Appears in *Proceedings of the 14th International Conference on Inductive Logic Programming (ILP)*. Porto, Portugal. September, 2004.

Learning Ensembles of First-Order Clauses for Recall-Precision Curves: A Case Study in Biomedical Information Extraction

Mark Goadrich, Louis Oliphant and Jude Shavlik

Department of Biostatistics and Medical Informatics and
Department of Computer Sciences,
University of Wisconsin-Madison, USA

Abstract. Many domains in the field of Inductive Logic Programming (ILP) involve highly unbalanced data. Our research has focused on Information Extraction (IE), a task that typically involves many more negative examples than positive examples. IE is the process of finding facts in unstructured text, such as biomedical journals, and putting those facts in an organized system. In particular, we have focused on learning to recognize instances of the protein-localization relationship in Medline abstracts. We view the problem as a machine-learning task: given positive and negative extractions from a training corpus of abstracts, learn a logical theory that performs well on a held-aside testing set. A common way to measure performance in these domains is to use *precision* and *recall* instead of simply using accuracy. We propose Gleaner, a randomized search method which collects good clauses from a broad spectrum of points along the recall dimension in recall-precision curves and employs an "at least N of these M clauses" thresholding method to combine the selected clauses. We compare Gleaner to ensembles of standard Aleph theories and find that Gleaner produces comparable testset results in a fraction of the training time needed for ensembles.

1 Introduction

Domains suitable for Inductive Logic Programming (ILP) can be roughly divided into two main groups. In one group, there are tasks in which each example has some inherent relational structure. One classic example of this domain is the trains dataset [20], where the goal is to discriminate between two types of trains, and the trains themselves are relational objects, having varying length and types of objects carried by each car. A more realistic example is the mutagenesis dataset [29], where the goal is to classify a chemical compound as mutagenic or not using the relational nature of the atomic structure of each chemical. ILP has proven successful in these domains by bringing the inherently relational attributes into the hypothesis space.

The other group contains tasks where examples, in addition to having a relational structure, have relations to other examples. One such domain is the

learning of friendship in social networks [2], where instead of classifying people, we try to determine the structural relationships of people based on a combination of their personal attributes and the attributes of their known friends. Another domain of this type is learning to suggest citations for scientific publications [21], where a correct citation can be a combination of data in this particular paper as well as the currently listed citations. The overall goal in these domains is to classify *links* between objects instead of the objects themselves.

Our research has focused on Information extraction (IE), the process of finding facts from unstructured text such as biomedical journals and putting those facts in an organized system. In particular, we have focused on learning multi-slot protein localization from Medline[1] abstracts, where the task is to identify *links* between phrases which correspond to a protein and the location of that particular protein in a cell. When seen as a relational data task, multi-slot IE clearly falls into the link-learning category described above.

Link-learning tasks present a number of problems to an ILP system. First, these domains tend to have a large number of objects and relations, causing a large explosion in the search space of clauses. A first approach is to sample these objects and bring the space down to a reasonable size. However, even a moderate number of objects brings about the second problem, a large skew of the data toward negative examples. Suppose in the social network domain we have 500 people, each of whom have 10 friends amongst these 500 people. This gives us 5000 positive examples, assuming that the friendship relationship is not necessarily symmetric. Our negative examples must include all other possible friendships, for $500 \times 500 - 5000 = 245,000$ negative examples, a skew of 1:49.

Information extraction is a domain that typically has unbalanced data; for example, only a very small number of phrases are protein names. Learning the relation between two entities, such as protein and location, only increases this imbalance, as the number of positive examples is now a subset of the cross-product of the entities, and the negative examples are every other pairing in the dataset.

These issues lead us away from using the standard performance measure of accuracy. Letting TP stand for true positives, FP for false positives, TN for true negatives and FN for false negatives, accuracy can be defined as $\frac{TP+TN}{TP+FP+TN+FN}$. With the positive class so small relative to the negative class, it is trivial to achieve high accuracy by labeling all test examples negative. To concentrate on the positive examples, more appropriate performance measures are *precision*, defined as $\frac{TP}{TP+FP}$, and *recall*, defined as $\frac{TP}{TP+FN}$. Precision can be seen as a measure of how accurate we are at predicting the positive class, while recall is a measure of how many of the total positives we are able to identify.

We chose to pursue IE from a machine-learning perspective. Given a set of journal abstracts manually tagged with protein-localization relationships, our goal is to learn a theory that extracts only these relations from a set of abstracts and performs well on unseen abstracts. We use five-fold cross validation, with approximately 250 positive and 120,000 negative examples in each fold. Our

[1] http://www.ncbi.nlm.nih.gov/pubmed

division of examples is not uniform because we chose to split our data into folds at the journal-abstract level (so that all the sentences in a given abstract are in the same fold), and the number of examples per abstract is variable.

We believe that ILP can be applied successfully for Information Extraction in biomedical domains as well as other link-learning tasks. ILP offers us the advantages of a straight-forward way to incorporate domain knowledge and expert advice and will produce logical clauses suitable for analysis and revision by humans to improve performance. We use Aleph [27], a mature ILP system, to learn first-order clauses.

The standard approach to ILP is to learn clauses sequentially until almost all of the positive examples are covered by at least one clause, thus creating a theory. By itself, an individual theory will produce one value for precision and recall, at least if one uses the standard logical approach of disjunction to combine the clauses in a theory. A more useful evaluation would be to create a recall-precision curve, which illustrates the trade-off between these two measurements. One way to create a recall-precision curve from a theory containing M clauses is to require that *at least* N of the clauses are satisfied. By varying N from 1 to M, one can obtain a variety of points in the recall-precision curve [10]. However, ILP systems have not traditionally been designed to produce recall-precision curves, and it is likely that specially designed algorithms will do better than simply counting the number of clauses that are satisfied by a given example.

To address the goal of efficiently producing good recall-precision curves with ILP, we propose the Gleaner algorithm. Gleaner is a randomized search method that collects good clauses from a broad spectrum of points along the recall dimension in recall-precision curves and employs an "at least N of these M clauses" thresholding method to combine the selected clauses. We compare Gleaner to ensembles of standard Aleph theories [11]. We find that Gleaner produces comparable results in a fraction of the training time needed for Aleph ensembles. These smaller theories will also reduce classification time, an important consideration when working with large domains.

2 Biomedical Information Extraction

Information Extraction (IE) is the process of scanning plain text files for objects of interest and facts about these objects. As a learning task, IE is defined as: given information in unstructured text documents, extract the relevant objects and relationships between them. There are two main IE tasks, Named Entity Recognition (NER) and Multi-Slot Extractions. NER can be seen as identifying a single type of object, for example the name of an individual, corporation, gene, or weapon. Successful rule-based approaches for named-entity IE include Rapier [8], a system which learns clauses with the format *prefix, extraction, postfix*, and Boosted Wrapper Induction (BWI) [14], a method for boosting weak rule-based classifiers of extraction boundaries into a powerful extraction method. BWI has been further examined by Kauchak et al. [17] showing results with high recall and high precision on a wide variety of tasks.

"We suggest that SMF1 and SMF2 are mitochondrial membrane proteins that influence PEP-dependent protein import, possibly at the step of protein translocation."

```
protein_location(SMF1, mitochondrial)
protein_location(SMF2, mitochondrial)
```

Fig. 1. Sample Sentence with its Correct Extractions

Multi-slot extraction builds upon the objects found in NER, and looks for a relationship between these items in the text, some examples being a parent-child relationship between individuals, the CEO of a particular company, or the interaction of two proteins in a cell. Multi-slot extraction is typically much harder; not only must the objects of the relation be identified, but also the semantic relationship between these two objects.

Recently, biomedical journal articles have been a major source of interest in the IE community for a number of reasons: the amount of data available is enormous, the objects, proteins and genes, do not have standard naming conventions, and there is a definite interest from biomedical practitioners to quickly find relevant information [3, 26]. Biomedical journals also contain highly domain-specific language, as seen in Figure 1.

Previous machine-learning work in the biomedical multi-slot domain includes a number of different approaches. Ray and Craven [23] use a Hidden Markov Model (HMM) modified to include part of speech tagging, and analyze their method on protein localization, genetic disorder and protein-protein interaction tasks. For the same datasets, Eliassi-Rad and Shavlik [13] implemented a neural network for IE primed with domain-specific prior knowledge. Aitken [1] uses FOIL to perform ILP, working with a closed ontology of entities, while Brunescu et al. [7] propose the use of ELCS, a bottom up approach to finding protein interactions with rule templates for sentences. Brunescu et al. have also extended Rapier and BWI to handle multi-slot extractions.

2.1 Data Labeling

In this paper, we focus on one particular dataset, learning the location of yeast proteins in a cell as illustrated in Figure 1. Our testbed comes from Ray and Craven [23]. The data consist of 7,245 sentences from 871 abstracts found in the Medline database, and contains 1,200 relations. In the original dataset, the labeling was performed semi-automatically, in order to avoid the laborious task of labeling by a human. Protein localizations were gathered from the Yeast Protein Database (YPD), and sentences which contained instances of both a protein and location pair were marked as positive by a computer program.

In our early exploration of the dataset, we found that there were a significant number of false positives that looked like true positives but were apparently missed by the automated labeling algorithm. Also, some of the labelings were

ambiguous at best, finding both parts of a positive protein localization, whereas the human-judged semantics of the sentence did not involve localization. In addition, by using this labeling scheme, we did not have data on all yeast proteins in the corpus, only those listed in YPD. Because of these issues, we decided to relabel the dataset by hand. We were assisted in this effort by Soumya Ray.

To label the positive examples, we manually performed both protein and location named-entity labeling and relational labeling. Our labeling standards differ from those used by other groups [16], as our task is to extract the locations of yeast proteins. If there was any disagreement among the labelers, we did not tag the protein or location, to make sure our training set was as precise as possible at the expense of some recall.

For the protein labeling, we strove to be specific rather than general, and only labeled those words that directly refered to a protein or gene molecule. This included gene names such as "SMF1", protein names like "fet3p" and full chemical names of enzymes, such as "qh2-cytochome c reductase". Therefore, while we would label SEC53 from "SEC53 mutant", we did not label "isp4delta" or "rrp1-1" as these gene products are defective and would not give rise to a functioning protein molecule. We did not label protein families such as "hsp70" unless it was an adjective to a protein, as in "hsp70 dnaK". Fusion proteins, such as when a gene is combined with a fluorescent tag, were labeled as proteins. Protein complexes, antibodies and open reading frames were never labeled as positive protein examples. Also, only proteins that are known to exist in yeast were labeled, not those which were found in other species, since our dataset dealt with the localization of yeast proteins.

Labeling the location words was much more direct. We used a list of known cellular locations listed in an introductory cellular biology text book, including locations and abbreviations such as "cytoskeleton", "membrane", "lumen", "ER", "npc", "bud", etc. Also labeled were location adjectives, such as "nucleoporin" and "ribosomal".

To determine if there was a relationship between any tagged proteins and tagged locations, we used three classifications: clear, ambiguous, or co-occurrence. Relationships directly implied by the text, as in `protein_location(YRB1p, cytosol)` from the sentence "YRB1p is located in the cytosol," were classified as *clear*, while those relationships where the protein location was implied rather than stated, such as `protein_location(LIP5, mitochondrial)` from the sentence "LIP5 mutants undergo a high frequency of mitochondrial DNA deletions," were labeled as *ambiguous*. The correct classification was agreed upon by all three labelers. For our experiments, we used the *clear* category as positive examples, and all other phrase pairings as negative examples. A future goal is to improve our manual-labeling interface.

2.2 Background Knowledge

Instead of the standard feature-vector machine learning setup, ILP uses logical relations to describe the data. Algorithms attempt to construct logical clauses based on this background structure that will separate positive and negative

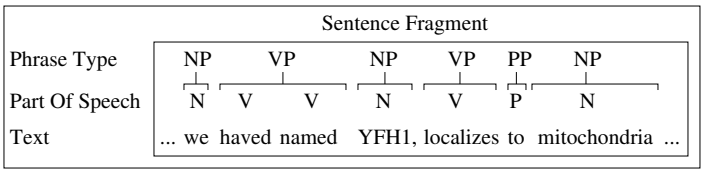

Fig. 2. Sample Sentence Parse from Sundance Sentence Analyzer (N=noun, V=verb, P=preposition or phrase)

examples. For our information extraction task, we construct background knowledge from sentence structure, statistical word frequency, lexical properties, and biomedical dictionaries.

Our first set of relations comes from the sentence structure. We use the Sundance sentence parser [24] to automatically derive a parse tree for all sentences in our dataset and the part-of-speech for all words and phrases of the tree. This tree is then flattened to some degree, so that there are no nested phrases; all phrases have the sentence as the root, and therefore all words are only members of one phrase. Figure 2 shows an example sentence parse.

Each word, phrase, and sentence is given a unique identifier based on its ordering within the given abstract. This allows us to create relations between sentences, phrases and words not based on the actual text of the document but on its structure, such as `sentence_child`, `phrase_previous` and `word_next` about the tree structure and sequence of words, and relations like `nounPhrase`, `article`, and `verb` to describe the sentence structure. To include the actual text of the sentence in our background knowledge, the predicate `word_ID_to_string` maps these identifiers to the words. In addition, the words of the sentence are stemmed using the Porter stemmer [22], and currently we only use the stemmed version of words.

Another group of background relations comes from looking at the frequency of words appearing in the target phrases in the training set. This is done on a per-fold basis to prevent learning from the test set. For example, the words "body", "npc", and "membrane" are at least 10 times more likely to appear in location phrases than in phrases in general in training set 1. We created predicates for several gradations from 2 times to 10 times the general word frequency across all abstracts in a given training set. These gradations are calculated for both arguments–protein and location–as well as for words that appear more frequently in between the two arguments or before or after them. We create semantic classes, consisting of these high frequency words. These semantic classes are then used to mark up all occurrences of these words in a given training and testing set.

A third source of background knowledge is derived from the lexical properties of each word. `Alphanumeric` words contain both numbers and alphabetic characters, whereas `alphabetic` words have only alphabetic characters. Other lexical and morphological features include `singleChar`, `hyphenated` and `capitalized`. Also, words are classified as `novelWord` if they do not appear in the standard `/usr/dict/words` dictionary in UNIX.

Sentence Structure Predicates
phrase_after(Phrase1,Phrase2)
phrase_contains_specific_word(Phrase,Word,WordString)

Statistical Word Frequency Predicates
phrase_contains_2x_word(Phrase,Argument)
phrase_contains_no_between_halfX_word(Phrase,Argument,PartOfSpeech)

Lexical Properties Predicates
alphabetic(Word)
few_wordPOS_in_sentence(Sentence,PartOfSpeech)

Biomedical Dictionaries Predicates
phrase_contains_mesh_term(Phrase,Term,StemmedTerm)
phrase_contains_go_term(Phrase,Term,StemmedTerm)

Fig. 3. Sample Predicates used in our Information Extraction Task

Finally, we incorporate semantic knowledge about biology and medicine into our background relations, such as the Medical Subject Headings (MeSH)[2], the Gene Ontology (GO)[3], and the Online Medical Dictionary[4]. As in sentence structure, we have simplified these hierarchies to only be one level. We have picked three categories from MeSH (protein, peptide and cellular structure), the cellular-localization category from GO, and the cellular-biology category from the Online Medical Dictionary, and have labeled phrases with these predicates if any of the words in the given phrase match any words in the category.

Sentence structure predicates like `word_before` and `phrase_after` are added allowing navigation around the parse tree. Phrases are also tagged as being the first or last phrase in the sentence, likewise for words. The length of phrases is calculated and explicitly turned into a predicate, as well as the length (by words and phrases) of sentences. Also, phrases are classified as short, medium or long. An additional piece of useful information is the predicate `different_phrases`, which is true when its arguments are distinct phrases.

Lexical predicates are augmented to make them more applicable to the phrase level. If a phrase contains an alphabetic word, the phrase is given the predicate `phrase_contains_alphabetic_word(A)`. Similarly phrases with specific words are marked with `phrase_contains_specific_word(A, ''lumen'')`. This is the equivalent of adding both `phrase_child(A,B)`, `word_ID_to_string(B, ''lumen'')` at once. These predicates are also created for pairs and triplets of words, so we can assert that a phrase has the word "golgi" labeled as a noun all in one search step.

[2] http://www.nlm.nih.gov/mesh/meshhome.html
[3] http://www.geneontology.org/
[4] http://cancerweb.ncl.ac.uk/omd/

Finally, predicates are added to denote the ordering between the phrases. `Target_arg1_before_target_arg2` asserts that the protein phrase occurs before the location phrase, similarly for `target_arg2_before_target_arg1`. Also created are `adjacent_target_args` (which is true when the protein and location phrases are adjacent to each other in the sentence), and `identical_target_args` (which says the same noun phrase contains both the protein and its location), as well as the count of phrases before and after the target arguments. A list of our predicate categories and some sample predicates are found in Figure 3. Overall, we have defined 251 predicates for use in describing the training examples.

2.3 Unbalanced Data Filtering

As previously mentioned, one of the difficulties we face with this domain is the large number of possible examples we must consider. Within each sentence, we need to examine each pair of phrases. With only a few positive examples, our positive:negative ratio is 1:600, leading to severely unbalanced data.

For this domain, we use prior knowledge to help reduce the number of false positive examples. We observe that 95% of our positive relations contain only noun phrases, while the overall ratio is 26%, and use this to limit the size of our training data to only those candidate extractions where both arguments are noun phrases. This reduces the positive:negative ratio in our data to 1:158. We must necessarily keep track of all missed positive in the testing set, those that have at most one non-noun phrase, and record them as false negatives in our recall-precision results.

To further reduce the positive:negative ratio we randomly under-sample the negatives, retaining only a fourth during training. This allows for faster clause learning. Future work includes selecting the "close" negative examples to use during training rather than randomly selecting them.

3 Aleph

Aleph [27], is a top-down ILP covering algorithm developed at Oxford University, UK. It is written completely in Prolog and is open source. As input, Aleph takes background information in the form of predicates, a list of modes declaring how these predicates can be chained together, and a designation of one predicate as the "head" predicate to be learned. Also required are lists of positive and negative examples of the head predicate.

As a high-level overview, Aleph generates clauses for the positive examples by picking a random example to be a seed. This example is saturated to create the bottom clause, i.e. every relation in the background knowledge that can be reached from this example. The bottom clause becomes the possible search space for clauses. Aleph heuristically searches through the space of possible clauses until the "best" clause is found or time runs out. The standard way to use Aleph is to combine these learned clauses into a theory when enough clauses are learned to cover almost all positive training examples.

Aleph is a very flexible ILP system with a wide variety of learning parameters available for modification. Some of the parameters we utilized were:

minimum accuracy. We can place a lower bound on the accuracy of all clauses learned by our system. This is only the accuracy of the clause on the examples covered by it, in other words, precision.

minimum positives. To prevent Aleph from learning narrow clauses, ones which only cover a few examples, we can specify that each acceptable clause must cover at least a certain number of positives.

clause length. The size of a particular clause can be constrained using clause length. By limiting the length, we can explore a wider breadth of clauses and prevent clauses from becoming too specific.

search strategy. As Aleph uses search to find good clauses, the type of search is a parameter. These include the standard search methods of breadth-first search, depth-first search, iterative beam search, iterative deepening, as well as heuristic methods requiring an evaluation function.

evaluation function. There are many ways to calculate the value of a node for further exploration. The most common heuristic used in ILP is *coverage*. This is defined as the number of positives covered by the clause minus the number of negatives ($TP - FP$). A very similar heuristic is *compression*, which is coverage minus the length of the clause ($TP-FP-L$). Since we are working within domains to generate precision/recall curves, we also explored as our heuristic-search's evaluation function (a) $precision \times recall$, and (b) the F1 measure, which is ($\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$). To improve clause quality and correct accuracy estimates for clauses that cover a small number of examples, one can also use the Laplace estimate, ($\frac{TP+1}{TP+FP+2}$).

coverage in tune set. To encourage our clauses to be more general, we added a parameter to Aleph requiring each recorded clause to have some small positive coverage in the tuneset. We believe this will help our clauses on the unseen examples in the test set.

4 Gleaner

Since our biomedical IE task is a link-learning task, we need to evaluate the success of our methods using precision and recall. In order to rapidly produce good recall-precision curves, we have developed Gleaner, a two-stage algorithm to (1) learn a broad spectrum of clauses and (2) then combine them into a thresholded disjunctive clause aimed at maximizing precision for a particular choice of recall. Our algorithm is summarized in Figure 4.

Our first stage of Gleaner learns a wide spectrum of clauses. We have Aleph search for clauses using K seed examples. We diversify the search by first uniformly dividing the recall dimension into B equal sized bins, for example, $[0, 0.05], [0.05, 0.10], \ldots, [0.95, 1]$. For each seed, we consider up to N possible clauses using a random local-search method. As these clauses are generated, we compute the recall of each clause and determine into which bin the clause falls.

Create B recall bins, uniformly dividing the range [0,1]
For $i = 1$ to K
    Pick a seed example to generate bottom clause
    Use Random Local Search to find clauses
    After each generation of a new clause r
        Find the recall bin b_k for r
        If the $Precision \times Recall$ of r is best yet
            Store r in b_k
For each bin b
    Find $L_b \in [1, K]$ on trainset such that
        recall of "At least L of K clauses match examples" $\approx$ recall for this bin
Find precision and recall of testset using each bin's "at least L of K" decision process

Fig. 4. Gleaner Algorithm

Each bin keep tracks of the highest precision clause learned in that bin so far and will be replaced when a more precise clause is found (actually, rather than finding the highest precision clause within each bin, we save the clause whose product of precision and recall is highest among those clauses falling into this recall bin). At the end of this search process, there will be B clauses collected for each seed and K seed examples for a total of $B \times K$ clauses (assuming a clause is found that falls into each bin for each seed).

To perform random local search, we considered four search methods, Rapid Random Restart (RRR), Stochastic Clause Selection (SCS), GSAT, and WalkSAT. SCS randomly picks clauses which are subsets of the bottom clause according to the distribution of clauses based on length. SCS has a hard time finding high quality clauses and is biased to select long clauses due to the heavy-tailed distribution of clause lengths. GSAT selects an initial clause at random and then chooses to either add or remove a randomly selected literal if the new clause is "better" according to the evaluation function; WalkSAT modifies GSAT by allowing a certain percent of "bad" moves. RRR works similarly to GSAT and WalkSAT in the initial clause selection, but only refines clauses by adding predicates (using best first search), restarting with a new clause after a specified number of evaluations. GSAT and WalkSAT occasionally make "downhill" moves in the search space, while RRR does not, and due to the interal workings of Aleph, adding predicates to a clause is much more efficient than removing them. We found that RRR both takes less time and produces higher quality clauses than the other methods, and we use it as Gleaner's search method in the remainder of this article.

The second stage takes place once we have gathered our clauses using random search. We need a way to combine these clauses into a single precision/recall point for each bin. We could choose the best clause collected from each bin, however this is likely to have poor generalization to the test set, especially for the low-recall bins. If we classify an example as positive only if it matches *all* K clauses collected for a bin, we obtain high precision, but our recall will be dras-

tically reduced. Alternatively, if we classify an example as positive if it matches *any* of our K clauses, we will probably have a theory with high recall but low precision. Instead, we need to find a balance between these two extremes, and classify examples to be positive if they are covered by a large enough subset of clauses. *Our hypothesis is that this method will produce a theory with about the same recall as the bin (by construction), but higher precision than any one clause, since we require that an example satisfy multiple clauses (assuming $L > 1$).*

Gleaner combines the clauses in each bin to create one large thresholded disjunctive clause, of the form "At least L of these K clauses must cover an example in order to classify it as a positive." We want this clause to have about the same recall as as that of the clauses in the bin (so that we cover the full range of possible recalls), thus we need to find the best threshold L for each bin. We can find this L on the training set for each bin by starting with $L = K$ and incrementally lowering the threshold to increase recall. We stop when any lower L would increase the distance between the recall of the best L of K clause and our desired recall. With this L, we now evaluate our disjunctive clause on the testset and record the precision and recall. We will end up with B precision/recall points, one for each bin, that span the recall-precision curve.

5 Ensembles in ILP

Bagging [6] is a popular ensemble approach to machine learning where multiple classifiers are trained using different subsamples of the training data. These classifiers then vote on the classification of testset examples, usually with the majority class being selected as the output classification. How they vote is user-dependent, with some common schemes being equal voting or weighted according to the tuneset accuracy of each voter. The main idea of bagging is that it will produce diverse classifiers that make their mistakes in different regions of the input space; when their votes are combined, prediction errors will be reduced.

The use of bagging for ILP has been previously investigated by Dutra et al. [11] where they demonstrate bagging to be helpful for modest improvements in accuracy as well as a straight-forward way to calculate the confidence of a particular example. We use their "random seeds" approach for creating ensembles. This approach, which Dutra et al. showed to have essentially equivalent predictive accuracy as bagging, produces diversity in its learned models by starting each run of its underlying ILP system with a different "seed" example.

We compare our Gleaner approach to that of using "random seeds" in Aleph. In this experimental control, we call Aleph N times and have it create N theories (i.e., sets of clauses that cover most of the positive training examples and few of the negative ones). To create a recall-precision curve from these N theories, we simply classify an example as positive if at least K of the theories classify it as positive; varying L from 1 to N produces a family of ensembles, and each of these ensembles produces a point on a recall-precision curve.

Aleph involves a large number of parameters, and we use the train and test sets to choose a good set (since this is the experimental control against which we

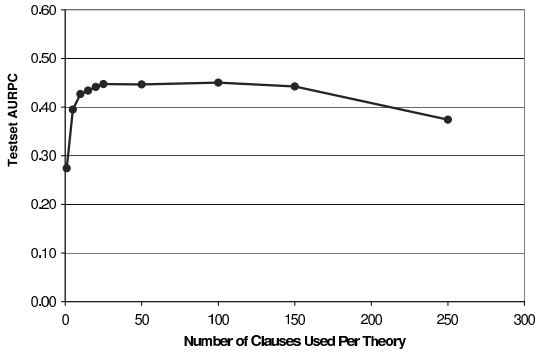

Fig. 5. Area Under the Recall-Precision Curve for 100 Aleph Ensembles With Varying Number of Clauses

compare our Gleaner system, it is "fair" to use the testset to tune parameters). We compare several different evaluation functions for judging clauses: Laplace (which essentially measures accuracy, but corrects for small coverage), coverage (the number of positive examples covered minus the number of negatives covered), $precision \times recall$, and $F1$ (the harmonic mean of precision and recall; F1 is the most commonly used performance measure in information extraction). We consider two settings for minimum accuracy for learned clauses: 0.75 and 0.90. We require all clauses to at least cover seven positive examples and to be no longer than ten terms (the same settings we use for random sampling of the hypothesis space in our Gleaner approach). We limit the number of clauses considered to 100 thousand and we also limit the number of reductions to 100 million (using the `call_counting` predicate available in YAP Prolog[5]).

We obtained our best area under the recall-precision curve using Laplace as the evaluation function and a minimum clause accuracy of 0.75. (Under this setting, the average number of clauses considered per constructed theory is approximately 35,000.)

One new finding we encountered that was not reported by Dutra et al. is that it is better to limit the size of theories. Figure 5 plots the area under the recall-precision curve (AURPC) as a function of the maximum number of clauses we allow in the learned theories. Running Aleph to its normal completion given the above parameters leads to theories containing 271 clauses on average. However, if we limit this to the first C clauses, the AURPC can be drastically better. The likely reason for this is that larger theories have less diversity amongst themselves than do smaller ones, and diversity is the key to ensembles [12]. A nice side-effect of limiting theory size is that the runtime of individual Aleph executions is substantially reduced.

In the next section, where we evaluate our Gleaner algorithm, we limit theory size in our "ensemble of Aleph theories" approach to 50 clauses, since as seen in Figure 5, testset AURPC has essentially peaked by then. In that section's

[5] http://www.ncc.up.pt/~vsc/Yap/yap.html

experiments we do vary the size of the ensemble (i.e., number of theories) and the number of clauses in each theory, in order to see the impact on AURPC as a function of the amount of time spent training.

While we are from having considered all possible parameters settings and algorithm designs with which one could use Aleph to create an ensemble of theories, we have evaluated a substantial number of variants and feel that our chosen settings provide a satisfactory experiment control against which to compare our new algorithm, Gleaner.

6 Results

For our experiments, we divided the protein localization data into five folds, equally divided at the journal-abstract level. Each training set consisted of three folds, with one fold held aside for tuning and another for testing. For our current experiments we only use the tuning set minimally, requiring each clause learned on the training set to cover at least two positive examples in the tuning set.

To evaluate the performance of our algorithms, we use recall-precision curves [19], or more precisely, we use the Area Under the Recall-Precision Curve (AURPC) to gather a single score for each algorithm. AUC has traditionally been used to analyze ROC curves [5], which plot the true positive rate versus the false positive rate. To calculate the AURPC, we first standardize our recall-precision curves to always cover the full range of recall values and then interpolate between the threshold points. From the first threshold point, which we designate (R_{first}, P_{first}), the curve is extended horizontally to the point $(0, P_{first})$, since we could randomly discard a fraction, f, of the extracted relations and expect the same precision on the remaining examples; the setting of f would determine the recall. An ending point of $(1, \frac{Pos}{Neg})$ can always be found by calling everything a positive example. This will give us a closed curve extending from 0 to 1 along the recall dimension.

For any two points A and B in a recall-precision curve, we must interpolate between their true positive (TP) and false positive (FP) counts in order to calculate the area. To do this, we create new points for each of $TP_A + 1, TP_A + 2, ..., TP_B - 1$, increasing the false positives for each new point by $\frac{FP_B - FP_A}{TP_B - TP_A}$. Interpolation for the recall-precision curve is different than for an ROC curve; whereas the ROC interpolation would be a linear connection between the two points, in recall-precision space the connection can be curved, depending the actual number of positive and negative examples covered by each point. The curve is especially pronounced when two points are far away in recall and precision. Consider a curve constructed from a single point of $(0.02, 1)$, and extended to the endpoints of $(0, 1)$ and $(1, 0.008)$ as described above (for this example, our dataset contains 433 positives and 56,164 negatives). Interpolating as we have described, would produce an AURPC of 0.031; a linear connection would overestimate with an AURPC of 0.50 (Figure 8 shows this graphically).

A sample clause found by Gleaner is shown in Figure 6. We can see for our dataset that it is important to require the protein phrase to contain

```
protein_location(P,L,S) :-
    first_word_in_phrase(L,A),
    phrase_after(L,_),
    target_arg1_before_target_arg2(P,L,S),
    after_both_target_phrases(S,B),
    phrase_contains_some_marked_up_location(L,_),
    few_POS_in_phrase(P,alphanumeric),
    few_wordPOS_in_sentence(S,alphanumeric),
    phrase_contains_no_between_halfX_word(B,between_arg1_and_arg2,verb),
    phrase_contains_some_art(L,A).
```

where P is the protein phrase, L is the location phrase, S is the sentence, and '_' indicates variables that only appear once in the clause.

Positive Extraction
"NPL3 encodes a nuclear protein with an RNA recognition motif and similarities to a family of proteins involved in RNA metabolism."
`protein_location('NPL3', 'a nuclear protein')`

Negative Extraction (i.e., a false positive)
"Subcellular fractionation studies further demonstrate that the 1455 amino acid Vps15p is peripherally associated with the cytoplasmic face of a late Golgi or vesicle compartment."
`protein_location('the 1455 amino acid Vps15p', 'the cytoplasmic face')`

Fig. 6. Sample Clause with 29% Recall and 34% Precision on Testset 1

`alphanumeric` words. Also important for this clause is the sentence structure, requiring that the protein phrase comes before the location phrase, and that the location phrase is not the last phrase in the sentence.

Our Aleph-based method for producing ensembles has two parameters that we vary: N, the number of theories (i.e., the size of the ensemble), and C, the number of clauses per ensemble. To produce ensemble points in Figure 7, we choose N from {10, 25, 50, 75, 100} and C from {1, 5, 10, 15, 20, 25, 50}, producing 20 combinations for each fold.

For the parameters of Gleaner, we used 20 recall bins and 100 seed examples to collect 2,000 clauses total. We told RRR to construct 1,000 clauses before restarting with a new random clause. We generate AURPC data points for Gleaner by choosing the number of seed examples from {25, 50, 75, 100}, and using the intervals of {1K, 10K, 25K, 50K, 100K, 250K, 500K} for the number of candidate clauses generated per seed.

The results of our comparison are found in Figure 7; the points are averaged over all five folds. Note this graph has a logarithmic scale in the number of clauses generated. We see that Gleaner can find comparable AURPC numbers using two orders of magnitude fewer clauses. It is interesting to note that the

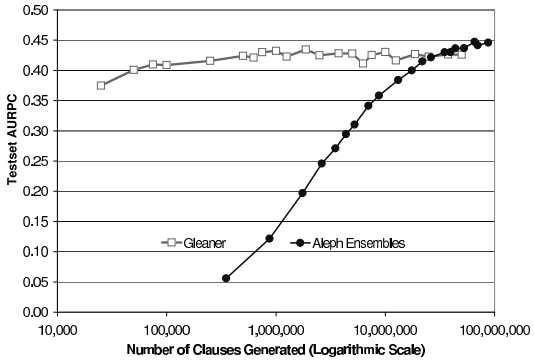

Fig. 7. Comparison of AURPC from Gleaner and Aleph Ensembles by Varying Number of Clauses Generated

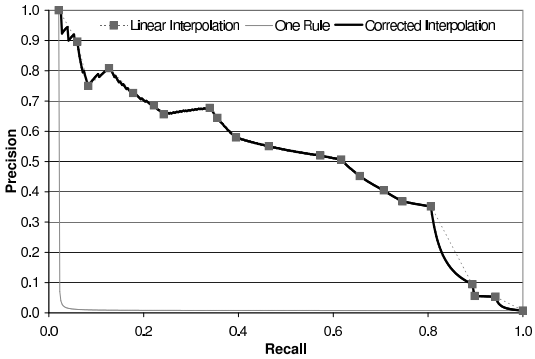

Fig. 8. A Sample Gleaner Recall-Precision Curve From Fold 5

Gleaner curve is very consistent across the number of clauses allowed, while the ensemble method increases when more clauses are considered. It is a topic of future work to devise a new version of Gleaner that is able to better utilize additional candidate clauses.

In Figure 8, we show one of the better recall-precision curve produced by Gleaner using 10,000 candidate clauses per seed and 100 seed examples (on fold 5). For comparison, we also show the one-point interpolation curve mentioned above. Gleaner's "L of K" clauses theoretically should produce higher precision than individual rules with the same recall, as long as coverage of positives is greater than coverage of negatives. In practice, our clauses are not as independent as we would like, especially in the high-recall bins, with many of the learned clauses being identical. This overlap degrades the performance.

7 Conclusions and Future Work

Multi-Slot Information Extraction is a an appealing challenge task for ILP, due to its large amount of examples and background knowledge, as well as the substantial skew of examples. We have developed a method called Gleaner, which gathers a wide spectrum of clauses and combines them within bins based on recall using an "at least N of these M clauses" thresholding method.

We find that Aleph ensembles can perform well when using early stopping (i.e., only learning a dozen or two rules); however, Aleph ensembles suffer when allotted a limited amount of time to create multiple theories. Our method of Gleaner results in similar curves to Aleph ensembles, and outperforms ensembles when both are only allowed to evaluate a limited number of clauses. There are not many large, heavily skewed datasets available for ILP research, and we believe this information-extraction task will provide a useful testbed for further ILP research. To aid in ILP research this dataset is being made available at our website (see Acknowledgements).

There are a number of approaches relating to the combination of learned clauses to produce a confidence measure, as opposed to combining multiple theories as in bagging or Gleaner. Propositionalization of the feature space has been examined by Lavrac et al. [18], which allows for any propositional learner that generates confidence measures to be used. Similarly, Srinivasan [28] investigated using ILP as a feature construction tool for propositional learners, namely linear regression. Craven and Slatterly [10] use a logical setup combined with Naive Bayes classifiers for IE and generate recall-precision curves with their resulting theories. We plan to compare these within-theory ensemble methods to the multiple theory ensemble methods and to Gleaner.

In this same vein, we see the use of boosting in ILP [15] as another alternative method to searching for clauses and learning how to combine them in one single step. Recent work has shown that a RankBoost, a variant of boosting, directly optimizes the area under the ROC curve [9]. We believe that a similar optimization of the area under the recall-precision curve can be achieved, and plan to implement this algorithm in Aleph for comparison to Gleaner.

We noticed that many of our learned clauses are focused on learning the individual entities of the relation, in our case, creating logical clauses for protein and location, and little of the clause is relevant to the relation *between* these two entities. We believe that using a named-entity classifier to identify promising pieces of our relation first could both reduce the number of examples as well as produce high quality clauses due to their direct focus on the relation. Blaschke et al. [4, 3] and Rindflesh et al. [25] have found success in biomedical information extraction using domain expert rules, and Temkin and Gilder [31] use handcrafted context-free grammars to similar ends. Another step in this direction is taking these clauses from a domain expert and learning to revise their advice, similar to work by Eliassi-Rad and Shavlik [13].

Finally, there are many more datasets in Information Extraction where we are planning to test our method for comparison, namely the genetic disorder and protein interaction from Ray and Craven [23] and a protein interaction dataset

from Brunescu et al. [7]. Other datasets outside of IE where we believe Gleaner will be useful include the nuclear smuggling dataset from Tang et al. [30], the social network dataset from Taskar et al. [2], and the CiteSeer citation dataset from Popescul et al. [21]

8 Acknowledgements

Our dataset can be found at ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/datasets/IE-protein-location

This work was supported by National Library of Medicine (NLM) Grant 5T15 LM007359-02, NLM Grant 1R01 LM07050-01, DARPA EELD Grant F30602-01-2-0571, and United States Air Force Grant F30602-01-2-0571. We would like to thank Ines Dutra and Vitor Santos Costa for their help with Yap, the UW Condor Group for Condor assistance, Soumya Ray and Marios Skounakis for their help with labeling the data, and David Page for his help with Aleph, as well as the anonymous reviewers for their informative comments.

References

1. S. Aitken. Learning Information Extraction Rules: An Inductive Logic Programming Approach. In F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence*, Amsterdam, 2002.
2. M.-F. W. Ben Taskar, Pieter Abbeel and D. Koller. Label and Link Prediction in Relational Data. In *IJCAI Workshop on Learning Statistical Models from Relational Data*, 2003.
3. C. Blaschke, L. Hirschman, and A. Valencia. Information Extraction in Molecular Biology. *Briefings in Bioinformatics*, 3(2):154–165, 2002.
4. C. Blaschke and A. Valencia. Can Bibliographic Pointers for Known Biological Data be Found Automatically? Protein Interactions as a Case Study. *Comparative and Functional Genomics*, 2:196–206, 2001.
5. A. Bradley. The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.
6. L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.
7. R. Bunescu, R. Ge, R. Kate, E. Marcotte, R. Mooney, A. Ramani, and Y. Wong. Comparative Experiments on Learning Information Extractors for Proteins and their Interactions. *Journal of Artificial Intelligence in Medicine*, 2004.
8. M. Califf and R. Mooney. Relational Learning of Pattern-Match Rules for Information Extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.
9. C. Cortes and M. Mohri. AUC Optimization vs. Error Rate Minimization. In *Neural Information Processing Systems NIPS2003*, 2003.
10. M. Craven and S. Slattery. Relational Learning with Statistical Predicate Invention: Better Models for Hypertext. *Machine Learning*, 43(1/2):97–119, 2001.
11. I. de Castro Dutra, D. Page, V. S. Costa, and J. Shavlik. An Empirical Evaluation of Bagging in Inductive Logic Programming. In *Twelfth International Conference on Inductive Logic Programming*, pages 48–65, Sydney, Australia, 2002.

12. T. Dietterich. Machine-Learning Research: Four Current Directions. *The AI Magazine*, 18(4):97–136, 1998.
13. T. Eliassi-Rad and J. Shavlik. A Theory-Refinement Approach to Information Extraction. In *Proceedings of the 18th International Conference on Machine Learning*, 2001.
14. D. Freitag and N. Kushmerick. Boosted Wrapper Induction. In *AAAI/IAAI*, pages 577–583, 2000.
15. S. Hoche and S. Wrobel. Relational Learning Using Constrained Confidence-Rated Boosting. In *11th International Conference on Inductive Logic Programming*, Strasbourg, France, 2001.
16. Z. Hu. Guidelines for Protein Name Tagging. Technical report, Georgetown University, 2003.
17. D. Kauchak, J. Smarr, and C. Elkan. Sources of Success for Boosted Wrapper Induction. *Journal of Machine Learning Research*, 5:499–527, May 2004.
18. N. Lavrac, F. Zelezny, and P. Flach. RSD: Relational Subgroup Discovery through First-order Feature Construction. In *Proceedings of the 12th International Conference on Inductive Logic Programming (ILP'02)*, Sydney, Australia, 2002.
19. C. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
20. R. Michalski and J. Larson. Inductive Inference of VL Decision Rules. In *Proceedings of the Workshop in Pattern-Directed Inference Systems*, May 1977.
21. A. Popescul, L. Ungar, S. Lawrence, and D. Pennock. Statistical Relational Learning for Document Mining. In *IEEE International Conference on Data Mining, ICDM-2003*, 2003.
22. M. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.
23. 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 (IJCAI-2001)*, 2001.
24. E. Riloff. The Sundance Sentence Analyzer. *http://www.cs.utah.edu/projects/nlp/*, 1998.
25. T. Rindflesch, T. Tanabe, L. Weinstein, and J. Hunter. Edgar: Extraction of drugs, genes and relations from the biomedical literature. In *Proceedings of the Pacific Symposium on Biocomputing.*, 2000.
26. H. Shatkay and R. Feldman. Mining the Biomedical Literature in the Genomic Era: An Overview. *Journal of Computational Biology*, 10(6):821–55, 2003.
27. A. Srinivasan. The Aleph Manual Version 4. *http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/*, 2003.
28. 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.
29. A. Srinivasan, S. Muggleton, M. Sternberg, and R. King. Theories for Mutagenicity: A Study in First-Order and Feature-Based Induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.
30. L. Tang, R. Mooney, and P. Melville. Scaling up ILP to Large Examples: Results on Link Discovery for Counter-Terrorism. In *KDD Workshop on Multi-Relational Data Mining*, 2003.
31. J. Temkin and M. Gilder. Extraction of Protein Interaction Information From Unstructured Text Using a Context-Free Grammar. *Bioinformatics*, 19(16):2046–2053, 2003.