Cornell University Computer Science Technical Report TR98-1702

Proposal for an Interactive Environment for Information Extraction

Claire Cardie and David Pierce
Department of Computer Science
Cornell University
Ithaca NY 14853 USA
cardie, pierce@cs.cornell.edu

July 6, 1998

Abstract

Information extraction systems have been successfully deployed for domains ranging from terrorist activities to medical records. However, building these systems remains costly for users who lack annotated training corpora or knowledge engineering expertise. This paper proposes a framework for an interactive information extraction environment in which the user trains the system by example and by feedback about performance. If successful, this will be the first system that allows end-users to create information extraction systems without the aid of computational linguists and NLP system designers.

1 Information Extraction

Information extraction (IE) systems are natural language processing (NLP) systems that produce structured summaries of input text (e.g. see Cardie (1997)). These systems are important both to users with information needs and to NLP researchers. From an NLP point of view, these systems are important because they attack a more tractable task than the fully general language understanding problem of determining the complete meaning of each sentence in a text; they focus instead on domain-specific text applications, such as extracting diagnoses and symptoms from medical patient records. From a user's point of view, information extraction systems perform important tasks such as identifying interesting documents, presenting their content in a concise and consistent manner, and even generating high-level document representations for information retrieval. To give some examples, IE systems for the terrorism domain of the MUC conferences aid defense analysts in monitoring terrorist activities in Latin America (MUC-3, 1991); while doctors use IE systems for the medical domain to collect and collate evidence for diagnoses by automatically examining a patient's medical record (Soderland *et al.*, 1995). In general, information extraction systems allow users to consider significantly more data by automatically structuring and filtering information.

As an introduction to information extraction, consider the following text from our "Quality Circle" domain. A Cornell sociologist wants to extract from a collection of journal abstracts several kinds of information, including: what motivated industries to organize quality circle innovations; what results the innovations delivered, whether good or bad; what caused innovations to succeed or fail.

> Title: A new spin on quality circles
>
> Authors: Vasilash, Gary S
>
> Subjects: Automobile industry; Case studies; Quality circles; R&D
>
> Codes: 9190 (United States); 9110 (Company specific); 8680 (Transportation equipment industry); 5320 (Quality control); 5400 (Research & development)
>
> The New Honda (NH) circle is something used throughout Honda. It is familiar to many people as a quality circle. However, whereas quality circles in many organizations became disappearing spirals largely due to lack of on-going focus, consistency and application, Honda people have made NH circle part of their on-going routine. Although the tendency of many manufacturing

companies was to organize quality circles almost exclusively on the shop floor, at Honda there are no such limits. At Honda R&D North America's Raymond, Ohio complex, an NH circle was used to solve problems with CAD. The paybacks for quality circles in engineering can be significant.

There is much useful information in this text. The second sentence describes the fate of *quality circles* in some organizations where they *became disappearing spirals*. The cause of this was *lack of on-going focus, consistency and application*. On the other hand, the penultimate sentence relates a positive result for *an NH circle* which was used *to solve problems with CAD*. IE systems typically collect these bits of information into report forms called *templates*.

Innovation Report	
Result:	Failure
Innovation:	"quality circles"
Innovation Fate:	"became disappearing spirals"
Cause of Problems:	"lack of on-going focus, consistency and application"

Innovation Report	
Result:	Success
Innovation:	"NH circle"
Innovation Use:	"to solve problems with CAD"

The information extraction task, then, is to identify and classify interesting portions of a document such as these. Accordingly, an information extraction system must be capable of performing many operations including: identifying patterns using lexical, syntactic, and semantic features; identifying conceptual relationships between pieces of information such as innovations and their results; and identifying coreferences between extractions that refer to the same entity, such as a pronoun and its antecedent.

While IE systems have been used successfully for real applications such as monitoring terrorists and diagnosing medical conditions, there are still obstacles to building and using them. The two obstacles addressed by this proposal are these:

1. **The cost of new domain-specific components.** Traditionally, new domain-specific components have been built by hand for each new application. This required many hours of painstaking work by a programmer who was familiar with the underlying NLP system and aware of the linguistic issues involved.

2. **The limitation of extracted structures.** Currently, information extraction systems are capable of extracting only noun phrases in particular syntactic positions to classify as instances of a concept.

Let's consider the first obstacle. The most crucial domain-specific component of typical IE systems is the *concept dictionary*. This is a set of patterns that describes the kinds of constituents that should be labeled as a particular extraction type. For example, an effective Innovation Use pattern (for the third slot in the second example template) might look something like this:

```
if X is a Clause constituent where
   subject(X) was extracted as an Innovation
   verb(X) = "use" in Passive Voice
   complement(X) is an Infinitive Clause
then
   extract complement(X) as an Innovation Use.
```

This pattern matches any clause stating that "an innovation was used to do something" and extracts the "to do something" part of the text. In particular, this pattern matches the sentence containing *an NH circle was used to solve problems with CAD*, extracting *to solve problems with CAD* as a Innovation Use concept. The extracted concept can later be combined with other extractions to form a template like the example.

Concept dictionaries typically consist of hundreds to thousands of patterns like this. New dictionaries are required for each new domain, and creating them consumes a great amount of time. Research efforts have begun to address this: e.g. Cardie (1993), Riloff (1993), Soderland et al. (1995), Huffman (1996). These corpus-based methods all employ a large training corpus annotated with examples for each concept or type of extraction. From these examples, machine learning algorithms induce conceptual patterns for extraction. However, these methods have not eliminated the cost of building information extraction systems, but rather simply deferred the cost of encoding linguistic knowledge bases for IE to the cost of annotating corpora for each IE task. So further effort has been necessary to address the cost of corpus annotation. Substituting less costly corpus information is one strategy. For example, the AutoSlog-TS system uses document classifications, in place of the full corpus annotations used by the original AutoSlog (Riloff, 1996).

The other obstacle to using information extraction is the limitation imposed on the extracted text. Each of the methods referenced above creates patterns to extract only noun phrases as the text to place in template slots such as Innovation Use. While noun phrases are useful, there are domains in which it is important to extract also verb phrases, clauses, adverbials, or noun modifiers.

The effect of these two obstacles has been to reduce the utility and accessibility of IE technology for non-expert users. First, users cannot provide large, annotated, domain-specific corpora, so the advances in corpus-based dictionary generation have limited utility. Second, the restriction on the structures extracted may eliminate extractions that users consider desirable.

> **Goal** Therefore, the goal of this research is to enable the automatic construction of information extraction systems for real users.

To achieve this goal, we address both of the obstacles outlined above. To address the first obstacle, the cost of domain-specific knowledge engineering for IE, we start with a different assumption from other corpus-based methods: we assume that *no* information is provided with the corpus of documents. With no corpus annotations, relevance judgments, or any other form of corpus information, the IE system must look elsewhere for the training information necessary to learn extraction patterns. Our proposal is that it be provided through interaction with the user. The system will learn both from examples supplied by the user and from the user's feedback regarding current performance.

> **Hypothesis** Thus, the primary hypothesis of this work is that information extraction systems can be automatically constructed in an "interactive information extraction environment."

In order to rely on interaction with users, the interactive information extraction environment must be a place where users can work effectively even though they are not linguists or NLP experts. One aspect of this philosophy is that users should be able to annotate and extract the parts of the text they want, not just noun phrases. This forces us to address the second obstacle and abandon traditional IE limitations. Consequently, the system will need effective ways of learning patterns to extract a wide variety of syntactic constructs and combinations of constituents.

> **Goal** Therefore, a second goal of this work is to broaden information extraction to include the task of imitating or reproducing a selection of arbitrary annotations on text.

The fundamental importance of this proposal is that it makes information extraction technology accessible to non-experts. But in so doing, we expect to reap many other benefits. First, effective, interactive, user-oriented training may eliminate a large share of the cost of engineering an IE system for a new domain, because the user and the system, cooperating, can locate and process concept examples much more efficiently than the user alone. Second, the flexibility of information extraction systems will increase if users are allowed to extract whatever text they want, rather than being compelled to extract only certain structures. Third, the user's feedback may provide high-level information that can be used to improve not only the extraction component, but also the parsing component. Suppose the parser finds competing syntactic analyses for an ambiguous portion of text. If one of these parses matches an extraction pattern that is known to be very accurate, and if the user judges this matching instance to be correct, then the extraction pattern constitutes compelling evidence for choosing the matching parse in other similarly ambiguous contexts that match the pattern. Finally, seeking to achieve our goals, we will contribute to the collection of useful techniques for interactive machine learning and human–computer cooperation.

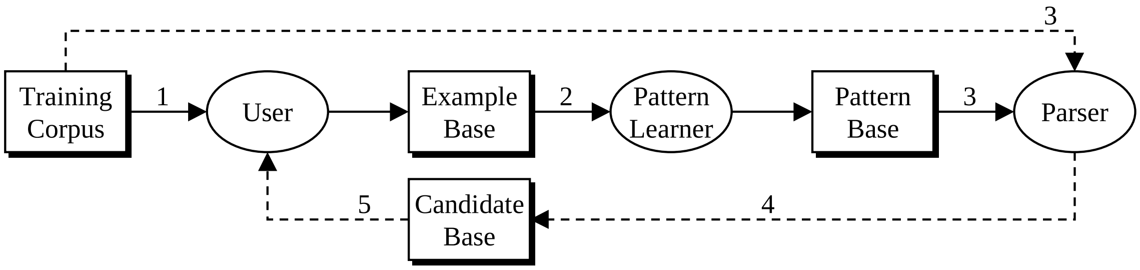

Figure 1: Interactive Information Extraction Environment Model

The rest of this document discusses the proposed interactive environment. Section 2 outlines how such an environment might work, using an extended example. Sections 3 and 4 consider the two most important components of the framework, the partial parser and the pattern learner, comparing related work to the needs of this new environment. Finally, Section 5 gives a brief research plan for accomplishing the goals set forth above.

2 Interactive Environment

Figure 1 depicts schematically the general approach to building an interactive information extraction environment (I^2E^2). The user initiates the training sequence, then the user and the system (parser and pattern learner) participate in an interactive learning loop. The numbered arrows in the diagram correspond to the following steps:

1. Drawing on the training corpus, the user annotates instances of some concept, adding them to the example base.

2. At any point the user chooses, the learning component begins to induce patterns for the examples provided in the example base.

3. The parser collects candidate matching instances of the patterns throughout the training corpus.

4. The candidate instances are presented to the user for review. The user rejects incorrect instances and accepts correct instances, thereby providing the system with a base of positive and negative examples for the concept at relatively low cost.

5. Using the positive and negative instances, the pattern learner adapts its pattern base accordingly. Ultimately the system finds a set of patterns that maximize the positive match instances while minimizing the negative instances.

In practice these five steps would be interleaved; so the user might only review a few of the candidate instances, rather than all of them, at step 4 before allowing the system to continue with step 5. The user's strategy for using the system may vary from simply operating the system as an annotation tool to mark the corpus, document by document, without taking advantage of the system's learning capability (i.e. continuously repeating step 1); to reviewing a single instance at a time, guiding the system through the modification of a single pattern, and using the pattern learner to find as many additional positive instances as possible (i.e. a fine-grained loop through steps 2, 3, 4, and 5). Presumably the most efficient strategy lies somewhere between these two extremes.

The interactive learning process is used to induce a pattern base for each type of annotation. The process completes when the user is satisfied with the performance of the extraction system or when the accuracy of the learned patterns is maximized over an annotated test corpus. The result is a partially (or fully) annotated training corpus and a partial (or complete) IE system. Ideally, the concept induction process will be quick and efficient for the user. What makes efficiency possible is that the user can direct the learning process while the system can very quickly generate instances and refine patterns.

To illustrate these ideas in more detail, let's consider how this interactive process might apply to our example "Quality Circle" text.

Annotating Text. To interact with the I^2E^2, the user must annotate the document with logical annotations denoting the interesting concepts in the text. For example, in the Quality Circle domain, the sociologist has identified concepts such as: Innovation (e.g. *quality circles*); Innovation Fate (e.g. *became disappearing spirals*); Cause of Problems (e.g. *lack of on-going focus, consistency and application*); and Innovation Use (e.g. *to solve problems with CAD*). The system component that deals with storing and handling these document annotations is based on the Tipster architecture (Grishman, 1997), an abstract architecture for managing collections, documents, and annotations.

In the Tipster view, a document consists of raw text associated with a database of annotations. Annotations identify portions of the text as logical objects. Annotations consist of

- an ID, which is unique among the document's annotations;
- a type, such as Innovation or Innovation Fate;
- a set of spans: a span is a pair of integer indices into the document character stream; the spans represent the subsequences of document text that comprise the annotation;
- a set of attribute-value pairs: a set of attribute names associated with a value that is a string or an annotation.

Figure 2 shows a database of annotations marked on the example text. These are the annotations that the sociologist wants the system to learn to imitate. Annotation 4, for example, has type Innovation Fate. It spans $\langle 447, 474\rangle$ over the text *became disappearing spirals*. It has an innovation attribute with the value 3 (a logical annotation reference) because it describes the fate of Innovation 3, namely *quality circles*. Since a table or database is not a very intuitive way of viewing annotations, we often describe them instead by "marking up" the text spans with which they associate.

> However, whereas ⟨Innovation id=3⟩quality circles⟨/Innovation⟩ in many organizations ⟨Innovation Fate id=4, innovation=3⟩became disappearing spirals⟨/Innovation Fate⟩ largely due to...

To begin interaction with the I^2E^2, the user provides examples of a particular type of annotation. The simplest annotation types should be supplied first, since the system obviously cannot learn complex annotations until their simpler constituents are known.[1] In our example, the user might mark some examples of the type Innovation. The system creates the annotations from these and stores them in the example base.

Learning Patterns. Now that the system has examples, it can begin learning. It considers one instance, say Innovation 1 from the first sentence.

> ⟨Innovation id=1⟩The New Honda (NH) circle⟨/Innovation⟩ is something used throughout Honda.

The learner generates a concept pattern to match the example. At an abstract level, a pattern specifies a certain combination of features. One possible pattern to extract Innovation 1 is this.

```
if X is a Clause constituent where
   subject(X) matches
      if Y is a Noun Phrase constituent where
         head(Y) = "circle"
   head(verb(X)) = "is"
   head(object(X)) contains "something"
then
   extract Y.
```

This pattern extracts a subject annotation whose category is noun phrase and head is *circle*, from a copular clause whose object contains the word *something*. Of course, this is an overly specific pattern that is likely to be useless to the system. The system instead begins with the most general pattern it can formulate.

[1] Complex annotations are, for example, annotations that require other annotations as attribute-values. Simple annotations, in contrast, only serve to identify a portion of text, rather than to relate multiple portions of text.

ID	Type	Spans	Attributes
1	Innovation	$\langle 279, 304 \rangle$	
	The New Honda (NH) circle		
2	Innovation	$\langle 376, 390 \rangle$	
	quality circle		
3	Innovation	$\langle 409, 424 \rangle$	
	quality circles		
4	Innovation Fate	$\langle 447, 474 \rangle$	innovation = 3
	became disappearing spirals		
5	Cause of Problems	$\langle 490, 541 \rangle$	innovation = 3
	lack of on-going focus, consistency and application		
6	Innovation	$\langle 566, 575 \rangle$	
	NH circle		
7	Innovation Fate	$\langle 543, 606 \rangle$	innovation = 6
	Honda people have made NH circle part of their on-going routine		
8	Innovation	$\langle 678, 693 \rangle$	
	quality circles		
9	Domain	$\langle 608, 730 \rangle$	
	Although the tendency of many manufacturing companies was to organize quality circles almost exclusively on the shop floor		
10	Domain	$\langle 732, 765 \rangle$	
	at Honda there are no such limits		
11	Innovation	$\langle 819, 831 \rangle$	
	an NH circle		
12	Innovation Use	$\langle 841, 867 \rangle$	innovation = 11
	to solve problems with CAD		
13	Innovation	$\langle 886, 901 \rangle$	
	quality circles		
14	Innovation Use	$\langle 869, 935 \rangle$	
	The paybacks for quality circles in engineering can be significant		

Figure 2: Example user annotations for the Quality Circle text.

if X is a Noun Phrase constituent
then extract X.

This pattern matches all noun phrases and is obviously too general. But it turns out that many bad matches from a too general pattern are more useful than no good matches from a too specific pattern. Using this general pattern, the system tentatively identifies all noun phrases in the text as Innovations.

Applying Feedback. These matches are presented to the user for evaluation. Most of the NPs are not innovations, and the user accordingly marks them incorrect, providing the system with very useful feedback. Each of the incorrect matches can be used to specialize the general pattern. A good specialization of the innovation pattern is

if X is a Noun Phrase constituent where
  head(X) = "circle"
then
  extract X.

The I^2E^2 will continue to refine the innovation pattern to maximize its performance. Our initial approximation to "maximum performance" is that the pattern matches no negative instances. (We note here that this criterion will tend to overfit each pattern; but we postpone discussion until Section 5.3.) Then the user may either select more Innovation examples to work on until the set of innovation patterns covers the example, or proceed to training for other annotation types.

3 Partial Parsing

The text analysis component of the interactive information extraction environment uses a *partial-parsing* methodology. Partial parsing aims at reliably recognizing unambiguous syntactic elements, rather than producing complete parses. Our partial parser is similar to the *finite-state transducer cascade* architecture popularized by Abney (1996). The finite-state cascade builds up syntactic structure in a series of levels, such as in Figure 3. Each level is formed from the previous levels by a transducer, which is a finite-state automaton for mapping input languages to output languages. Abney relates a number of advantages of partial parsing using finite-state cascades.

- The finite-state cascade is faster than traditional parsers, because there is no global optimization. This also contributes to robustness, since correct phrases never have to fit into a globally optimized scheme.
- The grammar is considered to be a programming language for recognizers rather than a linguistic description. Thus we write patterns to recognize reliable structure, even when the structure is not linguistically motivated—as is the case for patterns for phrase "boundaries" and "kernels."
- Patterns are reliable because the partial-parsing philosophy is to make easy decisions first and delay difficult decisions until they can be made with high precision.
- Also because of the partial-parsing philosophy, ambiguity in lower level decisions is contained by reliable high-level structure. For example, prepositional phrase attachments are difficult, but clause boundaries limit the possible attachment sites.

The finite-state cascade is an efficient architecture for building parsers. However, each level transducer must be specified as a pattern for mapping input to output. The patterns may be constructed by hand, but this places the now-familiar knowledge engineering burden upon the parser builder. A more attractive approach is to *learn* the parser much as we have described learning extraction patterns above. Fortunately, because of the existence of parsed corpora, learning syntax patterns need not burden the system user. Instead, corpus-based methods can learn syntax from the pre-parsed texts.

Thus, pattern learning is fundamental to our framework of language processing. It should not be surprising then, that the most important component of this research focuses on machine learning of patterns for information extraction.

Level 3	An NH circle was used {to solve problems with CAD} SUBJECT VERB COMPLEMENT	Clauses
Level 2	[An NH circle] was used to solve [problems] with [CAD] .	Base noun phrases
Level 1	An NH circle was used to solve problems with CAD . DT NNP NN VBZ VBD TO VB NNS IN NNP .	Part-of-speech tags
Level 0	An NH circle was used to solve problems with CAD .	Words

Figure 3: Syntactic levels in a finite-state cascade.

4 Pattern Learning

4.1 Syntax

Although it is not the primary focus of this proposal, syntactic pattern learning is important to our language processing framework. As noted above, *corpus-based* methods are an attractive approach to learning syntactic patterns. In this approach, the patterns are learned in various ways from a large corpus of pre-parsed text.

Our previous work on learning patterns for base noun phrases (Cardie and Pierce, 1998) exemplifies this approach. The algorithm learns the patterns from a corpus annotated with base noun phrases derived from the Penn Treebank parsed text. The patterns are sequences of part-of-speech tags. The pattern ⟨DT NNP NN⟩, for example, matches a sequence of words tagged as determiner, proper noun, singular common noun; an example would be the phrase *an NH circle*. The algorithm first tags the text with a part-of-speech tagger, then scans the text from left to right, at each word finding the longest noun phrase matching one of the patterns, bracketing it, and moving to the word following it. The set of base noun phrase patterns is initially generated by recording every tag sequence from the noun phrases in the annotated corpus. The pattern set is then pruned to improve its performance, by removing patterns that behave badly when checked against a held-out portion of the annotated corpus. The set of patterns can be evaluated against yet another held-out portion of the corpus. Upon evaluation we found that the algorithm performed comparably to current best methods for noun phrase bracketing.

4.2 Extraction

4.2.1 Related Work

Previous research has explored the learning of extraction patterns, using the corpus-based paradigm described above for syntactic patterns: for extraction patterns the corpus must be annotated with examples of the desired extraction from which the learning algorithm can induce the patterns.

The AutoSlog system (Riloff, 1993) uses the simplest learning algorithm. From each corpus example, AutoSlog derives one pattern based on the syntactic position of the instance and a “trigger” (usually the verb of the clause). AutoSlog performs no automatic learning or filtering on these patterns. Instead, a human must filter the entire set of patterns, throwing away patterns that look wrong.

AutoSlog-TS (Riloff, 1996) extends AutoSlog so that it does not require annotation of the entire corpus. Instead, it suffices to classify each document as relevant or irrelevant to the domain. AutoSlog-TS forms patterns just like AutoSlog, except that it forms *all* possible patterns from the training corpus (from the syntactic position of each noun phrase), instead of forming patterns only for the training instances (because there are none). These “anonymous” patterns are then ranked and filtered according to their relevancy rate, the number of occurrences in relevant documents versus number of occurrences in irrelevant documents. This filtering is effective, but a human is still required to accept the interesting patterns and label the information they extract.

The Crystal system (Soderland *et al.*, 1995) uses a specific-to-general machine learning algorithm to build a dictionary of patterns. As with AutoSlog, the dictionary is initialized with one pattern for each concept instance. Each pattern is maximally specific, encoding the entire context of the instance. Crystal generalizes each pattern by selecting the most similar pattern for the same concept and unifying the two patterns. Unification involves dropping pattern constraints until the two patterns are the same. By unifying similar patterns Crystal avoids having to choose from the combinatorial variety of pattern generalizations.

ANNOTATEINSTANCES
  allow the user to annotate an example e
  add e to N and to E^+

JUDGEINSTANCES
  allow the user to select e from $E^?$
  if the user marks e correct
    add e to E^+
  else
    add e to E^-

CREATEPATTERN
  remove an example e from N
  s = the most specific pattern from e
  g = the most general pattern from e
  add g to P

PROPOSEINSTANCES
  select p from P
  add new instances $p(T)$ to $E^?$

REFINEPATTERN
  select p from P
    with s the corresponding specific pattern
  unless p is already performing at its best
    specialize p in the direction of s

Figure 4: Algorithm for learning extraction patterns from user-supplied instances.

The LIEP system (Huffman, 1996) also uses a specific-to-general learning algorithm. It learns a new pattern for every example in the training corpus, unless a previously learned pattern matches the example or can be generalized to match it. LIEP generalizes a pattern by fixing a "non-generalizable portion" of the pattern; then, if a new example matches the non-generalizable part of the pattern, the pattern is generalized to fit the new example. In practice this means that patterns are generalized by adding more word choices to head constraints.

These systems have laid important groundwork, which I build upon in several directions. First, each of these systems extracts only noun phrases as concept slot fillers. Many IE tasks, including our example task, will need to extract arbitrary constituents. This research extends the pattern learning task to include patterns for phrases of other types and multiple phrases in syntactic relationships. Furthermore, the needs of a non-expert user are not considered in the design of these systems. LIEP is the most potentially interactive of the systems, but it does not take full advantage of the power of user interaction. It is interactive only in the sense that the learning algorithm is incremental, so that a user could supply examples one at a time. But the user must still supply all the examples. On the other hand, I^2E^2 takes advantage of the parser's ability to quickly find instances of a candidate pattern, the user's ability to judge their relevancy, and the pattern learner's ability to adapt the candidate pattern to conform to the user's wishes.

4.2.2 An Interactive Learning Algorithm

Given the shape of the I^2E^2 task, let's consider the design of the pattern learning component. Since the system needs to learn from examples, we need an algorithm to search the space of possible patterns. The systems described above search from specific patterns toward general patterns, because additional (positive) examples can be used to generalize a pattern. The I^2E^2, on the other hand, searches from general to specific, enabling it to identify candidate instances of the concept. This is because a general pattern can be used to propose many new instances. New instances that are correct save the user the cost of annotation; while the pattern learner can use new instances that are incorrect to specialize a general pattern.

The learning component also needs heuristics to guide the specialization process; otherwise the exponential number of possible patterns will cripple the search. Crystal guides its specific-to-general search by finding similar patterns to unify. LIEP severely limits the allowed generalization of its patterns, so that similar example instances can be found to guide generalization. Searching from general-to-specific, the I^2E^2 needs different heuristics. One plausible heuristic is to guide the general-to-specific search in the direction of the original instance used to form the pattern.

With these these learning criteria in mind, Figure 4 presents an appropriate high-level learning algorithm. The algorithm interleaves five procedures: the user annotates new concept instances; the pattern learner creates a pattern from an instance; the parser proposes new instances from pattern matches; the user judges the candidate instances; and the learner refines the affected patterns. The procedures in the left column interact with the user, while those on the right involve the learner and the parser. The procedures share some data structures: T, the training corpus; N, the set of new instances provided by the user; E^+, E^-, and $E^?$, the sets of positive, negative, and candidate (yet to be judged) instances (N overlaps with E^+); and P, the set of patterns. When p is a pattern and S is a corpus or a set of instances, $p(S)$ denotes the instances found in S that match p.

The condition whereby we cease to refine a pattern is deliberately left vague ("p is already performing at its best"). In Section 2 we proposed one criterion for best performance: that p does not make any mistakes, or using the notation here, that $p(E^-) = \emptyset$. However, this naive condition is not robust when the training data is noisy: it will overspecialize p to fit the original training instance described by s. Considering that the training data is provided by an ordinary user rather than a linguist, it is imperative that the learner be robust, so a more flexible condition may be required. This issue is discussed in Section 5.3.

5 Research Plan

5.1 Next Steps

Both of the crucial $\mathrm{I}^2\mathrm{E}^2$ components, the partial parser and the pattern learner, need further development. On the parsing side, we have introduced a new algorithm for bracketing base noun phrases, as described in Section 4. Now we need to develop the next stages of the parser, for identifying other syntactic features such as clauses, argument structure, and coreference, which will be needed later for use in extraction patterns. One of the nice aspects about this research is that the interactive learning algorithm is general and can be applied to any of these parsing tasks as well as to learning extraction patterns. So we could build up the parser by, for example, interactively learning patterns for argument structure using the tagging and base NPs we already have. By experimenting with some simple structures, we can gain experience with the framework to prepare for the larger challenge of users' annotations. Thus, part of the research will focus on adding interactive learning to our toolbox of corpus-based methods for building NLP systems.

On the extraction side, we need to validate the accuracy of the interactive IE system and demonstrate that it can be used to annotate a wider variety of constructions than previous IE learning algorithms. To validate the $\mathrm{I}^2\mathrm{E}^2$, we will apply it to one of the existing simple noun phrase extraction tasks, say the terrorism task (MUC-3, 1991), and compare its performance to that of the UMass Circus system using patterns learned with AutoSlog. The MUC-3 task also provides a limited opportunity to learn extractions other than noun phrases—in particular, some of the "set fills," or template slots that must be chosen from a fixed set of answers. For example, one of the set fills from the terrorism templates is a rating of confidence in the incident report; answers may be: Reported as Fact; Acquitted; Claimed or Admitted; Suspected or Accused. Previous systems have had difficulty finding these set fills. So this task is a relatively simple opportunity to try a different type of extraction for which we have well-defined and readily available answers.

5.2 Evaluation

The existence of corpora for parsing and information extraction expedites the evaluation of the early stages of this work. The parsing component can be evaluated with respect to treebank parses; and with a little effort, the noun phrase extractions from the templates of the MUC tasks could be converted to corpora of annotations for evaluation of the learning component. However, there are no corpora available for the more general task of imitating arbitrary annotations. In order to evaluate the final system, we will need to choose another task and construct a test corpus by hand.

Corpus-based evaluation is informative about the accuracy of the IE systems that we build, but not about the effectiveness of the $\mathrm{I}^2\mathrm{E}^2$ for the user. One way to measure the effectiveness of interaction is to run experiments with real users to determine how long it takes them to train an IE system to a certain level of accuracy. Another interesting alternative is to simulate users, using a hand-annotated corpus and a user model that selects annotations from this corpus to use for training. We can measure the amount of interaction (e.g. number of instances annotated and candidates judged) necessary for the psuedo-user to train the system. One advantage of this approach is that we can formalize various behaviors in the user model and measure their effect on training quickly and easily.

With these user or pseudo-user interactions arises the question of what a reasonable amount of interaction is. This will of course be subjective. Certainly the I^2E^2 will be a success if a user can train an IE system in about the amount of time needed to weed the set of extraction patterns produced by AutoSlog (about 8 hours). Arguably, it will even be a success if a user can train a reasonably complicated system in a matter of a few days.

Using the methods of evaluation outlined above, the progress of this research can be measured. Our ultimate goals, then, will be:

1. To demonstrate, if possible, that users' annotations can be imitated or reproduced with reasonable accuracy.

2. To demonstrate, if possible, that a reasonable amount of interactive learning can be used to produce an IE system with accuracy comparable to other leading systems such as UMass' Circus system with AutoSlog.

5.3 Key Research Issues

This section considers some of the most important issues faced by the proposed framework of interactive information extraction.

- The I^2E^2 relies upon the consistency of the users' annotations. It seems plausible to suppose that in the face of inconsistent annotation, the performance of learned patterns will degrade significantly. On the other hand, with the use of the I^2E^2, users are in a much better position to monitor the consistency of their corpora. When the system finds that a previously learned pattern begins to make an unexpected number of errors, it may notify the user about the pattern and previously matched instances. This allows the user and the system to adapt to, or reject, variation in the annotation style.

- The I^2E^2 is a learning framework that is independent of the "features" used in learned patterns. For example, the base noun phrase patterns use only the word and part-of-speech features for each lexical item. Higher-level extraction patterns will use syntactic and semantic features. The performance of the IE system will depend on the available features, so one research issue is to identify and provide important features of the text.

- This leads to another issue. When provided with a bag full of available features, how do we know which ones are crucial to the patterns we need to learn? In our framework, this is a question about specializing patterns. When specializing a general pattern, we need to add the features that are most useful. How to do this is perhaps the most important research question.

- The discussion about specializing patterns in Section 4 hinted at the problem of overspecialization. The naive stopping condition $p(E^-) \neq \emptyset$ dictates that the system specialize every pattern that makes a mistake. But many users are willing to sacrifice a little accuracy to save training time. That is, instead of a pattern that creates 5 correct annotations (for 100% accuracy), they might prefer a pattern that creates the same 5 annotations plus 90 other correct annotations, but makes 5 additional mistakes (for 95% accuracy). This calls for a more flexible "usefulness" measure that incorporates both accuracy and coverage.

Acknowledgments

This work was supported in part by NSF Grants IRI–9624639 and GER–9454149.

References

(Abney, 1996) Abney, S. 1996. Partial Parsing via Finite-State Cascades. In *Proceedings of the ESSLLI'96 Robust Parsing Workshop.*

(Cardie and Pierce, 1998) Cardie, C. and Pierce, D. 1998. Error-Driven Pruning of Treebank Grammars for Base Noun Phrase Identification. In *Proceedings of COLING-ACL98.* 218–224. Available as cmp-lg/9808015.

(Cardie, 1993) Cardie, C. 1993. A Case-Based Approach to Knowledge Acquisition for Domain-Specific Sentence Analysis. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*.

(Cardie, 1997) Cardie, C. 1997. Empirical Methods in Information Extraction. *AI Magazine* 18(4):65–79.

(Grishman, 1997) Grishman, R. 1997. TIPSTER Text Phase II Architecture Design 2.3. Technical report, TIPSTER. Available at http://www.tipster.org/.

(Huffman, 1996) Huffman, S. 1996. Learning Information Extraction Patterns from Examples. In Wermter, S.; Riloff, E.; and Scheler, G., editors, *Symbolic, Connectionist, and Statistical Approaches to Learning for Natural Language Processing.* Springer. 246–260.

(MUC-3, 1991) MUC-3, 1991. *Proceedings of the Third Message Understanding Conference (MUC-3).* Morgan Kaufmann.

(Riloff, 1993) Riloff, E. 1993. Automatically Constructing a Dictionary for Information Extraction Tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*. 811–816.

(Riloff, 1996) Riloff, E. 1996. Automatically Generating Extraction Patterns from Untagged Text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume 2. 1044–1049.

(Soderland *et al.*, 1995) Soderland, S.; Fisher, D.; Aseltine, J.; and Lehnert, W. 1995. CRYSTAL: Inducing a Conceptual Dictionary. In *Proceedings of the Fourteenth Joint Conference on Artificial Intelligence.* 1314–1319.