Proceedings of the Eleventh National Conference on Artificial Intelligence, 1993, AAAI Press / MIT Press, pages 811–816.

Automatically Constructing a Dictionary for Information Extraction Tasks

Ellen Riloff
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
riloff@cs.umass.edu

Abstract

Knowledge-based natural language processing systems have achieved good success with certain tasks but they are often criticized because they depend on a domain-specific dictionary that requires a great deal of manual knowledge engineering. This knowledge engineering bottleneck makes knowledge-based NLP systems impractical for real-world applications because they cannot be easily scaled up or ported to new domains. In response to this problem, we developed a system called AutoSlog that automatically builds a domain-specific dictionary of concepts for extracting information from text. Using AutoSlog, we constructed a dictionary for the domain of terrorist event descriptions in only 5 person-hours. We then compared the AutoSlog dictionary with a hand-crafted dictionary that was built by two highly skilled graduate students and required approximately 1500 person-hours of effort. We evaluated the two dictionaries using two blind test sets of 100 texts each. Overall, the AutoSlog dictionary achieved 98% of the performance of the hand-crafted dictionary. On the first test set, the AutoSlog dictionary obtained 96.3% of the performance of the hand-crafted dictionary. On the second test set, the overall scores were virtually indistinguishable with the AutoSlog dictionary achieving 99.7% of the performance of the hand-crafted dictionary.

Introduction

Knowledge-based natural language processing (NLP) systems have demonstrated strong performance for information extraction tasks in limited domains [Lehnert and Sundheim, 1991; MUC-4 Proceedings, 1992]. But enthusiasm for their success is often tempered by real-world concerns about portability and scalability. Knowledge-based NLP systems depend on a domain-specific dictionary that must be carefully constructed for each domain. Building this dictionary is typically a time-consuming and tedious process that requires many person-hours of effort by highly-skilled people who have extensive experience with the system. Dictionary construction is therefore a major knowledge engineering bottleneck that needs to be addressed in order for information extraction systems to be portable and practical for real-world applications.

We have developed a program called AutoSlog that automatically constructs a domain-specific dictionary for information extraction. Given a training corpus, AutoSlog proposes a set of dictionary entries that are capable of extracting the desired information from the training texts. If the training corpus is representative of the targeted texts, the dictionary created by AutoSlog will achieve strong performance for information extraction from novel texts. Given a training set from the MUC-4 corpus, AutoSlog created a dictionary for the domain of terrorist events that achieved 98% of the performance of a hand-crafted dictionary on 2 blind test sets. We estimate that the hand-crafted dictionary required approximately 1500 person-hours to build. In contrast, the AutoSlog dictionary was constructed in only 5 person-hours. Furthermore, constructing a dictionary by hand requires a great deal of training and experience whereas a dictionary can be constructed using AutoSlog with only minimal training.

We will begin with an overview of the information extraction task and the MUC-4 performance evaluation that motivated this work. Next, we will describe AutoSlog, explain how it proposes dictionary entries for a domain, and show examples of dictionary definitions that were constructed by AutoSlog. Finally, we will present empirical results that demonstrate AutoSlog's success at automatically creating a dictionary for the domain of terrorist event descriptions.

Information Extraction from Text

Extracting information from text is a challenging task for natural language processing researchers as well as a key problem for many real-world applications. In the last few years, the NLP community has made substantial progress in developing systems that can achieve good performance on information extraction tasks for limited domains. As opposed to in-depth natural language processing, information extraction is a more focused and goal-oriented task. For example, the MUC-4 task was to extract information about terrorist events, such as the names of perpetrators, victims, instruments, etc.

Our approach to information extraction is based on a technique called *selective concept extraction*. Selective concept extraction is a form of text skimming that selectively processes relevant text while effectively ignoring surrounding text that is thought to be irrelevant to the domain. The work presented here is based on a conceptual sentence analyzer called CIRCUS [Lehnert, 1990].

To extract information from text, CIRCUS relies on a domain-specific dictionary of *concept nodes*. A concept node is essentially a case frame that is triggered by a lexical item and activated in a specific linguistic context. Each concept node definition contains a set of enabling conditions which are constraints that must be satisfied in order for the concept node to be activated. For example, our dictionary for the terrorism domain contains a concept node called $kidnap-passive$ that extracts information about kidnapping events. This concept node is triggered by the word "kidnapped" and has enabling conditions that allow it to be activated only in the context of a passive construction. As a result, this concept node is activated by phrases such as "was kidnapped", "were kidnapped", etc. Similarly, the dictionary contains a second concept node called $kidnap-active$ which is also triggered by the word "kidnapped" but has enabling conditions that allow it to be activated only in the context of an active construction, such as "terrorists kidnapped the mayor".

In addition, each concept node definition contains a set of slots to extract information from the surrounding context. In the terrorism domain, concept nodes have slots for perpetrators, victims, instruments, etc. Each slot has a syntactic expectation and a set of hard and soft constraints for its filler. The syntactic expectation specifies where the filler is expected to be found in the linguistic context. For example, $kidnap-passive$ contains a victim slot that expects its filler to be found as the subject of the clause, as in "the mayor was kidnapped". The slot constraints are selectional restrictions for the slot filler. The hard constraints must be satisfied in order for the slot to be filled, however the soft constraints suggest semantic preferences for the slot filler so the slot may be filled even if a soft constraint is violated.

Given a sentence as input, CIRCUS generates a set of instantiated concept nodes as its output. If multiple triggering words appear in a sentence then CIRCUS can generate multiple concept nodes for that sentence. However, if no triggering words are found in a sentence then CIRCUS will generate no output for that sentence.

The concept node dictionary is at the heart of selective concept extraction. Since concept nodes are CIRCUS' only output for a text, a good concept node dictionary is crucial for effective information extraction. The UMass/MUC-4 system [Lehnert *et al.*, 1992a] used 2 dictionaries: a part-of-speech lexicon containing 5436 lexical definitions, including semantic features for domain-specific words and a dictionary of 389 concept node definitions for the domain of terrorist event descriptions. The concept node dictionary was manually constructed by 2 graduate students who had extensive experience with CIRCUS and we estimate that it required approximately 1500 person-hours of effort to build.

The MUC-4 Task and Corpus

In 1992, the natural language processing group at the University of Massachusetts participated in the Fourth Message Understanding Conference (MUC-4). MUC-4 was a competitive performance evaluation sponsored by DARPA to evaluate the state-of-the-art in text analysis systems. Seventeen sites from both industry and academia participated in MUC-4. The task was to extract information about terrorist events in Latin America from newswire articles. Given a text, each system was required to fill out a template for each terrorist event described in the text. If the text described multiple terrorist events, then one template had to be completed for each event. If the text did not mention any terrorist events, then no templates needed to be filled out.

A template is essentially a large case frame with a set of pre-defined slots for each piece of information that should be extracted from the text. For example, the MUC-4 templates contained slots for perpetrators, human targets, physical targets, etc. A training corpus of 1500 texts and instantiated templates (answer keys) for each text were made available to the participants for development purposes. The texts were selected by keyword search from a database of newswire articles. Although each text contained a keyword associated with terrorism, only about half of the texts contained a specific reference to a relevant terrorist incident.

Behind the Design of AutoSlog

Two observations were central to the design of AutoSlog. The first observation is that news reports follow certain stylistic conventions. In particular, the most important facts about a news event are typically reported during the initial event description. Details and secondary information are described later. It follows that the first reference to a major component of an event (e.g., a victim or perpetrator) usually occurs in a sentence that describes the event. For example, a story about a kidnapping of a diplomat will probably mention that the diplomat was kidnapped before it reports secondary information about the diplomat's family, etc. This observation is key to the design of AutoSlog. AutoSlog operates under the assumption that the *first* reference to a targeted piece of information is most likely where the relationship between that information and the event is made explicit.

Once we have identified the first sentence that contains a specific piece of information, we must determine which words or phrases should activate a concept node to extract the information. The second key observation behind AutoSlog is that the immediate linguistic context surrounding the targeted information usually contains the words or phrases that describe its role in the event. For example, consider the sentence "A U.S. diplomat was kidnapped by FMLN guerrillas today". This sentence contains two important pieces of information about the kidnapping: the victim ("U.S. diplomat") and the perpetrator ("FMLN guerrillas"). In both cases, the word "kidnapped" is the key word that relates them to the kidnapping event. In its passive form, we expect the subject of the verb "kidnapped" to be a victim and we expect the prepositional phrase beginning with "by" to contain a perpetrator. The word “kidnapped” specifies the roles of the people in the kidnapping and is therefore the most appropriate word to trigger a concept node.

AutoSlog relies on a small set of heuristics to determine which words and phrases are likely to activate useful concept nodes. In the next section, we will describe these heuristics and explain how AutoSlog generates complete concept node definitions.

Automated Dictionary Construction

Given a set of training texts and their associated answer keys, AutoSlog proposes a set of concept node definitions that are capable of extracting the information in the answer keys from the texts. Since the concept node definitions are general in nature, we expect that many of them will be useful for extracting information from novel texts as well. The algorithm for constructing concept node definitions is as follows. Given a targeted piece of information as a string from a template, AutoSlog finds the first sentence in the text that contains the string. This step is based on the observation noted earlier that the first reference to an object is likely to be the place where it is related to the event. The sentence is then handed over to CIRCUS which generates a conceptual analysis of the sentence. Using this analysis, AutoSlog identifies the first clause in the sentence that contains the string. A set of heuristics are applied to the clause to suggest a good *conceptual anchor point* for a concept node definition. If none of the heuristics is satisfied then AutoSlog searches for the next sentence in the text that contains the targeted information and the process is repeated.

The *conceptual anchor point heuristics* are the most important part of AutoSlog. A conceptual anchor point is a word that should activate a concept; in CIRCUS, this is a triggering word. Each heuristic looks for a specific linguistic pattern in the clause surrounding the targeted string. The linguistic pattern represents a phrase or set of phrases that are likely to be good for activating a concept node. If a heuristic successfully identifies its pattern in the clause then it generates two things: (1) a conceptual anchor point and (2) a set of enabling conditions to recognize the complete pattern. For example, suppose AutoSlog is given the clause “the diplomat was kidnapped” along with “the diplomat” as the targeted string. The string appears as the subject of the clause and is followed by a passive verb “kidnapped” so a heuristic that recognizes the pattern **<subject> passive-verb** is satisfied. The heuristic returns the word “kidnapped” as the conceptual anchor point along with enabling conditions that require a passive construction.

To build the actual concept node definition, the conceptual anchor point is used as its triggering word and the enabling conditions are included to ensure that the concept node is activated only in response to the desired linguistic pattern. For the example above, the final concept node will be activated by phrases such as “was kidnapped”, “were kidnapped”, “have been kidnapped”, etc.

The current version of AutoSlog contains 13 heuristics, each designed to recognize a specific linguistic pattern. These patterns are shown below, along with examples that illustrate how they might be found in a text. The bracketed item shows the syntactic constituent where the string was found which is used for the slot expectation (<dobj> is the direct object and <np> is the noun phrase following a preposition). In the examples on the right, the bracketed item is a slot name that might be associated with the filler (e.g., the subject is a victim). The underlined word is the conceptual anchor point that is used as the triggering word.

Linguistic Pattern	Example
<subject> passive-verb	<victim> was murdered
<subject> active-verb	<perpetrator> bombed
<subject> verb infinitive	<perpetrator> attempted to kill
<subject> auxiliary noun	<victim> was victim
passive-verb <dobj>[1]	killed <victim>
active-verb <dobj>	bombed <target>
infinitive <dobj>	to kill <victim>
verb infinitive <dobj>	threatened to attack <target>
gerund <dobj>	killing <victim>
noun auxiliary <dobj>	fatality was <victim>
noun prep <np>	bomb against <target>
active-verb prep <np>	killed with <instrument>
passive-verb prep <np>	was aimed at <target>

Several additional parts of a concept node definition must be specified: a slot to extract the information[2], hard and soft constraints for the slot, and a type. The syntactic constituent in which the string was found is used for the slot expectation. In the previous example, the string was found as the subject of the clause so the concept node is defined with a slot that expects its filler to be the subject of the clause.

The name of the slot (e.g., *victim*) comes from the template slot where the information was originally found. In order to generate domain-dependent concept nodes, AutoSlog requires three domain specifications. One of these specifications is a set of mappings from template slots to concept node slots. For example, information found in the human target slot of a template maps to a *victim* slot in a concept node. The second set of domain specifications are hard and soft constraints for each type of concept node slot, for example constraints to specify a legitimate victim.

Each concept node also has a type. Most concept nodes accept the event types that are found in the template (e.g., bombing, kidnapping, etc.) but sometimes we want to use special types. The third set of domain specifications are mappings from template types to concept node types. In general, if the targeted information was found in a kidnapping template then we use “kidnapping” as the concept node type. However, for the terrorism domain we used special types for information from the perpetrator and instrument template slots because perpetrators and instruments often

[1] In principle, passive verbs should not have objects. However, we included this pattern because CIRCUS occasionally confused active and passive constructions.

[2] In principle, concept nodes can have multiple slots to extract multiple pieces of information. However, all of the concept nodes generated by AutoSlog have only a single slot.

appear in sentences that do not describe the nature of the event (e.g., "The FMLN claimed responsibility" could refer to a bombing, kidnapping, etc.).

Sample Concept Node Definitions

To illustrate how this whole process comes together, we will show some examples of concept node definitions generated by AutoSlog. Figure 1 shows a relatively simple concept node definition that is activated by phrases such as "was bombed", "were bombed", etc. AutoSlog created this definition in response to the input string "public buildings" which was found in the physical target slot of a bombing template from text DEV-MUC4-0657. Figure 1 shows the first sentence in the text that contains the string "public buildings". When CIRCUS analyzed the sentence, it identified "public buildings" as the subject of the first clause. The heuristic for the pattern **<subject> passive-verb** then generated this concept node using the word "bombed" as its triggering word along with enabling conditions that require a passive construction. The concept node contains a single variable slot[3] which expects its filler to be the subject of the clause (*S*) and labels it as a target because the string came from the physical target template slot. The constraints for physical targets are pulled in from the domain specifications. Finally, the concept node is given the type *bombing* because the input string came from a bombing template.

Id: DEV-MUC4-0657 **Slot filler:** "public buildings"
Sentence: (in la oroya, junin department, in the central peruvian mountain range, public buildings were bombed and a car-bomb was detonated.)

CONCEPT NODE
Name: target-subject-passive-verb-bombed
Trigger: bombed
Variable Slots: (target (*S* 1))
Constraints: (class phys-target *S*)
Constant Slots: (type bombing)
Enabling Conditions: ((passive))

Figure 1: A good concept node definition

Figure 2 shows an example of a good concept node that has more complicated enabling conditions. In this case, CIRCUS found the targeted string "guerrillas" as the subject of the first clause but this time a different heuristic fired. The heuristic for the pattern **<subject> verb infinitive** matched the phrase "threatened to murder" and generated a concept node with the word "murder" as its trigger combined with enabling conditions that require the preceding words "threatened to" where "threatened" is in an active construction. The concept node has a slot that expects its filler to be the subject of the clause and expects it to be a perpetrator (because the slot filler came from a perpetrator template slot). The constraints associated with perpetrators are incorporated and the concept node is assigned the type "perpetrator" because our domain specifications map the perpetrator template slots to perpetrator types in concept nodes. Note that this concept node does not extract the direct object of "threatened to murder" as a victim. We would need a separate concept node definition to pick up the victim.

Id: DEV-MUC4-0071 **Slot filler:** "guerrillas"
Sentence: (the salvadoran guerrillas on mar_12_89, today, threatened to murder individuals involved in the mar_19_88 presidential elections if they do not resign from their posts.)

CONCEPT NODE
Name: perpetrator-subject-verb-infinitive-threatened-to-murder
Trigger: murder
Variable Slots: (perpetrator (*S* 1))
Constraints: (class perpetrator *S*)
Constant Slots: (type perpetrator)
Enabling Conditions: ((active)
(trigger-preceded-by? 'to 'threatened))

Figure 2: Another good concept node definition

Although the preceding definitions were clearly useful for the domain of terrorism, many of the definitions that AutoSlog generates are of dubious quality. Figure 3 shows an example of a bad definition. AutoSlog finds the input string, "gilberto molasco", as the direct object of the first clause and constructs a concept node that is triggered by the word "took" as an active verb. The concept node expects a victim as the direct object and has the type *kidnapping*. Although this concept node is appropriate for this sentence, in general we do not want to generate a kidnapping concept node every time we see the word "took".

Id: DEV-MUC4-1192 **Slot filler:** "gilberto molasco"
Sentence: (they took 2-year-old gilberto molasco, son of patricio rodriguez, and 17-year-old andres argueta, son of emimesto argueta.)

CONCEPT NODE
Name: victim-active-verb-dobj-took
Trigger: took
Variable Slots: (victim (*DOBJ* 1))
Constraints: (class victim *DOBJ*)
Constant Slots: (type kidnapping)
Enabling Conditions: ((active))

Figure 3: A bad concept node definition

AutoSlog generates bad definitions for many reasons, such as (a) when a sentence contains the targeted string but does not describe the event (i.e., our first observation mentioned earlier does not hold), (b) when a heuristic proposes the wrong conceptual anchor point or (c) when CIRCUS incorrectly analyzes the sentence. These potentially dangerous definitions prompted us to include a human in

[3] *Variable slots* are slots that extract information. *Constant slots* have pre-defined values that are used by AutoSlog only to specify the concept node type.

the loop to weed out bad concept node definitions. In the following section, we explain our evaluation procedure and present empirical results.

Empirical Results

To evaluate AutoSlog, we created a dictionary for the domain of terrorist event descriptions using AutoSlog and compared it with the hand-crafted dictionary that we used in MUC-4. As our training data, we used 1500 texts and their associated answer keys from the MUC-4 corpus. Our targeted information was the slot fillers from six MUC-4 template slots that contained string fills which could be easily mapped back to the text. We should emphasize that AutoSlog does not require or even make use of these complete template instantiations. AutoSlog needs only an annotated corpus of texts in which the targeted information is marked and annotated with a few semantic tags denoting the type of information (e.g., victim) and type of event (e.g., kidnapping).

The 1258 answer keys for these 1500 texts contained 4780 string fillers which were given to AutoSlog as input along with their corresponding texts.[4] In response to these strings, AutoSlog generated 1237 concept node definitions. AutoSlog does not necessarily generate a definition for every string filler, for example if it has already created an identical definition, if no heuristic applies, or if the sentence analysis goes wrong.

As we mentioned earlier, not all of the concept node definitions proposed by AutoSlog are good ones. Therefore we put a human in the loop to filter out definitions that might cause trouble. An interface displayed each dictionary definition proposed by AutoSlog to the user and asked him to put each definition into one of two piles: the "keeps" or the "edits". The "keeps" were good definitions that could be added to the permanent dictionary without alteration.[5] The "edits" were definitions that required additional editing to be salvaged, were obviously bad, or were of questionable value. It took the user 5 hours to sift through all of the definitions. The "keeps" contained 450 definitions, which we used as our final concept node dictionary.

Finally, we compared the resulting concept node dictionary[6] with the hand-crafted dictionary that we used for MUC-4. To ensure a clean comparison, we tested the AutoSlog dictionary using the official version of our UMass/MUC-4 system. The resulting "AutoSlog" system was identical to the official UMass/MUC-4 system except that we replaced the hand-crafted concept node dictionary with the new AutoSlog dictionary. We evaluated both systems on the basis of two blind test sets of 100 texts each. These were the TST3 and TST4 texts that were used in the final MUC-4 evaluation. We scored the output generated by both systems using the MUC-4 scoring program. The results for systems are shown in Table 1.[7]

Recall refers to the percentage of the correct answers that the system successfully extracted and *precision* refers to the percentage of answers extracted by the system that were actually correct. The *F-measure* is a single measure that combines recall and precision, in this case with equal weighting. These are all standard measures used in the information retrieval community that were adopted for the final evaluation in MUC-4.

System/Test Set	Recall	Precision	F-measure
MUC-4/TST3	46	56	50.51
AutoSlog/TST3	43	56	48.65
MUC-4/TST4	44	40	41.90
AutoSlog/TST4	39	45	41.79

Table 1: Comparative Results

The official UMass/MUC-4 system was among the top-performing systems in MUC-4 [Lehnert *et al.*, 1992b] and the results in Table 1 show that the AutoSlog dictionary achieved almost the same level of performance as the hand-crafted dictionary on both test sets. Comparing F-measures, we see that the AutoSlog dictionary achieved 96.3% of the performance of our hand-crafted dictionary on TST3, and 99.7% of the performance of the official MUC-4 system on TST4. For TST4, the F-measures were virtually indistinguishable and the AutoSlog dictionary actually achieved better precision than the original hand-crafted dictionary. We should also mention that we augmented the hand-crafted dictionary with 76 concept nodes created by AutoSlog before the final MUC-4 evaluation. These definitions improved the performance of our official system by filling gaps in its coverage. Without these additional concept nodes, the AutoSlog dictionary would likely have shown even better performance relative to the MUC-4 dictionary.

Conclusions

In previous experiments, AutoSlog produced a concept node dictionary for the terrorism domain that achieved 90% of the performance of our hand-crafted dictionary [Riloff and Lehnert, 1993]. There are several possible explanations for the improved performance we see here. First, the previous results were based on an earlier version of AutoSlog. Several improvements have been made to AutoSlog since then. Most notably, we added 5 new heuristics to recognize additional linguistic patterns. We also made a number of improvements to the CIRCUS interface and other parts of the system that eliminated many bad definitions[8] and gener-

[4] Many of the slots contained several possible strings ("disjuncts"), any one of which is a legitimate filler. AutoSlog finds the first sentence that contains any of these strings.

[5] The only exception is that the user could change the concept node type if that was the only revision needed.

[6] We augmented the AutoSlog dictionary with 4 meta-level concept nodes from the hand-crafted dictionary before the final evaluation. These were special concept nodes that recognized textual cues for discourse analysis only.

[7] The results in Table 1 do not correspond to our official MUC-4 results because we used "batch" scoring and an improved version of the scoring program for the experiments described here.

[8] The new version of AutoSlog generated 119 fewer definitions than the previous version even though it was given 794 additional string fillers as input. Even so, this smaller dictionary produced

ally produced better results. Another important factor was the human in the loop. We used the same person in both experiments but, as a result, he was more experienced the second time. As evidence, he finished the filtering task in only 5 hours whereas it took him 8 hours the first time.[9]

AutoSlog is different from other lexical acquisition systems in that most techniques depend on a "partial lexicon" as a starting point (e.g., [Carbonell, 1979; Granger, 1977; Jacobs and Zernik, 1988]). These systems construct a definition for a new word based on the definitions of other words in the sentence or surrounding context. AutoSlog, however, constructs new dictionary definitions completely from scratch and depends only on a part-of-speech lexicon, which can be readily obtained in machine-readable form.

Since AutoSlog creates dictionary entries from scratch, our approach is related to one-shot learning. For example, explanation-based learning (EBL) systems [DeJong and Mooney, 1986; Mitchell *et al.*, 1986] create complete concept representations in response to a single training instance. This is in contrast to learning techniques that incrementally build a concept representation in response to multiple training instances (e.g., [Cardie, 1992; Fisher, 1987; Utgoff, 1988]). However, explanation-based learning systems require an explicit domain theory which may not be available or practical to obtain. AutoSlog does not need any such domain theory, although it does require a few simple domain specifications to build domain-dependent concept nodes.

On the other hand, AutoSlog is critically dependent on a training corpus of texts and targeted information. We used the MUC-4 answer keys as training data but, as we noted earlier, AutoSlog does not need these complete template instantiations. AutoSlog would be just as happy with an "annotated" corpus in which the information is marked and tagged with event and type designations. NLP systems often rely on other types of tagged corpora, for example part-of-speech tagging or phrase structure bracketing (e.g., the Brown Corpus [Francis and Kucera, 1982] and the Penn Treebank [Marcus *et al.*]). However, corpus tagging for automated dictionary construction is less demanding than other forms of tagging because it is smaller in scope. For syntactic tagging, every word or phrase must be tagged whereas, for AutoSlog, only the targeted information needs to be tagged. Sentences, paragraphs, and even texts that are irrelevant to the domain can be effectively ignored.

We have demonstrated that automated dictionary construction is a viable alternative to manual knowledge engineering. In 5 person-hours, we created a dictionary that achieves 98% of the performance of a hand-crafted dictionary that required 1500 person-hours to build. Since our approach still depends on a manually encoded training corpus, we have not yet eliminated the knowledge engineering bottleneck. But we have significantly changed the nature of the bottleneck by transferring it from the hands of NLP experts to novices. Our knowledge engineering demands can be met by anyone familiar with the domain. Knowledge-based NLP systems will be practical for real-world applications only when their domain-dependent dictionaries can be constructed automatically. Our approach to automated dictionary construction is a significant step toward making information extraction systems scalable and portable to new domains.

better results than the larger one constructed by the earlier system.

[9] For the record, the user had some experience with CIRCUS but was not an expert.

Acknowledgments

We would like to thank David Fisher for designing and programming the AutoSlog interface and Stephen Soderland for being our human in the loop. This research was supported by the Office of Naval Research Contract N00014-92-J-1427 and NSF Grant no. EEC-9209623, State/Industry/University Cooperative Research on Intelligent Information Retrieval.

References

Carbonell, J. G. 1979. Towards a Self-Extending Parser. In *Proceedings of the 17th Meeting of the Association for Computational Linguistics*. 3–7.

Cardie, C. 1992. Learning to Disambiguate Relative Pronouns. In *Proceedings of the Tenth National Conference on Artificial Intelligence*. 38–43.

DeJong, G. and Mooney, R. 1986. Explanation-Based Learning: An Alternative View. *Machine Learning* 1:145–176.

Fisher, D. H. 1987. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning* 2:139–172.

Francis, W. and Kucera, H. 1982. *Frequency Analysis of English Usage*. Houghton Mifflin, Boston, MA.

Granger, R. H. 1977. FOUL-UP: A Program that Figures Out Meanings of Words from Context. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*. 172–178.

Jacobs, P. and Zernik, U. 1988. Acquiring Lexical Knowledge from Text: A Case Study. In *Proceedings of the Seventh National Conference on Artificial Intelligence*. 739–744.

Lehnert, W. 1990. Symbolic/Subsymbolic Sentence Analysis: Exploiting the Best of Two Worlds. In Barnden, J. and Pollack, J., editors 1990, *Advances in Connectionist and Neural Computation Theory, Vol. 1*. Ablex Publishers, Norwood, NJ. 135–164.

Lehnert, W.; Cardie, C.; Fisher, D.; McCarthy, J.; Riloff, E.; and Soderland, S. 1992a. University of Massachusetts: Description of the CIRCUS System as Used for MUC-4. In *Proceedings of the Fourth Message Understanding Conference (MUC-4)*. 282–288.

Lehnert, W.; Cardie, C.; Fisher, D.; McCarthy, J.; Riloff, E.; and Soderland, S. 1992b. University of Massachusetts: MUC-4 Test Results and Analysis. In *Proceedings of the Fourth Message Understanding Conference (MUC-4)*. 151–158.

Lehnert, W. G. and Sundheim, B. 1991. A Performance Evaluation of Text Analysis Technologies. *AI Magazine* 12(3):81–94.

Marcus, M.; Santorini, B.; and Marcinkiewicz, M. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*. Forthcoming.

Mitchell, T. M.; Keller, R.; and Kedar-Cabelli, S. 1986. Explanation-Based Generalization: A Unifying View. *Machine Learning* 1:47–80.

Proceedings of the Fourth Message Understanding Conference (MUC-4). 1992. Morgan Kaufmann, San Mateo, CA.

Riloff, E. and Lehnert, W. 1993. Automated Dictionary Construction for Information Extraction from Text. In *Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications*. IEEE Computer Society Press. 93–99.

Utgoff, P. 1988. ID5: An Incremental ID3. In *Proceedings of the Fifth International Conference on Machine Learning*. 107–120.