Boosting First-Order Learning

J. R. Quinlan

University of Sydney, Sydney N.S.W. 2006, Australia

Abstract. Several empirical studies have confirmed that boosting classifier-learning systems can lead to substantial improvements in predictive accuracy. This paper reports early experimental results from applying boosting to FFOIL, a first-order system that constructs definitions of functional relations. Although the evidence is less convincing than that for propositional-level learning systems, it suggests that boosting will also prove beneficial for first-order induction.

1 Introduction

Designers of induction systems are concerned with many aspects of the systems' performance, but the predictive accuracy of the theories that they learn is always high on the list. When a new technique offers the prospect of substantially increasing this accuracy at the cost of additional computation, it tends to receive a lot of attention!

This is certainly the case with *boosting*, a technique that allows a learning system to produce more accurate classifiers. Since it emerged from the theoretical side of the machine learning community, boosting has been enthusiastically taken up and investigated by empiricists. Early results indicate that it works well with most classifier-learning systems, the only known exceptions being those (such as instance-based methods) that are very *stable* – i.e., where small changes to the training data do not appreciably alter the classifier learned [4].

This paper explores the use of boosting beyond classifier-learning systems. After a discussion of boosting and an illustration of its usefulness with an attribute-value learner, the paper presents FFOIL, a system for learning Horn clause definitions of functional relations, and shows that it too can be boosted. Early results on relational learning tasks suggest that boosting should also prove beneficial for first-order induction.

2 Boosting

This technique for improving the performance of classifier-learning systems depends, like *bagging* [3], on the effect of aggregation. The system is invoked repeatedly and the learned classifiers are assembled into a composite classifier that usually has a higher predictive accuracy than any of its components.

The boosting procedure used in this paper follows[1] Freund and Schapire's *AdaBoost.M1* [9]. We assume a set of training instances $i = 1,2,...,N$. At each

[1] With some minor departures regarding the voting scheme.

repetition or *trial* $t = 1,2,...,T$, instance i has weight w_i^t, where $w_i^1 = 1/N$ for all i. The process at trial t can be summarised as follows:

- The learning system constructs classifier C^t from the training instances using the weights w_i^t.
- The error rate ϵ^t of this classifier on the training data is determined as the sum of the weights w_i^t of the misclassified instances.
- If $\epsilon^t = 0$ or $\epsilon^t \geq 1/2$, the process terminates. Otherwise the weights w_i^{t+1} for the next trial are set so that the error rate of C^t under weights w_i^{t+1} is exactly $1/2$. Freund and Schapire accomplish this by multiplying the weight of each correctly classified instance by $\epsilon^t/(1-\epsilon^t)$ and renormalising. Alternatively,
$$w_i^{t+1} = w_i^t \div \begin{cases} 2\epsilon^t & \text{if } C^t \text{ misclassifies instance } i \\ 2(1-\epsilon^t) & \text{otherwise.} \end{cases}$$

The composite classifier C^* is obtained by voting its component classifiers $\{C^t\}$. If C^t classifies some instance x as belonging to class k, the total vote for k is incremented by $\log((1-\epsilon^t)/\epsilon^t)$. C^* then assigns x to the class with the greatest total vote.

Freund and Schapire prove that the error rate of C^* on the unweighted training instances approaches zero exponentially quickly as T increases. Provided that the learning system can reliably generate from weighted instances a hypothesis that has less than 50% error on those instances, a sequence of "weak" classifiers $\{C^t\}$ can be boosted to a "strong" classifier C^* that is at least as accurate as, and usually more accurate than, the best weak classifier.

Some learning systems cannot readily employ weighted training instances. For these, the training set for trial $t = 2,3,...,T$ can be established by drawing from the instances a resample of size N, using w_i^t as the probability of selecting instance i at each draw. This is equivalent to approximating the weights w_i^t by another weight vector, all of whose components are multiples of $1/N$.

3 Boosting C4.5

The exponential reduction of error mentioned above applies to the *training* instances; there is no guarantee that the composite classifier C^* will more accurately classify unseen instances. Nevertheless, several authors have reported a sizeable reduction of classification error on test instances when boosting is applied to decision tree learning systems. For example, Breiman [4] describes experiments with a number of well-known learning tasks in which boosting leads to substantially improved performance.

This section summarises results from boosting C4.5 [16]; details may be found in [18]. The intention is to provide a yardstick against which to measure the effect of boosting on first-order learning, and also to note some interesting aspects of boosting decision trees.

C4.5 allows instances to have weights, so it was relatively easy to prepare a version incorporating boosting as above. One modification was found to lead

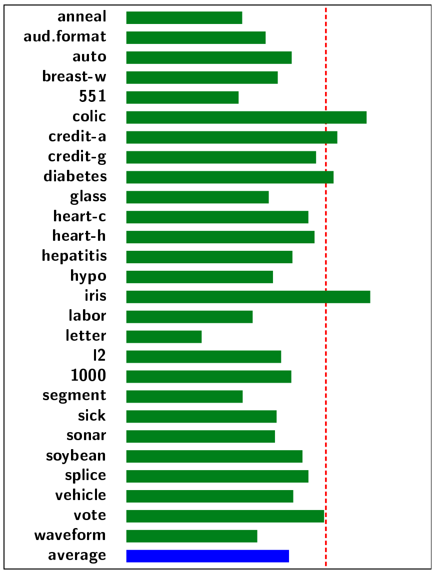

Fig. 1. Summary of boosting results

to higher classification accuracy: rather than using a constant voting weight for each classifier C^t, this weight was allowed to vary according to the confidence with which C^t classified each instance. The experiments employed a representative collection of 27 datasets from the UCI Machine Learning Repository that show considerable diversity in size, number of classes, and number and type of attributes. Ten complete 10-fold cross-validations were carried out with each dataset.[2] The parameter T governing the number of trials was set at the relatively low value of 10, so that constructing a composite classifier C^* required one order of magnitude more computation than building a single tree.

For each dataset, the boosted classifiers' average error rate on unseen instances was compared to the same rate for the unboosted classifiers. The ratios appear in Figure 1 in which the dotted line indicates a value of 1. Boosting leads to improved performance on 23 of the 27 datasets and reduces average error on unseen instances across all datasets by just under 20%. For the largest dataset (*letter*, with 20,000 instances, 16 attributes and 26 classes), boosting reduces this error rate from 12.0±0.1% to 4.5±0.1%.

These results confirm that boosting can be a powerful technique for improving the classification performance of learning systems. However, further investigations revealed aspects of boosting that are not evident from its underlying

[2] In a 10-fold (stratified) cross-validation, the training instances are partitioned into 10 equal-sized blocks with similar class distributions. Each block in turn is then used as test data for the classifier generated from the remaining nine blocks.

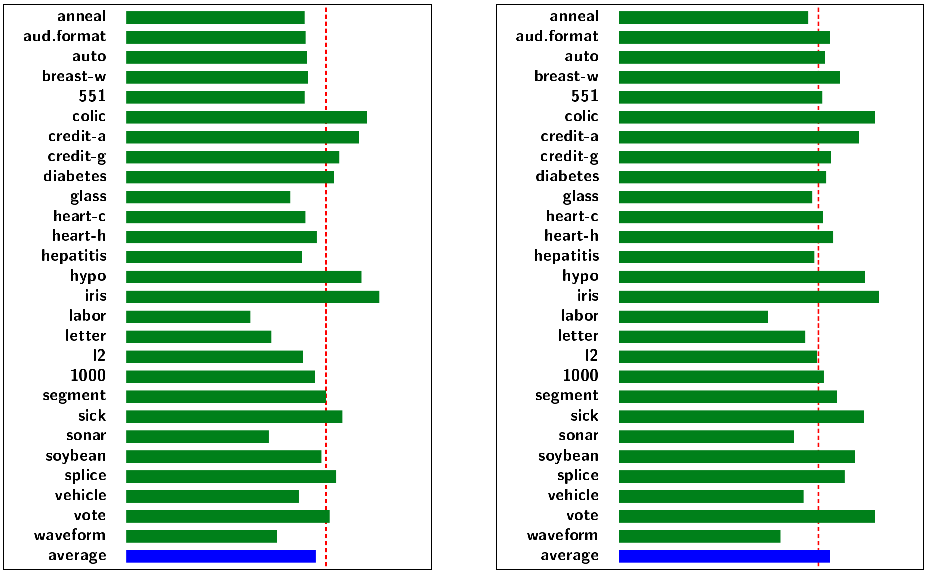

Fig. 2. Effect of 5% and 15% classification noise

theory:

- The stated objective of boosting is to reduce error on the training set. While the boosting trials stop if any C^t correctly classifies all the training instances, it may be that, after t trials, C^* has zero error even though C^t does not. Continuing with further trials would seem to be pointless since the error rate of C^* cannot be reduced further. However, if boosting is modified to stop at this point, (requiring only three trials on average for most of these datasets), the boosted classifiers show negligible improvement on unseen instances.
- Since boosting involves aggregating classifiers through voting, the boosted classifiers would be expected to be more robust in the presence of noise. This seems not to be the case – the advantage of boosting can be eliminated and even negated when the data is subject to non-trivial classification noise. Such noise can be added artificially by changing the given class of p% of the instances to some other class. The datasets were corrupted in this way and the experiments repeated; outcomes for two values of p are shown in Figure 2. Most of the advantage of boosting is lost with only 5% classification noise, while boosting degrades performance on 19 of the 27 datasets when p reaches 15%.

The reason that boosting increases classification accuracy on unseen instances may be unrelated to its proved convergence properties on the training data. Breiman [4] notes that, since boosting causes the previous classifier to have a 50% error rate on the reweighted training instances, the learning system is forced to produce a quite different classifier on the next trial. Boosting thus leads to more extensive exploration of the space of possible classifiers. Of course, these findings and speculations may apply only to systems that learn decision trees, but it will be interesting to see whether they have more general validity.

We now leave the area of propositional-level classification and move to first-order learning, the primary subject of this paper.

4 Overview of FFOIL

General first-order learning systems such as FOCL [14], FOIL [15, 5], GRENDEL [6], and PROGOL [13] try to find a clause-form definition of a *target* relation R using background relations $\{B_i\}$. R is specified extensionally by ground tuples of constants that belong in the target relation; the B_i's are specified similarly, although some systems permit them to be defined intensionally by clauses. Most systems require additional information such as tuples known not to belong to R, constraints on the form that the definition of R may take, or information about the modes of the arguments of the background relations.

On the other hand, more specialised systems such as DOLPHIN [21], FILP [2], and GOLEM [12] exploit some property of a class of learning tasks in order to improve their efficiency and/or effectiveness on such tasks. Examples of these classes include learning guard clauses for logic programs, learning definitions of functional relations, and learning definitions consisting of determinate clauses.

FFOIL [19] belongs in this second group, being an adaptation of FOIL for functional relations (and so strongly related to the work of Bergadano and Gunetti [2]). A target relation $R(V_1, V_2, ..., V_n)$ is *functional* if, for any ground values of v_1, v_2, ..., v_{n-1} of V_1, V_2, ..., V_{n-1}, there is a single value v_n for V_n such that $\langle v_1, v_2, ..., v_n \rangle$ belongs to R. The last variable V_n of a functional target relation will be referred to as the *output* variable. Relations of this kind are common and have several distinguishing characteristics:

- Queries to be answered by the learned definition cannot be assumed to be ground. For example, if $mult(A, B, C)$ denotes the functional relation $A \times B = C$, a sensible query is not "Is $4 \times 3 = 12$?", but rather "What is 4×3?".
- Systems that perform general relational learning do not necessarily bind every literal in the clause head by a body literal. In the case of functional relations, however, the output variable must be bound somewhere in the clause body in order to answer queries of this form.
- Such a query has only one correct answer. If a ground tuple $\langle v_1, v_2, ..., v_n \rangle$ belongs to the target relation R, then changing v_n is guaranteed to give tuples that do not belong to R. This fact can be used to generate negative examples without them having to be provided explicitly, and without recourse to the closed world assumption [1]. Similarly, no further clauses should be investigated after a clause succeeds, i.e. each clause should end with a cut. This implies that a learned definition consists of an ordered sequence of clauses instead of the unordered clauses generated by most general first-order induction systems.

```
/* Initialization */
definition := null program
remaining := all tuples belonging to target relation R

/* Construct definition */
While remaining is not empty
    /* Grow a new clause */
    clause := R(V1, V2, ..., Vn) :-
    While clause does not bind output or makes errors
        /* Specialize clause */
        Find appropriate literal(s) L
        - if unable, exit from outer while loop
        Add L to body of clause
    Remove from remaining tuples in R covered by clause
    Remove unnecessary literals from clause and append cut
    Add clause to definition

Add default clause

/* Global pruning */
While errors do not increase
    Remove literals from each clause
    Remove clauses from definition
```

Fig. 3. Outline of FFOIL

- Some queries might exhaust the list of clauses without finding one whose body is satisfied. It may then be useful to provide a final default clause of the form

$$R(V_1, V_2, ..., k).$$

  for some constant k in order to cover this possibility.

The above notwithstanding, the outline structure of FFOIL shown in Figure 3 is similar to that of FOIL. The main differences are the addition of cuts to clauses, the generation of a default clause, and a new global pruning phase. This last is required because FFOIL tends to produce heavily qualified clauses that overfit the data, and to give up while some tuples of R remain uncovered. The final pruning phase removes literals from clauses and clauses from the definition as long as doing so does not increase the number of errors on the training data.

5 Boosting FFOIL

Although FFOIL is not a classifier-learning system, there is an intuitive correspondence between attribute-value learning and finding definitions of functional

Given:

target relation R containing N tuples $P = P_1,...,P_N$
background relations

/ Initialization */*
For $i := 1$ to N do $w_i^1 := 1/N$

/ Form component definitions */*
For $t := 1$ to T do

If $t = 1$ then
training set $:= P$
else
training set $:=$ resample of size N from P using w_i^t

Run FFOIL *on training set giving definition D^t of R*

Calculate error rate ϵ^t of D^t on P
if $\epsilon^t > 0.5$, discard D^t and stop
if $\epsilon^t = 0$ or $\epsilon^t = 0.5$, stop

/ Adjust weights */*
For $i := 1$ to N do

$$w_i^{t+1} = w_i^t \div \begin{cases} 2\epsilon^t & \textit{if } D^t \textit{ gives wrong answer for } P_i \\ 2(1-\epsilon^t) & \textit{otherwise.} \end{cases}$$

To answer a query using boosted definition D^*:

Answer query with each definition D^t
Choose most common value for output variable

Fig. 4. Method of boosting FFOIL

relations. The analogue of an instance is a tuple of constants belonging to the target relation R, and instead of a classifier we have a Horn clause definition of R. This definition "misclassifies" a tuple $\langle v_1, v_2, ..., v_n \rangle$ if it gives the wrong answer to a query $R(v_1, v_2, ..., v_{n-1}, X)$, i.e. the value returned for X is not equal to v_n.

Unlike C4.5, FFOIL currently does not allow weights to be associated with training data. In order to boost FFOIL, then, it is necessary to use the resampling procedure described at the end of Section 2. The complete boosting scenario for FFOIL is set out in Figure 4. This again mirrors AdaBoost.M1 except that, lacking a measure of confidence with which a definition "classifies" a tuple, each constituent definition D^t has equal voting weight in the boosted definition D^*.

6 Experiments

The boosted version of FFOIL was tested on several datasets that have appeared in published studies. For each dataset, the true error rates of the "standard" and boosted versions of FFOIL were estimated either by cross-validation or by repeated runs with different training and test sets, using where possible the same divisions of the data as previous experiments. As with the experiments reported in Section 3, the maximum number T of boosting trials was set at 10.

6.1 Components of documents

The first learning task, reported in [8], involves identifying the purposes served by components of single-page letters such as that in Figure 5. The data consist

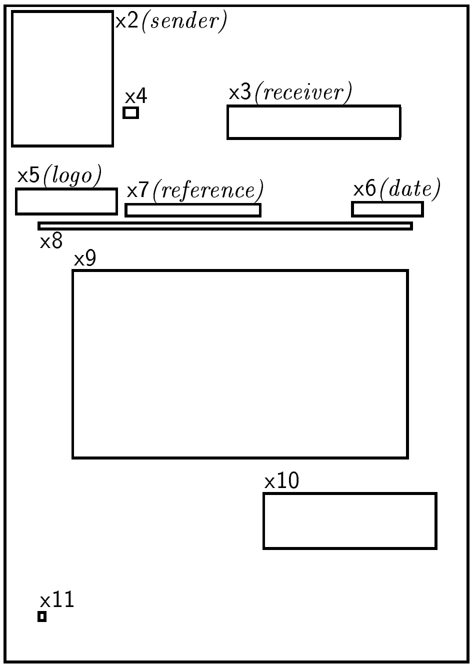

Fig. 5. Sample letter showing components

of thirty letters containing 364 components in all. The target relation used here is *function*(A, B), where A is a component of one of the letters and B is one of *sender, receiver, date, reference, logo* and *other*. 57 background relations describe properties of components such as their width and height, and relationships such as horizontal and vertical alignment with other components.

Version of FFOIL	Errors	Error Rate
standard	22	6.0%
boosted	11	3.0%

Table 1. Results on component function task

A 30-fold cross-validation was conducted, at each fold omitting all information about components of one letter, learning a definition for *function*, then testing this definition on all components of the omitted letter. The total errors over all folds can be seen in Table 1.

6.2 Finite element mesh design

This learning task, first reported in [7], deals with the division of edges of an object into an appropriate number of intervals for finite element simulation. Data is provided for five objects containing 277 edges in total, each edge being divided into 1 to 12 segments. The target relation $mesh(A, B)$ gives the number B of segments recommended by an expert for edge A. Thirty background relations cover properties of each edge and its topological relationship to other edges in the same object.

Version	Errors on object					Total	Error
of FFOIL	a(54)	b(42)	c(28)	d(57)	e(96)	Errors	Rate
standard	33	27	17	35	42	154	55.6%
boosted	32	28	17	39	70	186	67.1%

Table 2. Errors on mesh data

A leave-one-out cross-validation was carried out with this data, similar in form to that conducted for the previous task. Table 2 shows errors for both versions of FFOIL, where the number of edges in each object is indicated in parentheses. This is the only relational dataset for which boosting was observed to increase error, due mainly to very poor performance on the largest object.

6.3 Mutagenicity

This task, reported in [20], concerns learning a theory of mutagenesis. The data consist of 188 molecules, 66% of which are mutagenic, described by 18 background relations. Two of these provide global properties of the molecule, but the bulk of the relations address the molecule's structure and specify component atoms, bonds between them, and information about the presence and connectivity of rings.

Version	Errors on fold										Total	Error
of FFOIL	1	2	3	4	5	6	7	8	9	10	Errors	Rate
standard	3	5	2	1	0	3	1	6	2	2	25	13.3%
boosted	3	4	2	1	0	4	0	4	2	2	22	11.7%

Table 3. Errors on mutagenesis data

A 10-fold cross-validation was carried out using the same partition of the data as that in [20]. Table 3 shows that boosting delivers a modest improvement in prediction accuracy.

6.4 Past tense of English verbs

Learning how to transform a verb from present to past tense was first studied in the connectionist community, but relational representations of the task have also been explored [11, 17, 19]. These experiments imitate Ling's [10] in using ten training and test sets of verbs taken from a corpus of 1500 phonetic verbs. This formulation used two target relations, *delete*(A, B) and *add*(A, C), which jointly state that the past tense of verb A is found by removing the string B from its end and adding the string C. A single background relation *split*(X, Y, Z) indicates all ways in which verb X (in present or past tense) can be divided into non-null substrings Y and Z.

Version	Errors on run										Total	Error
of FFOIL	1	2	3	4	5	6	7	8	9	10	Errors	Rate
standard	64	57	54	55	59	49	77	44	56	51	566	11.3%
boosted	46	58	63	59	56	58	60	49	55	55	559	11.2%

Table 4. Errors on converting verbs to past tense

A run involved two separate learning tasks, one for each target relation, with the same training and test verbs used in each case. A test verb was judged to be correct only when both target relations gave the correct result. The summary of the errors over the 10 runs in Table 4 shows that boosting has only a marginal effect on the accuracy of the learned definition.

6.5 Attribute-value data

The final task does not involve relational data, but was included to check whether the boosted versions of FFOIL and C4.5 behave similarly on attribute-value data. The task chosen is the Wisconsin breast cancer database, one of the UCI datasets (breast-w) reported in Figure 1. A single 10-place relation was used to record the nine attributes and the output variable (the binary class).

In a single 10-fold cross-validation over this data, standard FFOIL gave an error rate of 5.2% and the boosted version 4.0%. These are very close to the results with C4.5, where boosting reduced the error rate from 5.3% to 4.0% (averaged over 10 10-fold cross-validations).

7 Summary

The results of the previous section can be summarised in a similar manner to those for C4.5. Figure 6 shows the ratios of the average errors of the boosted and

Fig. 6. Summary of boosting FFOIL

standard version of FFOIL. Boosting improves performance on four of the tasks and degrades it on one, reducing the error rate by an average of 13%. In terms of evidence in favour of boosting, though, this figure is less convincing than Figure 1 because there are fewer datasets and fewer results averaged for each dataset. Nevertheless, what evidence there is suggests that boosting is advantageous for FFOIL and, by implication, for first-order learning in general.

In order to use boosting for best effect in both attribute-value and relational learning, there remains a pressing need to better understand its occasional failures. This phenomenon could be partly attributable to classification noise (or its analogue, incorrect values of the output variable, when learning definitions of functional relations), as noted in Section 3. Breiman [4] also observes that such failures are more likely to occur with relatively small datasets. It will be interesting to see whether other indicators can be identified.

Acknowledgements

This research was supported by grants from the Australian Research Council. The UCI Machine Learning Repository was the source of many of the datasets and its assistance is gratefully acknowledged.

References

1. Bell, S., and Weber, S.: On the close logical relationship between FOIL and the frameworks of Helft and Plotkin. In *Proceedings Third International Workshop on Inductive Logic Programming*, Bled, Slovenia (1993) 127-147
2. Bergadano, F., and Gunetti, D.: An interactive system to learn functional logic programs. In *Proceedings Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France (Morgan Kaufmann, San Francisco, 1993) 1044-1049
3. Breiman, L.: Bagging predictors. *Machine Learning* (1996) to appear
4. Breiman, L: Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California, Berkeley, 1996
5. Cameron-Jones, R.M., and Quinlan, J.R.: Efficient top-down induction of logic programs. *SIGART 5* (1994) 33-42
6. Cohen, W.W.: Recovering software specifications with inductive logic programming. In *Proceedings AAAI-94 Twelfth National Conference on Artificial Intelligence*, Seattle, Washington (AAAI Press, Menlo Park, CA, 1994) 142-148.
7. Dolšak, B., and Muggleton, S.: The application of inductive logic programming to finite element mesh design. In S. Muggleton, ed., *Inductive Logic Programming* (Academic Press, London, 1992) 453-472
8. Esposito, F., Malerba, D., Semeraro, G., and Pazzani, M.: A machine learning approach to document understanding. In *Proceedings Second International Workshop on Multistrategy Learning* (West Virginia, 1993) 276-292
9. Freund, Y., and Schapire, R. E.: A decision-theoretic generalization of on-line learning and an application to boosting. Unpublished manuscript, available from the authors' home pages ("http://www.research.att.com/orgs/ssr/people/{yoav, schapire}"), 1996
10. Ling, C.X.: Learning the past tense of English verbs: the symbolic pattern associator versus connectionist models. *Journal of Artificial Intelligence Research 1* (1994) 209-229
11. Mooney, R.J., and Califf, M.E.: Induction of first-order decision lists: results on learning the past tense of English verbs. *Journal of Artificial Intelligence Research 3* (1995) 1-24
12. Muggleton, S., and Feng, C.: Efficient induction of logic programs. In S. Muggleton, ed., *Inductive Logic Programming* (Academic Press, London, 1992) 281-298
13. Muggleton, S.: Inverse entailment and Progol. *New Generation Computing 13* (1995) 245-286
14. Pazzani, M. and Kibler, D.: The utility of knowledge in inductive learning. *Machine Learning 9* (1992) 57-94
15. Quinlan, J.R.: Learning logical definitions from relations. *Machine Learning 5* (1990) 239-266
16. Quinlan, J. R.: *C4.5: Programs for Machine Learning.* (San Mateo: Morgan Kaufmann, 1993)
17. Quinlan, J.R.: Past tenses of verbs and first-order learning. In *Proceedings AI'94 Seventh Australian Joint Conference on Artificial Intelligence*, Armidale, Australia (World Scientific, Singapore, 1994) 13-20
18. Quinlan, J.R.: Boosting, bagging, and C4.5. In *Proceedings AAAI-96 Fourteenth National Conference on Artificial Intelligence*, Portland, OR (AAAI Press, Menlo Park, CA, 1996)

19. Quinlan, J.R.: Learning first-order definitions of functions. *Journal of Artificial Intelligence Research* (1996) to appear
20. Srinivasan, A., Muggleton, S.H., Sternberg, M.J.E., and King, R.D.: Theories for mutagenicity: a study in first-order and feature-based induction. *Artificial Intelligence* (1996) to appear
21. Zelle, J.M., and Mooney, R.J.: Combining FOIL and EBG to speed-up logic programs. In *Proceedings Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France (Morgan Kaufmann, San Francisco, 1993) 1106-1111