Relational Learning of Pattern-Match Rules for Information Extraction

Mary Elaine Califf and Raymond J. Mooney
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
{mecaliff,mooney}@cs.utexas.edu

Abstract

Information extraction systems process natural language documents and locate a specific set of relevant items. Given the recent success of *empirical* or *corpus-based* approaches in other areas of natural language processing, machine learning has the potential to significantly aid the development of these knowledge-intensive systems. This paper presents a system, RAPIER, that takes pairs of documents and filled templates and induces pattern-match rules that directly extract fillers for the slots in the template. The learning algorithm incorporates techniques from several inductive logic programming systems and learns unbounded patterns that include constraints on the words and part-of-speech tags surrounding the filler. Encouraging results are presented on learning to extract information from computer job postings from the newsgroup `misc.jobs.offered`.

1 Introduction

An increasing amount of information is available in the form of electronic documents. The need to intelligently process such texts makes information extraction (IE), the task of locating specific pieces of data from a natural language document, a particularly useful sub-area of natural language processing (NLP). In recognition of their significance, IE systems have been the focus of DARPA's MUC program (Lehnert and Sundheim, 1991). Unfortunately, IE systems are difficult and time-consuming to build and the resulting systems generally contain highly domain-specific components, making them difficult to port to new domains.

Recently, several researchers have begun to apply learning methods to the construction of IE systems (McCarthy and Lehnert, 1995, Soderland et al., 1995, Soderland et al., 1996, Riloff, 1993, Riloff, 1996, Kim and Moldovan, 1995, Huffman, 1996). Several symbolic and statistical methods have been employed, but learning is generally used to construct only part of a larger IE system. Our system, RAPIER (Robust Automated Production of Information Extraction Rules), learns rules for the complete IE task. The resulting rules extract the desired items directly from documents without prior parsing or subsequent processing. Using only a corpus of documents paired with filled templates, RAPIER learns unbounded Eliza-like patterns (Weizenbaum, 1966) that utilize limited syntactic information, such as the output of a part-of-speech tagger. Induced patterns can also easily incorporate semantic class information, such as that provided by WordNet (Miller et al., 1993). The learning algorithm was inspired by several Inductive Logic Programming (ILP) systems and primarily consists of a specific-to-general (*bottom-up*) search for patterns that characterize slot-fillers and their surrounding context.

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 extracting information from messages posted to the newsgroup `misc.jobs.offered`. Section 5 discusses related work in applying learning to IE, Section 6 suggests areas for future research, and Section 7 presents our conclusions.

2 Background

2.1 Information Extraction

In information extraction, the data to be extracted from a natural language text is given by a template specifying a list of slots to be filled. The slot fillers

Mary Elaine Califf and Raymond J. Mooney (1997) **Relational Learning of Pattern-Match Rules for Information Extraction.** In T.M. Ellison (ed.) *CoNLL97: Computational Natural Language Learning,* ACL pp 9-15.

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

may be either one of a set of specified values or strings taken directly from the document. For example, 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 USENET newsgroups, such as job announcements. Our overall goal is to extract a database from all the messages in a newsgroup and then use learned query parsers (Zelle and Mooney, 1996) to answer natural language questions such as "What jobs are available in Austin for C++ programmers with only one year of experience?". Numerous other Internet applications are possible, such as extracting information from product web pages for a shopping agent (Doorenbos, Etzioni, and Weld, 1997).

2.2 Relational Learning

Most empirical natural-language research has employed statistical techniques that base decisions on very limited 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 and other *relational learning* methods (Birnbaum and Collins, 1991) allow induction over *structured* examples that can include first-order logical predicates and functions and unbounded data structures such as lists, strings, and trees. Detailed 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 an English verb (Mooney and Califf, 1995). While RAPIER is not strictly an ILP system, its relational learning algorithm was inspired by ideas from the following ILP systems.

GOLEM (Muggleton and Feng, 1992) is a bottom-up (specific to general) ILP algorithm based on the construction of relative least-general generalizations, *rlggs* (Plotkin, 1970). The idea of least-general generalizations (LGGs) is, given two items (in ILP, two clauses), finding the least general item that covers the original pair. This is usually a fairly simple computation. *Rlggs* are the LGGs *relative* to a set of background relations. Because of the difficulties introduced by non-finite *rlggs*, background predicates must be defined extensionally. The algorithm operates by randomly selecting several pairs of positive examples and computing the determinate *rlggs* of each pair. Determinacy constrains the clause to have for each example no more than one possible valid substitution for each variable in the body of the clause. The resulting clause with the greatest coverage of positive examples is selected, and that clause is further generalized by computing the *rlggs* of the selected clause with new randomly chosen positive examples. The generalization process stops when the coverage of the best clause no longer increases.

The CHILLIN (Zelle and Mooney, 1994) system combines top-down (general to specific) and bottom-up ILP techniques. The algorithm starts with a most specific definition (the set of positive examples) and introduces generalizations which make the definition more compact. Generalizations are created by selecting pairs of clauses in the definition and computing LGGs. If the resulting clause covers negative examples, it is specialized by adding antecedent literals in a top-down fashion. The search for new literals is carried out in a hill-climbing fashion, using an information gain metric for evaluating literals. This is similar to the search employed by FOIL (Quinlan, 1990). In cases where a correct clause cannot be learned with the existing background relations, CHILLIN attempts to construct new predicates which will distinguish the covered negative examples from the covered positives. At each step, a number of possible generalizations are considered; the one producing the greatest compaction of the theory is implemented, and the process repeats. Chillin uses the notion of *empirical subsumption*, which means that as new, more general clauses are added, all of the clauses which are not needed to prove positive examples are removed from the definition.

Progol (Muggleton, 1995) also combines bottom-up and top-down search. Using mode declarations provided for both the background predicates and the predicate being learned, it constructs a most specific clause for a random seed example. The mode declarations specify for each argument of each predicate both the argument's type and whether it should be a constant, a variable bound before the predicate is called, or a variable bound by the predicate. Given this most specific clause, Progol employs a A*-like search through the set of clauses containing up to k literals from that clause in order to find the simplest consistent generalization to add to the definition. Advantages of Progol are that the constraints on the search make it fairly efficient, especially on some types of tasks for which top-down approaches are particularly inefficient, and that its search is guaranteed to find the simplest consistent generalization if such a clause exists with no more than k literals. The primary problems with the system are its need for the mode declarations and the fact that too small a k may prevent Progol from learning correct clauses while too large a k may allow the search to explode.

3 Rapier System

3.1 Rule Representation

Rapier's rule representation uses patterns that make use of limited syntactic and semantic information, using freely available, robust knowledge sources such as a part-of-speech tagger and a lexicon with semantic classes, such as the hypernym links in WordNet (Miller et al., 1993). The initial implementation does not use a parser, primarily because of the difficulty of producing a robust parser for unrestricted text and because simpler patterns of the type we propose can represent useful extraction rules for at least some domains. The extraction rules are indexed by template name and slot name and consist of three parts: 1) a pre-filler pattern that must match the text immediately preceding the filler, 2) a pattern that must match the actual slot filler, and 3) a post-filler pattern that must match the text immediately following the filler. Each pattern is a sequence (possibly of length zero in the case of pre- and post-filler patterns) of *pattern items* or *pattern lists*. A pattern item matches exactly one word or symbol from the document that meets the item's constraints. A pattern list specifies a maximum length N and matches 0 to N words or symbols from the document that each must match the list's constraints. Possible constraints are: a list of words, one of which must match the document item; a list of part-of-speech (POS) tags, one of which must match the document item's POS tag; a list of semantic classes, one of which must be a class that the document item belongs to. Figure 2 shows a rule created by hand that extracts the `area` filler from the example document in figure reftemplate. This rule assumes that the document has been tagged with the POS tagger of (Brill, 1994).

Pre-filler Pattern:	Filler Pattern:	Post-filler Pattern:
1) word: leading	1) list: len: 2 tags: [nn, nns]	1) word: [firm, company]

Figure 2: A Rule Extracting an Area Filler from the Example Document

3.2 The Learning Algorithm

As noted above, Rapier is inspired by ILP methods, and primarily consists of a specific to general (bottom-up) search. First, for each slot, most-specific patterns are created for each example, specifying word and tag 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 on the item consisting of the word and its assigned 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 and generalize the rules for each slot. New rules are created by selecting two existing rules and creating a generalization. The aim is to make small generalization steps, covering more positive examples without generating suprious fillers, so a standard approach would be to generate the least general generalization (LGG) of the pair of rules. However, in this particular representation which allows for unconstrained disjunction, the LGG may be overly specific. Therefore, in cases where the LGG of two constraints is their disjunction, we want to create two generalizations: one would be the disjunction and the other the removal of the constraint. Thus, we often want to consider multiple generalization of a pair of items. This, combined with the fact that patterns are of varying length, making the number of possible generalizations of two long patterns extremely large, makes the computational cost of

```
For each slot, S in the template being learned
   SlotRules = most specific rules from documents for S
   while compression has failed fewer than lim times
      randomly select 2 rules, R1 and R2, from S
      find the set L of generalizations of the fillers of R1
         and R2
      create rules from L, evaluate, and initialize
         RulesList
      let n = 0
      while best rule in RuleList produces spurious
            fillers and the weighted information value
            of the best rule is improving
         increment n
         specialize each rule in RuleList with general-
            izations of the last n items of the
            pre-filler patterns of R1 and R2 and
            add specializations to RuleList
         specialize each rule in RuleList with general-
            izations of the first n item of the
            post-filler patterns of R1 and R2 and
            add specializations of RuleList
      if best rule in RuleList produces only valid fillers
         Add it to SlotRules and remove empirically
            subsumed rules
```

Figure 3: RAPIER Algorithm for Inducing IE Rules

producing all interesting generalizations of two complete rules prohibitive. But, while 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, we start by computing the generalizations of the filler patterns of the two rules and create rules from those generalizations. We maintain a list of the best n rules created and specialize the rules under consideration by adding pieces of the generalizations of the pre- and post-filler patterns of the two seed rules, working outward from the fillers. The rules are ordered using an information value metric (Quinlan, 1990) weighted by the size of the rule (preferring smaller rules). When the best rule under consideration produces no negative examples, specialization ceases; that rule is added to the rule base, and all rules empirically subsumed by it are removed. Specialization will be 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 which fail to produce a compressing rule exceed either a pre-defined limit or the number of rules for that slot. An outline of the algorithm appears in Figure 3 where *RuleList* is a prioritized list of no more than *Beam-Width* rules. The search is somewhat similar to a beam search in that a limited number of rules is kept for consideration, but all rules in *RuleList* are expanded at each iteration, rather than only the best.

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." The rules created from these phrases for the city slot would be

```
Pre-filler Pattern:  Filler Pattern:    Post-filler Pattern:
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 Pattern:  Filler Pattern:    Post-filler Pattern:
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: .
```

The fillers are generalized to produce two possible rules with empty pre-filler 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 LGG'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 produced are:

```
Pre-filler Pattern:  Filler Pattern:    Post-filler Pattern:
                     1) list: len: 2
                        word: [atlanta, kansas, city]
                        tag: nnp
```

and

```
Pre-filler Pattern:  Filler Pattern:    Post-filler Pattern:
                     1) list: len: 2
                        tag: nnp
```

Either of these rules is likely to cover spurious examples, so we add pre-filler and post-filler LGGs. The items produced from the "in"'s and the commas are identical and, therefore, unchanged. Assuming that our lexicon contains a semantic class for states, generalizing the state names produces a semantic constraint of that class along with a tag constraint nnp and either no word constraint or the disjunction of the two states. Thus, a final best rule would be:

```
Pre-filler Pattern:  Filler Pattern:    Post-filler Pattern:
1) word: in          1) list: len: 2    1) word: ,
   tag: in              tag: nnp           tag: ,
                                        2) tag: nnp
                                           semantic: state
```

4 Evaluation

The task we have chosen for initial tests of RAPIER is to extract information from computer-related job

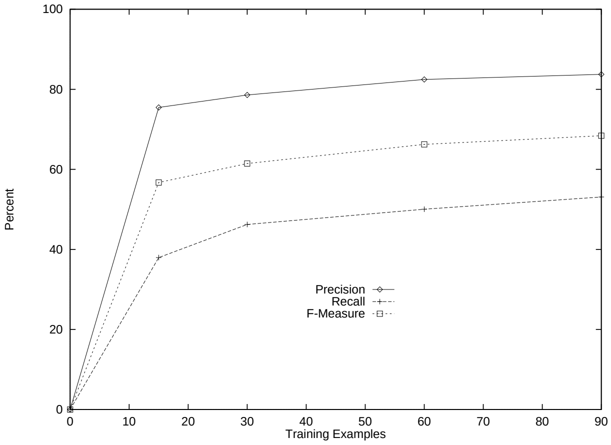

Figure 4: Performance on job postings

postings that could be used to create a database of available jobs. The computer-related job posting 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. The current results do not employ semantic categories, only words and the results of Brill's POS tagger.

The results presented here use a data set of 100 documents paired with filled templates. We did a ten-fold cross-validation, and also ran tests with smaller subsets of the training examples for each test set in order to produce learning curves. We use three measures: precision, the percentage of slot fillers produced which are correct; recall, the percentage of slot fillers in the correct templates which are produced by the system; and an F-measure, which is the average of the recall and the precision.

Figure 4 shows the learning curves generated. At 90 training examples, the average precision was 83.7% and the average recall was 53.1%. These numbers look quite promising when compared to the measured performance of other information extraction systems on various domains. This performance is comparable to that of CRYSTAL on a medical domain task (Soderland et al., 1996), and better than that of AUTOSLOG and AUTOSLOG-TS on part of the MUC4 terrorism task (Riloff, 1996). It also compares favorably with the typical system performance on the MUC tasks (ARPA, 1992, ARPA, 1993). All of these comparisons are only general, since the tasks are different, but they do indicate that RAPIER is doing relatively well. The relatively high precision is an especially positive result, because it is highly likely that recall will continue to improve as the number of training examples increases.

The rules RAPIER learns are of several different types. Some are fairly simple memorizations of words or phrases that consistently appear in particular slots: these include things like programming languages and operating systems. Others learn the context of the filler, usually also constraining the parts of speech of the filler: for example, a rule for the language slot where the prefix is constrained to "familiarity with", the suffix is "programming" and the filler is a list of up to three items which must be proper nouns or symbols.

5 Related Work

Previous researchers have generally applied machine learning only to parts of the IE task and their systems have typically required more human interaction than just providing texts with filled templates. RESOLVE uses decision trees to handle coreference decisions for an IE system and requires annotated coreference examples (McCarthy and Lehnert, 1995). 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, Soderland et al., 1996). AUTOSLOG creates a dictionary of extraction patterns by specializing a set of general syntactic patterns (Riloff, 1993, Riloff, 1996). It assumes that an expert will later examine the patterns it produces. PALKA learns extraction patterns relying on a concept hierarchy to guide generalization and specialization (Kim and Moldovan, 1995). AUTOSLOG, CRYSTAL, and PALKA all rely on prior 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). LIEP's primary limitations are that it also requires a sentence analyzer to identify noun groups, verbs, subjects, etc.; it makes no real use of semantic information; it assumes that all information it needs is between two entities it identifies as "interesting"; and it has been applied to only one domain in which the texts are quite short (1-3 sentences).

6 Future Research

Currently, RAPIER assumes slot values are strings taken directly from the document; however, MUC templates also include slots whose values are taken from a pre-specified set. We plan to extend the system to learn rules for such slots. Also, the current system attempts to extract the same set of slots from every document. RAPIER must be extended to learn patterns that first categorize the text to determine which set of slots, if any, should be extracted from a given document. Finally, the same pattern learning algorithm may prove applicable to other natural language processing tasks such as identifying the sense of an ambiguous word based on its surrounding context.

7 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 information extraction 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 information extraction 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 on extracting information from newsgroup jobs postings have shown that for one realistic application, fairly accurate rules can be learned from relatively small sets of examples. Future research will hopefully demonstrate that similar techiques will prove useful in a wide variety of interesting applications.

8 Acknowledgements

This research was supported by a fellowship from AT&T awarded to the first author and by the National Science Foundation under grant IRI-9310819.

References

ARPA, editor. 1992. *Proceedings of the Fourth DARPA Message Understanding Evaluation and Conference*, San Mateo, CA. Morgan Kaufman.

ARPA, editor. 1993. *Proceedings of the Fifth DARPA Message Understanding Evaluation and Conference*, San Mateo, CA. Morgan Kaufman.

Birnbaum, L. A. and G. C. Collins, editors. 1991. *Proceedings of the Eighth International Workshop on Machine Learning: Part VI Learning Relations*, Evanston, IL, June.

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

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

Doorenbos, R. B., O. Etzioni, and D. S. Weld. 1997. A scalable comparison-shopping agent for the world-wide web. In *Proceedings of the First International Conference on Autonomous Agents*.

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

Kim, Jun-Tae and Dan I. Moldovan. 1995. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE Transactions on Knowledge and DataEngineering*, 7(5):713–724, October.

Lehnert, Wendy and Beth Sundheim. 1991. A performance evaluation of text-analysis technologies. *AI Magazine*, 12(3):81–94.

McCarthy, J. and W. Lehnert. 1995. Using decision trees for coreference resolution. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1050–1055.

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

Mooney, R. J. and M. E. Califf. 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 C. Feng. 1992. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, New York, pages 281–297.

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

Plotkin, G. D. 1970. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence (Vol. 5)*. Elsevier North-Holland, New York.

Quinlan, J.R. 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*, pages 811–816.

Riloff, Ellen. 1996. Automatically generating extraction patterns from untagged text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1044–1049.

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

Soderland, Stephen, David Fisher, Jonathan Aseltine, and Wendy Lehnert. 1996. Issues in inductive learning of domain-specific text extraction rules. In Stefan Wermter, Ellen Riloff, and Gabriele Scheller, editors, *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, Lecture Notes in Artificial Intelligence. Springer, pages 290–301.

Weizenbaum, J. 1966. ELIZA – A computer program for the study of natural language communications between men and machines. *Communications of the Association for Computing Machinery*, 9:36–45.

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

Zelle, J. M. and R. J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, August.