising margins was observed in early experiments: the generalisation error often keeps dropping even after the training error reaches zero. However, it was later found that AdaBoost does overfit sometimes, especially when the data is noisy [11]. A strong connection between AdaBoost and logistic regression was also discovered [12], which showed that both algorithms essentially solve the same constrained optimisation problems.

The AdaBoost algorithm tries to construct an accurate combining classifier via a weighted majority vote of base classifiers. The base classifiers are obtained by repeatedly calling the base learner, which is supplied with a weighting that affects the evaluation of training errors of the base learner. Each time it is called,

the base learner is applied to the training examples and returns the classifier which minimises the weighted training error. AdaBoost then chooses a weight for the received base classifier according to the weighted training error and updates the weighting of training examples such that the total weight of correctly classified examples are the same as that of misclassified examples. This process is repeated until AdaBoost has received a specified number of base classifiers – a typical setting of 200 to 300 base classifiers is widely used. The final classifier is the weighted sum of all of the received base classifiers.

2.2 Descriptive ILP

In contrast to predictive learning, which learns a target concept from labelled examples, a descriptive learning system requires no class labels when performing non-predictive learning tasks such as association rule learning and frequent pattern discovery. Descriptive ILP systems often perform learning from interpretations [13], assuming each training example is an independent set of ground facts and using coverage tests to validate candidate rules or patterns. Such patterns are referred to as *classification rules* in this paper, as each rule specifies a binary classification of objects according to the truth-value. Without any limitation, descriptive ILP systems search over an excessively large rule space, and this may require an impractically long time to finish. To avoid this problem, searching is often limited to a specific type of rule specified by an explicit declarative *language bias*. Well known descriptive ILP systems include $\mathcal{C}$LAUDIEN [14] and HR [15].

$\mathcal{C}$LAUDIEN performs characterising induction on positive examples to produce classification rules which characterise training examples. To restrict the language to search over, $\mathcal{C}$LAUDIEN employs the $\mathcal{D}$LAB language bias. $\mathcal{D}$LAB defines the syntax of association rules by using a grammar that has the expressive power of a regular expression, but with a more convenient notation.

We refer to this type of language bias as a *syntactical language bias*. In contrast, other ILP systems use a *constructive language bias*, which operates by repeatedly applying production rules to existing classification rules to construct new ones. Note that in a syntactical language bias, production rules are used differently, namely to develop an intermediate rule into either another intermediate rule or a classification rule (as is the case in a context-free grammar). An important difference between the two types of language biases is that constructive language biases typically allow for recursive language definitions, producing infinite language spaces and usually requiring classification rules to meet other constraints, such as the maximum number of literals in a rule. In contrast, syntactical language biases generally do not take recursive definitions, and produce a finite search space.

HR is a descriptive ILP system that performs automated theory formation via a constructive language bias [15]. Starting from a set of initial classification rules provided as background knowledge, HR repeatedly applies a set of production rules to develop an existing rule or combine two existing rules. For instance, the *compose* rule makes conjunctions of two existing classification rules, while

the *split* rule instantiates some variables in an existing classification rule. HR employs a weighted sum – with weights provided by the user – of measures of interestingness to guide the search for classification rules.

3 Boosting Descriptive ILP

Our boosted descriptive ILP approach is composed of three steps:

Rule generation. In this step, a new descriptive ILP system, *WeakILP*, is used to produce a set of first order classification rules, which are specified in a syntactical language bias. The rules may have to meet certain criteria with respect to training examples and background knowledge.

Rule conversion. In this step, the received classification rules are converted into binary classifiers, from which boosting chooses base classifiers. Different rule conversion methods may be used, as discussed in section 3.2.

Boosting. In this step, we use an adaptation of AdaBoost to choose some classifiers to aggregate into the boosted classifier. Instead of specifying the number of base classifiers in advance, we employ cross validation sets to determine when to stop adding base classifiers.

Compared to existing boosted ILP approaches, our descriptive ILP based approach has certain advantages. Firstly, the new framework separates the ILP and boosting steps, which avoids the necessity of resampling weighted examples. Hence, the boosting step can handle weightings of examples more accurately. Secondly, the learning process is more efficient, because descriptive ILP is invoked only once. This enables boosting to run as many rounds as necessary without significant increase in computational time. In previous boosted ILP experiments, such as [5], the number of base classifiers was set to between 10 and 20, whereas the typical setting in our experiments is between 50 and 200 base classifiers. In general, this means that a higher predictive accuracy can be achieved. Thirdly, although some predictive ILP systems may produce multiple classification rules at once, descriptive ILP can make better use of boosting. Boosting's performance is conditional on the ability of the base learner to return a proper[1] base classifier for arbitrarily weighted training examples. As descriptive ILP does not have a target concept to learn, classification rules from it describe a wide variety of classifications of training examples. Hence, rules from a descriptive ILP system are much more likely to fit different weightings than those from predictive ILP, which enables boosting to perform properly.

3.1 Rule generation

Our approach employs descriptive ILP to exhaustively generate classification rules, regardless of accuracy. In contrast to predictive ILP which produces only

[1] The weighted training error of a proper base classifier must be less than 50% for binary learning tasks.

a few of the most accurate rules, the number of rules generated in this step may be quite large. Hence, to improve efficiency, it is essential to have an expressive language bias to specify a language where no rule is irrelevant to the learning target.

As an example, a typical learning task in bioinformatics is to predict a certain biological characteristic of a molecule given its structural information, usually atoms in the molecule and bonds between atoms. For simplicity, we assume the predicates are of the form: $atom(X, A, E)$ and $bond(X, A, B)$, where X is the unique identifier of a molecule, A and B are atoms, and E represents the element type of atom A. Continuing with the example, suppose that domain experts believe that the biological characteristic is determined by linear (i.e., non-cyclic) connected substructures of the molecule. In this case, we need a language which specifies a sequence of atoms of any type and any length.

This is not straightforward to specify with existing syntactical language biases, as the maximum number of variables in the rule is indefinite due to the indefinite length of the sequence. Hence, because existing syntactical language biases do not allow recursive production rules, to cover all rules in this language, we have to use a more powerful language specification. However, existing constructive language biases also have difficulty to restrict rules to non-cyclic sequences of atoms, and often an excessive number of irrelevant rules are produced. Our solution has been to develop a new light-weight descriptive ILP system, called WeakILP, which uses a syntactical language bias of more expressive power. WeakILP allows recursive production rules and can employ a novel rule conversion approach.

Language bias

A classification rule in WeakILP has the following form:

$$rule(X, \{X_1, X_2, \ldots\}) : \ Body$$

where X represents the object to classify (i.e., a training or test set example), $\{X_1, X_2, \ldots\}$ is a set of *key variables* which occur in the $Body$, which is a well-formed formula in first-order logic[2], though it is often a conjunction of literals. For example, below is a rule specifying a non-cyclic sub-molecule structure:

$$rule(X, \{A, B, C\}) : atom(X, A, o) \wedge bond(X, A, B) \wedge bond(X, B, C) \wedge atom(X, C, n)$$

where X is the molecule to classify and A, B and C are strictly different atoms.

The set of key variables is referred to as a *key set*, which is used to highlight interesting properties or structures of the rule. In the above example, the variables A, B and C can help identify each unique occurrence of the substructure. To specify key sets for classification rules and allow recursive definitions in the

[2] Note that $Body$ is not theoretically restricted to first-order logic, as the language bias is simply a set of grammatical definitions. Any logic can be accepted if the produced rules can be interpreted by the runtime system. Our current implementation is based on Prolog, hence rules are restricted to Prolog queries. In the experiments presented here, all classification rules are a conjunction of literals.

language, WeakILP adopts a new syntactical language bias. The language bias defines the grammar of classification rules by using production rules.

A grammar is composed of several production rules: $A_1 \rightarrow B_1, \ldots, A_k \rightarrow B_k$, where each A_i is a positive literal or a function symbol, to be replaced by one of the formulae on the right-hand side, and B_i is a well-formed formula. Each A_i is a *nonterminal symbol*, as defined below.

Definition 1 (Terminal and Nonterminal symbols). *A* nonterminal symbol *is any positive literal which occurs on the left-hand side of a production rule, and may or may not be ground. The nonterminal symbol is in fact a placeholder, which is absent from the produced classification rules. On the other hand, literals which must occur in the produced classification rules are called* terminal symbols.

Note that, to avoid confusion, any nonterminal symbol must not be used as a terminal symbol.

A replacing formula, B_i, can be any well-formed logic formula, which may include nonterminal symbols to allow recursive definitions of rules. In particular, to allow the specification of the key set, some variables may be enclosed by the function symbol $key/1$. In the generated classification rules (which include no nonterminal symbol), if a variable occurs in a $key/1$ functor, it will be put into the key set, and the functor itself will be ignored.

The following is an example of the syntactical grammar which defines a sequence of atoms of arbitrary length for the above bioinformatics problem.

$$\begin{aligned} rule(X) &\rightarrow sequence(X, key(A) \wedge key(B)) \\ sequence(X, A, B) &\rightarrow bond(X, A, B) \\ sequence(X, A, B) &\rightarrow bond(X, A, key(C)) \wedge sequence(X, key(C), B) \end{aligned}$$

The above grammar produces classification rules, of which the key set comprises all atoms in a sequence of atoms. For instance, the grammar can produce the rule $bond(key(A), key(C)), bond(key(C), key(B))$, which is interpreted as this classification rule, which defines a sequence of three connected atoms: $rule(X, \{A, B, C\}) : \ bond(X, A, C) \wedge bond(X, C, B)$. Note that the $key/1$ functor enclosing variables A, B and C has been removed, and these variables have been put into the key set (the importance of which becomes clearer when this classification rule is interpreted as a binary classifier, as described below).

When a production rule is chosen to apply to a formula which contains nonterminal symbols, one of the nonterminals is replaced by the formula defined in the production rule. The replacing continues until there are no nonterminal symbols left. More formally, the application of a production rule to a formula including nonterminals is defined as follows:

Definition 2 (Application of production rules). *Given a well-formed formula $F(A')$, where A' is an occurrence of some nonterminal symbol in $F(A')$, a production rule $A \rightarrow B$ can be applied to A' if and only if there is a unification of A' and A. Suppose θ is the most general unifier of A' and A. Application of the production rule to A' of $F(A')$ produces a well-formed formula $F(B)\theta$.*

The generation of classification rules starts from a rule including only one nonterminal symbol, known as the start symbol: $rule(X)$. The language defined by a grammar is the set of all classification rules that contain no nonterminal symbols and can be derived from the start symbol by applying production rules. Note that, when there are recursive production rules in a grammar, the set of rules defined by the grammar may be infinite. Therefore, it is often necessary to specify a maximum length of classification rules or the maximum steps to derive a classification. Some classification rules defined by the above grammar include:

$$\begin{aligned} rule(X, \{A, B\}) &: bond(X, A, B) \\ rule(X, \{A, C, B\}) &: bond(X, A, C) \wedge bond(X, C, B) \\ rule(X, \{A, C, D, B\}) &: bond(X, A, C) \wedge bond(X, C, D) \wedge bond(X, D, B) \end{aligned}$$

Importantly, the key set can be used for counting purposes. For instance, in the above example, suppose we require variables to be instantiated into strictly different constants. In this case, we can count different ground instantiations of rules for a molecule, which gives information about the occurrences of a specific substructure. This information is then used to obtain more sophisticated base classifiers, as described in section 3.2.

The following grammar extends the above example grammar with $atom/3$ predicates to restrict the element type of atoms. For the purposes of the example, we specify that the first and last atoms in a sequence must be assigned a specific element, with other atoms being optional.

$$\begin{aligned} rule(X) &\rightarrow atomtype(X, A) \wedge sequence(X, key(A), key(B)) \wedge atomtype(X, B) \\ sequence(X, A, B) &\rightarrow bond(X, A, B) \\ sequence(X, A, B) &\rightarrow bond(X, A, key(C)) \wedge optional(X, C) \wedge sequence(X, key(C), B) \\ optional(X, A) &\rightarrow atom(X, A, \_) \\ optional(X, A) &\rightarrow atomtype(X, A) \\ atomtype(X, A) &\rightarrow atom(X, A, o) \\ atomtype(X, A) &\rightarrow atom(X, A, n) \\ atomtype(X, A) &\rightarrow atom(X, A, c) \\ atomtype(X, A) &\rightarrow atom(X, A, h) \end{aligned}$$

Continuing the example, we add production rules to the above grammar, so that the generated classification rules will include combinations of genotoxicity properties of a molecule, such as $salmonella/1$, $cytogen/1$, and $drosophila/1$.

$$\begin{aligned} rule(X) &\rightarrow properties(X, 2) \wedge atom(X, A) \\ &\quad \wedge sequence(X, key(A), key(B)) \wedge atom(X, B) \\ property(X, Last, New) &\rightarrow salmonella(X) \\ properties(X, N) &\rightarrow salmonella(X) \wedge properties1(X, M) \\ properties(X, N) &\rightarrow properties1(X, N) \\ properties1(X, N) &\rightarrow cytogen(X) \wedge properties2(X, M) \end{aligned}$$

$$properties1(X, N) \rightarrow properties2(X, N)$$
$$properties2(X, N) \rightarrow drosophila(X, N)$$

Pruning
Given a grammar as above, WeakILP exhaustively produces all classification rules in the language, except those which are true for fewer examples than requested by a user-specified minimum coverage. Such pruning is mainly to improve computational efficiency. In the case that a classification rule covers no training examples, it cannot contribute to classification, because its training accuracy could not be higher than the default classifier regardless of weighting over training examples. As a consequence, boosting does not choose a base classifier derived from a classification rule of zero coverage. Moreover, those rules that cover very few training examples may be too specific to these examples, and might not affect the training result significantly. Pruning those examples can dramatically reduce the number of classifiers that boosting has to evaluate.

3.2 Rule conversion

For predictive learning tasks, we convert first-order classification rules from descriptive ILP into binary classifiers according to their evaluation for each training example, as described below. These classifiers are then used as candidate base classifiers for boosting. We have experimented with the conventional method for performing this conversion which uses truth values. We have also experimented with a novel method which finds coefficients of instantiations, as described below.

Truth-based Conversion Method
An intuitive means for converting a classification rule into a classifier is based on its truth-value for each example, which is the method adopted in previous attempts to combine ILP with propositional learning systems [6, 5]. In this case, supposing that $R(X, K)$ is a classification rule, then the corresponding classifier is defined as:

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

where $R(x_i, K)$ is the instantiation of $R(X, K)$ gained by replacing X with a specific example x_i.

Instantiation-based Conversion Method
We also propose a different rule conversion method based on the number of ground instantiations of the classification rule. Given a classification rule $R(X, K)$, the corresponding binary classifiers are defined as:

$$f(x_i, \beta) = \begin{cases} +1 & \text{if } \left| \{K\theta \,|\, R(x_i, K)\theta \text{ is ground and true}\} \right| \geq \beta \\ -1 & \text{otherwise} \end{cases}$$

where β is a non-negative integer and θ is a ground substitution that maps variables into ground terms.

As $R(x_i, K)\theta$ is ground, the substituted key set $K\theta$ is also ground. The set $\{K\theta \mid R(x_i, K)\theta$ is ground and true$\}$ is therefore the set of all ground instantiations of the key variables that make the classification rule $R(X, K)$ true for the example x_i. The cardinality of the instantiation set counts the different key sets that make the classification rule true for the example, which can be understood as the degree to which a rule holds for an example. Note that only instantiations of the key set to strictly different ground instances are counted, and permutations of an instantiation which has been counted already are similarly not counted.

To illustrate this novel rule conversion method, we consider the above bioinformatics problem. Each rule defined in the language represents a non-cyclic sequence of atoms. The key set is composed of the *atom* variables, therefore the cardinality of the instantiation set describes the number of distinct occurrences of the sequence in a molecule. Illustrated below are 6 instantiations of this classification rule: $rule(X, \{A, B, C\}) : bond(X, A, B), bond(X, B, C)$, for a particular example x_i.

$$rule(x_i, \{a, b, c\}) : bond(x_i, a, b), bond(X, b, c) \quad (1)$$
$$rule(x_i, \{c, b, a\}) : bond(x_i, c, b), bond(X, b, a) \quad (2)$$
$$rule(x_i, \{a, b\}) : bond(x_i, a, b), bond(X, b, a) \quad (3)$$
$$rule(x_i, \{b, a\}) : bond(x_i, b, a), bond(X, a, b) \quad (4)$$
$$rule(x_i, \{b, c\}) : bond(x_i, b, c), bond(X, c, b) \quad (5)$$
$$rule(x_i, \{c, b\}) : bond(x_i, c, b), bond(X, b, c) \quad (6)$$

We note that only instantiation (1) will be counted. This is because the key set in (2) is a permutation of that in (1), and instantiations (3), (4), (5) and (6) have instantiated two variables to the same ground term, hence are not counted. Therefore, for this classification rule, the instantiation coefficient used in the second rule conversion method described above will be 1. Note that the requirement to instantiate to strictly different terms is referred to in $\mathcal{C}$LAUDIEN as injectivity. Note also that the first conversion method is clearly a special case of the second method, namely when β is set to 1, and, when β is set to 0, $f(x_i, \beta)$ is a naive classifier that gives the same positive prediction for any example.

Further Pruning

When the instantiation-based conversion is adopted, each classification rule corresponds to multiple binary classifiers with different choices of the parameter β. To improve efficiency, we prune certain classifiers. In many descriptive ILP systems, a prover is used to determine whether two rules are logically equivalent. Logically equivalent rules can be safely pruned, as they always give the same prediction with respect to the background theory. In WeakILP, we choose to prune any classifier which gives the same predictions for training examples as another, i.e., predictively equivalent classifiers. Note that such pruning does not affect the learning process of boosting, as the boosting algorithm cannot distinguish those classifiers, and might randomly (depending on the implementation) choose one of them as a base classifier when appropriate. As all logically

equivalent classifiers must also be predictively equivalent, this approach is more efficient in reducing redundant classifiers than using a logic prover.

However, because the converse statement is not true, i.e., predictively equivalent classifiers are not necessarily logically equivalent, predictively equivalent classifiers (for training examples) might give different predictions for test examples. It has been suggested in [16] that a syntactically less complex classifier tends to have better generalization performance than a more specific one. Hence, we choose to prune the more complex classifiers, i.e., in WeakILP, given a set of predictively equivalent classifiers, we take the shortest one in terms of the number of literals in the classification rule.

3.3 Boosting

Once binary classifiers are produced, the AdaBoost algorithm will be applied to construct a combining classifier from them. The boosting algorithm is presented in Figure 1, in which boosting does not invoke a separate base learner to obtain base classifiers, but, instead, evaluates received binary classifiers against weighted examples directly and chooses the one of the highest weighted accuracy.

Given $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{+1, -1\}$
Initialise $d_1(x_i) = 1/m$ 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 minimise $\epsilon_t = \sum_i h_t(x_i) y_i d_t(x_i)$
- 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, where T is the specified number of base classifiers to combine in the boosted classifier.

In experiments with boosting, the learning results are often presented on a stepwise basis, i.e., results after every step are listed. This is particularly useful to demonstrate the efficiency of boosting for improving the accuracy of the base learner. However, it is more appropriate to evaluate the generalisation performance of the learning algorithm as a whole. This is because, in practical applications, we have to choose a combining classifier produced at a particular step. Hence, we need to estimate AdaBoost's parameter, i.e., determine when to stop adding base classifiers to the boosted classifier.

For our experiments, when n-fold cross validation is used for an experiment, we use $(n-1)$-fold cross validation to evaluate the parameter on the training set. This strategy roughly maintains the size of the validation sets employed comparable to that of the test set. Hence, after each base classifier is added to the boosted classifier, we use 9-fold cross validation over the training set to determine the performance of the boosted classifier. Our system then backtracks to the

boosted classifier which performed best, and outputs this as the final result. We have found that this improves performance over the usage of a single validation set. In the case of separate training and test sets, 10-fold cross validation is typically used.

4 Experiments with Bioinformatics Datasets

We performed experiments with four bioinformatics datasets: mutagenicity [17], DSSTox [3], carcinogenicity [18], and KDD Cup 2001 [19]. For each dataset, we evaluated our method using four different settings: WeakILP with and without boosting and using both of the rule conversion methods. When WeakILP was used without boosting, we chose the most accurate classifier in terms of predictive accuracy on training examples. We performed cross validation for all datasets except the KDD Cup dataset (which has a independent test set) to estimate the generalisation performance. In the KDD Cup dataset, for pruning purposes, the minimum coverage of classification rules was set to 15 to reduce the number of classifiers produced. In all the other experiments, we used a minimum coverage of one.

- **Mutagenicity.** The mutagenicity problem, reported in [17], is one of the most widely used datasets in ILP. The task regards learning a theory of mutagenesis from a set of 188 nitroaromatic molecules, of which 125 are mutagenic (active) and 63 are non-mutagenic (non-active). The background knowledge includes atoms which occur in a molecule, bonds between the atoms, certain chemical features, structural attributes, and predefined function groups in the molecule. The language bias used for this experiment is presented in table 1. The maximum length of classification rules was set to be 4. We performed 10-fold cross validation to estimate the generalisation performance on this dataset. Table 2 gives a partial example of an output combining classifier, which achieves a predictive accuracy of 89.47% (on both training and test examples) when six base classifiers are chosen.
- **DSSTox.** DSSTox is the predictive toxicity dataset used in [3], which consists of 576 molecules. The language bias and other settings were the same as in the mutagenicity experiment, except that 5-fold cross validation was used, to be consistent with the previous study [3].
- **Carcinogenicity.** The carcinogenicity dataset includes 337 chemicals, which is composed of both training and test datasets used from a previous predictive toxicology competition. The task is to predict the cancerous activity of the chemicals. Similar settings were used in this experiment as with the mutagenicity dataset, except that the language bias allowed arbitrary combinations of genotoxicity properties and structural indicators [18].
- **KDD Cup 2001.** This competition [19] was composed of three tasks, of which we consider only the second task, the prediction of *functions* of genes. The dataset consists of 862 genes as training examples and 381 genes as test

examples. Each gene can belong to any combination of 14 classes, so we can break down the leaning task into 14 binary classification sub-tasks. We used a language bias similar to that used in [20], except that no negation was allowed.

$$
\begin{aligned}
bound\_type(X,A,B) \rightarrow\ & bond(X,key(A),key(B),1) && \text{or}\\
& bond(X,key(A),key(B),2) && \text{or}\\
& \ldots\\
atom\_type(X,B) \rightarrow\ & atom(X,B,h) && \text{or}\\
& atom(X,B,c) && \text{or}\\
& \ldots\\
connection(X,A,B) \rightarrow\ & bond\_type(X,A,B) && \text{or}\\
& bond\_type(X,A,B) \wedge atom\_type(X,B)\\
\\
sequence(X,A,B) \rightarrow\ & connection(X,A,B) && \text{or}\\
& connection(X,A,C) \wedge sequence(X,C,B)\\
\\
structure(X,A,B) \rightarrow\ & arc(X,A,B) && \text{or}\\
& arc(X,A,B) \wedge structure(X,A,B) && \text{or}\\
& arc(X,A,C) \wedge arc(X,C,B) \wedge structure(X,C,D)\\
\\
rule(X) \rightarrow\ & structure(X,A,B) && \text{or}\\
& atom\_type(X,A) \wedge structure(X,A,B)
\end{aligned}
$$

Table 1. Language bias for the mutagenicity dataset. Production rules with the same left-hand side nonterminal symbol are grouped together for ease of reading.

Acc.	$Wt.$	*Pred.*	β	Body of the classification rule
73.68%	0.82	+	16	$bond(X,A,C,1) \wedge bond(X,C,B,7)$.
73.68%	0.45	+	6	$bond(X,A,C,2) \wedge bond(X,C,D,1) \wedge bond(X,D,E,7)$ $\wedge bond(X,E,B,7)$.
73.68%	0.37	+	28	$bond(X,A,B,7)$.
73.68%	0.43	−	2	$bond(X,A,C,1) \wedge atom(X,C,o) \wedge bond(X,C,B,1)$ $\wedge atom(X,B,c)$.
78.94%	0.38	+	8	$bond(X,A,C,1) \wedge bond(X,C,B,1) \wedge bond(X,C,D,1)$ $\wedge atom(X,D,c)$.
89.47%	0.28	−	16	$bond(X,A,C,1) \wedge bond(X,C,D,7) \wedge bond(X,D,B,1)$ $\wedge atom(X,B,h)$.

Table 2. The first six base classifiers of a boosted classifier for the mutagenicity dataset. *Acc.* represents the test accuracy of the corresponding combining classifier. *Wt.* is the weight assigned to the corresponding base classifier. *Pred.* is the prediction of the base classifier, which may be either active (+), or non-active (-). We only give the body of the classification rule, as all variables except X are key variables. The classifier is read as: *the molecule is* Pred. *if the number of ground instantiations is equal to or greater than the threshold,* β.

Rule conversion	Truth-value based		Instantiation based	
Boosting	WeakILP	boosted	WeakILP	boosted
Mutagenicity	66.5%	76.6%	80.9%	90.5%
DSSTox	63.0%	66.1%	68.4%	75.6%
Carcinogenicity	58.4%	57.5%	58.7%	61.1%
KDD Cup 2001	90.5%	91.8%	90.5%	91.8%

Table 3. Test accuracy or estimated generalization accuracy of WeakILP and boosted WeakILP for the four datasets, using truth-value and instantiation based rule conversion methods.

[t]

Category	Method	Mutagenesis	Carcinogenesis	DSSTox ††	KDD Cup 2001
ILP	Progol	88.0%		55.0%	
	FOIL	86.7%			
	STILL	93.6%‡			
	ICL	88.3%			92.2%
	Aleph	88.8%	57.9%		
Kernel	SVILP			73.0%	
based	CHEM			58.0%	
	MIK	93.0%		60.0%	
	PLS			71.0%	
Bagging/	RS	95.8%			
Boosting	Bagging Aleph		64.0%†		
Based	Boosted FFOIL	88.3%			
	Boosted WeakILP	90.5%	61.1%	59.3%	91.8%
Others	RELAGGS	88.0%			93.0%

Table 4. The comparison of our boosted WeakILP approaches with other state of the art systems. The results for the mutagenicity dataset are mostly taken from [21] and the results for DSSTox are from [3]. RELAGGS was the winner of KDD Cup 2001 task 2 [19] and ICL results are collected from [20]. Aleph results are based on [22]. It is worth noting that many experiments were done in different settings, including different background knowledge and performance estimation. † This result has large variations between 60% and 64% in different bagging steps. ‡ The STILL experiment did not perform 10-fold cross validation, but held 10% examples back as test examples. †† All other methods except boosted WeakILP had access to background knowledge which is not currently available in the public domain.

Table 2 shows a typical training result from our boosted WeakILP experiments. We found most weightings concentrate on the first few base classifiers and if we reduce the boosted classifier to ten base classifiers, in most cases, the result is still fairly accurate but much simpler. Hence, the training result can be made more understandable at a minor cost to predictive accuracy. Table 3 lists the predictive accuracies of WeakILP and boosted WeakILP using both rule conversion methods. The result shows that with only one exception, boosting is able to improve the generalization performance of WeakILP, and the improvement is more than 10% in two out of four datasets. We also observed that the new instantiation-based rule conversion method resulted in better test accuracy with no exceptions, and when used together with the boosted WeakILP approach, it always produces better predictive accuracy than with other settings. We compare

our results (using boosted WeakILP with the instantiation based rule conversion) with other methods in Table 4. The predictive accuracy we achieve is in line with the top ranking approaches for all experiments except DSSTox. Note that all the other methods had access to certain more sophisticated background information for the DSSTox dataset, which was not available to us. Our experiments involved minimum use of background knowledge, and we hope to improve our results by using more background knowledge.

5 Conclusions and Further Work

We have explored the use of boosted descriptive ILP for predictive learning tasks, and presented some experimental results for bioinformatics datasets. The main contributions of our study include the following:

- We distinguish two types of language biases, namely constructive and syntactical, and we highlight the limitation of existing language biases.
- To take advantage of both types of language biases, we suggest a new declarative language bias, used in our WeakILP system. The new language bias adopts a context-free style grammar to define languages, which is more expressive than $\mathcal{D}$LAB in that it allows recursive definitions of the language.
- We have proposed a new propositionalization technique, which counts the ground instantiations of a logic rule to indicate the degree that the classification rule supports its prediction. This approach has been shown to be effective in some learning tasks, in which both training and test accuracies have been improved significantly.
- We have shown that the boosted WeakILP approach performs well with four bioinformatics datasets, and it outperforms many widely used ILP systems. Hence, there is some evidence that the boosted WeakILP approach is competitive with state of the art ILP systems in terms of predictive accuracy. This is encouraging, especially given that the learning results are understandable compared to other propositionalization based methods.

In further work, we plan to perform further experiments to investigate both runtime and predictive performance of the proposed approach in non-bioinformatics domains. Also, further theoretical and experimental studies are necessary to compare the performance of boosting with other machine learning methods including SVM and logistic regression. Moreover, we aim to find a theoretical explanation to answer questions about when our approach is suitable and when it is not.

Acknowledgments

The authors thank Stephen Muggleton for making the DSSTox dataset available and the three reviewers, whose comments and references provided important information to improve the quality of this paper.

References

1. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming **13**(3-4) (1995) 245–286
2. Quinlan, J., Cameron-Jones, R.: FOIL: A midterm report. In: Machine Learning: ECML-93, European Conference on Machine Learning. Volume 667. (1993) 3–20
3. Muggleton, S., Lodhi, H., Amini, A., Sternberg, M.: Support vector inductive logic programming. In Holmes, D., Jain, L.C., eds.: Innovations in Machine Learning: Theory and Applications. Springer-Verlag (2006)
4. Schapire, R.: The boosting approach to machine learning: An overview. In: MSRI Workshop on Nonlinear Estimation and Classification. (2001)
5. Quinlan, J.: Boosting first-order learning. In: Algorithmic Learning Theory, 7th International Workshop, ALT '96. Volume 1160. (1996) 143–155
6. Kramer, S.: Demand-driven construction of structural features in ILP. Lecture Notes in Computer Science **2157** (2001) 132–141
7. Valiant, L.: A theory of the learnable. Communications of the ACM **27**(11) (1984) 1134–1142
8. Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. Technical report, Dept. of Statistics, Stanford University (1998)
9. Meir, R., Ratch, G.: An introduction to boosting and leveraging. In: Advanced lectures on machine learning, Springer-Verlag (2003) 118–183
10. Schapire, R., Freund, Y., Bartlett, P., Lee, W.: Boosting the margin: a new explanation for the effectiveness of voting methods. In: 14th International Conference on Machine Learning. (1997) 322–330
11. Jin, R., Liu, Y., Si, L., Carbonell, J., Hauptmann, A.: A new boosting algorithm using input-dependent regularizer. 20th International Conference on Machine Learning (2003)
12. Lebanon, G., Lafferty, J.: Boosting and maximum likelihood for exponential models. Advances in Neural Information Processing Systems **15** (2001)
13. Raedt, L., Džeroski, S.: First-order jk-clausal theories are pac-learnable. Artif. Intell. **70**(1-2) (1994) 375–392
14. Deraedt, L., Dehaspe, L.: Clausal discovery. Machine Learning **26** (1997) 99–146
15. Colton, S., Muggleton, S.: Mathematical applications of inductive logic programming. Machine Learning **64**(1-3) (2006) 25–64
16. Muggleton, S.: Learning from positive data. In Muggleton, S., ed.: ILP '96. Volume 1314 of LNAI., Springer-Verlag (1996) 358–376
17. Srinivasan, A., Muggleton, S., Sternberg, M., King, R.: Theories for mutagenicity: A study in first-order and feature-based induction. Artificial Intelligence **85**(1-2) (1996) 277–299
18. Srinivasan, A., King, R., Muggleton, S., Sternberg, M.: Carcinogenesis predictions using ILP. In: ILP 97. Volume 1297. (1997) 273–287
19. Cheng, J., Hatzis, C., Hayashi, H., Krogel, M., Morishita, S., Page, D., Sese, J.: KDD cup 2001 report. SIGKDD Explorations **3**(2) (2002) 47–64
20. Laer, W.: From Propositional to First Order Logic in Machine Learning and Data Mining - Induction of first order rules with ICL. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven (2002)
21. Lodhi, H., Muggleton, S.: Is mutagenesis still challenging? In: 15th International Conference on Inductive Logic Programming. (2005) 35–40
22. Zelezny, F., Srinivasan, A., Page, D.: Lattice-search runtime distributions may be heavy-tailed. In: The Twelfth International Conference on Inductive Logic Programming. (2002)