Boosting First-Order Clauses for Large, Skewed Data Sets

Louis Oliphant[1,3], Elizabeth Burnside[2,3] and Jude Shavlik[1,3]

[1]Computer Sciences Department
[2]Radiology Department
[3]Biostatistics and Medical Informatics Department
University of Wisconsin-Madison

Abstract. Creating an effective ensemble of clauses for large, skewed data sets requires finding a diverse, high-scoring set of clauses and then combining them in such a way as to maximize predictive performance. We have adapted the RankBoost algorithm in order to maximize area under the recall-precision curve, a much better metric when working with highly skewed data sets than ROC curves. We have also explored a range of possibilities for the weak hypotheses used by our modified RankBoost algorithm beyond using individual clauses. We provide results on four large, skewed data sets showing that our modified RankBoost algorithm outperforms the original on area under the recall-precision curves.
Keywords: Learning to Rank, Ensembles, Boosting

1 Introduction

Research over the past 15 years has shown an improvement in predictive accuracy by using an ensemble of classifiers [4] over individual classifiers. In the Inductive Logic Programming [6] domain ensembles have been successfully used to increase performance [5,9,10]. Successful ensemble approaches must both learn individual classifiers that work well with a set of other classifiers as well as combine those classifiers in a way that maximizes performance. AdaBoost [8] is a well known ensemble method that does both of these things. AdaBoost learns weak hypotheses iteratively, increasing the weight on previously misclassified examples so successive learners focus on misclassified examples. AdaBoost combines weak hypotheses into a single classifier by using a weighted sum, where each weak hypothesis is weighted according to its accuracy.

While AdaBoost focuses on improving accuracy of the final classifier, other boosting algorithms have been created that maximize other metrics. The objective of Freud et al.'s RankBoost algorithm [7] is to maximize the correct ordering of all possible pairs of examples in a list of examples. RankBoost maintains a probability distribution over all pairs of examples. The weak learner uses this distribution and finds a hypothesis that minimizes the weighted misorderings from the correct ordering of the examples.

Appears In the ILP-2009 Springer LNCS Post-conference Proceedings.

One version of RankBoost, named RankBoost.B, is designed to work with binary classification problems. Weights are only assigned to pairs of examples if the examples are from different classes. This focuses learning on ordering examples so that all positive examples will be ranked before the negative examples and ignoring the ordering of examples if they are of the same class. Cortes and Mohri [1] showed RankBoost.B maximizes the area under the receiver operator characteristic (AUROC) curve.

AUROC is a common metric used to discriminate between classifiers. Davis and Goadrich [3] however demonstrated that AUROC is not a good metric for discriminating between classifiers when working with highly skewed data where the negatives outnumber the positives. They recommend using area under the recall-precision curve (AURPC) when working with skewed data.

We present a modified version of the RankBoost.B algorithm that works well with skewed data which we name PRankBoost for *precision-recall RankBoost*. Its objective function seeks to maximize AURPC. We implement a top-down, heuristic-guided search to find high-scoring rules for the weak hypotheses and then use this modified RankBoost algorithm to combine them into a single classifier. We also evaluate several other possibilities for weak hypotheses that use sets of the best-scoring rules found during search.

2 PRankBoost–A Modified RankBoost Algorithm

PRankBoost, a modified version of Freud et al.'s RankBoost.B algorithm, appears in Table 1. We have modified the sum of the weights on the negative set to the skew between the size of the negative set and the size of the positive set. We make this change to expose enough information to the weak learner so that it can optimize the AURPC.

PRankBoost initializes weights on the positive examples uniformly to $\frac{1}{|X_1|}$ where X_1 is the set of positive examples. Negative examples are also uniformly initialized so that the sum of their weights is equal to the skew between positives and negatives. These initial weights preserve the same distribution between positive and negative examples as what exists in the unweighted data set. Calculating recall and precision for a model on the initial-weighted data set will be identical to calculating recall and precision on the unweighted version of the data set.

After PRankBoost initializes example weights, the algorithm enters a loop to learn a set of T weak learners. A weak learner is trained using the weighted examples. We have explored using several different weak learners which we will discuss shortly. The objective function used during training is the weighted AURPC. After training, PRankBoost assigns a weight to the weak learner. The weight is calculated analogous to the *third method* discussed by Freud et al. In this method α is an upper bound on the normalization factor, Z. Cortes and Mohri show that the r parameter used to calculate α is equivalent to a weighted version of the area under the ROC curve. We modify this approach for PRankBoost so that the r is a weighted version of AURPC.

PrankBoost updates weights using the parameter α, the weak learner $h(x)$, and a factor Z, which maintains the same weight distribution between the positive and negative examples as exists with the initial weights. An example's weight is decreased relative to how well the weak learner scores the example. The higher a positive example is scored by the weak learner the smaller the weight while be, while the opposite is true for negative examples. The effect is to place more weight on examples which the weak learner has difficulty classifying.

The final classifier, $H(x)$, assigns a score to a new example, x, as a weighted sum of the individual weak learners. We designed PRankBoost to be analogous to RankBoost. While RankBoost's final classifier maximizes AUROC, our modified version attempts to maximize AURPC. We hypothesize that this modified version will outperform RankBoost when comparing AURPC.

Table 1. PRankBoost–A modified RankBoost algorithm for optimizing area under the recall-precision curve.

```
**Modified RankBoost Algorithm**
Given: disjoint subsets of negative, $X_0$, and positive, $X_1$, examples
Initialize:
    $skew = \frac{|X_0|}{|X_1|}, \qquad w_1(x) = \begin{cases} \frac{skew}{|X_0|} \; if\, x \in X_0 \\ \frac{1}{|X_1|} \; if\, x \in X_1 \end{cases}$
for $t = 1, ..., T$:
    Train weak learner, $h_t$, using $w_t$ and $skew$.
    Get weak ranking $h_t : X \longrightarrow \mathbb{R}$.
    Choose $\alpha_t = 0.5 \ln\left(\frac{1+r}{1-r}\right)$ where $r = AURPC$(see text).
    Update
        $w_{t+1}(x) = \begin{cases} \frac{w_t(x)\exp(-\alpha_t h_t(x))}{Z_t^1} \; if\, x \in X_1 \\ \frac{w_t(x)\exp(\alpha_t h_t(x))}{Z_t^0} \; if\, x \in X_0 \end{cases}$
    where
        $Z_t^1 = \sum_{x \in X_1} w_t(x)\exp(-\alpha_t h_t(x))$
        $Z_t^0 = \frac{1}{skew} \times \sum_{x \in X_0} w_t(x)\exp(\alpha_t h_t(x))$
Output the final ranking: $H(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$.
```

3 Weak Learners

As shown in Table 1, a weak learner, $h_t(x)$ is a function that maps an example to a real value. A perfect weak learner maps all positive examples to higher values than negative examples. Often it is not possible to find a perfect weak learner and some objective function is used to decide among possible weak learners. In Adaboost the object function guides learning towards models that minimize a weighted version of misclassification error. In RankBoost the objective function maximizes a weighted area under the ROC curve. Our PRankBoost algorithm for finding weak learners uses area under the recall-precision curve as the object function.

When deciding what search algorithm to use for finding a weak learner we had several goals in mind. First, we wanted the search algorithm to find a clause that worked well with highly skewed data. This is the reason we use AURPC as the objective function. Second, we wanted to apply this algorithm to large data sets. Evaluation of clauses in large data sets is a costly time step and limits the number of weak learners that can be considered in a reasonable amount of time. Because of this we use a greedy hill-climbing algorithm to find weak learners.

We consider several possibilities for weak learners. The simplest weak learner we use consists of a single first-order rule. To find this rule we select a random positive example as a seed and saturate it to build the bottom clause. We begin with the most general rule from this bottom clause. All legal literals are considered to extend the rule. The extension that improves the AURPC the most is selected and added to the rule. The process repeats until no improvement can be found or some time limit or rule-length limit is reached. Each weak hypothesis, $h_t(x)$, is the best scoring individual rule found during this search.

This weak learner maps an example, x, to the range $\{0, 1\}$ where the mapping is 1 if the example is predicted as true, 0 otherwise. We call this learner PRankBoost.Clause.

We have also explored other possibilities for the weak learner and how the AURPC is calculated for the objective function. Our goal in developing other weak learners was to create more accurate models without increasing the number of rules evaluated on training data. One method of developing more complex first-order models is to retain more than just the best clause found during search. Taking an idea from the Gleaner algorithm [9] which retains an entire set of rules found during search that span the range of recall values, we have developed a second weak learner that retains a set of the best rules found during search. This weak learner, PRankBoost.Path, contains all rules along the path from the most general rule to the highest-scoring rule found during search. This set of rules will contain short, general rules that cover many examples and longer, more specific rules that have higher accuracy but lower coverage on the positive examples.

For example consider the rules that appear in Figure 1. A set of rules would contain the highest-scoring rule, h(X):-p(X),q(X,Y),r(Y), along with the subsets of the rule from the most general rule to this rule, h(X):-p(X,Y),p(Y,Z) and h(X):-p(X,Y). This weak hypothesis, $h_t(x)$, maps an example, x, to the range $[0, 1]$ by finding the most specific of these rules that covers the example. If the highest-scoring rule did not cover some new example then the next most specific rule would be considered until a rule is found that covers the example. $h_t(x)$ is the fraction of the total AURPC covered by this rule as illustrated in Figure 1.

The total AURPC, r, is the area under the entire path from the most specific rule to the most general rule (the total grayed area in Figure 1).

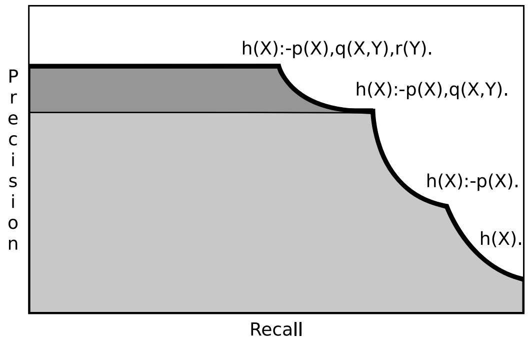

Fig. 1. Area under the recall-precision curve for a path of clauses learned during hill climbing. The total grayed area is the total AURPC, r. If h(X) :- p(X), q(X,Y) is the most specific clause in the path to cover an example then $h_t(x)$ maps the example to the value (light gray area / total grayed area).

4 Calculating AURPC

We use a weighted version of AURPC as both the objective function used to find weak learners as well as to weight weak learners when combining them into an ensemble. In general we follow the algorithm outlined by Goadrich et al. [9] to calculate AURPC, however We made two modifications to work in this ensemble setting and to improve accuracy and increase speed. First, we use a weighted version of recall and precision. Second, when calculating the area between two points in recall-precision space, A and B, Davis and Goadrich use a discretized version that estimates the area under the curve. We calculate the area under the curve exactly using a closed form solution to the integral for the curve between the two points,

$$\int_{TP_A}^{TP_B} \frac{x}{x + FP_A + s(x - TP_A)}\, dx$$

where TP is the true positive weight and FP is the false positive weight. Parameter s is the local skew of false positives to true positives between the two points A and B, $s = \frac{FP_B - FP_A}{TP_B - TP_A}$. The total AURPC is a piece-wise integral between each of the points in recall-precision space that correspond to the rules of a weak learner. For PRankBoost.Clause, which consists of a single clause, this would be a single point in recall-precision space. We use Goadrich et al.'s method for extending this point down to zero recall and up to 100% recall by using the most general clause. For PrankBoost.Path we perform the same extension down to zero recall and up to 100% recall but we use all point that correspond to the clauses in the set retained by the weak learner. This curve is shown in Figure 1.

5 Experimental Methodology and Results

We modified Aleph [12] to incorporate RankBoost and my modified versions, PRankBoost.Clause and PRankBoost.Path. RankBoost uses the same hill-climbing algorithm for finding weak learners as my two variants use. We used individual clauses for the weak learners in RankBoost. This makes the RankBoost algorithm directly comparable to PRankBoost.Clause. We compared these algorithms using AUROC and AURPC on four large, skewed data sets, two from the information-extraction domain and two from the mammography domain.

Protein Localization data set consists of text from 871 abstracts taken from the Medline database. The task is to find all phrase pairs that specify a protein and where it localizes in a cell. The data set comes from Ray and Craven [11]. Additional hand annotation was done by Goadrich et al. [9] The data set contains 281,071 examples with a positive/negative skew of 1:149.

Gene Disease data set also comes from Ray and Craven [11]. We utilized the ILP implementation by Goadrich et al. [9] The task is to find all phrase pairs showing genes and their associated disorder. the data set contains 104,192 examples with a positive/negative skew of 1:446.

Mammography1 data set is described by Davis et al. [2] It contains 62,219 findings. The objective with this data set is to determine if a finding is benign or malignant given descriptors of the finding, patient risk factors, and radiologist's prediction. The positive/negative skew is 1:121.

Mammography2 is a new data set that has the same task as Mammography1, however the data was collected from mammograms from a second institution, the University of Wisconsin Hospital and Clinics. The data set consists of 30,405 findings from 18,375 patients collected from mammograms at the radiology department. The positive/negative skew is 1:86.

We ran 10-fold, cross-validation for the mammography data sets and 5-fold for the IE data sets. We ran each fold 10 times using a different random seed to average out differences due to random effects such as seed selection. We calculated average AURPC, average AUROC, and standard deviations across the different runs and folds. Also, to compare how quickly the ensembles converged, we created learning curves with the x-axis showing the number of rules evaluated and the y-axis showing the average AURPC.

Table 2 shows average AURPC and AUROC results with standard deviations for ensembles containing 100 weak learners for RankBoost and PRankBoost.Clause. RankBoost outperforms PRankBoost.Clause when comparing AUROC on three of the four data sets. The AUROC scores are high and close together. This makes it more difficult to visually distinguish ROC curves from

Table 2. Average AUROC and AURPC percentages with standard deviations for several large, skewed data sets using the RankBoost and PRankBoost.Clause algorithms. Bold indicates statistically significant improvement at 5% confidence level.

Data set	AUROC		AURPC	
	RankBoost	PRankBoost.Clause	RankBoost	PRankBoost.Clause
Mammography 1	$\mathbf{89.9 \pm 4.2}$	88.1 ± 5.8	18.5 ± 5.7	$\mathbf{32.9 \pm 7.6}$
Mammography 2	92.5 ± 2.0	$\mathbf{96.7 \pm 1.1}$	16.2 ± 7.4	$\mathbf{41.3 \pm 10.6}$
Protein Localization	$\mathbf{98.9 \pm 0.1}$	97.9 ± 0.7	40.4 ± 7.9	40.5 ± 8.6
Gene Disease	$\mathbf{98.2 \pm 0.9}$	95.4 ± 2.4	32.9 ± 10.7	$\mathbf{46.6 \pm 11.9}$

each other. However when comparing AURPC the difference between the two algorithms is large. PRankBoost.Clause outperforms RankBoost on three of the four data sets. The variance is much larger for AURPC scores than for AUROC scores because when recall is close to zero variance in precision values is high.

Learning curves on the four data sets appear in Figure 2. Each graph shows the AURPC on the y-axis by the number of rules considered during training on the x-axis. Each curve extends until 100 weak hypotheses have been found. We do this as a way of showing that the various algorithms do different amounts of work to produce 100 hypotheses, a fact that would be lost if we simply extended all three to the full width of the x-axis.

My PRankBoost.Path algorithm reaches an AURPC of 0.44 on the Protein Localization data set after less than 20,000 clauses searched. The Gleaner algorithm takes over 100,000 clauses to surpass this level of performance [9]. On the Gene Disease data set my PRankBoost.Clause algorithm reaches 0.48 AURPC after 45,000 clauses searched, while the Gleaner algorithm does not reach this level of performance even after 10 million clauses searched.

The more complex weak learner, PRankBoost.Path does not appear to dominate the simple learner, PRankBoost.Clause, on all of the data sets. We believe this is because PRankBoost.Clause learns a very specific clause and reduces the weights on just a few positive examples. This forces search to focus on other positive examples and find other specific clauses that perform well on those positive examples. PRankBoost.Path on the other hand learns both specific and general clauses in the set of clauses used as a model for the weak learner. This means many positive examples will be down-weighted rather quickly. The remaining positive examples may consist of very difficult examples where it is not easy to find a good clause that covers those positive examples without also covering many negatives. After observing these characteristics We designed other weak learners that try to find a mix of models somewhere between PRankBoost.Clause and PRankBoost.Path.

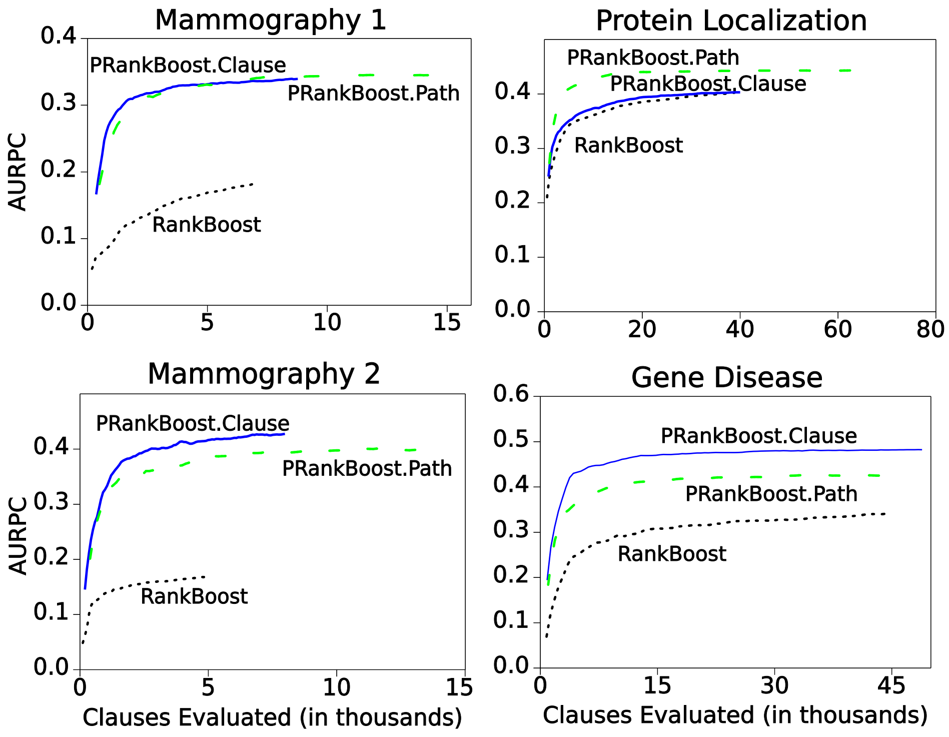

Fig. 2. Learning curves for Freund et al.'s RankBoost algorithm, our PRankBoost.Clause and PRankBoost.Path algorithms on four large, skewed data sets. Learning curves extend until 100 weak hypotheses are learned. This makes some curves extend farther than others.

6 Additional Experiments with Variations on Weak Learners

We have additional results using other weak learners that combine variations of PRankBoost.Clause and PRankBoost.Path. Remember that PRankBoost.Clause retains the single rule that is the best seen during search. Its score, α, is a weighted version of the area under the recall-precision curve of that single rule. PRankBoost.Path retains a set of rules along the trajectory from the most general rule to the best rule found during hill climbing. The weak learner's score is based upon the area under the entire path of rules. Figure 3 shows the two methods of scoring a weak learner based upon the single rule or the entire trajectory. The solid curve uses the entire trajectory from the most general rule to the rule itself while the dashed curve uses only the rule itself.

These two scoring methods create a very different search pattern. Consider scoring rules based upon the entire path from the most general rule. A portion of the score is fixed based upon the portion of the rule that has already been chosen. Any extension to the rule will only decrease recall or at best leave recall unchanged. The score will change only the left-most portion of the recall-precision curve. Any extension that increases precision will also increase the rule's overall score. This is not true when scoring a rule based upon only the rule itself. Adding a literal to a rule, even though it may increase the precision of the rule, may still decrease the overall rule's score because the curve to the single rule will also change. No portion of the curve is fixed. The difference between these two scoring methods means that using the entire path to score a rule will search more deeply in the search space and discover longer rules with higher precision but lower recall.

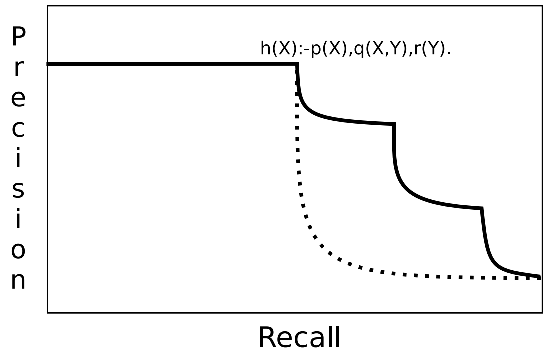

Fig. 3. Two scoring methods for a weak learner. One scoring method (solid curve) used by the PRankBoost.Path and Mix1 weak learners is based upon the entire trajectory of rules from the most general rule to the best rule. The second scoring method (dashed curve) used by the PRankBoost.Clause and Mix2 weak learners is based upon the single best rule alone. Mix3 alternates between using these two scoring methods.

As variations on PRankBoost.Path and PRankBoost.Clause we have created three other weak learners. The first retains the entire set of rules like PRankBoost.Path, but the scoring function of the learner is based upon the single best rule like PRankBoost.Clause. The second does just the reverse by retaining only the single best rule, but scoring it based upon the entire trajectory. As a final variation we have also alternated between PRankBoost.Clause and PRankBoost.Path for each weak learner created.

We ran experiments using the same experimental setup as my previous experiments. Results for these three new weak learners appear in Figure 4. It appears that these variations do not find models whose precision-recall performance is consistently higher than PRankBoost.Clause and PRankBoost.Path models when measuring AURPC. However the first mixed model (dashed line) does show some interesting properties. Its initial performance is very low compared to the other models. It has a more shallow learning curve and it does not appear to have reached its asymptotic performance after 100 weak learners have been included in the model. All of these observations make sense when considering the type of weak learner. Each weak learner is an individual clause that will have high precision but low recall due to the scoring function being the area under the entire path. After each weak hypothesis is learned the few positive examples that are covered will be down-weighted and a new weak hypothesis will be learned that covers new examples. Because of the small coverage of each individual clause, learning will be slow and consistent, showing improvement even after many clauses have been learned.

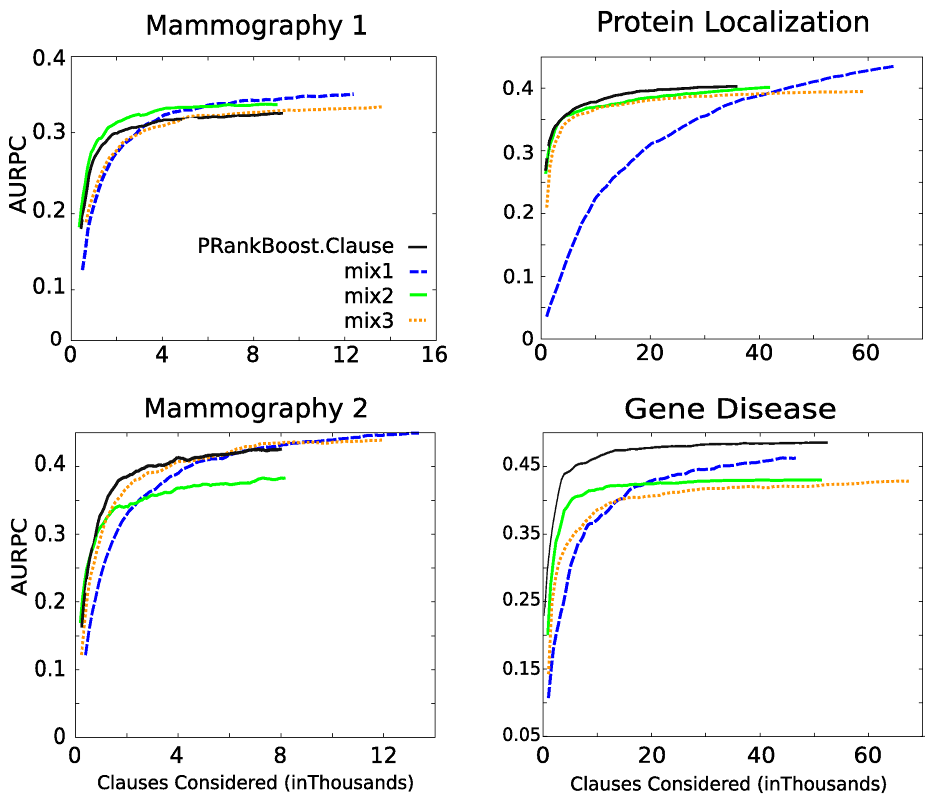

Fig. 4. Learning curves for three models that mix components of PRankBoost.Path and PRankBoost.Clause on four large data sets. *Mix1* includes clauses as weak learners like PRankBoost.Clause but scores them like PRankBoost.Path. *Mix2* includes entire paths of clauses as PRankBoost.Path but scores the path like PRankBoost.Clause. *Mix3* alternates between the method used in PRankBoost.Path and the one used in PRankBoost.Clause. PRankBoost.Clause is graphed for comparison purposes only.

7 Conclusion and Future Work

When working with skewed data sets metrics such as area under the recall-precision curve have been shown to discriminate well between competing models. We designed a modified RankBoost algorithm to maximize area under the recall-precision curve. We compared the original RankBoost algorithm, which is designed to maximize area under the ROC curve, with our modified version. When comparing AUROC on four large, skewed data sets the original RankBoost algorithm outperforms our modified PRankBoost version. However when comparing AURPC PRankBoost outperforms the original algorithm.

We created several first-order logic weak learners. The simplest weak learner, PRankBoost.Clause, consists of an individual rule. A second, more complex weak learner, PRankBoost.Path, consists of all rules along the path to the best rule. This more complex learner does not require any additional rules be evaluated on training data. This is especially important when working with large data sets because evaluation is a costly time step. Both weak learners have different strengths with neither learner dominating in performance across all data sets. In addition to these two weak learners We created several other weak learners that are a combination of these two. The most promising, Mix1, consists of the highest-scoring clause found during search, but its score is calculated using the entire trajectory of rules from the most general rule to this best rule.

For future work we would like to create additional mixed models that begin by learning more general clauses as seen in PRankBoost.Clause and then switching to learning more specific clauses as seen in the first mixed model. We believe this type of model will show good initial performance and will continue to show predictive improvement reaching a higher asymptote. We would also like to perform theoretical analysis to support our empirical work showing that PRankBoost maximizes AURPC following Freund et al.'s proof that RankBoost maximizes AUROC [7].

We gratefully acknowledge the funding from USA grants R01 LM07050 and 1 R01 CA127379, Houssam Nassif, David Page, and Ryan Woods and our anonymous reviewers for their comments and suggestions.

References

1. C. Cortes and M. Mohri. AUC optimization vs. error rate minimization. In *Neural Information Processing Systems (NIPS)*. MIT Press, 2003.
2. J. Davis, E. Burnside, I. Dutra, D. Page, and V. Costa. An integrated approach to learning Bayesian networks of rules. In *16th European Conference on Machine Learning*, pages 84–95. Springer, 2005.
3. J. Davis and M. Goadrich. The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 233–240, 2006.
4. T. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000.
5. I. Dutra, D. Page, V. Costa, and J. Shavlik. An empirical evaluation of bagging in inductive logic programming. In *Proceedings of the Twelfth International Conference on Inductive Logic Programming*, pages 48–65, Sydney, Australia, 2002.
6. S. Džeroski and N. Lavrac. An introduction to inductive logic programming. In *Proceedings of Relational Data Mining*, pages 48–66, 2001.

7. Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. In *Proceedings of 15th International Conference on Machine Learning*, pages 170–178, 1998.
8. Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148–156, 1996.
9. M. Goadrich, L. Oliphant, and J. Shavlik. Gleaner: Creating ensembles of first-order clauses to improve recall-precision curves. *Machine Learning*, 64(1-3):231–261, 2006.
10. John R. Quinlan. Relational learning and boosting. In *Relational Data Mining*, pages 292–306, 2001.
11. S. Ray and M. Craven. Representing sentence structure in hidden Markov models for information extraction. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 2001.
12. A. Srinivasan. The Aleph manual version 4. *http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/*, 2003.