Appears in *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 328-334, Orlando, FL, July 1999

Relational Learning of Pattern-Match Rules for Information Extraction

Mary Elaine Califf
Applied Computer Science Department
Illinois State University
Normal, IL 61790
mecalif@ilstu.edu

Raymond J. Mooney
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
mooney@cs.utexas.edu

Abstract

Information extraction is a form of shallow text processing that locates a specified set of relevant items in a natural-language document. Systems for this task require significant domain-specific knowledge and are time-consuming and difficult to build by hand, making them a good application for machine learning. We present a system, RAPIER, that uses pairs of sample documents and filled templates to induce pattern-match rules that directly extract fillers for the slots in the template. RAPIER employs a bottom-up learning algorithm which incorporates techniques from several inductive logic programming systems and acquires unbounded patterns that include constraints on the words, part-of-speech tags, and semantic classes present in the filler and the surrounding text. We present encouraging experimental results on two domains.

Introduction

As the amount of information available in the form of electronic documents increases, so does the need to intelligently process such texts. Of particular importance is information extraction (IE), the task of locating specific pieces of data from a natural language document, allowing one to obtain useful structured information from unstructured text. In recognition of their significance, IE systems have been the focus of DARPA's MUC program (DARPA 1995). Unfortunately, IE systems, although they do not attempt full text understanding, are still time-consuming to build and generally contain highly domain-specific components, making them difficult to port to new applications.

Thus, IE systems are an attractive application for machine learning. Several researchers have begun to use learning methods to aid the construction of IE systems (Soderland *et al.* 1995; Riloff 1993; Kim & Moldovan 1995; Huffman 1996). However, in these systems, learning is used for part of a larger IE system. Our system, RAPIER (Robust Automated Production of IE Rules), was one of the first systems to learn rules for the complete IE task. The resulting rules extract the desired items directly from documents without parsing or subsequent processing. Simultaneous with RAPIER's development, other learning systems have recently been developed for this task (Freitag 1999; Soderland 1999). Using a corpus of documents paired with filled templates, RAPIER learns unbounded patterns that use limited syntactic information, such as the output of a part-of-speech (POS) tagger, and semantic class information, such as that provided by WordNet (Miller *et al.* 1993).

The remainder of the paper is organized as follows. Section 2 presents background material on IE and relational learning. Section 3 describes RAPIER's rule representation and learning algorithm. Section 4 presents and analyzes results obtained on two domains and compares RAPIER's performance to a simple Bayesian learner and two relational learners. Section 5 discusses related work in applying learning to IE and Section 6 presents our conclusions.

Background

Information Extraction

Our system addresses the type of IE problem in which strings directly lifted from a document are used to fill slots in a specified template. Figure 1 shows part of a job posting and the corresponding slots of the filled computer-science job template.

IE can be useful in a variety of domains. The various MUC's have focused on domains such as Latin American terrorism, joint ventures, microelectronics, and company management changes. Others have used IE to track medical patient records (Soderland *et al.* 1995) or company mergers (Huffman 1996). A general task considered in this paper is extracting information from postings to

Copyright ©1999, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Posting from Newsgroup

```
Telecommunications. SOLARIS Systems
Administrator. 38-44K. Immediate need

Leading telecommunications firm in need
of an energetic individual to fill the
following position in the Atlanta
office:

  SOLARIS SYSTEMS ADMINISTRATOR
  Salary: 38-44K with full benefits
  Location: Atlanta Georgia, no
        relocation assistance provided
```

Filled Template

```
computer_science_job
title: SOLARIS Systems Administrator
salary: 38-44K
state: Georgia
city: Atlanta
platform: SOLARIS
area: telecommunications
```

Figure 1: Sample Message and Filled Template

USENET newsgroups, such as job announcements.

Relational Learning

Most empirical natural-language research has employed statistical techniques that base decisions on very short fixed-length contexts, or symbolic techniques such as *decision trees* that require the developer to specify a manageable, finite set of features for use in making decisions. Inductive Logic Programming (ILP) and other *relational learning* methods allow induction over *structured* examples that can include first-order logical representations and unbounded data structures such as lists, strings, and trees. Experimental comparisons of ILP and feature-based induction have demonstrated the advantages of relational representations in two language related tasks, text categorization (Cohen 1995) and generating the past tense of English verbs (Mooney & Califf 1995). While RAPIER is not strictly an ILP system, its learning algorithm was inspired by ideas from three ILP systems.

GOLEM (Muggleton & Feng 1992) employs a bottom-up algorithm based on the construction of relative least-general generalizations, *rlggs*. Rules are created by computing the *rlggs* of randomly selected positive examples. CHILLIN (Zelle & Mooney 1994) combines bottom-up and top-down techniques. The algorithm starts with a most specific definition (the set of positive examples) and introduces generalizations that compress the definition. The third system, PROGOL (Muggleton 1995), also combines bottom-up and top-down search. It constructs a most specific clause for a random seed example and employs a A* search to find the simplest consistent generalization.

Pre-filler:	Filler:	Post-filler:
1) tag: {nn,nnp}	1) word: undisclosed	1) sem: price
2) list: length 2	tag: jj	

Figure 2: Sample Rule Learned by RAPIER

The RAPIER System

Rule Representation

RAPIER's rule representation uses patterns that can make use of limited syntactic and semantic information. The extraction rules are indexed by template name and slot name and consist of three parts: 1) a pre-filler pattern that matches text immediately preceding the filler, 2) a pattern that matches the actual slot filler, and 3) a post-filler pattern that matches the text immediately following the filler. Each pattern is a sequence of pattern elements of one of two types: *pattern items* and *pattern lists*. A pattern item matches exactly one word that satisfies its constraints. A pattern list has a maximum length N and matches 0 to N words, each satisfying a set of constraints. RAPIER uses three kinds of constraints: on the specific word, on its assigned part-of-speech, and on its semantic class. The constraints are disjunctive lists of one or more words, tags, or semantic classes.

Figure 2 shows a rule constructed by RAPIER for extracting the transaction amount from a newswire concerning a corporate acquisition. This rule extracts the value "undisclosed" from phrases such as "sold to the bank for an undisclosed amount" or "paid Honeywell an undisclosed price". The pre-filler pattern consists of two pattern elements: 1) a word whose POS is noun or proper noun, and 2) a list of at most two unconstrained words. The filler pattern requires the word "undisclosed" tagged as an adjective. The post-filler pattern requires a word in the WordNet semantic category "price".

The Learning Algorithm

RAPIER's learning algorithm is compression-based and primarily consists of a specific to general search. We chose a bottom-up approach in order to limit search without imposing artificial limits on the constants to be considered, and in order to prefer high precision (by preferring more specific rules), which we believe is relatively more important in many IE tasks.

Like CHILLIN, RAPIER begins with a most specific definition and compresses it by replacing sets of rules with more general ones. To construct the initial definition, most-specific patterns for each slot

are created for each example, specifying words and tags for the filler and its complete context. Thus, the pre-filler pattern contains an item for each word from the beginning of the document to the word immediately preceding the filler with constraints listing the specific word and its POS tag. Likewise, the filler pattern has one item from each word in the filler, and the post-filler pattern has one item for each word from the end of the filler to the end of the document.

Given this maximally specific rule-base, RAPIER attempts to compress the rules for each slot. New rules are created by selecting pairs of existing rules and creating generalizations (like GOLEM). The aim is to make small generalization steps so a standard approach is to generate the least general generalization (LGG) of each pair of rules. However, since our pattern language allows for unconstrained disjunction, the LGG may be overly specific. Therefore, in cases where the LGG of two constraints is a disjunction, we create two alternative generalizations: 1) the disjunction and 2) the removal of the constraint. Since patterns consist of a sequence of items, this results in a combinatorial number of potential generalizations, and it is intractable to compute complete generalizations of two initial rules.

Although we do not want to arbitrarily limit the length of a pre-filler or post-filler pattern, it is likely that the important parts of the pattern will be close to the filler. Therefore, RAPIER starts with rules containing only generalizations of the filler patterns and performs a kind of top-down beam search to efficiently specialize the pre and post fillers. It maintains a priority queue of the best k rules and repeatedly specializes them by adding pieces of the generalizations of the pre and post-filler patterns of the seed rules, working outward from the fillers. The rules are ordered using an information gain metric (Quinlan 1990) weighted by the size of the rule (preferring smaller rules). When the best rule in the queue produces no spurious fillers when matched against the training texts, specialization ceases and it is added to the final rule base, replacing any more specific rules it renders superfluous. Specialization is abandoned if the value of the best rule does not improve across k specialization iterations. Compression of the rule base for each slot is abandoned when the number of successive iterations of the compression algorithm that fail to produce a compressing rule exceed a pre-defined limit. Figure 3 gives psuedocode for the basic algorithm. *SpecializePreFiller* and *SpecializePostFiller* create specializations of *CurRule* using the n items from the context preceding or following the filler.

As an example of the creation of a new rule, consider generalizing the rules based on the phrases "located in Atlanta, Georgia." and "offices in Kansas City, Missouri." These phrases are sufficient to demonstrate the process. The initial, specific rules created from these phrases for the city slot for a job template would be

```
For each slot, S in the template being learned
   SlotRules = most specific rules for S from examples
   while compression has failed fewer than CLim times
      RuleList = an empty priority queue of length k
      randomly select M pairs of rules from SlotRules
      find the set L of generalizations of the fillers of
         each rule pair
      for each pattern P in L
         create a rule NewRule with filler P and empty
            pre and post-fillers
         evaluate NewRule and add NewRule to RuleList
      let n = 0
      loop
         increment n
         for each rule, CurRule, in RuleList
            NewRL = SpecializePreFiller(CurRule,n)
            evaluate rules in NewRL and add to RuleList
         for each rule, CurRule, in RuleList
            NewRL = SpecializePostFiller(CurRule,n)
            evaluate rules in NewRL and add to RuleList
      until best rule in RuleList produces only valid fillers
            or the value of the best rule in RuleList has
            failed to improve over the last Lim iterations
      if best rule in RuleList covers no more than an
            allowable percentage of spurious fillers
         add to SlotRules, removing empirically
            subsumed rules
```

Figure 3: RAPIER Algorithm

```
Pre-filler:          Filler:          Post-filler:
1) word: located     1) word: atlanta 1) word: ,
   tag: vbn             tag: nnp         tag: ,
2) word: in                           2) word: georgia
   tag: in                               tag: nnp
                                      3) word: .
                                         tag: .
and
Pre-filler:          Filler:          Post-filler:
1) word: offices     1) word: kansas  1) word: ,
   tag: nns             tag: nnp         tag: ,
2) word: in          2) word: city    2) word: missouri
   tag: in              tag: nnp         tag: nnp
                                      3) word: .
                                         tag: .
```

For the purposes of this example, we assume that there is a semantic class for states, but not one for cities. The fillers are generalized to produce two possible rules with empty pre and post-filler patterns. Because one filler has two items and the other only one, they generalize to a list of no more than two words. The word constraints generalize to either a disjunction of all the words or no constraint. The tag constraints on all of the items are the same, so the generalized rule's tag constraints are also the same. Since the three words do not belong to a single semantic class in the lexicon, the semantics remain unconstrained. The fillers pro-

duced are:

Pre-filler:	Filler:	Post-filler:
	1) list: max length: 2 word: {atlanta, kansas, city} tag: nnp	

and

Pre-filler:	Filler:	Post-filler:
	1) list: max length: 2 tag: nnp	

Either of these rules is likely to cover spurious examples, so we add pre-filler and post-filler generalizations. At the first iteration of specialization, the algorithm considers the first pattern item to either side of the filler. The items produced from the "in"'s and commas are identical and, therefore, unchanged. Continuing the specialization, the algorithm considers the second to last elements in the pre-filler pattern. The generalization of these elements produce several possible specializations for each of the rules in the current beam, but none is likely to improve the rule, so specialization proceeds to the second elements of the post-fillers. Generalizing the state names produces a 'state' semantic tag and a 'nnp' (proper noun) POS tag, creating the final best rule:

Pre-filler:	Filler:	Post-filler:
1) word: in tag: in	1) list: max length: 2 tag: nnp	1) word: , tag: , 2) tag: nnp semantic: state

Experimental Evaluation

RAPIER has been tested on two data sets: a set of 300 computer-related job postings from `austin.jobs` and a set of 485 seminar announcements from CMU. [1] In order to analyze the effect of different types of knowledge sources on the results, three different versions of RAPIER were tested. The full representation used words, POS tags as assigned by Brill's tagger (Brill 1994), and semantic classes taken from WordNet. The other two versions are ablations, one using words and tags (labeled RAPIER-WT in tables), the other words only (labeled RAPIER-W).

We also present results from three other learning IE systems. One is a Naive Bayes system which uses words in a fixed-length window to locate slot fillers (Freitag 1998). Very recently, two other systems have been developed with goals very similar to RAPIER's. These are both relational learning systems which do not depend on syntactic analysis. Their representations and algorithms; however, differ significantly from each other and from RAPIER. SRV (Freitag 1999) employs a top-down, set-covering rule learner similar to FOIL (Quinlan 1990). It uses four pre-determined predicates which allow it to express information about the length of a fragment, the position of a particular token, the relative positions of two tokens, and various user-defined token features (e.g. capitalization, digits, word length). The second system is WHISK (Soderland 1999) which like RAPIER uses pattern-matching, employing a restricted form of regular expressions. It can also make use of semantic classes and the results of syntactic analysis, but does not require them. The learning algorithm is a covering algorithm, and rule creation begins by selection of a single seed example and creates rules top-down, restricting the choice of terms to be added to a rule to those appearing in the seed example (similar to PROGOL).

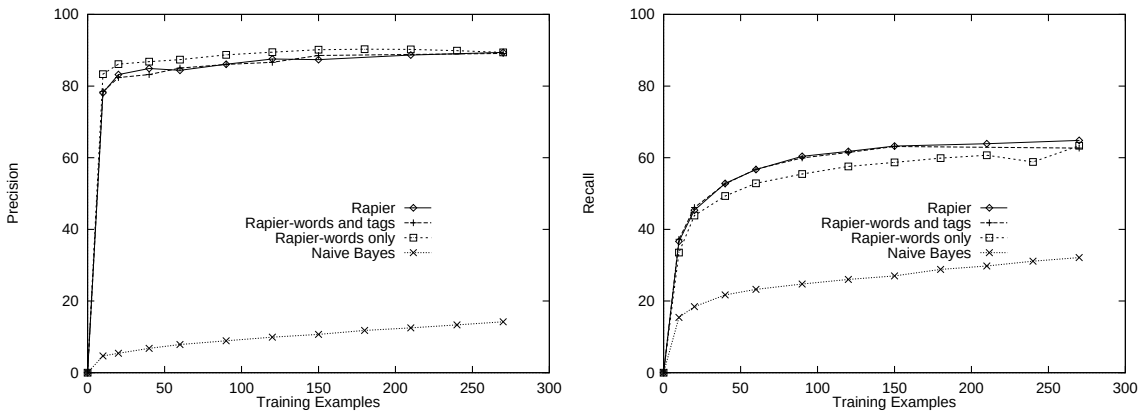
Computer-Related Jobs

The first task is extracting information from computer-related job postings that could be used to create a database of available jobs. The computer job template contains 17 slots, including information about the employer, the location, the salary, and job requirements. Several of the slots, such as the languages and platforms used, can take multiple values. We performed ten-fold cross-validation on 300 examples, and also trained on smaller subsets of the training examples for each test set in order to produce learning curves. We present two measures: precision, the percentage of slot fillers produced which are correct, and recall, the percentage of slot fillers in the correct templates which are produced by the system. Statistical significance was evaluated using a two-tailed paired t-test.

Figure 4 shows the learning curves for precision and recall. Clearly, the Naive Bayes system does not perform well on this task, although it has been shown to be fairly competitive in other domains, as will be seen below. It performs well on some slots but quite poorly on many others, especially those which usually have multiple fillers. In order to compare at reasonably similar levels of recall (although Naive Bayes' recall is still considerably less than Rapier's), Naive Bayes' threshold was set low, accounting for the low precision. Of course, setting the threshold to obtain high precision results in even lower recall. These results clearly indicate the advantage of relational learning since a simpler fixed-context representation such as that used by Naive Bayes appears insufficient to produce a useful system.

By contrast, RAPIER's precision is quite high, over 89% for words only and for words with POS tags. This fact is not surprising, since the bias of the bottom-up algorithm is for specific rules. High precision is important for such tasks, where

[1] The seminar dataset was annotated by Dayne Freitag, who graciously provided the data.



Precision Recall

Figure 4: Performance on job postings

having correct information in the database is generally more important than extracting a greater amount of less-reliable information. Also, the learning curve is quite steep. The RAPIER algorithm is apparently quite effective at making maximal use of a small number of examples. The precision curve flattens out quite a bit as the number of examples increases; however, recall is still rising, though slowly, at 270 examples. The use of *active learning* to intelligently select training examples can improve the rate of learning even further (Califf 1998). Overall, the results are very encouraging.

In looking at the performance of the three versions of RAPIER, an obvious conclusion is that word constraints provide most of the power. Although POS and semantics can provide useful classes that capture important generalities, with sufficient examples, these relevant classes can be implicitly learned from the words alone. The addition of POS tags does improve performance at lower number of examples. The recall of the version with tag constraints is significantly better at least at the 0.05 level for each point on the training curve up to 120 examples. Apparently, by 270 examples, the word constraints are capable of representing the concepts provided by the POS tags, and any differences are not statistically significant. WordNet's semantic classes provided no significant performance increase over words and POS tags only.

One other learning system, WHISK (Soderland 1999), has been applied to this data set. In a 10-fold cross-validation over 100 documents randomly selected from the data set, WHISK achieved a precision of 85% and recall of 55%. This is slightly worse than RAPIER's performance at 90 examples with part-of-speech tags with precision of 86% and recall of 60%.

Seminar Announcements

For the seminar announcements domain, we ran experiments with the three versions of RAPIER, and we report those results along with previous results on this data using the same 10 data splits with the Naive Bayes system and SRV (Freitag 1999). The dataset consists of 485 documents, and this was randomly split approximately in half for each of the 10 runs. Thus training and testing sets were approximately 240 examples each. The results for the other systems are reported by individual slots only. We also report results for WHISK. These results are from a 10-fold cross-validation using only 100 documents randomly selected from the training set. Soderland presents results with and without post-pruning of the rule set. Table 1 shows results for the six systems on the four slots for the seminar announcement task. The line labeled WHISK gives the results for unpruned rules; that labeled WH-PR gives the results for post-pruned rules.

All of the systems perform very well on the start time and end time slots, although RAPIER with semantic classes performs significantly worse on start time than the other systems. These two slots are very predictable, both in contents and in context, so the high performance is not surprising. Start time is always present, while end time is not, and this difference in distribution is the reason for the difference in performance by Naive Bayes on the two slots. The difference also seems to impact SRV's performance, but RAPIER performs comparably on the two, resulting in better performance on the end time slot than the two CMU systems. WHISK also performs very well on the start time task with post-pruning, but also performs less well on the end time task.

Location is a somewhat more difficult field and one for which POS tags seem to help quite a bit. This is not surprising, since locations typically con-

System	stime		etime		loc		speaker	
	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec
RAPIER	93.9	92.9	95.8	94.6	91.0	60.5	80.9	39.4
RAP-WT	96.5	95.3	94.9	94.4	91.0	61.5	79.0	40.0
RAP-W	96.5	95.9	96.8	96.6	90.0	54.8	76.9	29.1
NAIBAY	98.2	98.2	49.5	95.7	57.3	58.8	34.5	25.6
SRV	98.6	98.4	67.3	92.6	74.5	70.1	54.4	58.4
WHISK	86.2	100.0	85.0	87.2	83.6	55.4	52.6	11.1
WH-PR	96.2	100.0	89.5	87.2	93.8	36.1	0.0	0.0

Table 1: Results for seminar announcements task

sist of a sequence of cardinal numbers and proper nouns, and the POS tags can recognize both of those consistently. SRV has higher recall that RAPIER, but substantially lower precision. It is clear that all of the relational systems are better than Naive Bayes on this slot, despite the fact that building names recur often in the data and thus the words are very informative.

The most difficult slot in this extraction task is the speaker. This is a slot on which Naive Bayes, WHISK, and RAPIER with words only perform quite poorly, because speaker names seldom recur through the dataset and all of these systems are using word occurrence information and have no reference to the kind of orthographic features which SRV uses or to POS tags, which can provide the information that the speaker names are proper nouns. RAPIER with POS tags performs quite well on this task, with worse recall than SRV, but better precision.

In general, in this domain semantic classes had very little impact on RAPIER's performance. Semantic constraints are used in the rules, but apparently without any positive or negative effect on the utility of the rules, except on the start time slot, where the use of semantic classes may have discouraged the system from learning the precise contextual rules that are most appropriate for that slot. POS tags help on the location and speaker slots, where the ability to identify proper nouns and numbers is important.

Discussion

The results above show that relational methods can learn useful rules for IE, and that they are more effective than a propositional system such as Naive Bayes. Differences between the various relational systems are probably due to two factors. First, the three systems have quite different learning algorithms, whose biases may be more or less appropriate for particular extraction tasks. Second, the three systems use different representations and features. All use word occurrence and are capable of representing constraints on unbounded ordered sequences. However, RAPIER and SRV are capable of explicitly constraining the lengths of fillers (and, in RAPIER's case, sequences in the pre and post fillers), and WHISK cannot. RAPIER makes use of POS tags, and the others do not (but could presumably be modified to do so). SRV uses orthographic features, and neither of the other systems have access to this information (though in some cases POS tags provide similar information: capitalized words are usually tagged as proper nouns; numbers are tagged as cardinal numbers). Many of the features used in SRV seem quite specific to the seminar announcements domain. Since all of the systems are fairly easily extended to include the lexical features used by the others, it would be useful to examine the effect of various features, seeing how much of the differences in performance depends upon them versus basic representational and algorithmic biases.

Related Work

Some of the work closest to RAPIER was discussed in the previous section. In this section, we briefly mention some other related systems. Previous researchers have generally applied machine learning only to parts of the IE task and have required more human interaction than providing texts with filled templates. CRYSTAL uses a form of clustering to create a dictionary of extraction patterns by generalizing patterns identified in the text by an expert (Soderland *et al.* 1995). AUTOSLOG creates a dictionary of extraction patterns by specializing a set of general syntactic patterns (Riloff 1993), and assumes that an expert will later filter the patterns it produces. PALKA learns extraction patterns relying on a concept hierarchy to guide generalization and specialization (Kim & Moldovan 1995). These systems all rely on prior detailed sentence analysis to identify syntactic elements and their relationships, and their output requires further processing to produce the final filled templates. LIEP also learns IE patterns (Huffman 1996), but also requires a a sentence analyzer to identify noun groups, verbs, subjects, etc. and assumes that all relevant information is between two entities it identifies as "interesting." Finally, ROBOTAG uses decision trees to learn the locations of slot-fillers in a document (Bennett, Aone, & Lovell 1997). The features available to

the decision trees are the result of pre-processing the text and are based on a fixed context. ROBOTAG learns trees to identify possible start and end tokens for slot-fillers and then uses a matching algorithm to pair up start and end tokens to identify actual slot-fillers.

Conclusion

The ability to extract desired pieces of information from natural language texts is an important task with a growing number of potential applications. Tasks requiring locating specific data in newsgroup messages or web pages are particularly promising applications. Manually constructing such IE systems is a laborious task; however, learning methods have the potential to help automate the development process. The RAPIER system described in this paper uses relational learning to construct unbounded pattern-match rules for IE given only a database of texts and filled templates. The learned patterns employ limited syntactic and semantic information to identify potential slot fillers and their surrounding context. Results from two realistic applications demonstrate that fairly accurate rules can be learned from relatively small sets of examples, and that its results are superior to a probabilistic method applied to a fixed-length context.

Acknowledgements

Thanks to Dayne Freitag for supplying his seminar announcements data. This research was supported by a fellowship from AT&T awarded to the first author and by the National Science Foundation under grant IRI-9704943.

References

Bennett, S. W.; Aone, C.; and Lovell, C. 1997. Learning to tag multilingual texts through observation. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, 109–116.

Brill, E. 1994. Some advances in rule-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 722–727.

Califf, M. E. 1998. *Relational Learning Techniques for Natural Language Information Extraction.* Ph.D. Dissertation, Department of Computer Sciences, University of Texas, Austin, TX. Available from http://www.cs.utexas.edu./users/ai-lab.

Cohen, W. W. 1995. Text categorization and relational learning. In *Proceedings of the Twelfth International Conference on Machine Learning*, 124–132. San Francisco, CA: Morgan Kaufman.

DARPA., ed. 1995. *Proceedings of the 6th Message Understanding Conference.* San Mateo, CA: Morgan Kaufman.

Freitag, D. 1998. Multi-strategy learning for information extraction. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 161–169.

Freitag, D. 1999. Machine learning for information extraction in informal domains. *Machine Learning* in press.

Huffman, S. B. 1996. Learning information extraction patterns from examples. In Wermter, S.; Riloff, E.; and Scheler, G., eds., *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing.* Berlin: Springer. 246–260.

Kim, J.-T., and Moldovan, D. I. 1995. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE Transactions on Knowledge and Data Engineering* 7(5):713–724.

Miller, G.; Beckwith, R.; Fellbaum, C.; Gross, D.; and Miller, K. 1993. Introduction to WordNet: An on-line lexical database. Available by ftp to clarity.princeton.edu.

Mooney, R. J., and Califf, M. E. 1995. Induction of first-order decision lists: Results on learning the past tense of English verbs. *Journal of Artificial Intelligence Research* 3:1–24.

Muggleton, S., and Feng, C. 1992. Efficient induction of logic programs. In Muggleton, S., ed., *Inductive Logic Programming.* New York: Academic Press. 281–297.

Muggleton, S. 1995. Inverse entailment and Progol. *New Generation Computing Journal* 13:245–286.

Quinlan, J. 1990. Learning logical definitions from relations. *Machine Learning* 5(3):239–266.

Riloff, E. 1993. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 811–816.

Soderland, S.; Fisher, D.; Aseltine, J.; and Lehnert, W. 1995. Crystal: Inducing a conceptual dictionary. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1314–1319.

Soderland, S. 1999. Learning information extraction rules for semi-structured and free text. *Machine Learning* 34.

Zelle, J. M., and Mooney, R. J. 1994. Combining top-down and bottom-up methods in inductive logic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, 343–351.