Automatically Generated DAML Markup for Semistructured Documents

William Krueger, Jonathan Nilsson, Tim Oates, Timothy Finin

CSEE Department, UMBC, 1000 Hilltop Circle,
Baltimore, MD 21250, USA
{wkrueg1, jnilss1, oates, finin}@cs.umbc.edu

Abstract. The semantic web is becoming a realizable technology due to the efforts of researchers to develop semantic markup languages such as the DARPA Agent Markup Language (DAML). A major problem that faces the semantic web community is that most information sources on the web today lack semantic markup. To fully realize the potential of the semantic web, we must find a way to automatically upgrade information sources with semantic markup. We have developed a system based on the STALKER algorithm that automatically generates DAML markup for a set of documents based on previously seen labeled training documents. Our rule-learning approach to semantic markup is highly effective when dealing with semistructured documents.

1 Introduction

Imagine a world-wide information network in which seamlessly integrated intelligent agents perform complex tasks by retrieving and processing information from anywhere in the world. This is the vision of the semantic web (Hendler, 2001). To make this vision a reality, the information stored on the world-wide web must be machine-understandable. Ideally, the expensive chore of manually marking up web documents with a semantic markup language like DAML (Hendler & McGuinness, 2000) can be avoided. Even hand-crafting rules is essentially an expensive manual approach as great effort must be made to write the many rules needed to perform accurate information extraction. Our research consists of finding an alternative to manual markup. Our alternative is a machine learning approach.

Although our system can be applied to any ontology, any semi-structured information source, and any semantic markup language, we applied the system to the problem of marking up talk announcements from three different domains with DAML. These specifics were chosen to support UMBC's ITTALKS system. ITTALKS (Cost et al., 2002) is an application that utilizes an agent architecture to discover talks that are of interest to users.

Our system uses the Talk ontology developed at UMBC as a guide for hierarchical wrapper induction. An ontology is the structured vocabulary used to semantically markup a document. Given the hierarchical nature of ontologies, it is natural to implement a hierarchical approach to performing semantic markup. The STALKER algorithm (Muslea, Minton, & Knoblach, 2001) uses such a hierarchical approach,

L. van Elst, V. Dignum, and A. Abecker (Eds.): AMKM 2003, LNAI 2926, pp. 276-287, 2003.

meaning that it limits its search within a document for leaves like *Hour* and *Minute* according to where their ontological parent *BeginTime* occurs.

Information agents use wrappers to extract information from documents. A wrapper is a set of extraction rules along with the code required to apply those extraction rules to a given document. Using this terminology, STALKER is a hierarchical wrapper induction algorithm. Using machine learning terminology, STALKER is a sequential covering algorithm. As explained in (Muslea et al., 2001), the STALKER algorithm requires up to two fewer orders of magnitude fewer training examples than other current wrapper induction algorithms and can handle information sources that cannot be wrapped by any other existing techniques. We have extended STALKER. Our implementation is based on the algorithm as it is described in (Muslea et al., 2001). As a result, our system applies to semi-structured information sources, meaning that relevant information should be locatable using a formal grammar.

We first describe a hierarchical characterization of the data in documents and then explain how data is extracted from documents using rules. Moving on, we describe the wrapper induction process in detail. After showing the method we use for extracting data, we focus in on several enhancements we made to the STALKER algorithm as well as how those enhancements improved our system's performance. We then provide a statistical discussion of our empirical results. Finally, we present our future considerations.

2 A Hierarchical Characterization of Data

Ideally, we would like to view each talk announcement as a collection of segments. These segments are *Title*, *Abstract*, *Date*, *BeginTime*, *EndTime*, *Location*, *Speaker*, *Host*, and *Type*. Each of these segments is itself either data (a leaf in the ontology) or a collection of other segments. For example, *Title* and *Abstract* are data, whereas *BeginTime* consists of both *BeginTime*:*Hour* and *BeginTime*:*Minute*. Notice that we have used the notation *BeginTime*:*Hour* to convey that the hour we are talking about pertains to the talk's begin time as opposed to the talk's end time. We make the assumption that it is most likely to find data pertaining to a single entity, such as *Location*, grouped closely together. This assumption enables us to view each document as a collection of embedded segments, and we can focus our search for data within a restricted view of the entire document.

However, it is possible that the city in which the talk is given does not appear near the rest of the information about the talk's location. Because of this problem, we attempt to learn rules for extracting two different entities that represent the same piece of data, one which is embedded and one which is not. We would like it if all talk announcements were structured hierarchically in a way that is perfectly consistent with the definition of our Talk ontology, but that is not the case. We might have chosen to tailor the talk ontology to each specific domain, but we took a more general approach.

3 Wrappers

A wrapper is the set of rules used to extract data along with the code required to perform the extraction. Before describing the extraction process, let us first define some terms. Viewing each document as a long sequence of tokens, let us define a landmark to be a sequence of one or more consecutive tokens, that is, a nonempty subsequence of the entire document. At this point, we should agree on what a token is. We have already said that a document is a long sequence of tokens, meaning that each document is parsed, or tokenized, into elementary pieces of text. Such tokens include HTML tags, all-lowercase words, two-digit numbers, symbols, alphanumeric words, etc. A *rule clause* is simply a landmark along with a qualifier indicating how to traverse the landmark. The two types of rule clauses are: *SkipTo* and *SkipUntil*. Let us form an example to clarify the distinction between the two possible types of rule clauses. Suppose our rule clause is *SkipTo*(<b> SYMBOL), the token sequence "<b> SYMBOL" of type *SkipTo*. The meaning of this rule clause would be to skip over everything until an HTML bold tag was found, followed immediately by any symbol. When this sequence of tokens has been found, then either the data we are searching for begins immediately *after* the symbol or we resume searching immediately *after* the symbol. On the other hand, suppose our rule clause is *SkipUntil*(<b> SYMBOL). The meaning of this rule clause would again be to skip over everything until an HTML bold tag was found, followed immediately by any symbol, only this time, either the data we are seeking begins immediately *at* the first token in the *SkipUntil* rule clause or we resume searching *at* the first token in the *SkipUntil* rule clause. Whether or not we continue searching depends on whether or not the rule clause is the last one in its rule.

Let us define a *rule* in this context to be an ordered list of rule clauses. Now, we argue that a single rule can always be represented as some sequence of *SkipTo* rule clauses possibly followed by one *SkipUntil* rule clause. This claim is justified because any rule containing a *SkipUntil* rule clause *rc* in any position within the rule but the last is equivalent to that same rule with the *SkipUntil* rule clause *rc* replaced by a *SkipTo* rule clause containing the same tokens. Therefore, we have two types of rules: ones ending with a *SkipTo* rule clause and ones ending with a *SkipUntil* rule clause.

Now that we have defined our two types of rules, we can take the next step and see how to extract data from a document. Rules can be applied both backward and forward through a sequence of tokens. When applied in the backward direction, a rule simply acts on a token sequence in reverse order. To extract data consisting of a sequence of tokens, we must locate both the beginning and the end of that sequence. For this, we must have some rules that locate the beginning and other rules that locate the end. Given the beginning and end tokens of some piece of data, the entire piece consists of the beginning token, the end token, and everything in between.

Consider the following excerpt from a UC Berkeley talk announcement: "This talk will be held in the Main Lecture Hall at ICSI. 1947 Center Street, Sixth Floor, Berkeley, CA 94704-1198 (on Center between Milvia and Martin Luther King Jr. Way)."

Here, we will assume that the location data for the talk has previously been extracted by some set of rules and that the excerpted passage is the extracted location data. Suppose our goal is to extract Location:City, the city in which the talk takes place. One way to do this would be to use the forward rule *SkipTo*(,) *SkipTo*(,) to find the first token of the city and to use the backward rule *SkipTo*(,) to find the last token of the city. In this case, the first and last tokens are the same; they are both the initial-cap word "Berkeley." There are many other rules that could be used to extract the same data. Now we see why a hierarchical extraction approach is helpful. It is likely that a rule as simple as *SkipTo*(,) could not have been used to help find the city if it had been applied to the entire document and not just to the portion of the document containing the location.

4 Hierarchical Wrapper Induction

It is now time to describe the STALKER algorithm (Muslea et al., 2001) upon which our system is based. Let us assume we are finding only beginning tokens for data since finding end tokens is analogous. For each element in the talk ontology, we do the following: starting with a set *T* of labeled training documents, we learn a single rule based on the documents in *T* to find the location of the beginning token for the current ontology element. Next, we remove all documents from *T* whose beginning token for the current ontology element is correctly identified by the learned rule. We then repeat the process of learning one rule, only now with a smaller set of documents, until *T* contains no documents. What we are left with is sets of rules that can be applied in some optimal ordering to find the beginning tokens for the elements in the talk ontology. For a detailed description of the basic STALKER algorithm, see (Muslea et al., 2001).

Since our system uses *supervised* machine learning, we should describe how we labeled the training documents. The double angle brackets “<<” and “>>” are used to avoid conflicts with HTML tags. Examples of begin tags are <<Talk>> and <<Talk:Title>>, and their corresponding end tags are <</Talk>> and <</Talk:Title>>. Each tag corresponds directly to an element with the same name in the Talk ontology, so sometimes we may refer to both DAML tags and ontology elements interchangeably.

When we are learning where to place a DAML tag, we make use of the location of that tag’s ontological parent. To make this possible, we order the learning of tags by ontological depth. Therefore, when we are learning where to place tags for some ontology element of depth two, for example, we assume we will know where its ontological parent of depth one is at markup time. When performing DAML markup for some element, we can limit our view to only the portion of the document corresponding to the parent of that element. Let us make this clear with an example. Say we are determining where to place tags for the speaker of a talk. First, we insert tags for the element *Talk:Speaker*. Then, we look only in between these tags when determining where to insert tags for *Talk*:*Speaker*:*FirstName*, *Talk*:*Speaker*:*Email*, etc. If we could not insert tags for *Talk:Speaker*, then we also fail to insert tags for ontological children of *Talk:Speaker*. If we are unable to insert tags for the tag *Talk*, then our hierarchical approach is dissolved.

To motivate a few items we will address later on, it is convenient for us now to discuss one component of the STALKER algorithm. Rule refinement works as follows. Each time we want to learn a single rule for determining some ontology element's starting or end location, the rule refinement process occurs. Each rule starts out as a single-token, single-landmark rule, such as *SkipUntil*(<b>). Then, the rule is refined, that is, landmarks are added to the rule and tokens are added to the rule's landmarks, until it has either achieved perfect accuracy (though not necessarily perfect recall) on the current set of training documents or arrived at the threshold for the maximum allowable rule size. Applying rule refinement to imperfect rules serves three purposes. First, it serves to increase the accuracy of rules. Remember that our system for learning rules is based on a sequential covering algorithm, so a rule is not learned until it has attained perfect accuracy. Second, by refining a rule, we decrease the chance that the rule will match some arbitrary sequence of tokens, and as a result our system's accuracy and recall on unseen documents is improved. Finally, rule refinement allows one rule to be used in marking up many documents. That is, fewer rules are learned because each learned rule has a greater coverage on the training documents.

5 Contributions

We began with our goal being to implement a system based on the STALKER algorithm to generate DAML markup for talk announcements on the web. DAML is similar in structure to HTML, and it allows us to provide metadata for documents so that they become understandable by machine applications. For instance, in a talk announcement marked up with DAML, one might find the following text: "<City>Berkeley</City>."

We developed along the way some techniques to increase the accuracy and recall of the basic system. With explanations to follow immediately, our added techniques can be referred to as: Minimum Refinements, Rule Score, Refinement Window, and Wildcards.

5.1 Minimum Refinements

Consider the case where an initial one-landmark, one-token rule matches only one document in the training set of documents and where that match is correct according to the labeling. Having learned a perfect rule, the document covered by that rule is removed, and the system continues learning more rules. This case can occur quite frequently and is highly undesirable. For instance, to learn the first name of a speaker, the rule SkipUntil(George) might be found to be perfect. When applied to unseen documents, however, this rule would probably be very ineffective in general for finding the speaker's first name. Our system, unlike the original STALKER algorithm, forces rules to be refined at least a certain number of times, and we usually choose that number to be five. In this way, we allow rules to be generalized with the hope that they will ultimately more closely reflect the structure of the documents rather than the attributes of the data being extracted.

5.2 Rule Score

For each DAML tag, the system generates both a forward rule and a backward rule. When it is time to mark up a new document, the forward and backward rules will sometimes disagree on the location of the tag. The system must have a way to resolve these disagreements. In the original STALKER algorithm, forward rules were always used for begin tags and backward rules were always used for end tags. With the rule score system both forward rules and backward rules are generated for every tag and evaluated based on how well they perform on a validation set of documents. The validation set consists of documents that have been marked up by hand but are not used by the system to generate rules. Whenever forward and backward rules disagree on the location of a tag, the system will use the rule with the highest score. Rules with lower scores are only used when higher scored rules fail to find the locations for tags.

5.3 Refinement Window

The system learns rules that find the position of tags relative to the position of their ontological parents. This means that the system only has to consider information that is contained within the ontological parent when looking for the location of a DAML tag. Consider the tag *BeginTime:Hour*. Its ontological parent is the *BeginTime* tag which will likely contain only a few tokens. However, for a tag like *Talk:Title* that has the entire talk as its parent the system will consider all of the tokens in the talk announcement to learn a rule. Many of these tokens will be far enough away from the title to be considered irrelevant. At best the system will consider many irrelevant tokens. At worst it will actually use some of them in the rules and as a result generate a rule that will work on one training document and nothing else. The refinement window limits the number of tokens the system will consider when learning rules to the n nearest tokens on each side of the tag. We usually use $n = 10$ in our system. This not only improves the quality of the rules the system learns, but also significantly decreases the running time. For example, in the UCSB domain the time required to learn the forward and backward rules for *all* tags was reduced from 246 seconds to 66 seconds after adding the token window to the basic system. In this experiment, we used a training set of 20 documents. Using the same training set, the full system produced rules in 117 seconds.

5.4 Wildcards

The domain-independent wildcards used in the system are based on the structure of tokens. For example, the INITIAL_CAP_WORD wildcard matches any token that consists of an uppercase letter followed by zero or more lower case letters. We included one domain-independent wildcard for each elementary token type as well as some more general wildcards such as ANY, which matches any token. In addition to domain-independent wildcards, sometimes domain-specific wildcards can improve the system's performance. In determining the location of the *Date:Month* tag, the system might use the INITIAL_CAP_WORD wildcard in a rule because it will match every month. Notice, though, that there may be many tokens in the document that match this wildcard. For instance, even when the system is restricted to the date information, not only the month but also the day of the week will match the INITIAL_CAP_WORD wildcard. Because this wildcard is too general, the system would have to use a literal landmark like September instead. "September" is too specific, though. Our solution to this dilemma is to use domain-specific wildcards. Adding the wildcard MONTH that matches any of the twelve months of the year gives the system a wildcard that is exactly as general as it needs to be. Our system includes several domain-specific wildcards.

6 Experimental Results

To test the contribution of each technique when added to the basic system we formed six different systems. One was the basic system which only used the STALKER algorithm. One used the minimum refinement threshold. One used the refinement window. One used domain specific wildcards. One used a rule score for marking up documents. The full system used all of these components.

Let us define *recall* to be the number of *correctly* extracted data items divided by the total number of data items that existed in the documents. Recall, defined in this way, measures the system's precision because only *correctly* extracted data items are considered successes. In the discussion that follows, what we mean by the "performance" of some system is its recall for either one particular ontology element or all ontology elements, depending on the context.

Three domains, each contributing a corpus of talk announcements, were chosen fairly arbitrarily. The criterion we used in selecting two of these domains was that within each domain there should be a plentiful supply of talk announcements having a generally consistent structure. These two will be referred to as the UC Berkeley and UCSB (University of California, Santa Barbara) domains, reflecting the origins of the documents. The UC Berkeley and UCSB corpuses each contain 60 documents. We will refer to the third domain as the ITTALKS domain, for this domain contains talk announcements from UMBC's ITTALKS system. This third domain was chosen specifically because it had a very consistent structure and could serve as a sanity check for our system. The ITTALKS corpus contains 30 documents.

We randomly chose ten ways to partition the sixty labeled training documents from UC Berkeley into equal training, validation, and testing sets. Then we ran each system on each of the partitions. We ran the same experiments on the UCSB and ITTALKS domains, but first we will present our results from what proved to be our most challenging domain, UC Berkeley. Because we ran each system on 10 different partitions, we will typically refer to a system's performance as *average recall*, which is the average recall achieved over all 10 partitions.

6.1 UC Berkeley Results

For most ontology elements the full system performed the best and the basic system performed the worst. There was only one DAML tag for which the basic system had better average recall than the full system. In several cases the differences between the average performances on an element was dramatic. Examining those cases will show what each component adds to the full system.

In attempting to extract date information, it makes sense to use domain specific wildcards. There are only seven possible days of the week so it is simple to add a wildcard to the system that matches any day. This improved average recall for the day of the week over the basic system from 87.5% to 91.5%. However, using score to select the best rule for mark up yielded a 94% average recall even without the wildcards. The full system which made use of both achieved a 96% average recall for day of the week.

In most documents the title occurs near the beginning. Since the title's ontological parent is the entire talk the system has almost the entire document to look at to determine a rule for the end of the title. Most of this information is far enough away from the title to be worthless. However, the basic system is allowed to use it to develop a rule that works on the training data but probably not on any unseen documents. By only allowing the system to use the 10 closest tokens in refinement we force it to use the most relevant information surrounding data to learn new rules. The refinement window improved average recall for title from 22% on the basic system to 49.5%. The full system achieved 60.5% average recall for the title.

It is hard for the system to learn rules to extract names because it is easy to learn rules that will work on one document but not any others. For instance it may learn that George indicates the speaker's first name, but that only works for a small fraction of speakers. Including a minimum refinement threshold forces the system to ignore initial candidates that are too specific. The basic system only had 2.5% average recall for the speaker's first name, but with the minimum refinement threshold it reached 50%.

The most effective component was using the score to mark up documents. With this system, the rules are tried on a validation set and ordered based on effectiveness. Then when the system is given new documents to mark up, the most effective rules take precedence over less effective rules. In the basic system begin tags are always found using forward rules and end tags are always found using backward rules. Unlike the basic system the score system will always use the most historically accurate rule. The score system did better than the basic system on almost all tags.

The full system's average recall over all tags was 83.1% with a standard deviation of 2.7%. These numbers are statistically significant when compared to those of the basic system. The basic system's average recall over all tags was 69.6% with a standard deviation of 2.1%. See figure 1 for a comparison of the recall of the two systems.

The average recall for several tags was significantly improved by the features present in the full system. These instances are displayed in figure 2.

UC Berkeley was our most challenging domain due to a higher degree of inconsistency in the documents. For instance, the begin time appeared as "2:00" in one docu-

document and "2pm" in another. Such inconsistencies were abundant in the UC Berkeley domain.

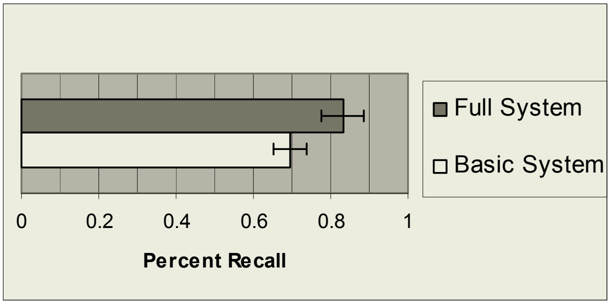

Fig. 1. Average recall for all tags in the UC Berkeley domain. The error bars represent 95% confidence intervals

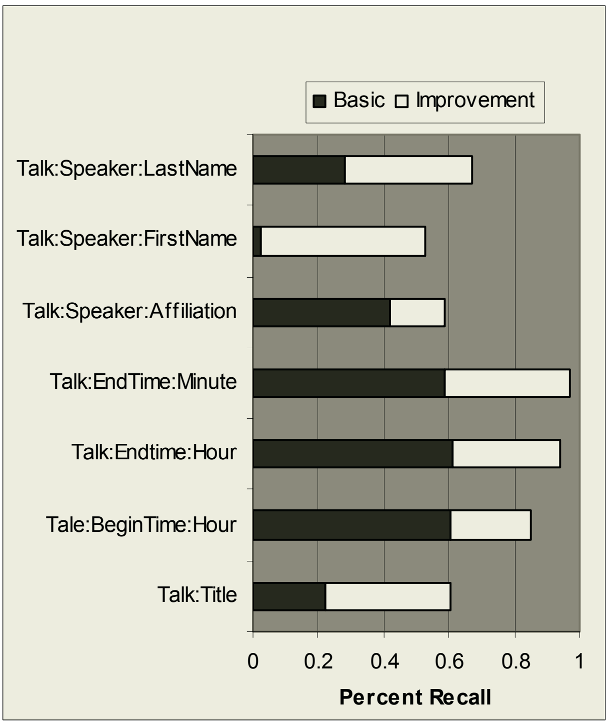

Fig. 2. Average recall for selected tags in the UC Berkeley domain. An entire bar represents the average recall achieved by the full system for that tag. The black and white portions of a bar represent the average recall achieved by the basic system and the improvement made by the full system, respectively

6.2 UCSB and ITTALKS Results

The results from the UCSB domain were similar to those from the UC Berkeley domain. See figure 3 for a comparison between the basic and full systems. The average recall for several tags was significantly improved by the features present in the full system, and these instances are displayed in figure 4.

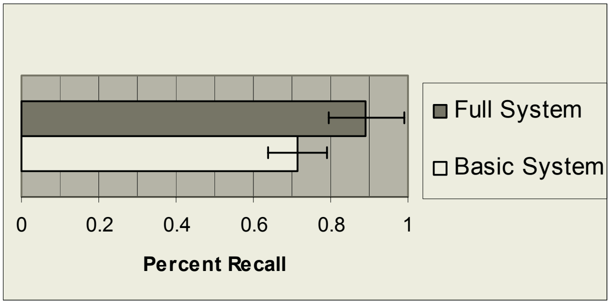

Fig. 3. Average recall for all tags in the UCSB domain. The error bars represent 95% confidence intervals

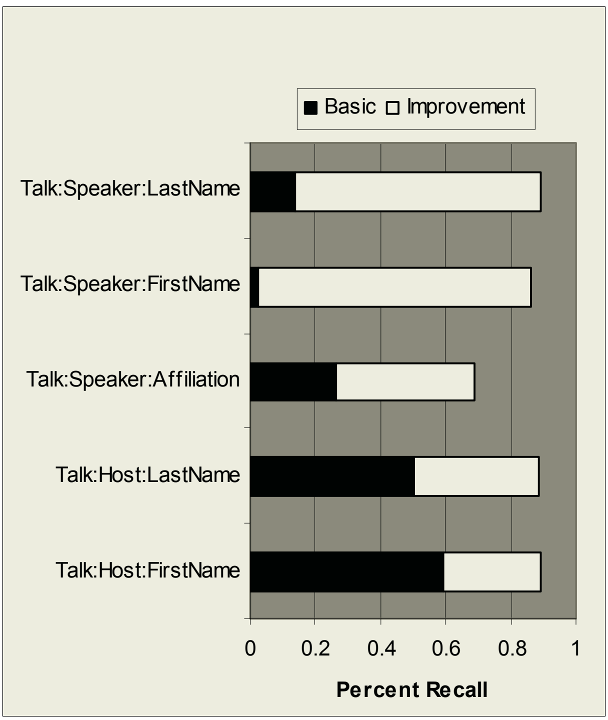

Fig. 4. Average recall for selected tags in the UCSB domain. An entire bar represents the average recall achieved by the full system for that tag. The black and white portions of a bar represent the average recall achieved by the basic system and the improvement made by the full system, respectively

As depicted in figure 5, the full system provided no significant increase in average recall in the ITTALKS domain. Notice, however, that the basic system performed much better in the ITTALKS domain than in the other two domains. This is because the ITTALKS domain contains talk announcements that have regularly appearing HTML comments that serve as useful landmarks. Since high-accuracy rules were often learned using these HTML comments as landmarks, the rules learned by the full system were often the same as those learned by the basic system.

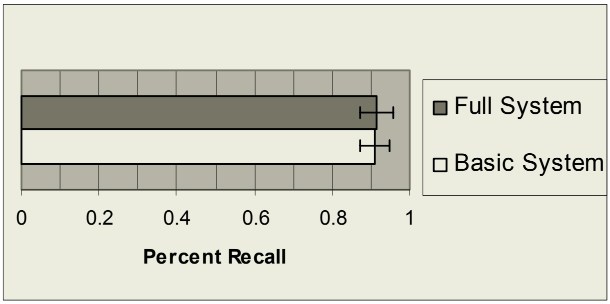

Fig. 5. Average recall for all tags in the ITTALKS domain. The error bars represent 95% confidence intervals

6.3 Mixing the Domains

Up to this point, our system was tested only on one domain at a time. To test our system's ability to generate a single set of rules for two unrelated domains, we ran an experiment in which the UC Berkeley and UCSB corpuses were combined.

Three results were possible. First, the learned rules that cover the UC Berkeley domain could have poor coverage on the UCSB domain, and vice versa. Second, the learned rules that accurately cover one of the domains could also *inaccurately* cover the other domain. Finally, the learned rules that accurately cover one of the domains could also *accurately* cover the other domain. This last case would occur if the two domains shared a common structure. The first case would occur if the landmarks used to locate tags in one domain were disjoint from the landmarks used to locate tags in the other domain. In either case one or case three, the system would maintain a performance on the mixed domains similar to the performance on each of the individual domains.

In our experiment, however, the second case occurred. As a result, the performance on the mixed domains was much worse than the performance on each individual domain. Because the rules that covered one domain contained many landmarks that appeared throughout the other domain in different contexts, those rules were refined to the point where they ultimately suffered from overfitting. That is, when applied to the testing set, the rules were too specific to achieve a high recall.

7 Conclusion and Future Considerations

In an effort to support the development of the semantic web, we have developed a system that performs automatic DAML markup on talk announcements. Our system is general enough to be applied to other domains as well. We plan to build upon our current system in several ways. First, we plan to incorporate active learning. Not only does active learning decrease the number of training examples needed when learning rules, but also the number of rules learned is reduced, resulting in greater coverage for individual rules. The number of rules learned is reduced because we can select documents for our training set according to which ones produced rules with the greatest coverage. When rules have greater coverage, we feel more confident that they have discovered the regularity that exists in the documents. Our second goal is to expand our system to support ontology elements that are found in lists in the documents. For example, a talk might have several speakers. Another goal is to incorporate into our system more linguistic information so that we can better handle domains in which documents are less structured. By using a system that performs an additional level of markup, labeling things like names, dates, and places, we could take advantage of the additional tokens (the additional markup) available for landmarks when learning rules. Finally, in order to increase the accuracy of learned rules, we have explored using the internet, in particular the Google API, to tell us whether or not our tagged data makes sense.

References

1. Ciravegna, F.: $(LP)^2$, an Adaptive Algorithm for Information Extraction from Web-related Texts. In Proceedings of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining held in conjunction with 17th International Joint Conference on Artificial Intelligence (IJCAI) (2001)
2. Cost, R. S., T. Finin, A. Joshi, Y. Peng, C. Nicholas, I. Soboroff, H. Chen, L. Kagal, F. Perich, Y. Zou, and S. Tolia.: ITtalks: A Case Study in the Semantic Web and DAML+OIL. IEEE Intelligent Systems (2002) 17(1):40-47
3. Hendler, J.: Agents and the Semantic Web. IEEE Intelligent Systems (2001) 16(2):30-37
4. Hendler, J. and D. L. McGuinness: The Darpa Agent Markup Language. IEEE Intelligent Systems (2000) 15(6):67-73
5. Knoblock, C. A., K. Lerman, S. Minton, and I. Muslea: Accurately and reliably extracting data from the web: A machine learning approach. Data Engineering Bulletin
6. Muslea, I., S. Minton, and C. Knoblock: Hierarchical wrapper induction for semistructured information sources. Journal of Autonomous Agents and Multi-Agent Systems (2001)