Journal of Machine Learning Research 4 (2003) 177-210 Submitted 12/01; Revised 2/03; Published 6/03

Bottom-Up Relational Learning of Pattern Matching Rules for Information Extraction

Mary Elaine Califf MECALIF@ILSTU.EDU
Department of Applied Computer Science
Illinois State University
Normal, IL 61790, USA

Raymond J. Mooney MOONEY@CS.UTEXAS.EDU
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712, USA

Editor: David Cohn

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 an algorithm, 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 is a bottom-up learning algorithm that incorporates techniques from several inductive logic programming systems. We have implemented the algorithm in a system that allows patterns to have 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.

Keywords: Natural Language Processing, Information Extraction, Relational Learning

1. Introduction

In the wake of the recent explosive growth in on-line text on the web and in other places, has come a need for systems to help people cope with the information explosion. A number of researchers in language processing have begun to develop *information extraction* systems: systems that pull specific data items out of text documents.

Information extraction systems seem to be a promising way to deal with certain types of text documents. However, a difficulty with information extraction systems is that they are difficult and time-consuming to build, and they generally contain highly domain-specific components, making porting to new domains also time-consuming. Thus, more efficient means for developing information extraction systems are desirable.

This situation has made information extraction systems an attractive application for machine learning. Several researchers have begun to use learning methods to aid in the construction of information extraction systems (Soderland et al., 1995, Riloff, 1993, Kim and Moldovan, 1995, Huffman, 1996). However, in these systems, learning is used for part of a larger information extraction system. Our system RAPIER (Robust Automated Production of Information Extraction Rules) learns rules for the complete information extraction task, rules producing the desired information pieces directly from the documents without prior parsing or any post-processing. We do this by using a structured (relational) symbolic representation, rather than learning classifiers or rules developed from general rule templates.

Using only a corpus of documents paired with filled templates, RAPIER learns Eliza-like patterns (Weizenbaum, 1966). In the current implementation, the patterns make use of limited syntactic and semantic information, using freely available, robust knowledge sources such as a part-of-speech tagger or a lexicon. The rules built from these patterns can consider an unbounded context, giving them an advantage over more limited representations which consider only a fixed number of words. This relatively rich representation requires a learning algorithm capable of dealing with its complexities. Therefore, RAPIER employs a relational learning algorithm which uses techniques from several Inductive Logic Programming (ILP) systems (Lavrač and Džeroski, 1994). These techniques are appropriate because they were developed to work on a rich, relational representation (first-order logic clauses). Our algorithm incorporates ideas from several ILP systems, and consists primarily of a specific to general (bottom-up) search. We show that learning can be used to build useful information extraction rules, and that relational learning is more effective than learning using only simple features and a fixed context. Simultaneous with RAPIER's development other learning systems have recently been developed for this task which also use relational learning (Freitag, 2000, Soderland, 1999). Other recent approaches to this problem include using hidden Markov models (HMMs) (Freitag and McCallum, 2000) and combining boosting with the learning of simpler "wrappers" (Freitag and Kushmerick, 2000).

Experiments using RAPIER were performed in two different domains. One of the domains was a set of computer-related job postings from Usenet newsgroups. The utility of this domain is evident in the success of FlipDog, a job posting website (www.flipdog.com) developed by WhizBang! (www.whizbanglabs.com) using information extraction techniques. It should be noted that our template is both much more detailed than the one used for FlipDog and that it is specific to computer-related jobs. The second domain was a set of seminar announcement compiled at Carnegie Mellon University. The results were compared to the two other relational learners and to a Naive Bayes-based system. The results are encouraging.

The remainder of the article is organized as follows. Section 2 presents background material on information extraction and relational learning. Section 3 describes RAPIER's rule representation and learning algorithm. Section 4 presents and discusses experimental results, including comparisons to a simple Bayesian learner and two other relational learners. Section 5 suggests some directions for future work. Section 6 describes some related work in applying learning to information extraction, and Section 7 presents our conclusions.

2. Background

This section provides background on the task of information extraction and on the relational learning algorithms that are the most immediate predecessors of our learning algorithm.

2.1 Information Extraction

Information extraction is a shallow form of natural language understanding useful for certain types of document processing, which has been the focus of ARPA's Message Understanding Conferences (MUC) (Lehnert and Sundheim, 1991, DARPA, 1992, 1993). It is useful in situations where a set of text documents exist containing information which could be more easily used by a human or

Posting from Newsgroup

```
Subject: US-TN-SOFTWARE PROGRAMMER
Date: 17 Nov 1996 17:37:29 GMT
Organization: Reference.Com Posting Service
Message-ID: <56nigp$mrs@bilbo.reference.com>

SOFTWARE PROGRAMMER

Position available for Software Programmer experienced
in generating software for PC-Based Voice Mail systems.
Experienced in C Programming.  Must be familiar with
communicating with and controlling voice cards; preferable
Dialogic, however, experience with others such as Rhetorix
and Natural Microsystems is okay. Prefer 5 years or more
experience with PC Based Voice Mail, but will consider as
little as 2 years.  Need to find a Senior level person who
can come on board and pick up code with very little training.
Present Operating System is DOS.  May go to OS-2 or UNIX
in future.

Please reply to:
Kim Anderson
AdNET
(901) 458-2888 fax
kimander@memphisonline.com
```

Figure 1: A sample job posting from a newsgroup.

computer if the information were available in a uniform database format. Thus, an information extraction system is given the set of documents and a template of slots to be filled with information from the document. Information extraction systems locate and in some way identify the specific pieces of data needed from each document.

Two different types of data may be extracted from a document: more commonly, the system is to identify a string taken directly from the document, but in some cases the system selects one from a set of values which are possible fillers for a slot. The latter type of slot-filler may be items like dates, which are most useful in a consistent format, or they may simply be a set of terms to provide consistent values for information which is present in the document, but not necessarily in a consistently useful way. In this work, we limit ourselves to dealing with strings taken directly from the document in question.

Information extraction can be useful in a variety of domains. The various MUC's have focused on tasks such as the Latin American terrorism domain mentioned above, joint ventures, microelectronics, and company management changes. Others have used information extraction to track medical patient records (Soderland et al., 1995), to track company mergers (Huffman, 1996), and to extract biological information (Craven and Kumlien, 1999, Ray and Craven, 2001). More recently, researchers have applied information extraction to less formal text genres such as rental ads (Soderland, 1999) and web pages (Freitag, 1998a, Hsu and Dung, 1998, Muslea et al., 1998).

Filled Template

```
computer_science_job
id: 56nigp$mrs@bilbo.reference.com
title: SOFTWARE PROGRAMMER
salary:
company:
recruiter:
state: TN
city:
country: US
language: C
platform: PC \ DOS \ OS-2 \ UNIX
application:
area: Voice Mail
req_years_experience: 2
desired_years_experience: 5
req_degree:
desired_degree:
post_date:  17 Nov 1996
```

Figure 2: The filled template corresponding to the message shown in Figure 1. All of the slot-fillers are strings taken directly from the document. Not all of the slots are filled, and some have more than one filler.

Another domain which seems appropriate, particularly in the light of dealing with the wealth of online information, is to extract information from text documents in order to create easily searchable databases from the information, thus making the wealth of text online more easily accessible. For instance, information extracted from job postings in USENET newsgroups such as `misc.jobs.offered` can be used to create an easily searchable database of jobs. An example of the information extraction task for such a system limited to computer-related jobs appears in Figures 1 and 2.

2.2 Relational Learning

Since much empirical work in natural language processing has employed statistical techniques (Manning and Schütze, 1999, Charniak, 1993, Miller et al., 1996, Smadja et al., 1996, Wermter et al., 1996), this section discusses the potential advantages of symbolic relational learning. In order to accurately estimate probabilities from limited data, most statistical techniques base their decisions on a very limited context, such as bigrams or trigrams (2 or 3 word contexts). However, NLP decisions must frequently be based on much larger contexts that include a variety of syntactic, semantic, and pragmatic cues. Consequently, researchers have begun to employ learning techniques that can handle larger contexts, such as *decision trees* (Magerman, 1995, Miller et al., 1996, Aone and Bennett, 1995), *exemplar* (*case-based*) methods (Cardie, 1993, Ng and Lee, 1996), and a maximum entropy modeling method (Ratnaparkhi, 1997). However, these techniques still require the system developer to specify a manageable, finite set of features for use in making decisions. Developing this set of features can require significant representation engineering and may still exclude important contextual information.

In contrast, *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 and trees. We consider a learning algorithm to be relational if it uses a representation for examples that goes beyond finite feature vectors and can handle an unbounded number of entities in its representation of an example and uses some notion of relations between entities in an example even if that is just a single "precedes" relation between token entities that allows representing unbounded sequences or strings. In particular, *inductive logic programming* (ILP) (Lavrač and Džeroski, 1994, Muggleton, 1992) studies the induction of rules in first-order logic (Prolog programs). ILP systems have induced a variety of basic Prolog programs (e.g. `append, reverse, sort`) as well as potentially useful rule bases for important biological problems (Muggleton et al., 1992, Srinivasan et al., 1996). Detailed experimental comparisons of ILP and feature-based induction have demonstrated the advantages of relational representations in two language related tasks, text categorization (Cohen, 1995a) and generating the past tense of an English verb (Mooney and Califf, 1995). Recent research has also demonstrated the usefulness of relational learning in classifying web pages (Slattery and Craven, 1998).

Although a few information extraction learning algorithms prior to RAPIER's development used structured representations of some kind, such as Autoslog (Riloff, 1993) and Crystal (Soderland et al., 1995), they artificially limited the possible rules to be learned, Autoslog by learning only rules that fit particular provided templates, Crystal less drastically by limiting examples to the sentence in which extracted phrase(s) occurred. In contrast, RAPIER's patterns are not limited in these ways.

While RAPIER is not an ILP algorithm, it is a relational learning algorithm learning a structured rule representation, and its algorithm was inspired by ideas from ILP systems. The ILP-based ideas are appropriate because they were designed to learn using rich, unbounded representations. Therefore, the following sections discuss some general design issues in developing ILP and other rule learning systems and then briefly describe relevant aspects of the three ILP systems that most directly influenced RAPIER's learning algorithm: GOLEM, CHILLIN, and PROGOL.

2.2.1 GENERAL ALGORITHM DESIGN ISSUES

One of the design issues in rule learning systems is the overall structure of the algorithm. There are two primary forms for this outer loop: compression and covering. Systems that use compression begin by creating an initial set of highly specific rules, typically one for each example. At each iteration a more general rule is constructed, which replaces the rules it subsumes, thus compressing the rule set. At each iteration, all positive examples are under consideration to some extent, and the metric for evaluating new rules is biased toward greater compression of the rule set. Rule learning ends when no new rules to compress the rule set are found. Systems that use compression include DUCE, a propositional rule learning system using inverse resolution (Muggleton, 1987), CIGOL, an ILP system using inverse resolution (Muggleton and Buntine, 1988), and CHILLIN (Zelle and Mooney, 1994).

Systems that use covering begin with a set of positive examples. Then, as each rule is learned, all positive examples the new rule covers are removed from consideration for the creation of future rules. Rule learning ends when all positive examples have been covered. This is probably the more common way to structure a rule learning system. Examples include FOIL (Quinlan, 1990), GOLEM

(Muggleton and Feng, 1992), PROGOL (Muggleton, 1995), CLAUDIEN (De Raedt and Bruynooghe, 1993), and various systems based on FOIL such as FOCL (Pazzani et al., 1992), MFOIL (Lavrač and Džeroski, 1994), and FOIDL(Mooney and Califf, 1995).

There are trade-offs between these two designs. The primary difference is the trade-off between a more efficient search or a more thorough search. The covering systems tend to be somewhat more efficient, since they do not seek to learn rules for examples that have already been covered. However, their search is less thorough than that of compression systems, since they may not prefer rules which both cover remaining examples and subsume existing rules. Thus, the covering systems may end up with a set of fairly specific rules in cases where a more thorough search might have discovered a more general rule covering the same set of examples.

A second major design decision is the direction of search used to construct individual rules. Systems typically work in one of two directions: bottom-up (specific to general) systems create very specific rules and then generalize those to cover additional positive examples, and top-down (general to specific) systems start with very general rules– typically rules which cover all of the examples, positive and negative, and then specialize those rules, attempting to uncover the negative examples while continuing to cover many of the positive examples. Of the systems above, DUCE, CIGOL, and GOLEM and pure bottom-up systems, while FOIL and the systems based on it are pure top-down systems. CHILLIN and PROGOL both combine bottom-up and top-down methods.

Clearly, the choice of search direction also creates tradeoffs. Top-down systems are often better at finding general rules covering large numbers of examples, since they start with a most general rule and specialize it only enough to avoid the negative examples. Bottom-up systems may create overly-specialized rules that don't perform well on unseen data because they may fail to generalize the initial rules sufficiently. Given a fairly small search space of background relations and constants, top-down search may also be more efficient. However, when the branching factor for a top-down search is very high (as it is when there are many ways to specialize a rule), bottom-up search will usually be more efficient, since it constrains the constants to be considered in the construction of a rule to those in the example(s) that the rule is based on. The systems that combine bottom-up and top-down techniques seek to take advantage of the efficiencies of each.

2.2.2 ILP ALGORITHMS

As mentioned above, GOLEM (Muggleton and Feng, 1992) uses a greedy covering algorithm. The construction of individual clauses is bottom-up, based on the construction of *least-general generalizations* (LGGs) of more specific clauses (Plotkin, 1970). In order to take background knowledge into account GOLEM actually creates *Relative* LGGs (RLGGs) of positive examples with respect to the background knowledge. GOLEM randomly selects pairs of examples, computes RLGGs of the pairs, and selects the best resulting clause. Additional examples are randomly selected to generalize that clause further until no improvement is made in the clause's coverage.

The second ILP algorithm that helped to inspire RAPIER is CHILLIN (Zelle and Mooney, 1994), an example of an ILP algorithm that uses compression for its outer loop. CHILLIN combines elements of both top-down and bottom-up induction techniques including a mechanism for demand-driven predicate invention. CHILLIN starts with a most specific definition (the set of positive examples) and introduces generalizations which make the definition more compact (as measured by a CIGOL-like size metric (Muggleton and Buntine, 1988)). The search for more general definitions is carried out in a hill-climbing fashion. 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. To determine which clauses in the current theory a new clause should replace, CHILLIN uses a notion of *empirical subsumption*. If a clause *A* covers all of the examples covered by clause *B* along with one or more additional examples, then *A* empirically subsumes *B*.

The individual clause creation algorithm attempts to construct a clause that empirically subsumes some clauses of the current definition without covering any of the negative examples. The first step is to construct the LGG of the input clauses. If the LGG does not cover any negative examples, no further refinement is necessary. If the clause is too general, an attempt is made to refine it using a FOIL-like mechanism which adds literals derivable either from background or previously invented predicates. If the resulting clause is still too general, it is passed to a routine which invents a new predicate to discriminate the positive examples from the negatives that are still covered.

PROGOL (Muggleton, 1995) also combines bottom-up and top-down search. Like FOIL and GOLEM, PROGOL uses a covering algorithm for its outer loop. As in the propositional rule learner AQ (Michalski, 1983), individual clause construction begins by selecting a random seed example. Using mode declarations provided for both the background predicates and the predicate being learned, PROGOL constructs a most specific clause for that random seed example, called the *bottom* clause. 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 the bottom clause, PROGOL employs an A*-like search through the set of clauses containing up to *k* literals from the bottom clause in order to find the simplest consistent generalization to add to the definition.

3. The RAPIER System

In the following discussion, it is important to distinguish between the RAPIER algorithm and the RAPIER system. The basic concept of the rule representation and the general algorithm are applicable to a variety of specific choices in terms of background knowledge or features. We also describe our specific implementation choices. As will be seen below in Section 4, some of these choices were more effective than others.

A note on terminology: for convenience and brevity, we use the term *slot-filler* to refer to the string to be extracted. We then use the term *pre-filler pattern* to refer to a pattern that matches the left context of a string to be extracted and the term *post-filler pattern* to refer to the right context of a string to be extracted.

3.1 Rule Representation

RAPIER's rule representation uses Eliza-like patterns (Weizenbaum, 1966) that can make use of limited syntactic and semantic information. Each extraction rule is made up of three patterns: 1) a pre-filler pattern that must match the text immediately preceding the slot-filler, 2) a slot-filler pattern that must match the actual slot-filler, and 3) a post-filler pattern that must match the text immediately following the filler. The extraction rules also contain information about what template and slot they apply to.

The purpose of the patterns is, of course, to limit the strings each rule matches to only strings that are correct extractions for a given slot of a particular template. Therefore, each element of a pattern has a set of constraints on the text the element can match. Our implementation of RAPIER allows for three kinds of constraints on pattern elements: constraints on the words the element can match,

```
Pre-filler Pattern:         Filler Pattern:          Post-filler Pattern:
1) syntactic:  {nn,nnp}     1) word:  undisclosed    1) semantic:  price
2) list:  length 2             syntactic:  jj
```

Figure 3: A rule for extracting the transaction amount from a newswire concerning a corporate acquisition. "nn" and "nnp" are the part of speech tags for noun and proper noun, respectively; "jj" is the part of speech tag for an adjective.

on the part-of-speech tags assigned to the words the element can match, and on the semantic class of the words the element can match. The constraints are disjunctive lists of one or more words, tags, or semantic classes and document items must match one of those words, tags, or classes to fulfill the constraint.

A note on part-of-speech tags and semantic classes: in theory, these could be from any source. RAPIER's operation does not depend on any particular tagset or tagging method. In practice, we used Eric Brill's tagger as trained on the Wall Street Journal corpus (Brill, 1994). Although the rule representation does not require a particular type of semantic class, we used WordNet synsets as the semantic classes (Miller et al., 1993), and RAPIER's handling of semantic classes is heavily tied to that representation.

Each pattern is a sequence (possibly of length zero in the case of pre- and post-filler patterns) of pattern elements. RAPIER makes use of two types of pattern elements: *pattern items* and *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 (a limited form of Kleene closure), each of which must match the list's constraints.

Figure 3 shows an example of a rule that shows the various types of pattern elements and constraints. This is a rule constructed by RAPIER for extracting the transaction amount from a newswire concerning a corporate acquisition. Rules are represented in three columns: each column representing one of the three patterns that make up the rule. Within each column, the individual pattern elements are numbered. Note that pattern elements with multiple constraints will take up multiple lines. The rule in Figure 3 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. The first is an item with a part-of-speech constraining the matching word to be tagged as a noun or a proper noun. The second is a list of maximum length two with no constraints. The filler pattern is a single item constrained to be the word "undisclosed" with a POS tag labeling it an adjective. The post-filler pattern is also a single pattern item with a semantic constraint of "price".'

In using these patterns to extract information, we apply all of the rules for a given slot to a document and take all of the extracted strings to be slot-fillers, eliminating duplicates. Rules may also apply more than once. In many cases, multiple slot fillers are possible, and the system seldom proposes multiple fillers for slots where only one filler should occur.

3.2 The Learning Algorithm

The following describes RAPIER's learning algorithm, along with some specific implementation issues and decisions.

3.2.1 ALGORITHM DESIGN CHOICES

RAPIER, as noted above, is inspired by ILP methods, particularly by GOLEM, CHILLIN, and PROGOL. It is compression-based and primarily consists of a specific to general (bottom-up) search. The choice of a bottom-up approach was made for two reasons. The first reason is the very large branching factor of the search space, particularly in finding word and semantic constraints. Learning systems that operate on natural language typically must have some mechanism for handling the search imposed by the large vocabulary of any significant amount of text (or speech). Many systems handle this problem by imposing limits on the vocabulary considered–using only the *n* most frequent words, or considering only words that appear at least *k* times in the training corpus (Yang and Pederson, 1997). While this type of limitation may be effective, using a bottom-up approach reduces the consideration of constants in the creation of any rule to those appearing in the example(s) from which the rule is being generalized, thus limiting the search without imposing artificial hard limits on the constants to be considered.

The second reason for selecting a bottom-up approach is that we decided to prefer overly specific rules to overly general ones. In information extraction, as well as other natural language processing task, there is typically a trade-off between high precision (avoiding false positives) and high recall (identifying most of the true positives). For the task of building a database of jobs which partially motivated this work, we wished to emphasize precision. After all, the information in such a database could be found by performing a keyword search on the original documents, giving maximal recall (given that we extract only strings taken directly from the document), but relatively low precision. A bottom-up approach will tend to produce specific rules, which also tend to be precise rules.

Given the choice of a bottom-up approach, the compression outer loop is a good fit. A bottom-up approach has a strong tendency toward producing specific, precise rules. Using compression for the outer loop may partially counteract this tendency with its tendency toward a more thorough search for general rules. So, like CHILLIN (Zelle and Mooney, 1994), RAPIER begins with a most specific definition and then attempts to compact that definition by replacing rules with more general rules. Since in RAPIER's rule representation rules for the different slots are independent of one another, the system actually creates the most specific definition and then compacts it separately for each slot in the template.

3.2.2 ALGORITHM OVERVIEW

The basic outer loop of the algorithm appears in Figure 4. It is a fairly standard compression-based algorithm. Note that the learning is done separately for each slot. The algorithms for constructing the initial most-specific rules and for rule generalization are discussed in some detail in the following sections. *CompressLim* is a parameter to the algorithm that determines the maximum number of times the algorithm can fail to compress the rulebase. We allow for multiple attempts to find an acceptable rule because of the randomness built in to the rule generalization algorithm. Once the maximum number of compression failures is exceeded, the algorithm ends. By default, *CompressLim* is 3.

```
For each slot, S in the template being learned
    SlotRules = most specific rules for S from example documents
    Failures = 0
    while Failures < CompressLim
       BestNewRule = FindNewRule(SlotRules, Examples)
       if BestNewRule is acceptable
          add BestNewRule to SlotRules
          remove empirically subsumed rules from SlotRules
       else
          Add 1 to Failures
```

Figure 4: RAPIER Algorithm

The definition of an acceptable rule is considered below in the discussion evaluating rules. In short, an acceptable rule is one that covers positive examples and may cover a relatively small number of negative examples. Once an acceptable rule is found, RAPIER uses the notion of empirical subsumption to determine which rules should be considered covered by the new rule and therefore removed from the rulebase.

3.2.3 INITIAL RULEBASE CONSTRUCTION

The first step of a compression-based learning algorithm is to construct an initial rulebase. In ILP algorithms, this is done simply by taking the set of examples as the set of facts that make up the initial definition of the predicate to be learned. For RAPIER, we must construct most-specific rules from the example documents and filled templates. RAPIER begins this process by locating every instance of a string to be extracted in the corresponding document. It then constructs a rule for each instance, using the string to be extracted for the filler pattern, everything in the document preceding the extracted string for the pre-filler pattern, and everything in the document following the extracted string for the post-filler pattern. Since the rules are to be maximally specific, the constructed patterns contain one pattern item for each token of the document. Pattern lists will arise only from generalization.

The constraints on each item are the most specific constraints that correspond to the token that the item is created from. For the word and part-of-speech tag constraints used by our implementation, this is quite straightforward: the word constraint is simply the token from the document, and the POS tag constraint is simply the POS tag for the token.

Using semantic class information creates some issues is the construction of the initial rulebase. The semantic class is left unconstrained because a single word often has multiple possible semantic classes because of the homonymy and polysemy of language. If semantic constraints were immediately created, RAPIER would have to either use a disjunction of all of the possible classes at the lowest level of generality (in the case of WordNet– the synsets that the word for which the item is created belongs to) or choose a semantic class. The first choice is somewhat problematic because the resulting constraint is quite likely to be too general to be of much use. The second choice is the best, if and only if the correct semantic class for the word in context is known, a difficult problem in and of itself. Selecting the most frequent choice from WordNet might work for some cases, but certainly not in all cases, and there is the issue of domain specificity. The most frequent meaning of

```
initialize RuleList to be 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
       NewRuleList = SpecializePreFiller (CurRule, n)
       evaluate each rule in NewRuleList and add it to RuleList
    for each rule, CurRule, in RuleList
       NewRuleList = SpecializePostFiller (CurRule, n)
       evaluate each rule in NewRuleList and add it 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 LimNoImprovements iterations
```

Figure 5: RAPIER Algorithm for Inducing Information Extraction Rules

a word in all contexts may not be the most frequent meaning of that word in the particular domain in question. And, of course, even within a single domain words will have multiple meanings so even determining the most frequent meaning of a word in a specific domain may often be a wrong choice. Our implementation of RAPIER avoids the issue altogether by waiting to create semantic constraints until generalization. Thus, it implicitly allows the disjunction of classes, selecting a specific class only when the item is generalized against one containing a different word. By postponing the choice of a semantic class until there are multiple items required to fit the semantic constraint, we narrow the number of possible choices for the semantic class to classes that cover two or more words. Details concerning the creation of semantic constraints are discussed below.

3.2.4 RULE GENERALIZATION

RAPIER's method for learning new rules was inspired by aspects of all three ILP algorithms discussed above. The basic idea is to take random pairs of rules and generalize each pair, then select the best generalization as the new rule. Due to issues of computational complexity, RAPIER actually employs a combination top-down and bottom-up approach to produce the generalizations of each pair of rules. Figure 5 outlines the algorithm for learning a new rule.

The parameters to this algorithm are the length of the priority queue, k, which defaults to 6; the number of pairs of rules to be generalized, m, which defaults to 5; and the number of specialization iterations allowed with no improvements to the value of the best rule, *LimNo Improvements*, which defaults to 3.

This algorithm is a bit more complex than the straightforward method of generalizing two rules, which would be to simply find the least general generalization (LGG) of corresponding patterns in the two rules. Thus, we would find the LGG of the two pre-filler patterns and use that as the pre-filler pattern of the new rule, make the filler pattern of the new rule be the LGG of the two filler patterns, and then do the same for the post-filler pattern. However, there are two problems with the straightforward approach that led us to the hybrid search that RAPIER employs.

The first problem is the expense of computing the LGGs of the pre-filler and post-filler patterns. These patterns may be very long, and the pre-filler or post-filler patterns of two rules may be of different lengths. Generalizing patterns of different lengths is computationally expensive because each individual pattern element in the shorter pattern may be generalized against one or more elements of the longer pattern, and it is not known ahead of time how elements should be combined to produce the LGG. Thus, the computation of the LGG of the pre-filler and post-filler patterns in their entirety may be prohibitively computationally expensive. The issues involved in generalizing patterns of different lengths are discussed in further detail below. Besides the expense of computing generalizations of some length, we have to deal with the fact that our generalization algorithm will return multiple generalizations of patterns of different lengths.

The second problem has to do with the issue of having multiple generalizations for a given pair of pattern elements. In our implementation, we allow constraints on words and tags to have unlimited disjunctions of words and tags, so the LGG of two word or tag constraints is always their union. However, while this disjunct may be the desirable generalization, the generalization that simply removes the constraint is the more desirable generalization for some cases. Thus, it is useful to consider multiple generalizations of pattern elements with these constraints, and this is the approach that our implementation of RAPIER uses. This choice clearly aggravates the problems with generalizing lengthy patterns, causing the process to produce many possible generalizations.

RAPIER's rule generalization method operates on the principle that the relevant information for extracting a slot-filler is likely to be close to that filler in the document. Therefore, RAPIER begins by generalizing the two filler patterns and creates rules with the resulting generalized filler patterns and empty pre-filler and post-filler patterns. It then specializes those rules by adding pattern elements to the pre-filler and post-filler patterns, working outward from the filler. The elements to be added to the patterns are created by generalizing the appropriate portions of the pre-fillers or post-fillers of the pair of rules from which the new rule is generalized. Working in this way takes advantage of the locality of language, but does not preclude the possibility of using pattern elements that are fairly distant from the filler.

RuleList is a priority queue of length *k* which maintains the list of rules still under consideration, where *k* is a parameter of the algorithm. The priority of the rule is its value according to RAPIER's heuristic metric for determining the quality of a rule (see Section 3.2.5). RAPIER's search is basically a beam-search: a breadth-first search keeping only the best *k* items at each pass. However, the search does differ somewhat from a standard beam-search in that the nodes (or rules) are not fully expanded at each pass (since at each iteration the specialization algorithms only consider pattern elements out to a distance of *n* from the filler), and because of this the old rules are only thrown out when they fall off the end of the priority queue.

The following sections discuss some of the specifics of the rule-generalization algorithm: looking at how rules are evaluated, some of the issues involved in generalizing the constraints, and the issues involved in generalizing patterns.

3.2.5 RULE EVALUATION

One difficulty in designing the RAPIER algorithm was in determining an appropriate heuristic metric for evaluating the rules being learned. The first issue is the measurement of negative examples. Clearly, in a task like information extraction there are a very large number of possible negative examples – strings which should not be extracted – a number large enough to make explicit enumeration of the negative examples difficult, at best. Another issue is the question of precisely which substrings constitute appropriate negative examples: should all of the strings of any length be considered negative examples, or only those strings with lengths similar to the positive examples for a given slot. To avoid these problems, RAPIER does not enumerate the negative examples, but uses a notion of implicit negatives instead (Zelle et al., 1995). First, RAPIER makes the assumption that all of the strings which should be extracted for each slot are specified, so that any strings which a rule extracts that are not specified in the template are assumed to be spurious extractions and, therefore, negative examples. Whenever a rule is evaluated, it is applied to each document in the training set. Any fillers that match the fillers for the slot in the training templates are considered positive examples; all other extracted fillers are considered negative examples covered by the rule.

Given a method for determining the negative as well as the positive examples covered by the rule, a rule evaluation metric can be devised. Because RAPIER does not use a simple search technique such as hill-climbing, it cannot use a metric like information gain (Quinlan, 1990) which measures how much each proposed new rule improves upon the current rule in order to pick the new rule with the greatest improvement. Rather, each rule needs an inherent value which can be compared with all other rules. One such value is the informativity of each rule (the metric upon which information gain is based (Quinlan, 1986)).

$$I(T) = -\log_2(T_+/|T|). \tag{1}$$

However, while informativity measures the degree to which a rule separates positive and negative examples (in this case, identifies valid fillers but not spurious fillers), it makes no distinction between simple and complex rules. The problem with this is that, given two rules which cover the same number of positives and no negatives but different levels of complexity (one with two constraints and one with twenty constraints), we would expect the simpler rule to generalize better to new examples, so we would want that rule to be preferred. Many machine learning algorithms encode such a preference; all top-down hill-climbing algorithms which stop when a rule covering no negatives is found have this preference for simple rather than complex rules. Since RAPIER's search does not encode such a preference, but can, because of its consideration of multiple ways of generalizing constraints, produce many rules of widely varying complexities at any step of generalization or specialization, the evaluation metric for the rules needs to encode a bias against complex rules. Finally, we want the evaluation metric to be biased in favor of rules which cover larger number of positive examples.

The metric RAPIER uses takes the informativity of the rule and weights that by the size of the rule divided by the number of positive examples covered by the rule. The informativity is computed using the Laplace estimate of the probabilities. The size of the rule is computed by a simple heuristic as follows: each pattern item counts 2; each pattern list counts 3; each disjunct in a word constraint counts 2; and each disjunct in a POS tag constraint or semantic constraint counts 1. This size is then divided by 100 to bring the heuristic size estimate into a range which allows the informativity and the rule size to influence each other, with neither value being overwhelmed by the other. Then the evaluation metric is computed as:

$$ruleVal = -\log_2\left(\frac{p+1}{p+n+2}\right) + \frac{ruleSize}{p}$$

where p is the number of correct fillers extracted by the rule and n is the number of spurious fillers the rule extracts.

RAPIER does allow coverage of some spurious fillers. The primary reason for this is that human annotators make errors, especially errors of omission. If RAPIER rejects a rule covering a large number of positives because it extracts a few negative examples, it can be prevented from learning useful patterns by the failure of a human annotator to notice even a single filler that fits that pattern which should, in fact, be extracted. If RAPIER's specialization ends due to failure to improve on the best rule for too many iterations and the best rule still extracts spurious examples, the best rule is used if it meets the criteria:

$$\frac{p-n}{p+n} > noiseParam$$

where p is the number of valid fillers extracted by the rule and n is the number of spurious fillers extracted. This equation is taken from RIPPER (Cohen, 1995b), which uses it for pruning rules measuring p and n using a hold-out set. Because RAPIER is usually learning from a relatively small number of examples, it does not use a hold-out set or internal cross-validation in its evaluation of rules which cover spurious fillers, but uses a much higher default value of *noiseParam* (Cohen uses a default of 0.5; RAPIER's default value is 0.9).

Note that RAPIER's noise handling does not involve pruning, as noise handling often does. Pruning is appropriate for top-down approaches, because in noise handling the goal is to avoid creating rules that are too specialized and over-fit the data and, in pure top-down systems, the only way to generalize a too-specific rule is some sort of pruning. Since RAPIER is a primarily bottom-up compression-based system, it can depend on subsequent iterations of the compression algorithm to further generalize any rules that may be too specific. The noise handling mechanism need only allow the acceptance of noisy rules when the "best" rule, according to the rule evaluation metric, covers negative examples.

3.2.6 CONSTRAINT GENERALIZATION

The starting point of generalization of a pair of pattern elements is necessarily the generalization of the corresponding constraints on the pattern. The basic concept of this generalization, and its application to simple binary constraints, is straightforward. If the constraints are the same, the corresponding constraint on the generalized pattern element will be the same. Otherwise, the constraint on the generalized pattern element should be the LGG of the two constraints. In the case of binary constraints, this would be simply the removal of the constraint. However, our implementation of RAPIER uses constraints that are more complicated to generalize. The discussion below emphasizes the issues of using the particular types of constraints used by our implementation. The algorithm could be applied using a variety of constraint types, but the discussion below should indicate what issues may arise in constraint generalization.

As mentioned above, the issue introduced by the word and tag constraints is simply that we allow unlimited disjunction in those constraints, leading us to prefer to produce two different generalizations in those cases where constraints differ: the disjunction of the words or tags in the constraint and the removal of the constraint.

Dealing with semantic constraints based on a hierarchy such as WordNet introduces complications into the generalization of constraints.

The first issue is that of ambiguous words. As indicated above, the most specific rules that RAPIER constructs do not create a semantic class constraint, because of the difficulties of selecting a particular semantic class for the word. Therefore, the algorithm for creating the generalization of two words must take this into account when generalizing the semantic constraint.

The second issue that arises is that of dealing with the semantic hierarchy. As stated above, for our semantic constraints, we are using the word senses (synsets) in WordNet.

Because of the structure of the hierarchy, there are typically several possible generalizations for two word senses, and we are interested in finding the least general of these. Since we cannot easily and efficiently guarantee finding the correct generalization for our purposes, especially given the issue that we may have multiple possible senses per word, we use a breadth-first search to try to minimize the total path length between the two word senses through the generalization. If we are generalizing from a semantic class (determined from a previous generalization step), we simply search upward from the WordNet synset that represents that class. When generalizing from words, we simultaneously search upwards through the hypernym hierarchy from all of the synsets containing the word. The first synset reached that is an ancestor of some possible synset from each of the two pattern elements being generalized is assumed to be the least general generalization of the two elements. Figure 6 shows possible results of this generalization process. Three different generalizations for the word "man" are shown in bold. Generalizing "man" and "world" results in the synset in bold that is a meaning for each of those words. Generalizing "man" and "woman" results in a synset for "person". Generalizing "man" and "rock" results "physical object". For purposes of simplicity, the figures show only a few of the possible meanings for each word, but it does indicate that there are other directions in which generalization might go for each individual word. In the generalization process, if no common ancestor exists, the generalized pattern element is semantically unconstrained.

It should be noted that this implementation of semantic constraints and their generalization is very closely tied to WordNet (Miller et al., 1993) since that is the semantic hierarchy used in this research. However, the code has been carefully modularized in order to make the process of substituting an alternative source for semantic information or modifying the generalization method to allow for disjunctions of classes relatively easy.

3.2.7 GENERALIZING PATTERN ELEMENTS

Given the rules for generalizing constraints, the generalization of a pair of pattern elements is fairly simple. First, the generalizations of the word, tag and semantic constraints of the two pattern elements are computed as described above. From that set of generalizations, RAPIER computes all combinations of a word constraint, a tag constraint, and the semantic constraint and creates a pattern element with each combination. See Figure 7 for an example of this combination. If both of the original pattern elements are pattern items, the new elements are pattern items as well. Otherwise, the new elements are pattern lists. The length of these new pattern lists is the maximum length of the original pattern lists (or the length of the pattern list in the case where a pattern item and a pattern list are being generalized).

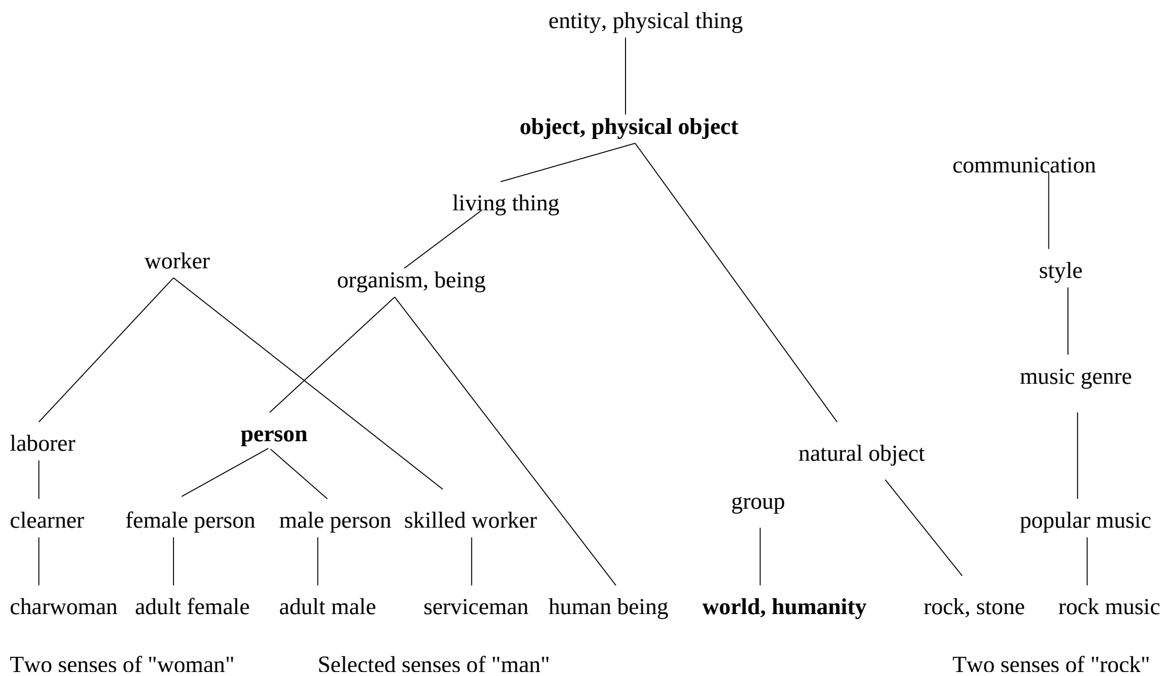

Figure 6: Portion of the WordNet hypernym hierarchy showing three different generalizations of the word "man"

3.2.8 GENERALIZING PATTERNS

Generalizing a pair of patterns of equal length is also quite straightforward. RAPIER pairs up the pattern elements from first to last in the patterns and computes the generalizations of each pair. It then creates all of the patterns made by combining the generalizations of the pairs of elements in order. Figure 8 shows an example.

Generalizing pairs of patterns that differ in length is more complex, and the problem of combinatorial explosion is greater. Suppose we have two patterns: one five elements long and the other three elements long. We need to determine how to group the elements to be generalized. If we assume that each element of the shorter pattern must match at least one element of the longer pattern, and that each element of the longer pattern will match exactly one element of the shorter pattern, we have a total of three ways to match each element of the shorter pattern to elements of the longer pattern, and a total of six ways to match up the elements of the two patterns. As the patterns grow longer and the difference is length grows larger, the problem becomes more severe.

In order to limit this problem somewhat, before creating all of the possible generalizations, RAPIER searches for any exact matches of two elements of the patterns being generalized, making the assumption that if an element from one of the patterns exactly matches an element of the other pattern then those two elements should be paired and the problem broken into matching the segments of the patterns to either side of these matching elements. However, the search for matching elements is confined by the first assumption of matching above: that each element of the shorter pattern should be generalized with at least one element of the longer pattern. Thus, if the shorter pattern, A, has three elements and the longer, B, has five, the first element of A is compared to elements 1 to 3 of

Elements to be generalized

```
Element A                     Element B
word:  man                    word:  woman
syntactic:  nnp               syntactic:  nn
semantic:                     semantic:
```

Resulting Generalizations

```
word:                         word:  {man, woman}
syntactic:  {nn, nnp}         syntactic:  {nn, nnp}
semantic:  person             semantic:  person

word:                         word:  {man, woman}
syntactic:                    syntactic:
semantic:  person             semantic:  person
```

Figure 7: An example of the generalization of two pattern elements. The words "man" and "woman" form two possible generalizations: their disjunction and dropping the word constraint. The tags "nn" (noun) and "nnp" (proper noun) also have two possible generalizations. Thus, there are a total of four generalizations of the two elements.

B, element 2 of A to elements 2-4 of B, and element 3 of A to elements 3-5 of B. If any matches are found, they can greatly limit the number of generalizations that need to be computed.

Any exact matches that are found break up the patterns into segments which still must be generalized. Each pair of segments can be treated as a pair of patterns that need to be generalized, so if any corresponding pattern segments are of equal length, they are handled just like a pair of patterns of the same length as described above. Otherwise, we have patterns of uneven length that must be generalized.

There are three special cases of different length patterns. First, the shorter pattern may have 0 elements. In this case, the pattern elements in the longer pattern are generalized into a set of pattern lists, one pattern list for each alternative generalization of the constraints of the pattern elements. Each of the resulting pattern lists must be able to match as many document tokens as the elements in the longer pattern, so the length of the pattern lists is the sum of the lengths of the elements of the longer pattern, with pattern items naturally having a length of one. Figure 9 demonstrates this case.

The second special case is when the shorter pattern has a single element. This is similar to the previous case, with each generalization again being a single pattern list, with constraints generalized from the pattern elements of both patterns. In this case the length of the pattern lists is the greater of the length of the pattern element from the shorter pattern or the sum of the lengths of the elements of the longer pattern. The length of the shorter pattern must be considered in case it is a list of length greater than the length of the longer pattern. A example of this case appears in Figure 10.

The third special case is when the two patterns are long or very different in length. In this case, the number of generalizations becomes very large, so RAPIER simply creates a single pattern list with no constraints and a length equal to the longer of the two patterns (measuring sums of lengths of elements). This case happens primarily with slot fillers of very disparate length, where there is

Patterns to be Generalized

```
Pattern A                  Pattern B
1) word:  ate              1) word:  hit
   syntactic:  vb             syntactic:  vb
2) word:  the              2) word:  the
   syntactic:  dt             syntactic:  dt
3) word:  pasta            3) word:  ball
   syntactic:  nn             syntactic:  nn
```

Resulting Generalizations

```
1) word:  {ate, hit}       1) word:
   syntactic:  vb             syntactic:  vb
2) word:  the              2) word:  the
   syntactic:  dt             syntactic:  dt
3) word:  {pasta, ball}    3) word:  {pasta, ball}
   syntactic:  nn             syntactic:  nn

1) word:  {ate, hit}       1) word:
   syntactic:  vb             syntactic:  vb
2) word:  the              2) word:  the
   syntactic:  dt             syntactic:  dt
3) word:                   3) word:
   syntactic:  nn             syntactic:  nn
```

Figure 8: Generalization of a pair of patterns of equal length. For simplicity, the semantic constraints are not shown, since they never have more than one generalization.

Pattern to be Generalized

```
1) word:  bank
   syntactic:  nn
2) word:  vault
   syntactic:  nn
```

Resulting Generalizations

```
1) list:  length 2         1) list:  length 2
   word:  {bank, vault}       word:
   syntactic:  nn             syntactic:  nn
```

Figure 9: Generalization of two pattern items matched with no pattern elements from the other pattern.

unlikely to be a useful generalization, and any useful rule is likely to make use of the context rather than the structure of the actual slot filler.

Patterns to be Generalized

```
Pattern A                    Pattern B
1) word:  bank               1) list:  length 3
   syntactic:  nn               word:
2) word:  vault                 syntactic:  nnp
   syntactic:  nn
```

Resulting Generalizations

```
1) list:  length 3           1) list:  length 3
   word:                        word:
   syntactic:  {nn,nnp}         syntactic:
```

Figure 10: Generalization of two pattern items matched with one pattern element from the other pattern. Because Pattern B is a pattern list of length 3, the resulting generalizations must also have a length of 3.

When none of the special cases holds, RAPIER must create the full set of generalizations as described above. RAPIER creates the set of generalizations of the patterns by first creating the generalizations of each of the elements of the shorter pattern against each possible set of elements from the longer pattern using the assumptions mentioned above: each element from the shorter pattern must correspond to at least one element from the longer pattern and each element of the longer pattern corresponds to exactly one element of the shorter pattern for each grouping. Once all of the possible generalizations of elements are computed, the generalizations of the patterns are created by combining the possible generalizations of the elements in all possible combinations which include each element of each pattern exactly once in order.

In the case where exact matches were found, one step remains after the various resulting pattern segment pairs are generalized. The generalizations of the patterns are computed by creating all possible combinations of the generalizations of the pattern segment pairs.

3.2.9 SPECIALIZATION PHASE

The final piece of the learning algorithm is the specialization phase, indicated by calls to SpecializePreFiller and SpecializePostFiller in Figure 5. These functions take two parameters, the rule to be specialized and an integer *n* which indicates how many elements of the pre-filler or post-filler patterns of the original rule pair are to be used for this specialization. As *n* is incremented, the specialization uses more context, working outward from the slot-filler. In order to carry out the specialization phase, each rule maintains information about the two rules from which it was created, which are referred to as the base rules: pointers to the two base rules, how much of the pre-filler pattern from each base rule has been incorporated into the current rule, and how much of the post-filler pattern from each base rule has been incorporated into the current rule. The two specialization functions return a list of rules which have been specialized by adding to the rule generalizations of the appropriate portions of the pre-fillers or post-fillers of the base rules.

One issue arises in these functions. If the system simply considers adding one element from each pattern at each step away from the filler, it may miss some useful generalizations since the lengths

```
SpecializePreFiller(CurRule,n)
Let BaseRule1 and BaseRule2 be the two rules from which CurRule was created
Let CurPreFiller be the pre-filler pattern of CurRule
Let PreFiller1 be the pre-filler pattern of BaseRule1
Let PreFiller2 be the pre-filler pattern of BaseRule2
Let PatternLen1 be the length of PreFiller1
Let PatternLen2 be the length of PreFiller2
Let FirstUsed1 be the first element of PreFiller1 that has been used in CurRule
Let FirstUsed2 be the first element of PreFiller2 that has been used in CurRule
GenSet1 = Generalizations of elements (PatternLen1 + 1 − n) to FirstUsed1 of
        PreFiller1 with elements (PatternLen2 + 1 − (n − 1)) to FirstUsed2 of
        PreFiller2
GenSet2 = Generalizations of elements (PatternLen1 + 1 − (n − 1)) to
        FirstUsed1 of PreFiller1 with elements (PatternLen2 + 1 − n) to
        FirstUsed2 of PreFiller2
GenSet3 = Generalizations of elements (PatternLen1 + 1 − n) to FirstUsed1 of
        PreFiller1 with elements (PatternLen2 + 1 − n) to FirstUsed2 of
        PreFiller2
GenSet = GenSet1 ∪ GenSet2 ∪ GenSet3
NewRuleSet = empty set
For each PatternSegment in GenSet
    NewPreFiller = PatternSegment concatenate CurPreFiller
    Create NewRule from CurRule with pre-filler NewPreFiller
    Add NewRule to NewRuleSet
Return NewRuleSet
```

Figure 11: RAPIER Algorithm for Specializing the Pre-Filler of a Rule

of the two patterns being generalized would always be the same. For example, assume we have two rules for required years of experience created from the phrases "6 years experience required" and "4 years experience is required." Once the fillers were generalized, the algorithm would need to specialize the resulting rule(s) to identify the number as years of experience and as required rather than desired. The first two iterations would create items for "years" and "experience," and the third iteration would match up "is" and "required." It would be helpful if a fourth iteration could match up the two occurrences of "required," creating a list from "is." In order to allow this to happen, the specialization functions do not only consider the result of adding one element from each pattern; they also consider the results of adding an element to the first pattern, but not the second, and adding an element to the second pattern but not the first.

Pseudocode for SpecializePreFiller appears in Figure 11. SpecializePostFiller is analogous. In order to allow pattern lists to be created where appropriate, the functions generalize three pairs of pattern segments. The patterns to be generalized are determined by first determining how much of the pre-filler (post-filler) of each of the original pair of rules the current rule already incorporates. Using the pre-filler case as an example, if the current rule has an empty pre-filler, the three patterns to be generalized are: 1) the last n elements of the pre-filler of *BaseRule1* and the last $n-1$ elements of the pre-filler of *BaseRule2*, 2) the last $n-1$ elements of the pre-filler of *BaseRule1* and the last n elements of the pre-filler of *BaseRule2*, and 3) the last n elements of the pre-filler of each of the base rules. If the current rule has already been specialized with a portion of the pre-filler, then whatever elements it already incorporates will not be used, but the pattern of the pre-filler to be used will start at the same place, so that *n* is not the number of elements to be generalized, but rather specifies the portion of the pre-filler which can be considered at that iteration.

The post-filler case is analogous to the pre-filler case except that the portion of the pattern to considered is that at the beginning, since the algorithm works outward from the filler.

3.2.10 COMPLETE SAMPLE INDUCTION TRACE

As an example of the entire process of creating 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, though rules in practice would be much longer. The initial, specific rules created from these phrases for the city slot for a job template 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:  .
```

For the purposes of this example, we assume that there is a semantic class for states, but not one for cities. For simplicity, we assume the beam-width is 2. 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 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 produced are:

```
Pre-filler Pattern:         Filler Pattern:               Post-filler Pattern:
                            1) list:  max length:  2
                               word:  {atlanta, kansas, city}
                               tag:  nnp
and
Pre-filler Pattern:         Filler Pattern:               Post-filler Pattern:
                            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. This results in:

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

The items produced from the "in"'s and the commas are identical and, therefore, unchanged. Alternative, but less useful rules, will also be produced with lists in place of the items in the pre-filler and post-filler patterns because of specializations produced by generalizing the element from each pattern with no elements from the other pattern. Continuing the specialization with the two alternatives above only, the algorithm moves on to look at the second to last elements in the pre-filler pattern. This generalization of these elements produce six possible specializations for each of the rules in the current beam:

```
list: length 1              list: length 1              word: {located, offices}
word: located               word: offices               tag: {vbn, nns}
tag: vbn                    tag: nns

word:                       word:                       word: {located, offices}
tag: {vbn, nns}             tag:                        tag:
```

None of these specializations is likely to improve the rule, and specialization proceeds to the second elements of the post-fillers. Again, the two pattern lists will be created, one for the pattern item from each pattern. Then the two pattern items will be generalized. Since we assume that the 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: max length: 2       1) word: ,
   tag: in                      tag: nnp                     tag: ,
                                                          2) tag: nnp
                                                             semantic: state
```

4. Experimental Evaluation

We present here results from two data sets: a set of 300 computer-related job postings from the `austin.jobs` newsgroup 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). For all experiments, we used the default values for all of RAPIER's parameters.

We also present results from three other learning information extraction systems. One is a Naive Bayes system which uses words in a fixed-length window to locate slot fillers (Freitag, 1998b). Very

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

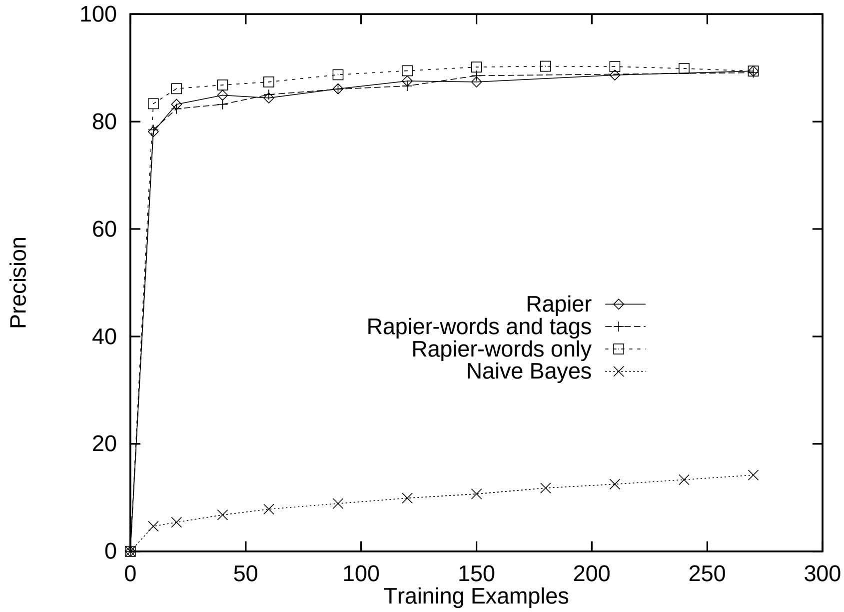

Figure 12: Precision on job postings

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, 2000) 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).

We ran the Naive Bayes system on the jobs data set using splits identical to those for RAPIER. All other results reported for these systems are from the author's cited papers.

4.1 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.

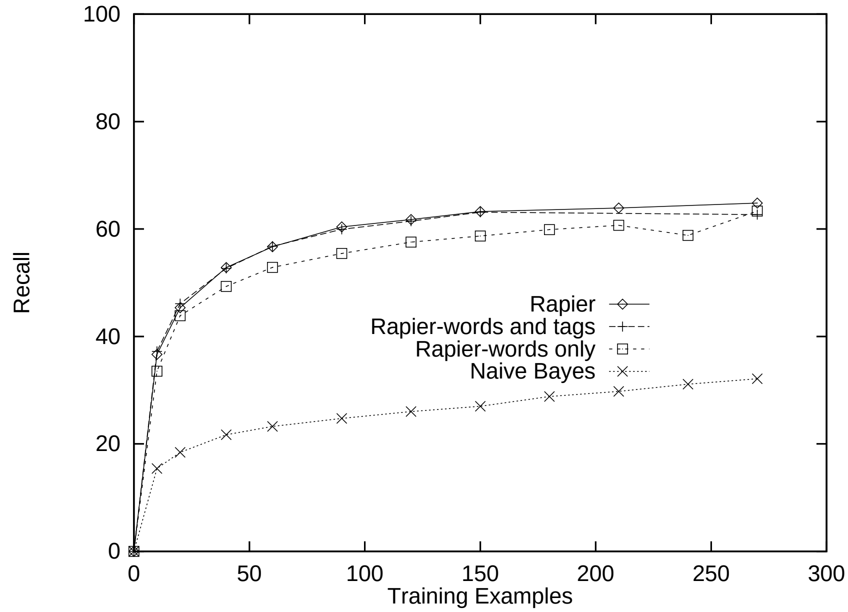

Figure 13: Recall on job postings

Figure 12 shows the learning curve for precision and Figure 13 shows the learning curve for 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), we set the Naive Bayes' threshold 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 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

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

differences are not statistically significant. WordNet's semantic classes provided no significant performance increase over words and POS tags only. Although recall increases with the use of semantic classes, precision decreases. This probably indicates that the semantic classes in WordNet are not a good fit for this problem, so that semantic generalizations are producing overly general rule.

One other learning system, WHISK, has been applied to this data set. Soderland reports that, 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% (Soderland, 1999). This is slightly worse than RAPIER's performance at 90 examples with part-of-speech tags with precision of 86% and recall of 60%. In making this comparison, it is important to note that the test sets are different and that Whisk system's performance was actually counted a bit differently, since duplicates were not eliminated. It's not entirely clear why WHISK does not perform quite as well as RAPIER, though one possibility is the restriction of context in WHISK to a single sentence.

4.2 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, 2000). 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. In looking at performance on these slots, it should be noted that results were counted a bit differently between the CMU systems and RAPIER. Freitag's systems assume only one possible answer per slot. These slots may have multiple correct answers (eg. either "2pm" or "2:00"), but either answer is considered correct, and only one answer is counted per slot. RAPIER makes no such assumption, since it allows for the possibility of needing to extract multiple, independent strings. Thus, performance is measured assuming that all of the possible strings need to be extracted. This is a somewhat harder task, at least partially accounting for RAPIER's weaker performance. It should also be noted that SRV makes use of binary features that look at orthographic issues and length of tokens; these kinds of features may be more useful for recognizing times than the word, POS tag, and semantic class features that our implementation of RAPIER has available to it.

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 consist of a sequence of cardinal numbers and proper nouns, and the POS tags can recognize both of those consistently. SRV has higher recall than 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.

4.3 Discussion

The results above show that relational methods can learn useful rules for information extraction, 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 none 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). One issue that should be addressed in future work is to examine the effect of various features, seeing how much of the differences in performance depend upon the features rather than basic representational and algorithmic biases. All of the algorithms should be adaptable to different features; certainly adapting RAPIER to use binary features of the type that SRV can employ should be straightforward.

4.4 Sample Learned Rules

One final interesting thing to consider about RAPIER is the types of rules it creates. One common type of rule learned for certain kinds of slots is the rule that simply memorizes a set of possible slot-fillers. For example, RAPIER learns that "mac," "mvs," "aix," and "vms" are platforms in the computer-related jobs domain, since each word only appears in documents where it is to be extracted as a platform slot-filler. One interesting rule along these lines is one which extracts "C++" or "Visual C++" into the language slot. The pre-filler and post-filler patterns are empty, and the filler pattern consists of a pattern list of length 1 with the word constraint "visual" and then pattern items for "c", "+" and "+". In the seminar announcements domain, one rule for the location slot extracts "doherty," "wean" or "weh" (all name of buildings at CMU) followed by a cardinal number. More often, rules which memorize slot-fillers also include some context to ensure that the filler should extracted in this particular case. For example, a rule for the area slot in the jobs domain extracts "gui" or "rpc" if followed by "software."

Other rules rely more on context than on filler patterns. Some of these are for very formal patterns, such as that for the message id of a job posting:

```
Pre-filler Pattern:       Filler Pattern:       Post-filler Pattern:
1) word:  message         1) list:  length 5    1) word:  >
2) word:  -
3) word:  id
4) word:  :
5) word:  <
```

Probably the majority of rules have some mix of context and the contents of the filler. An example is the following rule for a title in the computer-related jobs domain:

```
Pre-filler Pattern:       Filler Pattern:       Post-filler Pattern:
1) word:  {:, seeking}    1)
                          2) {consultant, dba}
```

In the seminar announcements domain, the following rule for the start time relies on a combinations of the structure of the filler and its context:

```
Pre-filler Pattern:       Filler Pattern:       Post-filler Pattern:
1) word:  {:, for}        1) syntactic:  cd
2)                        2) word:  :
                             syntactic:  :
                          3) syntactic:  cd
                          4) syntactic:  nn
```

5. Future Work

There are number of directions in which this work could be extended. Here we list a few of those that we consider immediately promising or helpful.

As mentioned above, our understanding of the various relational learning systems for information extraction would benefit from a systematic study of these systems that used the same features for each system and thus distinguished the contributions of available features vs. algorithms. At present, there is very little evidence to indicate which of the systems is best or most promising as a step toward future improvements. Adding additional constraint types to RAPIER's representation (such as the orthographical features employed by SRV) is a straightforward task.

Another useful extension to RAPIER would be the ability to learn rules which extract fillers for multiple slots. There are two advantages to learning such rules. First, if two slots often appear in a particular relationship to one another is a document, then learning a single rule for both slots may help to focus the search for a good rule. This could be helpful for learning start time and end time in the seminar announcements domain or for a position and a company in a management changes domain. There are also certain situations where there are multiple fillers for slots, but the fillers are in some way connected. For instance, in a rental ads domain on which WHISK has been tested (Soderland, 1999), it is common to have different sized apartments at different prices listed in the same ad. To simply extract the numbers of bedrooms and the prices without connecting them is not helpful. Learning rules that extract both the number of bedrooms and the related price can help to solve this problem.

Modifying RAPIER to handle multiple slot extraction rules should be fairly straightforward. A rule would simply have additional patterns: one for each slot-filler to be extracted, patterns between the slot-fillers, and the context patterns before the first slot-filler and after the last slot-filler. The primary issue that might be problematic would be the generalization of patterns in between two slot fillers. These would probably contain useful information, but might be long and of different sizes, causing generalizing a pair of them to be prohibitively expensive. It might be necessary to take a more top-down approach to learning the connecting patterns: starting with an unconstrained pattern list and breaking it up into pattern items or smaller lists and adding constraints (taken from those in the original rule pair to limit search) as long as such constraints improved rule quality.

Another desirable modification to RAPIER would be to enable the algorithm to take advantage of other kinds of pre-processing which somehow tag more than one token of text, such as parsing, phrase chunking, or named-entity recognition. This is less-straightforward than handling additional constraint types on a single token and would clearly complicate the generalization and specialization phases of RAPIER's algorithm. However, named-entity recognition in particular could prove very useful in identifying filler patterns, and phrase chunking can provide useful information for information extraction rules (Ray and Craven, 2001).

A final direction of research would be considering applying RAPIER's representation and algorithm to other natural language processing tasks. It seems to us quite likely that this language-specific representation would fit well with other types of tasks and possibly allow success with less engineering of features than many of the learning systems currently used must employ.

6. 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 information extraction 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 and 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 information extraction patterns (Huffman, 1996), but also requires 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 et al., 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.

Also requiring mention is work on learning information extraction and text categorization rules using ILP (Junker et al., 2000). Unlike RAPIER, WHISK, and SRV, which use text-specific representations and algorithms informed by ILP methods, they use a logic representation with an algorithm focused on text. Comparisons to this work is not yet possible, since they present no results.

Other effective approaches to learning information extraction rules have been studied recently. These approaches have been shown to work well on some portions of the tasks used in this paper. However, none of them has been tested on the complete set of tasks. Particularly, most have not been tested on the slots that require the ability to produce multiple fillers for a given slot.

One approach has been learning hidden Markov models (Freitag and McCallum, 2000, McCallum et al., 2000, Lafferty et al., 2001). The HMM learning methods seem to work well, but they have been tested on only a few of the slots in the job and seminar datasets: tasks that are generally generally the more difficult for SRV and RAPIER. This work has been limited to locating a single string to extract and has not been applied to slots with multiple correct fillers.

Another approach has been to use a simpler learner used to develop "wrappers" for accomplishing information extraction from very structured texts such as web pages and applying boosting (Freitag and Kushmerick, 2000). The BWI system has been tested on a few slots of the seminar announcement and job posting tasks, but has not been applied to the full job postings task. It also has not been applied to extraction of multiple fillers per slot. The results using BWI are therefore inconclusive, but also positive.

Roth and Yih (2001) have presented a system–SNoW-IE–which they use to learn the relational representation desirable for the information extraction task using propositional learning mechanisms. Their system performs very well on the seminar task; however, it has not been tested on the job task, and is, in fact, geared specifically toward extraction of at most one filler per slot.

Another very successful recent approach to the information extraction task has been that of Ciravegna (2001). His symbolic rule learner learns rules that insert tags into the text independently, proceeding in four phases: an initial tag insertion phase; a contextual rules phase, in which rules are dependent on the presence of other tags; a correction phase, in which rules are applied to adjust the placement of tags; and a validation phase, in which tags without matches are removed (beginning tags without ending tags or vice versa). This system benefits from use of morphological and dictionary linguistic evidence. It outperforms existing work overall, although RAPIER does outperform it on several slots of the job task, indicating that there is room for improvement in both systems.

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 here 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 from 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.

Acknowledgments

Thanks to Dayne Freitag for supplying his seminar announcements data. Thanks to the anonymous reviewers for comments on earlier drafts of this paper. 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

C. Aone and S. W. Bennett. Evaluating automated and manual acquisition of anaphora resolution strategies. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 122–129, Cambridge, MA, 1995.

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

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

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

Mary Elaine Califf. *Relational Learning Techniques for Natural Language Information Extraction*. PhD thesis, Department of Computer Sciences, University of Texas, Austin, TX, May 1998. Available from http://www.cs.utexas.edu./users/ai-lab.

C. Cardie. A case-based apprach to knowledge acquisition for domain-specific sentence analysis. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 798–803, Washington, DC, 1993.

Eugene Charniak. *Statistical Language Learning*. MIT Press, 1993.

Fabio Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1251–1256, Seattle, WA, 2001.

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

William Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, San Francisco, CA, 1995b.

Mark Craven and J. Kumlien. Constructing biological knowledge bases by extracting information from text sources. In *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology*, pages 77–86, Heidelberg, Germany, 1999.

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

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

L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1058–1063, Chambery, France, 1993.

Dayne Freitag. Information extraction from HTML: Application of a general learning approach. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 517–523, Madison, Wisconsin, 1998a.

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

Dayne Freitag. Machine learning for information extraction in informal domains. *Machine Learning*, 39:169–202, 2000.

Dayne Freitag and Nicholas Kushmerick. Boosted wrapper induction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 577–583, Austin, TX, 2000.

Dayne Freitag and Andrew McCallum. Information extraction with HMM structures learned by stochastic optimization. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 584–589, Austin, TX, 2000.

Chun-Nan Hsu and Nim-Tzung Dung. Wrapping semistructured web pages with finite-state transducers. In *Proceedings of the Conference on Automated Learning and Discovery*, Pittsburgh, PA, 1998.

S. B. Huffman. 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*, pages 246–260. Springer, Berlin, 1996.

M. Junker, M. Sintek, and M. Rinck. Learning for text categorization and information extraction with ILP. In J. Cussens and S. Džeroski, editors, *Learning Language in Logic*. Springer, 2000. Lecture Notes in Computer Artificial Intelligence 1925.

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

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, Williamstown, MA, 2001.

N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

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

D. M. Magerman. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Cambridge, MA, 1995.

C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 591–598, Williamstown, MA, 2000.

R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–161, 1983.

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

Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. A fully statistical approach to natural language interfaces. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 55–61, Santa Cruz, CA, 1996.

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

S. Muggleton. Duce, an oracle based approach to constructive induction. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 287–292, Milan, Italy, Aug 1987.

S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352, Ann Arbor, MI, June 1988.

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

S. Muggleton, R. King, and M. Sternberg. Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647–657, 1992.

S. H. Muggleton, editor. *Inductive Logic Programming*. Academic Press, New York, NY, 1992.

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

Ion Muslea, Steve Minton, and Craig Knoblock. Wrapper induction for semistructured, web-based information sources. In *Proceedings of the Conference on Automated Learning and Discovery*, Pittsburgh, PA, 1998.

H. T. Ng and H. B. Lee. Integrating multiple knowledge sources to disambiguate word sense: An exemplar-based approach. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 40–47, Santa Cruz, CA, 1996.

M. J. Pazzani, C. A. Brunk, and G. Silverstein. An information-based approach to integrating empirical and explanation-based learning. In S. Muggleton, editor, *Inductive Logic Programming*, pages 373–394. Academic Press, New York, 1992.

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

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

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

Adwait Ratnaparkhi. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 1–10, Providence, RI, 1997.

S. Ray and M. Craven. Representing sentence structure in hidden Markov models for information extraction. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1273–1279, Seattle, WA, 2001.

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

Dan Roth and W. Yih. Relational learning via propositional algorithms: An information extraction case study. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1257–1263, Seattle, WA, 2001.

Sean Slattery and Mark Craven. Learning to exploit document relationships and structure: The case for relational learning on the web. In *Proceedings of the Conference on Automated Learning and Discovery*, Pittsburgh, PA, 1998.

Frank Smadja, Kathleen R. McKeown, and Vasileios Hatzivassiloglou. Translating collocations for bilingual lexicons: A statistical approach. *Computational Linguistics*, 22(1):1–38, 1996.

Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34:233–272, 1999.

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

A. Srinivasan, S.H. Muggleton, M.J. Sternberg, and R.D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85:277–300, 1996.

J. Weizenbaum. 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, 1966.

S. Wermter, E. Riloff, and G. Scheler, editors. *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing.* Springer Verlag, Berlin, 1996.

Yiming Yang and Jan O. Pederson. A comparative study in feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 412–420, Nashville, TN, 1997.

J. M. Zelle and R. J. Mooney. 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 1994.

J. M. Zelle, C. Thompson, M. E. Califf, and R. J. Mooney. Inducing logic programs without explicit negative examples. In *Proceedings of the Fifth International Workshop on Inductive Logic Programming*, pages 403–416, Leuven, Belgium, 1995.