Appears in *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP)*. Corvallis, Oregon, USA. June, 2007.

Combining Clauses with Various Precisions and Recalls to Produce Accurate Probabilistic Estimates

Mark Goadrich and Jude Shavlik

University of Wisconsin - Madison

Abstract. Statistical Relational Learning (SRL) combines the benefits of probabilistic machine learning approaches with complex, structured domains from Inductive Logic Programming (ILP). We propose a new SRL algorithm, GleanerSRL, to generate the probability that an example is positive within highly-skewed relational domains. In this work, we combine clauses from Gleaner, an ILP algorithm for learning a wide variety of first-order clauses, with the propositional learning technique of support vector machines to learn well-calibrated probabilities. We find that our results are comparable to SRL algorithms SAYU and SAYU-VISTA on a well-known relational testbed.

1 Introduction

Inductive Logic Programming (ILP) is the process of learning first-order clauses to correctly categorize domains of relational data. ILP uses relations expressed in mathematical logic to describe examples, and can handle variable-sized structures and sequences [5]. Statistical Relational Learning (SRL) [8] builds on the benefits of relational data and introduces methods for learning from large and noisy datasets, typically in combination with producing probabilistic outputs as opposed to strict classifications. Prominent work within SRL includes the generative approaches of Probabilistic Relational Models by Friedman et al. [7] and Markov Logic Networks from Richardson and Domingos [17], as well as discriminative algorithms such as SAYU and SAYU-VISTA from Davis et al. [4].

In this work we propose the use of Gleaner [9] as the foundation for a new discriminative SRL algorithm called GleanerSRL. Gleaner is a two-stage algorithm developed to first learn a broad spectrum of clauses and then combine them into thresholded theories aimed at maximizing precision for a particular choice of recall. Gleaner can run quickly on large datasets when one has a set of available processors. Already new desktop computers include multiple cpu's (called 'cores'), and within a few years it will be common for desktop computers to have 32, 64, 128, or more cores. Also we have previously shown that Gleaner can achieve good performance from only a relatively small number of clause evaluations per seed, because it keeps more than one good clause per seed, and we believe the clauses learned from Gleaner will be more diverse than those found with other approaches.

We modify the two-stage approach used by Gleaner into GleanerSRL, which learns clauses, produces feature vectors, and generates probabilities. We then

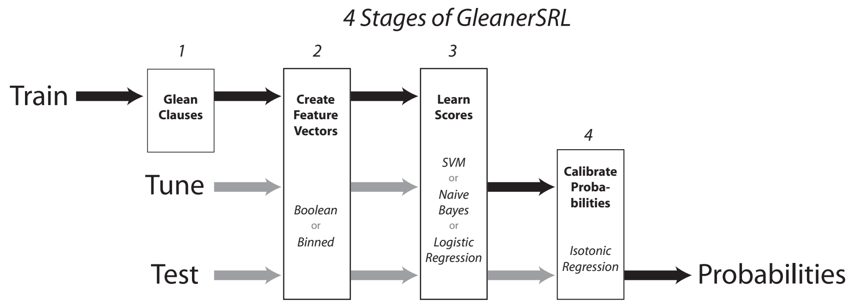


Fig. 1. GleanerSRL takes training, tuning and testing examples and returns a probability estimate for the testing examples after four stages of processing. Black arrows denote dependencies in training for a stage, while grey arrows denote only data transformations. Note that the testset is not examined until training is complete in order to allow us unbiased estimates of future performance.

evaluate the quality of these approaches using Mean Cross Entropy in comparison to SAYU and SAYU-VISTA. Finally, we conclude by discussing future directions and related work.

2 Learning Probabilities with GleanerSRL

GleanerSRL is a four-stage algorithm to directly estimate probabilities for relational domains, as shown in Figure 1. The first stage learns a wide variety of clauses from a large number of seed examples. The second stage uses the clauses learned to generate a feature vector for each example, while the third stage uses this feature vector in propositional learners to learn a numeric score for each example, and the fourth stage calibrates these scores into probabilities. In essence, we will be transforming our tasks into propositional domains through the medium of our learned clauses and then using standard propositional learners to estimate these probabilities.

2.1 Gleaning Clauses

The first stage of GleanerSRL is identical to that of the original Gleaner, and learns a wide spectrum of clauses, illustrated in Figure 2. Gleaner brings in a training set of positive and negative examples along with the background knowledge. Each clause examined will cover a subset of examples; those that are positive we call true positives (TP) and those that are negative we call false positives (FP). The precision of a clause is then defined as $\frac{TP}{TP+FP}$. Not all positives will be necessarily covered by a clause; those that are missed are called false negatives (FN), and we define the recall of a clause as $\frac{TP}{TP+FN}$.

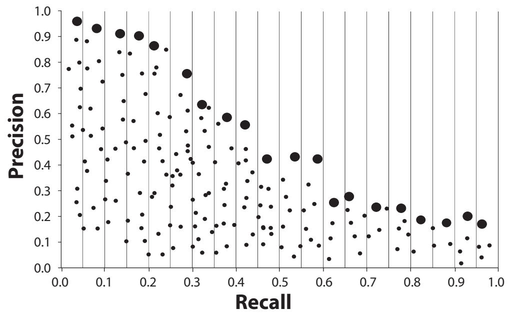


Fig. 2. A hypothetical run of Gleaner for one seed and 20 bins on the training set, showing each considered clause as a small circle, and the chosen clause per bin as a large circle. This is repeated for K seeds to gather $B \times K$ clauses (assuming a clause is found that falls into each bin for each seed).

Gleaner uses Aleph [18] to search for clauses using K seed examples to encourage diversity. The recall dimension is uniformly divided into B equal sized bins; in our experiments that appear in Section 3 our bins will be $[0, 0.05], [0.05, 0.10], \ldots, [0.95, 1]$. For each seed, we consider up to N possible clauses using stochastic local-search methods [10]. As these clauses are generated, we compute the recall of each clause and determine into which bin the clause falls. Each bin keeps track of the best clause appearing in its bin for the current seed. We use the heuristic function $\textit{precision} \times \textit{recall}$ to determine each bin's best clause, since we believe this will increase the generality of our clauses.

At the end of this search process, there will be B clauses collected for each seed and K seed examples for a maximum of $B \times K$ clauses (assuming a clause is found that falls into each bin for each seed). Since clauses can be learned independently for each seed, Gleaner is fast for large datasets because each seed can be explored in parallel.

2.2 Creating Features

Whereas the second stage of Gleaner combines these learned clauses in an attempt to maximize precision-recall (PR) curves on an unseen testset, here we wish to instead estimate the probability that an example is positive. We cannot directly convert Gleaner's final PR curve into numeric scores, since each point in the test curve may come from a distinct theory and threshold combination. Gleaner requires the user to find the point closest to their desired recall and then uses this theory to rank the testset examples based on the particular theory which generated this point. Since we are interested in directly generating probabilities instead of recall-precision curves, we introduce here a new second stage for GleanerSRL to transform the learned clauses into propositional features.

Our first transformation is the *Boolean* feature method. We create one feature for each clause and assign the feature a value of 1 if the clause is true and 0 if the clause is false. In a scenario with 20 bins and 100 seeds, this would generate 2000 features, given that there is a clause found within each bin for all seeds and all clauses are unique. We have found in practice that there are many less features generated than the complete 2000 due to duplicate clauses within the high-recall bins. These Boolean feature vectors are created for the trainset, tuneset and testset examples.

A second approach is the *binned* feature method. We make use of the theories and thresholds as previously calculated by the second stage Gleaner, making one feature per bin. For each example, the value of a feature is equal to the cumulative precision of each clause in this bin's theory that match this example. This reduces our features to only the number of bins no matter how many seeds are explored. In our earlier work with Gleaner, we noticed that duplicate clauses were found more often in the high-recall bins. This binning feature method will retain a more uniform coverage of the recall space and will also take advantage of combining similar clauses. We look at two binning feature methods, one with the features as raw score of the cumulative precision for each bin, and one with the cumulative precision *normalized* to between 0 and 1 by the maximum score found in that bin. The precision for each clause is calculated on the trainset, and bin feature vectors are created for the trainset, tuneset and testset examples. Using the tuneset to calculate the precision is also recommended, but I reuse the trainset to maintain a suitably large number of positive examples for these calculations.

2.3 Learning to Predict Scores

With the feature vectors calculated from the second stage, our problem is now propositional in nature. The third stage of GleanerSRL uses standard propositional approaches to estimate the probability for each example. We will be using classifiers where each feature f is assigned a weight w_f through training. For a new example x_i, where $0 < i \leq N$ for a testing set of size N and $x_{i,j}$ is the value of feature f on example x_i, we discriminate between positive and negative using a threshold b as follows:

$$\text{If} \quad \sum_{f \in feats} (w_f \times x_{i,f}) > b \quad \text{then +, else -}$$

We can achieve a richer feature space by using a kernel matrix K to give us a notion of similarity between example x_i and the examples in our training set (minus those set aside in the tuning set). The simplest kernel is constructed by taking the dot product of x_i and example x_j, such that $K(x_i, x_j) = \sum_{f \in feats}(x_{i,f} \times x_{j,f})$. We can then replace our weighted feature model from above with

$$\text{If} \quad \sum_{j \in examples} (\alpha_j \times K(x_i, x_j)) > b \quad \text{then +, else -}$$

Table 1. We examine five different kernel methods for calculating $K(x_i, x_j) = \sum_{f \in features} k(x_{i,f}, x_{j,f})$ for Boolean feature vectors.

Kernel	$k(x_{i,f}, x_{j,f})$
Dot-Prod	$k(1,1)\ :\ 1,\ \text{else} : 0$
Precision	$k(1,1)\ :\ precision_f,\ \text{else} : 0$
Recall	$k(1,1)\ :\ 1 - recall_f,\ \text{else} : 0$
Both-Pos	$k(1,1)\ :\ precision_f^2,\ \text{else} : 0$
Info	$k(1,1)\ :\ -log_2((\frac{TP+FP}{\lvert trainset \rvert})^2)$ $k(1,0)$ or $k(0,1)\ :\ -log_2(2 \times \frac{TP+FP}{\lvert trainset \rvert} \times (1 - \frac{TP+FP}{\lvert trainset \rvert}))$ $k(0,0)\ :\ -log_2((1 - \frac{TP+FP}{\lvert trainset \rvert})^2)$

where α_j is a weight on each kernel-induced feature. For the purposes of estimating probabilities, we are only really interested in the weighted sum from the above thresholded classification, and we use this as a numeric score s for each example.

Our particular classifier choice for this paper is the Support Vector Machine (SVM) [2]. SVMs learn weights for α_j that maximize the margin between the classification hyperplane and the training data by solving a linear or quadratic program. In practice (especially when using linear programming), most α_j values will be 0, thus ignoring a large number of our kernel-induced features. In our preliminary testing, we also investigated using naïve Bayes and Logistic Regression. We found them to be significantly outperformed by the SVM approach and therefore do not include those results.

We examine here five different kernels, shown in Table 1, for use within our SVM. First we use a simple *dot-product* kernel discussed above in combination with both of the binning feature methods. As for kernels on Boolean features, we also use a dot-product kernel, as well as four attempts to incorporate statistics from the training set about each clause.

Since the similarity under the dot-product kernel is only increased when two examples match on a feature (when features are all Boolean valued), we can score each match instead by the *precision* of that clause as calculated on the training set example. This means that examples will be more similar when they are both covered by high-precision clauses. Similarly for *recall*, we use the score $1 - recall_f$. Since we aim to collect clauses that have high precision in the first stage, matching an example on a low-recall clause should be more meaningful in relation to the positive examples.

We also explore two kernel methods related to the probability that a given clause is expected to match a particular example, called *both-pos* and *info*. Precision equals the probability of an example being truly positive given that it matched the clause, therefore $precision_f^2$ is the probability that any two examples are truly positive given that they both match on $x_{i,f}$, assuming independence, and we use this as our weight for *both-pos*. The actual probability of a given clause matching any example is based on the number of true and false positives for that clause: $prob_f = \frac{TP+FP}{|dataset|}$. For the *info* kernel, we consider the information content for the probabilities of both, only one, or none of the examples matching (using $-log_2(p(X))$ for each case). Two other kernel method explored but not reported here are the Hamming distance between two clauses (where clauses are more similar if they return the same classification on a given example, be it positive or negative) and a Gaussian kernel, as they were outperformed by our above kernels in preliminary tests.

2.4 Calibrating Probabilities

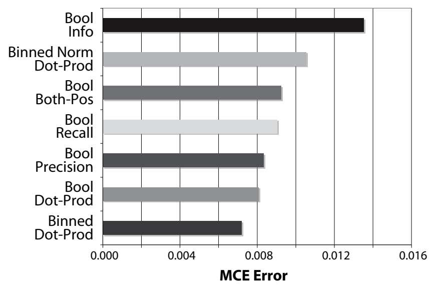
The SVM weighted sums from above will not be strict probabilities between 0 and 1. Therefore in the final stage of GleanerSRL, we calibrate these scores into proper probabilities. Zadrozny and Elkan [21] and Niculescu-Mizil and Caruna [14] recommend Isotonic Regression for large highly-skewed datasets. The main idea behind isotonic regression is to transform the sorted list of SVM scores into monotonically increasing probability scores which minimize the probability errors, and can be seen as an adaptive method for automatically finding the proper bin widths based on the tuning data. We achieve this isotonic regression by using the Pool Adjacent Violators (PAV) algorithm. Given a set of examples (s_i, c_i), where each example i consists of the weighted SVM score s along with the classification c, where c is now 1 for positive examples and 0 for negative examples, PAV will return a mapping for a range of s_i scores to their calibrated c_i values. We calibrate our probabilities on a tuning set and then use the found mapping to assign probabilities $p(x_i)$ on our testing set. Note that this step is not necessary but still recommended when using naïve Bayes or logistic regression.

3 Experimental Results

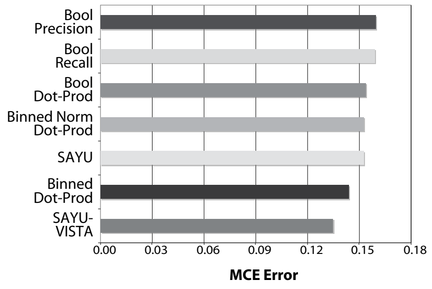
We follow the methodology of Caruna and Niculescu-Mizil [1], and evaluate our probability estimates from GleanerSRL using the metric of Mean Cross Entropy (MCE). In preliminary work we found the results from using mean squared error to be very similar, thus those results are not included here. Cross entropy calculates the difference of predicted probability from the true probability; the formula is derived from information theory and Kullback-Liebler divergence. Formally,

$$MCE = -\frac{\sum_{i=0}^{n}(a(x_i)log(p(x_i)) + (1 - a(x_i))log(1 - p(x_i))}{n}$$

where n is the testset size, $a(x_i)$ is the actual probability of example i (in our case 0 or 1) and $p(x_i)$ is our estimate. To properly compute these numbers, we enforced a bound on the probability estimates so that $0 < prob_{min} \leq p(X) \leq 1 - prob_{min} < 1$. We tune this bound on our tuning set using leave-one-out cross-validation. This bound is helpful when there is a complete mistake in probability,



(a) Cross Entropy Error for OMIM



(b) Cross Entropy Error for Advisor

Fig. 3. Comparison of Mean Cross Entropy GleanerSRL kernel methods and SAYU, ordered from least to most on each dataset.

where the actual probability is 1 and the predicted probability is 0, or vice versa, since the cross entropy error will be infinity.

We report results on two highly-skewed domains, OMIM and Advisor:

OMIM This is the Online Mendelian Inheritance in Man genetic-disorder biomedical information extraction dataset from Ray and Craven [16]. From a sentence such as "Mutations in the COL3A1 gene have been implicated as a cause of type IV Ehlers-Danlos syndrome, a disease leading to aortic rupture in early adult life," the task is to extract a relationship between the gene COL3A1 and Ehlers-Danlos syndrome. We use the ILP dataset construction of Goadrich et al. [9], which contains five disjoint folds with 233 positive and 103,959 negative examples.

Advisor This dataset is derived from the University of Washington CS Department. It was constructed by Richardson and Domingos [17]. The goal is to predict the advisor of a graduate student, where students, professors, courses and papers are known to be related by author, instructor, and teaching assistant relations. This dataset contains five disjoint folds with a total of 113 positive examples and 2,711 negative examples.

For the parameters of GleanerSRL, we ran Gleaner with 20 bins, 100 seeds for OMIM and 50 seeds for Advisor until 25,000 clauses were examined for each seed. In combination with the SVM for stage three, we tuned with nine values for the complexity parameter C ranging from 10,000 to 0.0001, and in stage four we tested nine values for $prob_{min}$ from 0.25 to 0.0001. Different C and $prob_{min}$ values were chosen for each fold.

Figure 3(a) shows the results of our kernel choices for GleanerSRL on OMIM. Binnned feature vectors combined with the dot product kernel outperforms the rest, however, this is only a statistically significant difference with the Boolean match kernel. It is interesting to note that the highest scoring approaches use the dot product kernels for both types of feature vectors.

The results on Advisor in Figure 3(b) again show that Binned Dot Product outperforms our other approaches. Once again the dot product kernel is the best choice. We also compare to SAYU and SAYU-VISTA from Davis et al. [4], using a tree-augmented network [6] and an eager rule-adoption policy. SAYU learns a Bayesian network for classification by continually adding features when a clause makes a significant improvement in the Area Under the Curve for Precision and Recall (AUC-PR). VISTA builds on SAYU by incorporating new predicates throughout the learning process. We find that the difference between GleanerSRL both SAYU-VISTA and SAYU is not statistically significant. We separately explored directly optimizing the MCE for SAYU, and found the results were slightly worse than optimizing for AUC-PR, but the difference was not statistically significant.

4 Related and Future Work

One typical approach to weighting a theory in ILP is propositionalization, where each clause in a theory is translated into a Boolean feature. This allows for a number of propositional learning algorithms to be used for learning weights on each clause. Pompe and Kononenko [15] use a naïve Bayes classifier to find their weights, while Srinivasan and King [19] use logistic regression, a technique to find weights that will maximize the likelihood of the data.

Koller and Pfeffer [11] learn the weights for clauses in a theory by first creating a Bayesian network model for the theory. They then use an Expectation Maximization algorithm to set the parameters to maximize the likelihood of the data. Their results are on a toy dataset with three clauses, so it is unknown how well this would extend to the very large datasets we propose to investigate here. Richardson and Domingos [17] extend work with Relational Markov Networks [20] to formulate Markov Logic Networks. Their setup can take clauses from either ILP or a domain expert, translate them to a Markov Network, and then learn the weights on the clauses using logistic regression. Davis *et al.* [3] compare naïve Bayes, TAN and the sparse candidate algorithm as alternate methods of learning appropriate weight parameters. As in the above methods, there is no attempt to modify the learned theory, only the weights.

Support Vector Machines are a recent addition to the SRL toolkit, with contributions of Support Vector Inductive Logic Programming (SVILP) from Muggleton et al. [13], and kFOIL by Landwehr et al. [12]. SVILP is most similar to our work, in that both use learned first-order clauses to create a kernel for probabilistic output. However, where they use mainly a Gaussian kernel with a prior probability over the clauses, we explore kernel methods and clause generation that are informed by precision and recall on the training set. kFOIL presents a dynamic kernel construction process, where the choice of clauses to add is informed by the current classification accuracy. Conversely, GleanerSRL learns clauses first and then constructs the kernel, and through the use of Gleaner we can quickly and in parallel explore a large area of large hypothesis spaces.

We have explored the use of GleanerSRL through comparisons on two relational domains. In future work, we plan to compare with other SRL methods and apply GleanerSRL to much larger testbeds, where we hope to see significant speedups in search time due to using Gleaner over other methods. We also plan to investigate other kernel methods and propositional learning algorithms, as well as alternate feature vector transformations.

5 Acknowledgements

We would like to thank Ameet Soni, Jesse Davis, Louis Oliphant, the UW Condor Group, and our anonymous reviewers for their helpful comments and suggestions regarding this work. This project is supported by DARPA grant HR0011-04-1-0007 and DARPA IPTO under contract FA8650-06-C-7606.

References

1. R. Caruana and A. Niculescu-Mizil. An Empirical Comparison of Supervised Learning Algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.
2. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000.
3. J. Davis, I. C. Dutra, D. Page, and V. S. Costa. Establish Entity Equivalence in Multi-Relation Domains. In *Proceedings of the International Conference on Intelligence Analysis*, Vienna, Va, 2005.
4. J. Davis, I. Ong, J. Struyf, E. Burnside, D. Page, and V. S. Costa. Change of Representation for Statistical Relational Learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007.
5. S. Džeroski and N. Lavrac. An Introduction to Inductive Logic Programming. In *Relational Data Mining*, pages 48–66. Springer-Verlag, 2001.
6. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian Network Classifiers. *Machine Learning*, 29(2-3):131–163, 1997.
7. N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning Probabilistic Relational Models. In *Proceedings of the 16th International Conference on Artificial Intelligence*, pages 1300–1309, 1999.
8. L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning.* MIT Press, 2007.
9. M. Goadrich, L. Oliphant, and J. Shavlik. Gleaner: Creating Ensembles of First-order Clauses to Improve Recall-Precision Curves. *Machine Learning*, 64:231–262, 2006.
10. H. Hoos and T. Stutzle. *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann, 2004.
11. D. Koller and A. Pfeffer. Learning Probabilities for Noisy First-Order Rules. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, Nagoya, Japan, August 1997.
12. N. Landwehr, A. Passerini, L. D. Raedt, and P. Frasconi. kFOIL: Learning Simple Relational Kernels. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 2006.
13. S. Muggleton, A. Amini, H. Lodhi, and M. Sternberg. Support Vector Inductive Logic Programming. In *Proceedings of the 8th International Conference on Discovery Science*, 2005.
14. A. Niculescu-Mizil and R. Caruana. Predicting Good Probabilities with Supervised Learning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 625–632, 2005.

15. U. Pompe and I. Kononenko. Naive Bayesian Classifier within ILP-R. In *Fifth International Workshop on Inductive Logic Programming*, pages 417–436, 1995.
16. 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.
17. M. Richardson and P. Domingos. Markov Logic Networks. *Machine Learning*, 62:107–136, 2006.
18. A. Srinivasan. The Aleph Manual Version 4. *http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/*, 2003.
19. A. Srinivasan and R. King. Feature Construction with Inductive Logic Programming: A Study of Quantitative Predictions of Biological Activity Aided by Structural Attributes. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 352–367. Stockholm University, Royal Institute of Technology, 1996.
20. B. Taskar, P. Abbeel, M.-F. Wong, and D. Koller. Label and Link Prediction in Relational Data. In *IJCAI Workshop on Learning Statistical Models from Relational Data*, 2003.
21. B. Zadrozny and C. Elkan. Transforming Classifier Scores into Accurate Multiclass Probability Estimates. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.