Appears in the *Proceedings of the* 18^{th} *International Conference on Machine Learning (ICML 2001).*

A Theory-Refinement Approach to Information Extraction

Tina Eliassi-Rad ELIASSI@CS.WISC.EDU
Jude Shavlik SHAVLIK@CS.WISC.EDU
Computer Sciences Department, University of Wisconsin, 1210 West Dayton Street, Madison, WI 53706 USA

Abstract

We investigate applying theory refinement to the task of extracting information from text. In theory refinement, partial domain knowledge (which may be incorrect) is given to a supervised learner. The provided knowledge guides the learner in its task, but the learner can refine or even discard this knowledge during training. Our supervised learner is a "knowledge-based" neural network that initially contains "compiled" prior knowledge about a particular information extraction (IE) task. The prior knowledge needs to specify the extraction slots for the specific IE task. Our approach uses *generate-and-test* to address the IE task. In the *generation* step, we produce candidate extractions by intelligently searching the space of possible extractions. In the *test* step, we use the trained network to judge each candidate and output those that exceed a system-selected threshold. Experiments on the CMU seminar-announcements and the Yeast subcellular-localization domains demonstrate our approach's value.

1. Introduction

The rapid growth of on-line information has created a surge of interest in tools that are able to extract information from on-line documents. Information extraction (IE) is the process of pulling desired pieces of information out of a document.

Unfortunately, building an IE system requires either a large number of annotated examples[1] or an expert to provide sufficient (and correct) knowledge about the domain of interest. Both of these requirements make it time-consuming and difficult to build an IE system. In this paper, we demonstrate how the theory-refinement approach (*e.g.*, Towell & Shavlik, 1994) can be used to build an IE system. By using theory refinement, we are able to strike a balance between needing a large number of labeled examples and having a complete (and correct) set of domain knowledge.

Our system takes advantage of the intuition that information retrieval (IR) and IE are nearly inverse problems of each other. An IR system is given a set of keywords and is asked to rate the relevance of documents. An IE system is given a set of documents and is asked to fill in the slots in a given template. We explore how what is essentially an IR system can be used to address the IE task.

Our system is called WAWA-IE.[2] The basic WAWA system has been described previously (Shavlik & Eliassi-Rad, 1998; Shavlik et al., 1999) and we will only briefly review it here.

The user provides a set of instructions in the form of IF-THEN statements to WAWA-IE. These instructions describe how the system should score possible bindings to the slots being filled during the the IE process. (We will call the names of the slots to be filled *variables*, and use "binding a variable" as a synonym for "filling a slot.") These initial instructions are then "compiled" (Towell & Shavlik, 1994) into a neural network (called *ScorePage*), which rates the goodness of a document in the context of the given variable bindings.

We refer to the user-provided instructions as *advice* to emphasize that our system does not blindly follow the user's instructions, but instead refines them based on the training examples. The use of user-provided advice typically leads to higher accuracy from fewer user-provided training examples.

WAWA-IE uses a *generate-and-test* approach to extract information. In the *generate* step, the user first specifies the slots to be filled (along with their part-of-speech tags or parse structures), and WAWA-IE gen-

[1] By annotated examples, we mean the result of the tedious process of reading the training documents and tagging each extraction by hand.

[2] WAWA-IE is the extraction subsystem of a larger system called WAWA (*Wisconsin Adaptive Web Assistant*).

erates a large list of candidates from the document. In the *test* step, WAWA-IE scores each possible candidate. The candidates that produce scores that are greater than a system-defined threshold are returned as the extracted information. A critical component of WAWA-IE is an intelligent selector (explained later) that eliminates the need to create an exhaustive list of all possible candidate bindings.

The key aspect of WAWA-IE is its advice language. An advice sentence is a conditional statement whose antecedent describes a property of the page (*e.g.*, whether whether the word "fly" appears as a verb on the page) and whose consequent specifies an estimate of the page's relevancy to the domain of interest.

As already mentioned, of particular relevance to our approach is the fact that WAWA-IE's advice language contains *variables*. To understand how WAWA-IE uses variables, assume that we want to extract speaker names from a collection of seminar announcements. We might wish to give such a system some (good) advice like: *If the page contains the phrase "Speaker . ?FirstName/NNP ?LastName/NNP", then score this page highly.* The leading question marks indicate slots to be filled, and '**.**' matches any single word. Also, the advice language allows the user to specify the required part of speech for a slot (e.g., *NNP* denotes a proper noun). The precondition of our example rule matches phrases like "*Speaker is Joe Smith*".

Figure 1 illustrates an example of extracting speaker names from a seminar announcement using WAWA-IE. The announcement is fed to the candidate generator and selector, which produces a list of speaker candidates. Each entry in the candidates list is then bound to the variables in advice.[3] The output of the (trained) network is a real number that represents our confidence in the speaker candidate being a correct slot filler for the given document.

2. The WAWA-IE System

We require the user to provide the following information to WAWA-IE:

1. The set of on-line documents from which the information is to be extracted.

2. The extraction slots like speaker names, etc.

3. The possible part-of-speech (POS) tags (*e.g.*, noun, proper noun, verb, etc) or the parse structures (*e.g.*, noun phrase, verb phrase, etc) for each extraction slot.

[3]The variable *?Speaker* is a record that holds multiple variables such as *?FirstName* and *?LastName*.

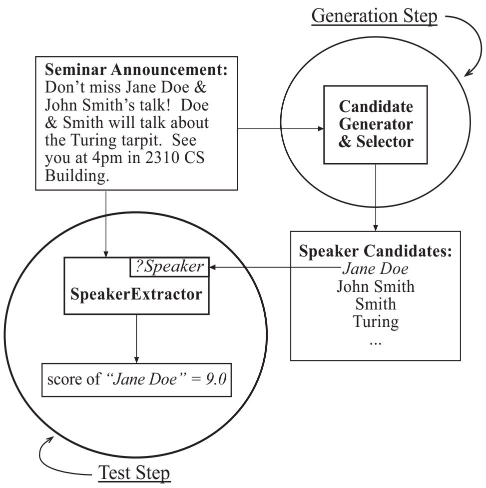

Figure 1. Extraction of speaker names with WAWA-IE

4. A set of advice rules containing variables which refer to the extraction slots.

5. A set of annotated examples, *i.e.*, training documents in which extraction slots have been marked.

Actually, the user does not have to explicitly provide the extraction slots and their POS tags separately from advice since they can be extracted from the advice rules. For example, in one of our case studies, we want to extract names of proteins and their subcellular locations from the yeast database of Ray and Craven (2001). One of our advice rules for this task is: *When the phrase "?ProteinName/Nphrase ./Vphrase ?LocationName/Nphrase" appears in the document, then score it very high.*

The variables *?ProteinName* and *?LocationName* represent the protein names and their subcellular structures. The "/Nphrase" trailing the variables indicates the required parse structure of the variables ("Nphrase" refers to a noun phrase). The "./Vphrase" matches any verb phrase. The precondition of this rule matches phrases such as "UBC6 localizes to the endoplasmic reticulum."

During training, WAWA-IE first compiles the user's advice into the SCOREPAGE network. WAWA-IE next uses what we call an *individual-slot candidate generator* and a *combination-slots candidate selector* to create training examples for the SCOREPAGE network. The same candidate generation and selection process is used after training to generate the possible extractions that the trained network scores.

2.1 Individual-Slot Candidate Generator

The first step WAWA-IE takes (both during training and after) is to generate all possible *individual* fillers for each slot on a given document. Fillers can be individual words or phrases.

Individual words are collected by using Brill's (1994) tagger to annotate each word in a document with its POS. For each slot, we collect every word in the document that has a POS tag that matches a tag assigned to this variable somewhere in the IE task's advice.

For cases where a variable is associated with a phrase, we apply a sentence analyzer called Sundance (Riloff, 1998) to each document. Sundance builds a shallow parse tree that segments sentences into noun, verb, or prepositional phrases. We collect those phrases that match the parse structure for the extraction slot and also generate all possible subphrases of consecutive words (since Sundance only does shallow parsing).

At this point we typically have lengthy lists of candidate fillers for each slot, and we need to focus on generating good *combinations* that fill *all* the slots. Obviously, this process can be combinatorially demanding. In the next section, we present and evaluate a heuristic method for choosing good combinations.

2.2 Combination-Slots Candidate Selector

Wawa-IE contains several methods for creating complete assignments to the slots from the lists of individual slot bindings:

Exhaustive Candidate Selector: Exhaustively produce the cross-product of all entries in the lists of individual-slot candidates for a given document. For some tasks, there may be sufficient computational resources to do this (also, it provides a "control" when we later evaluate the other options).

High-Scoring Simple Random Selection (SRS): Randomly select combination-slots candidates from the lists of individual-slot candidates. When training, only use the N combinations that produce the highest scores on the *untrained* ScorePage network.[4] This is mainly a control against which we compare the following "intelligent" selector of combined bindings.

Modified WSAT: Use a modified version of the WalkSAT algorithm (Selman et al., 1996). Figure 2 describes this algorithm.

For a given document, the algorithm starts with an empty list of combination-slots candidates. For each extraction slot, it then iteratively and randomly selects an item from that slot's list of individual-slot candidates. This produces a combination-slots candidate extraction that contains a candidate filler for each slot in the template. If the score produced by the (possibly untrained) ScorePage network is high enough (*i.e.*, over 9 on a -10 to 10 scale) for this set of variable bindings, then add this combination to the list of combination-slots candidates. Otherwise, repeatedly and randomly select a slot in the template. With probability p, randomly select a candidate for the selected slot and add the resulting combination-slots candidate to the list of combination-slots candidates. With probability 1-p, iterate over *all* possible candidates for this slot and choose the candidate that produces the highest network score for the document. The resulting combination-slots candidate is then added to the list of combination-slots candidates.

[4]By untrained, we mean a network containing only compiled (initial) advice and without any further training via backpropagation and labeled examples.

Inputs: MAX-TRIES, MAX-ALTERATIONS, p, MAX-CANDS, *doc*, L (where L is the lists of individual-slot candidate extractions for *doc*)

Output: *TL* (where *TL* is the list of combination-slots candidate extractions of size MAX-CANDS)

Algorithm:

1. *TL* := { }
2. **for** i:=1 **to** MAX-TRIES
   S := randomly selected combination-slots candidate from L.
   if (score of S w.r.t. *doc* is in [9.0, 10.0]), **then** add S to *TL*.
   otherwise,
      for j:=1 **to** MAX-ALTERATIONS
         s := Randomly select a slot in S to change
         With probability p, randomly select a candidate for s.
            Add S to *TL*.
         With probability 1-p, select a candidate for s that
            maximizes the score of S w.r.t. *doc*. Add S to *TL*.
3. **Sort** *TL* in decreasing order of score of its entries.
4. **Return** the top MAX-CANDS entries as *TL*.

Figure 2. Our Modified WalkSAT Algorithm

2.3 Training an IE agent

Figure 3 shows the process of building a trained IE agent. Since (usually) only positive training examples are provided in IE domains, we first need to generate some negative training examples. To this end, we use the candidate generator and selector described above. The list of negative training examples collected by the selector contains informative negative examples (*i.e.*, near misses) because the heuristic search used in the selector scores the training documents on the untrained ScorePage network. That is, the (user-provided) prior knowledge scored these "near miss" extractions highly (as if they were true extractions).

After the negative examples are collected, we train the

ScorePage neural network using these negative examples and all the provided positive examples. By training the network to recognize (i.e., produce a high output score for) a correct extraction in the context of the document as a whole (Shavlik et al., 1999), we are able to take advantage of the global layout of the information available in the documents of interest.

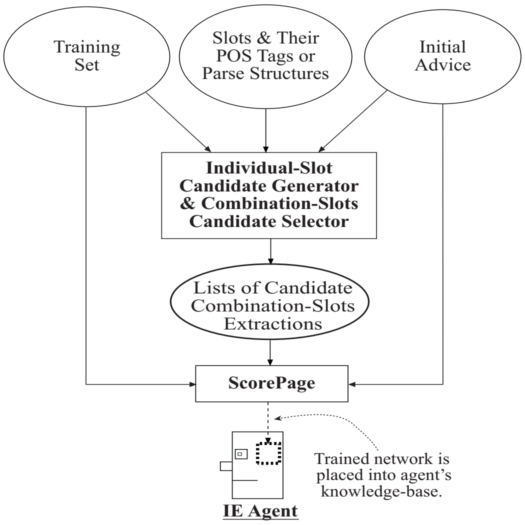

Figure 3. Building a Trained IE agent

Since the ScorePage network outputs a real number, Wawa-IE needs to define a threshold on this output such that the bindings for the scores above the threshold are returned to the user as extractions and the rest are discarded. Note that the value of the threshold can be used to manipulate the performance of the IE agent. For example, if the threshold is set to a high number (*e.g.*, 8.5), then the agent might miss a lot of the correct fillers for a slot (*i.e.*, have low *recall*), but the number of correct fillers it extracts should be higher (*i.e.*, high *precision*). *Recall* (van Rijsbergen, 1979) is the ratio of the number of correct fillers extracted to the total number of fillers in correct extraction slots. *Precision* (van Rijsbergen, 1979) is the ratio of the number of correct fillers extracted to the total number of fillers extracted. The commonly used F_1-*measure* combines precision and recall using the following formula: $F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$.

To avoid overfitting the ScorePage network and to find the best threshold on its output after training is done, we actually divide the training set into two disjoint sets. One of the sets is used to train the ScorePage network. The other set, the *validation* set, is first used to "stop" the training of the ScorePage network. Specifically, we cycle through the training examples 100 times. After each iteration over the training examples, we use the lists of combination-slots candidates associated with the validation set to evaluate the F_1-*measure* produced by the network for various settings of the threshold. We pick the network that produced the highest F_1-measure on our validation set as our final trained network and use the associated validation-set threshold when processing subsequent (e.g., "test set") examples.

2.4 Testing a Trained IE agent

Figure 4 depicts the steps a trained IE agent takes to produce extractions.

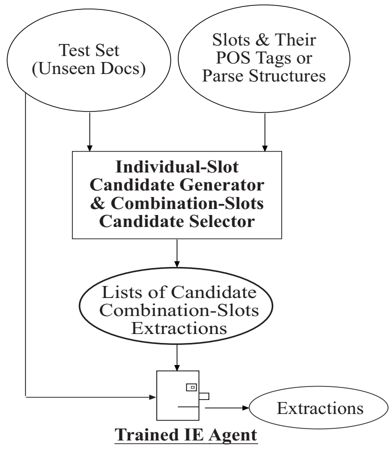

Figure 4. Testing a Trained IE Agent

For each entry in the list of combination-slots extraction candidates, we first bind the variables to their candidate values. Then, we perform a forward propagation on the trained ScorePage network and output the score of the network for the test document based on the candidate bindings. If the output value of the network is greater than the threshold defined during the validation step, we record the bindings as an extraction. Otherwise, these bindings are discarded.

2.5 Discussion

A novel aspect of Wawa-IE is its exploitation of the relationship between IR and IE. That is, we build IR agents that treat possible extractions as keywords, which are in turn judged within the context of the entire document.

The use of theory refinement allows us to take advantage of user's prior knowledge, which need not be perfectly correct since Wawa-IE is a learning system. This, in turn, reduces the need for labeled examples, which are very expensive to get in the IE task. Also, compiling users' prior knowledge into the ScorePage network provides a good method for finding informative negative training examples (*i.e.*, near misses).

One cost of using our approach is that we require the user to provide us with the POS tags or parse structures of the extraction slots. We currently assume that the Brill's tagger and Sundance are perfect (*i.e.*, they tag words and parse sentences with 100% accuracy). Brill's tagger annotates the words on a document with 97.2% accuracy (Brill, 1994), so 2.9% error rate propagates into our results. We were not able to find accuracy estimates for Sundance, though do remember that we also consider all subphrases of the phrases Sundance produces.

Our approach is computationally demanding, due to its use of a generate-and-test approach. But, CPU cycles are abundant; and, as our second experiment below shows, WAWA-IE still performs well when only using a subset of all possible combinations of slot fillers.

3. Experimental Evaluation

We evaluate our approach using two IE tasks. The first involves extracting speaker and location names from a collection of seminar announcements. This task has been widely used in the literature and allows us to directly compare the performance of WAWA-IE to several existing systems. For this domain, we follow existing methodology and independently extract speaker and location names, because each document is assumed to contain only one announcement. That is, for each announcement, we do not try to pair up speakers and locations, instead we return a list of speakers and a separate list of locations. Hence, we do not use any "combination slots selector" on this task.

The second IE task involves extracting protein names and their locations within the cell from a collection of abstracts from biomedical articles on yeast. We chose this domain because it illustrates a harder IE task than the first domain. In our second domain, the fillers for extraction slots depend on each other because a single abstract can contain multiple proteins and locations. Hence, for each document, a single list of <*protein*, *location*> pairs is extracted.

3.1 Seminar Announcements Domain

In our first experiment, we compare WAWA-IE to five other information extraction systems using the CMU seminar announcements domain (Freitag, 1998). These systems are SRV (Freitag, 1998), Naive Bayes (Freitag, 1998), WHISK (Soderland, 1999), RAPIER (Califf, 1998), and RAPIER-WT (Califf, 1998). Except for Naive Bayes, they are all relational learning algorithms that do not exploit prior knowledge.

Freitag (1998) first randomly divided the 485 documents in the seminar announcements domain into ten splits, and then randomly divided each of the ten splits into approximately 240 training examples and 240 testing examples. Except for WHISK, the results of the other systems are all based on the same 10 data splits; the results for WHISK are from a single trial with 285 documents in the training set and 200 documents in the testing set.

We give WAWA-IE 9 and 10 advice rules in *Backus Naur Form* (BNF) (Aho et al., 1986) notation about speakers and locations, respectively.[5] None of the advice rules are written with the specifics of the CMU seminar announcements in mind. The rules describe our prior knowledge about what might be a speaker or a location in a *general* seminar announcement. It took us about half a day to write these rules and we did not *manually* refine these rules over time.

For this domain, we create the same number of negative training examples (for speaker and location independently) as the number of positive examples. We choose 95% of the negatives, from the complete list of possibilities, by collecting those that score the highest on the untrained SCOREPAGE network; the remaining 5% are chosen randomly from the complete list.

Tables 1 and 2 show the results of our system and the other five systems for the speaker and location slots, respectively. The results reported are averages across the ten splits. The precision, recall, and F_1-measure for a split is determined by the optimal threshold found for that split using the validation set.

Remember that *precision* (P) is the ratio of the number of correct fillers extracted to the total number of fillers extracted, and *recall* (R) is the ratio of the number of correct fillers extracted to the total number of fillers in correct extraction slots. An ideal system has a precision and recall of 100%. Finally, the commonly used F_1*-measure* combines precision and recall using the following formula: $F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$.

Since the speaker's name and the location of the seminar may appear in multiple forms in an announcement, an extraction is considered correct as long as any one of the possible correct forms is extracted. For example, if the speaker is "John Doe Smith", the words "Smith", "Joe Smith", "John Doe Smith", "J. Smith", and "J. D. Smith" might appear in a document. Any one of these extractions is considered correct. This method of marking correct extractions is also used in the other IE systems against which we compare our approach.

[5]The complete list of these advice rules will appear in Eliassi-Rad (2001).

Table 1. Speaker slot results for seminar announcements

System	P	R	F_1
WAWA-IE's Trained Agent	61.5	86.6	71.8
SRV	60.4	58.3	59.3
RAPIER-WT	79.0	40.0	53.1
RAPIER	80.9	39.4	53.0
WAWA-IE's Untrained Agent	29.5	96.8	45.2
Naive Bayes	36.1	25.6	30.0
WHISK	71.0	15.0	24.8

Table 2. Location slot results for seminar announcements

System	P	R	F_1
WAWA-IE's Trained Agent	73.9	84.4	78.8
RAPIER-WT	91.0	61.5	73.4
SRV	75.9	70.1	72.9
RAPIER	91.0	60.5	72.7
WHISK	93.0	59.0	72.2
RAPIER-W	90.0	54.8	68.1
Naive Bayes	59.6	58.8	59.2
WAWA-IE's Untrained Agent	29.2	98.2	45.0

For both speaker and location slots, the F_1-measure of our system is the highest. Our F_1-measures are high because we generate many extraction candidates; hence, we are able to extract a lot of the correct fillers from the data set, which leads to higher recall than the other systems. After training, we are able to reject enough candidates that we obtain reasonable precision. However, several of the other systems have higher precision, so depending on the user's tradeoff between recall and precision, different systems would be preferred on this testbed.

The increase in performance from WAWA-IE's untrained agent[4] to WAWA-IE's trained agent shows that our system is not "hard-wired" to perform well on this domain and that training helped our performance.

For this first experiment, we train one network for finding the speaker slots and another one to find the location slots. In the next section, we discuss an experiment where we train only one network to extract fillers for two slots simultaneously.

3.2 Yeast Protein-Localization Domain

For our second experiment, the task is to extract protein names and their subcellular localization from the yeast protein-localization database produced by Ray and Craven (2001). They call their extraction template the *subcellular-localization relation* and created their *yeast* database by first collecting target instances of the subcellular-localization relation from the *Yeast Protein Database* (YPD) Web site. Then, they collected abstracts from articles in the *MEDLINE* database (National Library of Medicine, 2001) that have references to the entries selected from YPD. We followed Ray and Craven's methodology for our experiments on this domain and used their "tuple-level" method for measuring accuracy (where a tuple is an instance of the subcellular-localization relation).

In this yeast data set, each training and test instance is an individual sentence. A positive sentence is labeled with target tuples. There are 545 positive sentences containing 645 tuples, of which 335 are unique. A negative sentence is not labeled with any tuples. There are 6,700 negative sentences in this data set. Note that a sentence that does not contain both a protein and its subcellular location is considered to be negative.

WAWA-IE is given 12 advice rules in BNF (Aho et al., 1986) notation about a protein and its subcellular location.[5] None of the advice rules are written with the specifics of the yeast data set in mind; the rules describe our prior knowledge about what might be an instance of the subcellular-localization relation. It took us about half a day to write these rules and we did not *manually* refine these rules over time.

Ray and Craven (2001) split the yeast data set into five disjoint sets and ran 5-fold cross-validation. We use the same folds with WAWA-IE and compare our results to theirs. But first we investigate whether or not we can intelligently select good training examples (and, hence, reduce training time) by comparing test set F_1-measure to the case where we use *all* possible negative examples.

Figure 5 illustrates the difference in F_1-measure between Section 2.2's modified WalkSAT selector, "high-scoring SRS," and exhaustive candidate selector. The horizontal axis depicts the percentage of negative training examples used during the learning process, and the vertical axis depicts the F_1-measure of the trained IE-agent. WAWA-IE is able to achieve very good performance by using less than 20% of the negative training candidates. The "high-scoring SRS" selector does much worse than our modified WalkSAT.

In F_1-measures, WAWA-IE's trained agents outperform the untrained agents[4] by approximately 50% (results not shown). This further demonstrates that WAWA-IE is able to refine initial advice.

Figure 6 shows the precision and recall curves for (a) WAWA-IE's trained agent with the modified WalkSAT selector (using 17% of the negative examples), (b) WAWA-IE's trained agent without a selector (*i.e.*,

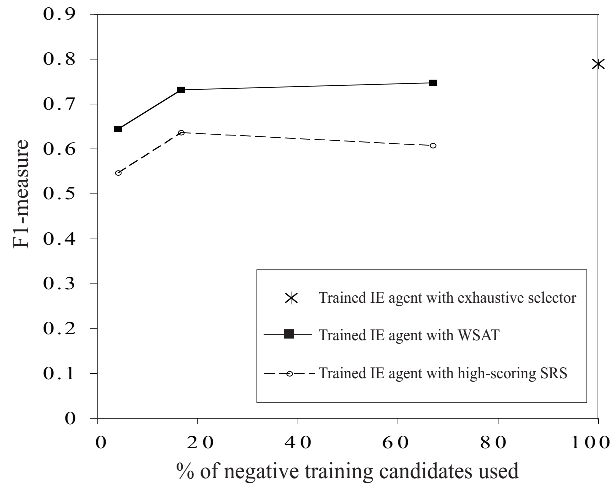

Figure 5. F_1-measure vs. percentage of negative training candidates used for different selector algorithms

using all the negative training examples), and (c) the system of Ray and Craven.

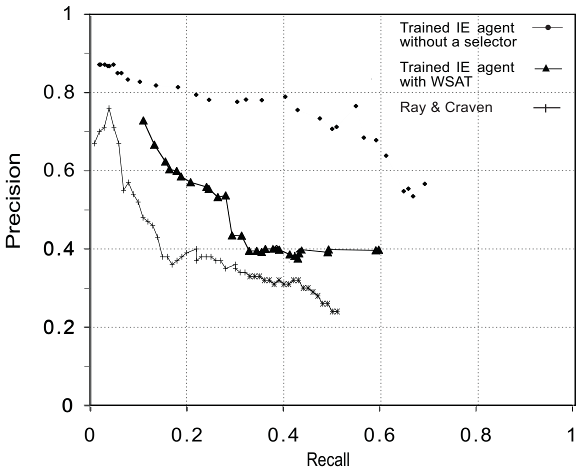

Figure 6. Precision/recall curves

The trained IE agent without any selector algorithm produces the best results. But, it is computationally very expensive since it needs to take the cross-product of all entries in the lists of individual-slot candidates. The trained IE agent with modified WalkSAT selector performs quite well, outperforming Ray and Craven's system. In results not shown, the "high-scoring SRS" selector performs similarly to Ray and Craven's.

Our results both illustrate the value of using theory refinement for IE and justify using an intelligent candidate-selection algorithm to reduce the computational burden of our "IE via IR" approach, which uses a generate-and-test strategy. Wawa-IE with the "modified WalkSAT" selector is able to improve on the state of the art using only 17% of the possible negative training candidates during training.

Finally, recall that we also use our variant of WalkSAT during *testing*. Thus, Figure 6 also shows that we obtain good precision and recall without needing to exhaustively score every possible candidate.

4. Related Work

We were unable to find any system in the literature that applies theory refinement to IE. Most IE systems break down into two groups. The first group uses some kind of relational learning to learn extraction patterns (Califf, 1998; Freitag, 1998; Soderland, 1999). The second group learns parameters of hidden Markov models (HMMs) and uses the HMMs to extract information (Bikel et al., 1999; Freitag & McCallum, 1999; Leek, 1997; Ray & Craven, 2001; Seymore et al., 1999).

Leek (1997) uses HMMs for extracting information from biomedical text. His system uses a lot of initial knowledge to build the HMM model before using the training data to learn the parameters of HMM. However, his system is not able to refine the knowledge.

Several authors use statistical methods to reduce the need for a lot of training examples. Freitag and McCallum (1999) use HMMs to extract information from text. They employ a statistical technique called "shrinkage" to get around the problem of not having sufficient labeled examples. Seymore et al. (1999) also use HMMs to extract information from on-line text. They get around the problem of not having sufficient training data by using data that is labeled for another purpose in their system. Similarly, Craven and Kumlien (1999) use "weakly" labeled training data to reduce the need for labeled training examples.

One advantage of our system is that we are able to utilize prior knowledge, which reduces the need for a large number of labeled training examples. However, we do not depend on the initial knowledge being 100% correct. We believe that it is relatively easy for users to articulate some useful domain-specific advice (especially when a user-friendly interface is provided that converts their advice into the specifics of Wawa-IE's advice language). The second advantage of our system is that the entire content of the document is used to estimate the correctness of a candidate extraction. This allows us to learn about the extraction slots and the documents in which they appear. The third advantage of Wawa-IE is that we are able to utilize the untrained ScorePage network to produce some informative negative training examples (*i.e.*, near misses).

5. Current and Future Work

Our catalog of candidate-selector algorithms includes a modified version of the GSAT algorithm (Selman et al., 1996), a modified version of random local search, and a hill climber with multiple random restarts. We are currently running tests on these selector algorithms; due to space limitations we did not discuss them.

In addition, we are also measuring performance as a function of the number of positive training examples and a function of the number of advice rules.

Finally, we are starting experiments on other IE domains such as the WebKB domain (Freitag, 1998). We also are looking into incorporating the candidate generation and selection steps directly into our connectionist framework, whereby we would use the current ScorePage network to help find new candidate extractions during the training process.

6. Conclusion

We describe and evaluate a system for using theory refinement to perform information extraction. Wawa-IE uses a neural network, which accepts advice containing variables, to rate candidate variable bindings in the content of the document as a whole. Our extraction process first generates a large set of candidate variable bindings for each slot, then selects a subset of the possible combinations of individual slot bindings via heuristic search, and finally uses the trained network to judge which are "best." Those bindings that score higher than a system-computed threshold are returned as the extracted information. By using theory refinement, we are able to take advantage of prior knowledge in the domain of interest and produce some informative training examples, both of which lead to an increase in the performance of the IE agent.

In our experiments in the CMU seminar-announcements domain, Wawa-IE achieved F_1-measures significantly higher than those of previous approaches. In the Yeast protein-localization data set, we empirically show the benefits of using an intelligent algorithm for selecting possible candidates for multiple slots and provided additional evidence that our approach improves on the state of the art.

Acknowledgements

This research was supported in part by NLM Grant 1 R01 LM07050-01, NSF Grant IRI-9502990, & UW Vilas Trust. Thanks to P. Andreae & M. Craven for helpful comments.

References

Aho, A., Sethi, R., & Ullman, J. (1986). *Compilers, Principles, Techniques and Tools*. Addison Wesley.

Bikel, D., Schwartz, R., & Weischedel, R. (1999). An algorithm that learns what's in a name. *Mach. Learn*, *34*.

Brill, E. (1994). Some advances in rule-based part of speech tagging. *Proc. AAAI94* (pp. 722–727). Seattle, WA.

Califf, M. E. (1998). *Relational Learning Techniques for Natural Language Information Extraction*. Doctoral dissertation, U. of Texas, Austin.

Craven, M., & Kumlien, J. (1999). Constructing biological knowledge-bases by extracting information from text sources. *Proc. of ISMB99* (pp. 77–86). Heidelberg.

Eliassi-Rad, T. (2001). *Building Intelligent Agents that Learn to Retrieve and Extract Information*. Doctoral dissertation, CS Dept, U. of Wisconsin, Madison.

Freitag, D. (1998). *Machine Learning for Information Extraction in Informal Domains*. Doctoral diss. CMU.

Freitag, D., & McCallum, A. (1999). Information extraction with HMMs and shrinkage. *Proc. AAAI99 Workshop on Machine Learning for Information Extraction*.

Leek, T. (1997). Information Extraction Using Hidden Markov Models. Master's Thesis, UCSD.

National Library of Medicine (2001). *The MEDLINE Database*. http://www.ncbi.nlm.nih.gov/PubMed/.

Ray, S., & Craven, M. (2001). Representing sentence structure in hidden markov models for information extraction. *Proc. of IJCAI01*. Seattle, WA.

Riloff, E. (1998). *The Sundance Sentence Analyzer*. http://www.cs.utah.edu/projects/nlp/.

Selman, B., Kautz, H., & Cohen, B. (1996). Local search strategies for satisfiability testing. *DIMACS Series in Discrete Mathematics and Theoretical CS*, *26*, 521–531.

Seymore, K., McCallum, A., & Rosenfeld, R. (1999). Learning hidden Markov model structure for information extraction. *Proc. AAAI99 Workshop on Machine Learning for Information Extraction* (pp. 37–42).

Shavlik, J., Calcari, S., Eliassi-Rad, T., & Solock, J. (1999). An instructable, adaptive interface for discovering and monitoring information on the world-wide web. *Proc. Intell. User. Interf.* (pp. 157–160). Redondo Beach, CA.

Shavlik, J., & Eliassi-Rad, T. (1998). Intelligent agents for web-based tasks: An advice-taking approach. *Proc. AAAI98 Workshop on Learning for Text Categorization*.

Soderland, S. (1999). Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, *34*.

Towell, G. G., & Shavlik, J. W. (1994). Knowledge-based artificial neural networks. *Artif. Intell.*, *70*, 119–165.

van Rijsbergen, C. J. (1979). *Information Retrieval*. London: Buttersworths. 2nd edition.