[1]iASA: Learning to Annotate the Semantic Web

Jie Tang, Juanzi Li, Hongjun Lu, Bangyong Liang, Xiaotong Huang, Kehong Wang

Department of Computer Science, Tsinghua University, Beijing, 100084, P.R.China
{j-tang02, liangby97}@mails.tsinghua.edu.cn, {ljz, x.huang, wkh}@keg.cs.tsinghua.edu.cn

Abstract. With the advent of the Semantic Web, there is a great need to upgrade existing web content to semantic web content. This can be accomplished through semantic annotations. Unfortunately, manual annotation is tedious, time consuming and error-prone. In this paper, we propose a tool, called iASA, that learns to automatically annotate web documents according to an ontology. iASA is based on the combination of information extraction (specifically, the Similarity-based Rule Learner—SRL) and machine learning techniques. Using linguistic knowledge and optimal dynamic window size, SRL produces annotation rules of better quality than comparable semantic annotation systems. Similarity-based learning efficiently reduces the search space by avoiding pseudo rule generalization. In the annotation phase, iASA exploits ontology knowledge to refine the annotation it proposes. Moreover, our annotation algorithm exploits machine learning methods to correctly select instances and to predict missing instances. Finally, iASA provides an explanation component that explains the nature of the learner and annotator to the user. Explanations can greatly help users understand the rule induction and annotation process, so that they can focus on correcting rules and annotations quickly. Experimental results show that iASA can reach high accuracy quickly.

1. Introduction

The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [4, 5]. In recent years, semantic web has made significant progress, in particular through the development of infrastructure such as: ontology language like RDF and OWL, ontology editor like Protégé, and reasoning engine like Racer.

In order to provide semantic web with 'understandable' data, it is necessary to conduct annotation for at least two kinds of metadata. Specifically, commonly used ontologies for semantic web need to be created; and existing web contents need to be upgraded to semantic web content, i.e. semantic annotation. The later issue is exactly the problem addressed in this paper.

Semantic annotation aims to markup the web pages by an ontology, which defines the meaning of contents in the pages. Figure 1 shows an example of semantic

[1] Supported by the National Natural Science Foundation of China under Grant No. 60443002
Supported by the Tsinghua-ITF co-laboratory.

annotation. The inputs of semantic annotation are web document and ontology, the output is the annotated result. In this example, the text “4:00 PM” and “6:00 PM” are annotated as “stime” and “etime” respectively.

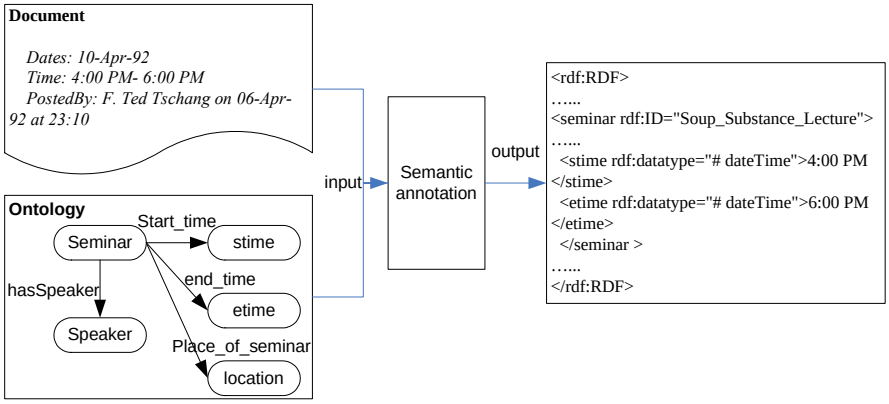


Fig. 1. An example of semantic annotation

Many existing tools have semantic annotation features. However, most of them support only manual annotation [26, 27]. Manual annotation is tedious, time consuming and error-prone. More recent efforts make use of existing wrapping method (e.g. Amilcare [9] and Rapier [7]) and disambiguation technology to automate this process [1, 17, 24, 28, 42, 49]. In information extraction, many models are proposed, such as: Hidden Markov Model [21, 44], Maximum Entropy Model [3, 8], Support Vector Machines [13], and Conditional Random Field [32]. See section 8.4 for details. The methodologies proposed in the previous work can be used in semantic annotation. However, they seem not sufficient for the task.

In this paper, we try to address semantic annotation in a new approach. In our approach, the annotation mainly consists of two stages: learning and annotation.

In learning, we generalize the learned rules. For each rule, we define the extracted text and its context text as features and assign a label. The label represents which type of metadata the extracted text should be annotated. We use the labeled documents to generalize the learned rule set in advance.

In annotation, we identify, extract, and annotate the string in given documents using learned rules.

We propose a method called Similarity based Rule Learner (SRL) to generate the rules. We utilize an empirical method to select the optimal dynamic window size for the rule. We make use of machine learning techniques to improve the annotation results by selecting the correct annotated instances and by predicting the missing annotated instances. Finally, we provide a mechanism for explaining the nature of the rule learner and annotator. The explanation can be very useful in system analysis both for development and usage scenario.

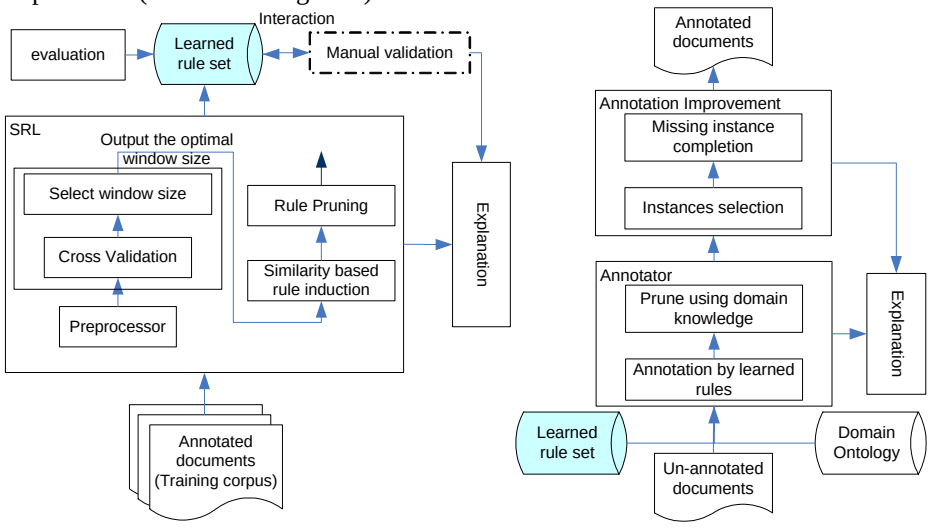
We have developed a tool based on the approach that is call iASA. iASA is targeting structured web data. In iASA, we learn the annotation rules by SRL, and apply the learned rules to un-annotated documents. We also make use of the explanation to help users refine the learned rules or to correct the annotation results.

We conducted the experiments on two data sets, and performed the comparison with existing methods. The experimental results indicate that the proposed method performs well for semantic annotation. We applied the method in a practical project: TIPSI. In TIPSI, we are aimed to extract the semi-structured information from company annual reports for Stock Exchange. Both of the results of the analysis on a user feedback and the result of an analysis on annotation results show that the features in iASA are helpful. We are trying to apply the tool to Contact Search on internet.

The rest of the paper is organized as follows. In section 2, we give an overview of the architecture of iASA and introduce the terminology and notations used throughout the paper. In section 3, we describe our rule learning method: similarity based Rule Learner (SRL). In section 4, we introduce the annotator. In section 5, we propose to improve the annotation results by using machine learning methods and in section 6, we provide a mechanism for explaining the nature of the rule learner and annotator. Section 7 gives our experimental results. Finally, before making concluding marks, we give the survey of related works.

2. iASA: An Automated Semantic Annotator

We perform semantic annotation in four main passes of procedures: learning, annotation, refinement, and the explanation. These procedures correspond to the following four components: SRL, Annotator, Annotation Improvement and Explanation (as shown in figure 2).



(a) Rule Learning (b) Annotator

Fig. 2. The architecture of iASA

In Rule Learning, the input is the annotated documents. We preprocess the annotated documents and construct an initial rule sets. We then use an empirical method to find the optimal window size for each concept's/property's rules. Next, we

employ the similarity based rule induction on the initial rule set and obtain a set of annotation rules. After the pruning procedure, the output is a learned rule set. On the learned rule set, explanation component supports a user interaction, which helps the user to refine the learned rule sets.

In Annotation, the input is un-annotated documents. We apply the learned rules to the un-annotated documents and annotate them according to an ontology. The ontology is defined to represent the annotated or un-annotated documents. After that, we try to refine the annotation results by using machine learning techniques. The output is the annotated documents (an example of annotated document is shown in figure 9). In the procedure, the explanation component supports a user interaction to help the user understand the annotation and the refinement process.

In the rest of this section, we will present the necessary terminologies and definitions.

2.1. Terminology and Notation

The following terminologies are used throughout this paper.

For short, we use entity to denote concept and property in ontology hereafter. Let $E=\{e_i|i\in[1,m]\}$ be a set of entities, where m is the number of entities. In annotated documents, the texts that are annotated as entity e_i are called the instances of e_i. An instance's content includes one or more words/phrases. Let i be an instance and let I denote a set of instances. Notation $< I_i, C_i >= \{< i_{i1}, c_{i1} >, < i_{i2}, c_{i2} >, \cdots, < i_{in_i}, c_{in_i} >\}$ denotes a set of instance-occurring time pairs, in which I_i is the set of instances of e_i and i_{i1} is one possible value of the instance and c_{i1} is the corresponding occurring time of the value in the instance set I_i. n_i is the number of entity in I_i.

2.1.1. Token Definition

1. Token

In both Rule Learning and Annotator, the annotated and un-annotated documents are split into a sequence of tokens $\{t_0, t_1, \ldots, t_n\}$. Each token can be a word (A word is a set of contiguous upper or lowercase letters), a gazetteer entry (e.g. person's first name, currency unit, location, etc) or a name entity (e.g. organization, person' name and date, etc).

Each token is associated with linguistic attributes. Specifically, linguistic attributes of word include: "kind", "orth", "type", "pos", and "name"; linguistic attributes of gazetteer entry (lookup) include: "name" and "type"; and linguistic attribute of name entity is its "name". Table 1 shows the details of the three types of tokens and their attributes. The columns respectively represent token, its attributes, description of the attribute, example of the attribute's value, and description of the example value.

2. Token similarity

According to the attributes of tokens, we define the similarity of two tokens as:

$$Token\_S(t_i, t_j^{'}) = \sum_k w_k match(a_{ik}, a_{jk})$$

where a_{ik} and a_{jk} respectively represent the k-th attribute vaule of token t_i and $t_i^{'}$. Function *match*() is a zero-one function, which ends up with 1 when a_{ik} equals to a_{jk},

0 otherwise. w_k is the weight of the k-th attribute. (In experiments, the weight of attributes "name", "pos", "kind", "type", and "orth" for word are tentatively set as 0.45, 0.25, 0.1, 0.1, and 0.1, respectively; the weight of attributes "name" and "type" for gazetteer are tentatively set as 0.7 and 0.3.)

Table 1. Details of the three types of tokens and their attributes

Token	Attribute	Comment	Examples	Value Description
Word	Name	Original string	"Time"	
	Orth	Orthography of the word	"OneHyphen"	Indicates the word is a simple hyphen ("-" or "_")
			"AllCap"	Indicates all letters in the word are capitalized ("CHINA")
			"Capitalized"	Indicates the first letter is capitalized ("Time")
	Kind	Kind of the word	"punctuation"	Indicates the word is a punctuation (",", or ".")
			"word"	Normal word ("Time")
			"number"	Indicates the word is numeric ("486")
	POS	Part-Of-Speech of the word	"NN"	Singular common noun with word initial capital
			"CD"	Cardinal number (one, 100)
	Type	Indicate whether word is a space token or not	"Spacetoken"	Indicates the word is a enter or space
			"Token"	Other words except whose type is "spacetoken"
Lookup (gazetteer entry)	Name	Major type of the gazetteer entry	person's first name	E.g. "Jeanne"
			Organization	E.g. "Motorola"
			Location	E.g. "Oregon"
	Type	Minor type of the gazetteer entry	Female	Minor type of person's first name
			company	Minor type of organization ("Motorola")
			Province	Minor type of location ("Oregon")
Name entity	Name	Recognized over the original words.	Organization	Includes company or governmental organization ("Oracle", "Intel")
			person's name	E.g. "Jeanne Heembrock"
			date	E.g. "17 Nov 1996"

3. Pattern definition

A pattern is a sequence of tokens with length n (n could be zero). Formally, a pattern can be written as:

$$pattern = \{t_0, t_1, \cdots, t_n\}$$

where t_i is the *i*-th token in the pattern.

As the example in figure 1, the string "4:00 PM" is annotated as the property *stime*. It can be viewed as a body pattern (defined in the following section) containing one token: name entity "date" or a body pattern containing four word tokens: "4", ":", "00", and "PM").

We also define similarity between the two patterns. See section 3.3 for details.

2.1.2 Rule definition

Semantic annotator needs abstract patterns that are defined based on all the features potentially helpful for the ontology annotation. These abstract patterns are defined as rules. In iASA, the rule is represented in XML.

Each rule has an entity name and consists of three parts: 1) left pattern: it is used to match the text that immediately precedes the instance (*w* tokens to the left); 2) body pattern: it corresponds to the instance of an entity (tokens contained); 3) right pattern: it is used to match the text that immediately follows the instance (*w* tokens to the right).

Figure 3 shows an example of a rule of entity *etime*. In the rule, Left, body and right pattern are denoted by "leftpattern", "bodypattern" and "rightpattern", respectively, and the token in these patterns is denoted by "tag". The attribute "indicator" of "tag" denotes the type of the token. The type can be name entity, gazetteer entry (lookup), or word. Attributes "kind", "name", "orth", "type", and "pos" represent the attributes of corresponding token (as shown in table 1). We use "<tag indicator="unknown"/>" to denote a placeholder.

```
<rule name="etime" no="12">
 <leftpattern>
  <tag indicator="word" kind="word" name="Time" orth="O: Capitalized" type="token" />
  <tag indicator="word" kind="punctuation" name=":" orth="O: OtherPunct" pos=":" type="token" />
  <tag indicator="unknown" />
  <tag indicator="word" kind="punctuation" name="-" orth="O: OneHyphen" pos=":" type="token" />
 </leftpattern>
 <bodypattern>
  <tag indicator="nameentity" name="date" />
 </bodypattern>
 <rightpattern>
  <tag indicator="word" name="" type="spacetoken" />
  <tag indicator="word" kind="word" orth="O: Capitalized" type="token" />
  <tag indicator="word" kind="punctuation" name=":" orth="O: OtherPunct" pos=":" type="token" />
 </rightpattern>
</rule>
```

Fig. 3. An example of rule

In order to learn the target rules, the user typically needs to provide a set of annotated documents (also called training documents). In the training documents, we label a set of training instances in advance. Each instance is enclosed by a start tag (e.g. <stime rdf:datatype="http://www.w3.org/2001/XMLSchema#string">) and an end tag (e.g. </stime>). Context of annotated instance (viz. leftpattern, bodypattern and rightpattern) is constructed according to the start and end tags. Specifically, the text that enclosed by a pair of tags is viewed as an instance and is transformed into a

bodypattern; *w* tokens that precedes the instance are transformed into leftpattern; and *w* tokens that follows the instance are transformed into rightpattern.

3. SRL—Similarity based Rule Learner

SRL has four main modules: preprocessor, rule set initialization, rule induction, and rule set pruning. The input is annotated documents. Preprocessor exploits Natural Language Processing (NLP) techniques to process the documents. Rule set initialization takes the preprocessed corpus as input and generates the initial rule set using dynamic window size based context. Rule induction performs rule generalization in an iterative mode. In each iteration, we select pairs of rules with high similarity in the initial rule set according to the rule similarity (see section 3.3 for definition of rule similarity), generalize new rules, evaluate each new rule, and then insert the new rule into the learned rule set if it survives the pruning phrase. Finally, SRL outputs the learned rule set.

3.1. Preprocessor

In preprocessing, we use NLP techniques to process the document, which has been proved to be useful for machine learning and information processing [7, 9]. NLP associates additional knowledge to each word in the document.

We make use of GATE as the NLP toolkit [14]. GATE is a general toolkit for text processing. It integrates many tools for NLP, including morphological analyzer, a POS tagger, gazetteer lookup, and named entity recognition (recognition of person name, dates, number and organization names, etc). For example, in the document snippet: "…; Patrick Stroh, assistant professor, SDS…", "Patrick Stroh" is annotated as an instance of entity *speaker*. After processing by GATE, each word is associated with linguistic knowledge: part of speech (POS), token kind (Kind), lookup, and name entity, etc. Table 2 gives an example about how GATE provides the linguistic knowledge.

Table 2. An example of preprocessed result with NLP knowledge

Instance with NLP Knowledge					**Annotated as**
Word	**POS**	**Kind**	**Lookup**	**Name Entity**	
;	:	Punctuation			
Patrick	NNP	Word	Person's first name	Person	Speaker
Stroh	NNP	Word			
,	,	Punctuation			
assistant	NN	Word	Jobtitle		
professor	NN	Word			
,	,	Punctuation			
SDS	NNP	Word			

The linguistic knowledge seems very useful in rule induction. For example, tags in the rule can be relaxed by substituting constraints on words by constraints on some parts of the linguistic knowledge. In this way, we can generalize the rules by name entity (e.g. "person's first name (male)") or POS (e.g. "NNP") instead of only flat words.

3.2. Rule Set Initialization

For rule induction, we first need to construct the initial rule set that contains the mostly specified rules. In annotated documents, each instance is enclosed by a start tag and an end tag. We transform the instance and its context in the preprocessed document into an initial rule. Existing methods usually define a window of fixed size (size=*w*) as the instance context and build the initial rule by using *w* tokens to the left and *w* tokens to the right for a given annotated instance.

In the rule set initialization, we exploit the linguistic attributes, and we also use the dynamic window size by an empirical method.

1. Linguistic knowledge based context

Suppose *w* is "3", for the annotated instance of *stime* in figure 1, table 3 gives a comparison of linguistic knowledge based context and word based context. The second column represents the linguistic knowledge based context and the third column represents the flat word based context.

Table 3. Linguistic knowledge based context vs. word based context

Initial Rule	Linguistic Context	Words' Context	Semantic Entity
Left Pattern	Date (name entity)	"10-Apr-92"	
	"Time" (word)	"Time"	
	":" (punctuation)	":"	
Body Pattern	Date (name entity)	4:00 PM	*stime*
Right Pattern	"-" (word)	"-" (word)	
	Date (name entity)	"6:00 PM" (word)	
	Return	Return	

In table 3, we see that by the comparison of the word based context, the linguistic knowledge based context substitutes a name entity 'Date' for both '10-Apr-92' and '6:00 PM'. The linguistic knowledge based context seems more reasonable, because the rules derived from it intuitively can be applied to broader cases.

2. Optimal dynamic window size based context

Analysis on our preliminary experimental results shows that different entities prefer to contexts with different window sizes. For example, in the task of annotating CMU Seminar announcements, with the increase of window size (tested from 2 to 8), the performance (evaluated by F-measure) for entity *etime* becomes better; however for entity *stime*, the performance becomes worse.

We perform the window size selection for each entity by using cross-validation, a typical approach for experimental selection [43]. Cross validation is a commonly used

technique in machine learning to prevent bias. The main idea is based on the following assumption: if the rules learned from a subset of the training data produce accurate result, it is also likely that it will produce highly accurate predictions when trained on the entire dataset.

In this method, to determine w for each entity, we evaluate the performance on training corpus with contexts of different window sizes using cross-validation. Specifically, let T be the training corpus. The examples in T are first randomly divided into d equal parts T_1, T_2, ⋯, T_d (we use d = 10 in our experiments). Next, for each part T_i, $i \in [1,d]$, we try to learn the rules from the other (d-1) parts, then apply the learned rules to the examples in T_i. Finally, we select the window size that performs best for each entity. Section 7 will show the comparison of dynamic window size and fixed one.

3.3. Rule Induction

The input of rule induction is an initial rule set, rule induction aims to learn rules over the initial rule set and to output a learned rule set.

Within rule induction, the following problems should be considered:

- Tokens in body patterns may be very sparse, which make it difficult for generalization. For example: instances of *address*: “Baker Hall 237B”, “room 1001”, “WeH 5403”, etc.
- Sometimes token in the rule may be only a placeholder instead of a meaning ones, e.g. in examples: “the speaker is <speaker>”, “the speaker: <speaker>”, and “the speaker, <speaker>”. A placeholder can be defined so as to match “is”, “:” and “,”.
- Each learned rule should be scored. The higher the score is, the higher probability that the rule is accepted in the final rule set.

In regular expression language, “*” usually is exploited to represent any group of characters and “?” usually is exploited to represent any character. In our rule definition, we extend this idea slightly and make use of “*” and “?” to respectively represent any group of tokens and any token. In this way, for the first problem, we use “*” to represent the body pattern and for the second problem, we use “?” to represent the placeholder.

Unfortunately, most of the existing rule induction methods do not have the feature of generalizing the wildcard “*” and “?”. In this section, we introduce how SRL solves these two problems. Moreover, we describe the evaluation metric that is used to score each learned rule.

1. Rule similarity

Rule similarity is the basic idea in SRL. SRL runs in an iterative mode. In each iteration, we always try to select the most similar pair of rules for generalization. Califf et al adopt a random strategy for the rule selection [7]. The similarity based selection method seems more reasonable. There are two reasons for this: similarity based method is more efficient than random method by avoiding the pseudo generalization and the learned rules by the random method may be inconsistent.

We then need to define the similarity of a pair of rules. Typically, a rule can have three patterns (see section 2.1.2 for the definition). We calculate the similarities of the

three corresponding patterns and sum them into an aggregate one. But soon, we found that there are mainly two kinds of rules: rules with sparse body patterns and rules with non-sparse body patterns. Examples of the former include rules of entity *speaker*; examples of the later include rules of entity *platform* or entity *database*.

Our proposal is that similarity of rules with sparse body patterns are calculated by the similarities of corresponding left patterns and right patterns, and similarity of rules with non-sparse body patterns includes only the similarity of body patterns. We, therefore, define rule similarity as:

$$sim(r_1, r_2) = \begin{cases} sim(lp_1, lp_2) + sim(rp_1, rp_2), & sparse(bp) > \mu \\ sim(bp_1, bp_2), & sparse(bp) \le \mu \end{cases}$$

where r_1 and r_2 are two rules. $sim(lp_1, lp_2)$, $sim(bp_1, bp_2)$, and $sim(rp_1, rp_2)$ respectively represent the similarities of corresponding left patterns, right patterns, and body patterns in rules r_1 and r_2. $sparse(bp)$ is a measurement indicating whether the body pattern is sparse or not. It is calculated by $count(value)/count(instance)$. $count(value)$ is the number of instance values for the given entity, and $count(instance)$ is the total number of instances (e.g. "4:00 PM" and "4:00 PM" are two instances of *etime* with only one value). Parameter μ is a threshold (we tentatively set it as 0.5).

2. Pattern similarity

We calculate the similarity of patterns by a recursive procedure called Multiple Layer Recursive Matching (MLRM) algorithm. This idea is derived from [46].

In MLRM algorithm, the input is two patterns: $pattern_1$ and $pattern_2$. Output is the similarity score of the two patterns.

The MLRM algorithm is described in Figure 4.

```
Input: pattern1[t0,t1,…,tn]
       pattern2[t0',t1',…,tm'];
Output: Seq_S(pattern1,pattern2); //pattern similarity
MLRM(pattern1[t0,t1,…,tn], pattern2[t0',t1',…,tm'])
{
   if(either pattern1 or pattern2 is empty or no more higher layer)
      return 0;
   A=0;B=0;
   while(A<=n and B<=m)
   {
      for( i from A to n )
        for( j from B to m )
          select the most similar token pair : argmax(Token_S ( ti, tj'));
          {
              Seq_S(pattern1,pattern2) += Token_S ( ti, tj') * weightk;
              Seq_S(pattern1,pattern2) += MLRM(pattern1[tA,…,ti-1],
                                          patterr2[tB',…,tj-1']);
              A=i+1; B=j+1; break to while;
          }
   }
   Seq_S(pattern1, pattern2) += MLRM(pattern1[tA,…,tn],
                                          patterr2[tB',…,tm']);
   return Seq_S(pattern1, pattern2);
}
```

Fig. 4. The MLRM algorithm

MLRM uses the function *MLRM*() for estimating the similarity between two patterns. $Token\_S(t_i, t_i')$ calculates the similarity between token t_i and t_i' using their

linguistic information (token similarity is defined in section 2). *Seq_S* is the similarity of the two patterns by aggregating the similarities of individual pairs of tokens. MLRM recursively computes similarity between two patterns at multiple layers with different similarity weights ($weight_k$). MLRM attempts to find a pair of the most similar tokens from $pattern_1$ and $pattern_2$. In each layer, the pair of tokens essentially divides each pattern into two sub-patterns. MLRM continues the process until one of the sub-patterns becomes empty. The weight at each layer is currently empirically assigned to reflect the relative importance of the token similarity in that layer. We tentatively set the weights w_1, w_2, w_3, and w_4 as 10, 9, 8 and 7 respectively.

Figure 5 illustrates MLRM with an example. At step 0, a pair of the most similar tokens is found (connected by a dark line) and divides each pattern into two sub-patterns. Each pair of sub-patterns is recursively processed by MLRM as indicated in step 1. In this step, two additional similarity pairs are found and the two patterns are further divided into four sub-patterns to be processed recursively at step 2. MLRM stops when no more nodes are left in the sub-patterns.

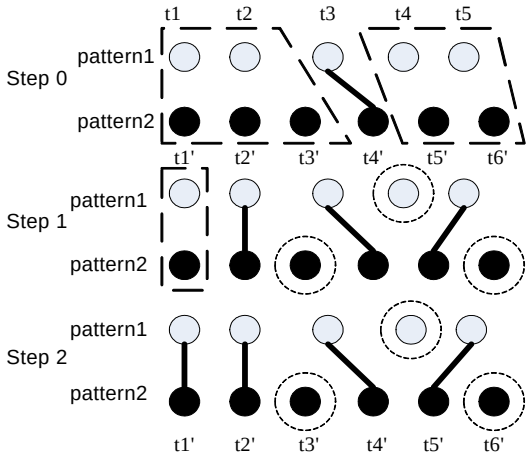


Fig. 5. An example of recursive matching in MLRM

3. Rule generalization

The task of rule generalization is to induce new rules from the pair of similar rules. The new rules should cover the pair of rules.

In rule generalization, we take a strategy of divide-and-conquer. For a pair of similar rules, our method is to generalize the three pairs of patterns (i.e. left patterns, body patterns, and right patterns), and then combine the generalized patterns into a new rule. For each pair of patterns, the generalization is performed by starting from constrains of all linguistic information on tokens to some relaxation. The algorithm of rule generalization is shown in figure 6.

In rule generalization, we first select the most similar rules by *getSimilarRules*(). The number of selected rules is not necessary two. Hence the algorithm calls function *getRealPairs*() to generate pairs of rules. Then, for each pair of rules, the algorithm uses *generalizePatterns*() to generalize left pattern, body pattern, and right pattern respectively. *generalizePatterns*() returns a collection of possible induced patterns for

the two input patterns (note: for two patterns, there might induce multiple patterns with different linguistic information). Function *getAllPossibleRules*() returns all possible rules by combining the three collections of patterns. For each rule, algorithm evaluates it both on the initial rule set and on the original training corpus, and then adds it to the learned rule set if its score exceeds the minimal rule score (a threshold predefined). (For the detail of rule evaluation, please refer to the following section.)

```
RuleInduction(RuleSet ruleset)
{
    rssimilar=getSimilarRules(ruleset);
    collect_RP=getRealPairs(rssimilar);
    foreach rulepair in collect_RP
    {
        rule1 and rule2 in rulepair
        col_leftpat=generalizePatterns(rule1.leftpattern, rule2.leftpattern);
        col_bodypat=generalizePatterns(rule1.bodypattern, rule2.bodypattern);
        col_rightpat=generalizePatterns(rule1.rightpattern, rule2.rightypattern);
        col_rules=getAllPossibleRules(col_leftpat,col_bodypat,col_rightpat);
        foreach rule in col_rules
        {
            score=EvaluateRule(rule);
            if( score > minRuleScore)
                col_learnedrules.add(rule);
        }
    }
    return col_learnedrules;
}
```

(a) Generalization of rules

```
generalizePatterns ( Pattern pattern1, Pattern pattern2)
{
    if( pattern1==null or pattern2==null)
        return null;
    relaxlevel = 0;
    while( true )
    {
        collectpair = getMatchedTagPair(pattern1,pattern2, relaxlevel);
        newpattern = getPattern(pattern1, pattern2, collectpair );
        pattern_collection.add( newpattern );
        relaxlevel++;
        if( relaxlevel>maxrelax )
            break;
    }
    return pattern_collection;
}
```

(b) Generalization of patterns

Fig. 6. The algorithm of rules generalization

In pattern generalization, *generalizePatterns*() searches for the matchable pairs of tokens (by function *getMatchedTagPair*()). The search starts from full constraints (*relaxlevel*=0) to maximum relaxation on linguistic information (*relaxlevel*= *maxrelax*-1). In the case of full constraints, two tokens are considered to be matched only when all of their attributes are equal, including “indicator”, “kind”, “name”, “orth”, “pos”, and “type” (see table 1 for details). With the increase of *relaxlevel*, the matcher relaxes the condition by the order of “name”, “pos”, “kind”. By relaxing the condition, we mean ignoring the corresponding attributes in the matching process. Parameter *maxrelax* is the stop condition to control the relaxation progress. The discovered matchable token pairs are used as the initial points to generalize the pattern. The function *getPattern*() returns the generalized results of the two patterns based on their matchable token pairs. Figure 7 gives an example to illustrate the strategy used in *getPattern*().

In this example, t2-t2’, t3-t4’ and t5-t5’ (linked by dark line) are three matchable token pairs returned by function *getMatchedTagPair*(). The three pairs of tokens divide the two patterns into four token groups: (t1-t1’), (t3’), (t4) and (t6’). For each token group, tokens in the two patterns are induced one by one from left to right for body pattern and right pattern or from right to left for left pattern. As for uneven length, such as (t3’), (t4) and (t6’), the algorithm adds wildcard “?” to the remainder tokens. Symbol C(t1,t1’) means the induction of two tokens: t1 and t1’. “(t3’)?” indicates that the token t3’ can either occur or absent in the target cases.

RuleInduction() is run iteratively until the stop condition is met. The stop condition is that no more general rules can be induced. Finally, rule generalization outputs the learned rule set.

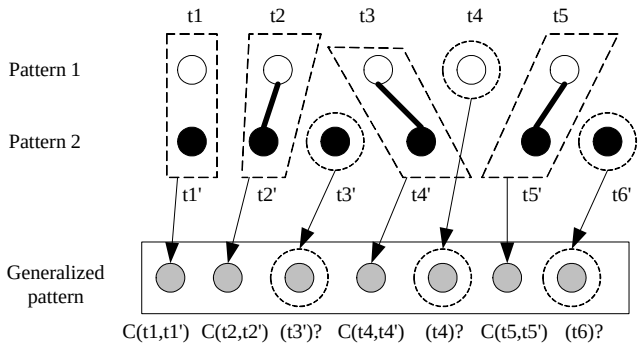


Fig. 7. An example to describe how two patterns are generalized in *getPattern*()

4. Rule evaluation

Each rule is scored by the function *Evaluation*() (shown in figure 6(a)). The score is the combination of two factors: score on the initial rule set and score on the original training corpus. Both scores are evaluated by F-measure. F-measure is defined as:

$$F = \frac{1+\beta}{\frac{\beta}{precision} + \frac{1}{recall}}$$

where *precision* is the percentage of correctly annotated instances in the annotated instances; *recall* is the percentage of correctly annotated instances in the correct instances. Parameter β indicates the degree of preference on precision/recall.

We call the score on the initial rule set as *rulescore*. We take the annotated instances of current entity as positive examples and instances of the other entity as negative ones. Score of each rule is evaluated by F-measure on the initial rule set. The metric *rulescore* should concern more precision than recall, because the generalized rule may only cover the two source rules after the first iteration, which leads to a very low recall. On the other hand, low precision indicates that the new rule covers negative examples. Therefore we set the weight β as 16.

We call the score on original training corpus as *realscore*. We apply the learned rule on the original training corpus (i.e. take training corpus set as test data), which can avoid over-learned rules to a certain extent. The over-learned rules may produce good result on the initial rule set, but they can also import noise. Such noise may lead to low precision. For calculating *realscore*, we set the parameter β as 1.

Finally, by multiplying *rulescore* by *realscore*, we obtain the final score of the new rule.

3.4. Rule Set Pruning

In rule set pruning, we aim to remove the redundant and unreliable rules in the learned rule set.

To remove redundant rules, it is necessary to define what a redundant rule is. We give the definition of redundant rule.

Redundant Rule: If a rule is covered by other rule(s) in the rule set, then this rule is redundant. In other words, if all instances that annotated by this rule can also be annotated by other rule(s), then this rule is a redundant rule.

In terms of the definition, we developed methods to judge whether a rule is redundant. Our methods include two steps: (1) judging whether a rule is covered by another rule; (2) judging whether a rule is covered by other rules. We exploit the annotated instances to judge whether a rule is covered by other rules.

Whether or not a rule is covered by another rule is based on the observation: if every pattern of a rule is more general than the corresponding pattern of another rule, we say that the latter rule is covered by the former rule.

Figure 8 gives an example. In the example, pattern 2 is covered by pattern 1 because token (<tag type="">) of pattern 1 is more general than token (<tag type="token">). <tag type=""> denotes a token that can be any type.

Pattern 1
```
<pattern>
  <tag type="" >
</pattern>
```

Pattern 2
```
<pattern>
  <tag type="word" >
</pattern>
```

Fig. 8. An example of two patterns

We exploit the annotated instances to determine whether a rule is covered by other rules: if instances annotated by a rule can be also annotated by some other rules, we say that the rule is a redundant rule.

To remove unreliable rules, we make use of ontology knowledge. We use entity type to infer whether the rule is reliable or not. For example, in the seminar ontology the property *stime* is defined as:

```
<owl:DatatypeProperty rdf:ID="stime">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime" />
  <rdfs:domain rdf:resource="# seminar " />
</owl:DatatypeProperty>
```

Then we use the date type *dateTime* to judge whether the body pattern of a rule conforms to it or not. After pruning, SRL outputs the final learned rule set.

4. Annotator

Annotator takes the un-annotated documents as input, preprocesses them by NLP which is the same as that in SRL, applies the learned rules on them, and annotates the documents according to an ontology.

An example of output by the annotator is shown in figure 9. The format conforms to the ontology standard language OWL DL [15], which can facilitate further reasoning process.

Using iASA in the experiments, we have found that domain knowledge can greatly improve the accuracy of annotation. Domain knowledge can be used to evaluate annotations and prune wrong annotations, and it is also helpful for directing the search process. In semantic annotation, domain knowledge is usually represented by ontology. We then use the restrictions in the ontology for improving the annotation.

```
<rdf:RDF>
 <seminar rdf:ID="Soup_Substance_Lecture">
  <location rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Wherrett Room, Skibo</location>
  <stime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">12:00 PM</stime>
  <etime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">1:00 PM</etime>
  <hasSpeaker  rdf:resource="#Erik_Devereux" />
      <speaker rdf:ID="Erik_Devereux">
        <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Erik Devereux</name>
      </speaker >
  </hasSpeaker >
  <hasSpeaker >
      <speaker rdf:ID="Patrick_Stroh">
        <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Patrick Stroh</name>
      </speaker >
  </hasSpeaker >
  <hasSpeaker rdf:resource="#Richard_Smith" />
      <speaker rdf:ID="Richard_Smith">
        <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Richard Smith</name>
      </speaker >
  </hasSpeaker >
 </seminar >
</rdf:RDF>
```

Fig. 9. An example output of annotator

Domain restriction: The constraints that are defined in the ontology include “date type”, “cardinality”, “minCardinality”, etc. Table 4 shows examples of domain restrictions currently used in iASA.

Table 4. Examples of domain restrictions that are exploited to improve the annotation

Constraint Types	Examples
date type	If the data type of annotated instance does not match the data type of entity *x* in ontology, then remove the instance.
Cardinality	If cardinality of entity *x* is 1, and annotator finds multiple candidate instances, then select the candidate annotated by the best rule (with highest score) as the instance of entity *x*.
	If cardinality of entity *x* is 1, and annotator doesn’t find any candidates, then relax the condition in the rule and search the document again (the relaxation strategy is same as that in rule induction).

5. Improving the Annotation by Machine Learning Methods

In annotation, we met two problems: correct instances selection and missing instances prediction. In this section, we try to make use of machine learning methods to solve the two problems.

5.1. Problem Statement

Now we give the definitions of the two problems.

(1) Correct instance selection. In automatic annotation, entity may be annotated with multiple instances even in one document. Some annotated instances are correct, but some may be wrong. The problem is how to identify the correct instances.

(2) Missing instance prediction. There may be some instances that are not annotated by automatic annotation, we call it missing instances. Since typically recall of semantic annotation is significantly lower than its precision, it is indeed necessary to deal with the problem.

In existing methods for correct instances selection, Califf et al propose to select the instance(s) that annotated by the highest scored rule [7]. The method can solve some problems, but it is not sufficient, because: (a) even the same rule might annotate multiple instances including both correct and erroneous ones; (b) some correct instances can be annotated by the other rules instead of the highest scored rule.

In existing methods for missing instance prediction, Nahm et al propose to induce predictive rules which are then used to predict the missing instances [39]. The method mines the association rules from the data. For example, suppose following rule is discovered from data on entity *program language* and *topic area*: "SQL"$\in$*language* $\rightarrow$"Database"$\in$*area*. If the annotation system annotated only "SQL" for language, but failed to annotate "Database" for *area*, then the method can assign the "Database" as the value of *area*.

In this paper, we formalize the correct instance selection problem as that of classification. When selecting the correct instances, we use a classification model to identify whether or not an instance is correct.

For missing instance prediction, it is difficult to accurately predict the missing instances that have random values, even by human. We confine ourselves to predict the instances that have enumerative values in missing instance prediction. It seems reasonable for predicting instances that have enumerative values, because we have observed that 17.4% of the entities have enumerative values in our experimental data.

We then formalize the missing instance prediction problem as that of multi-class classification. When predicting a missing instance, we use a classification model to predict which value has the highest probability as the missing instance. The classification model is trained using the training data in advance.

5.2. Classification Model

We make use of SVM (Support Vector Machines) as the classification model [47].

Let us first consider a two class classification problem. Let $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ be a training data set, in which x_i denotes an instance (a feature vector) and $y_i \in \{-1,+1\}$ denotes a classification label. In learning, one attempts to find an optimal separating hyper-plane that maximally separates the two classes of training instances (more precisely, maximizes the margin between the two classes of instances). The hyper-plane corresponds to a classifier (linear SVM). It is theoretically guaranteed that the linear classifier obtained in this way has small generalization errors. Linear

SVM can be further extended into non-linear SVMs by using kernel functions such as Gaussian and polynomial kernels.

We choose polynomial kernel, because our preliminary experimental results show that it works best for our current task. When there are more than two classes, we adopt the "one class versus all others" approach, i.e., take one class as positive and the other classes as negative.

Now, we have two kinds of "instances": annotated instance of an entity and instance in SVM model. To distinguish them from each other, we use sample to denote instance in SVM hereafter.

5.3. Correct Instance Selection

We divide the problem of instance selection into three categories:

(1) Multiple instances are annotated by the same rule. For example, "a MIS Manager" and "MIS Manager" might be annotated as "title" by a same rule. (The two examples are instances of entity "title" in the task of misc.job.offered. Correct one should be "MIS Manager".)

(2) Multiple instances are annotated by different rules. For example, "Marian D' Amico", "Charles E. Leiserson", and "Jeffrey V. Hill" may be annotated as "speaker" by different rules. (The three examples are instances of entity "speaker" in CMU seminar announcement. Correct one should be "Charles E. Leiserson".)

(3) Hybrid of the two situations. For example, "Leiserson", "Charles E. Leiserson", and "to Charles" may be annotated as "speaker", in which "Leiserson" and "Charles E. Leiserson" are annotated by a same rule, and "to Charles" is annotated by another rule. (The three examples are instances of entity "speaker" in CMU seminar announcement. Correct one should be "Charles E. Leiserson")

The method of selecting by highest scored rule can only deal with the problem of the second category, and will fail on the other two categories. Even on the second category, the method may fail when the correct instance is annotated by a lower scored rule.

We view the instance selection problem as that of classification. The correct instance selection consists of two stages: training and identification.

In identification, when problem of multiple instances occurs, we identify whether or not each instance is correct using SVM model. Then we rank the instances by scores output by SVM. We select the instance ranked top as the correct one.

In training, we construct the SVM model that can be used to identify the instance. In the SVM model, we view an instance as a sample in SVM. For each sample, we define a set of features and assign a label. The label represents whether the instance is correct or not. For each entity, we use instances of the entity in the annotated documents as positive samples for SVM and the others as negative samples. We use the labeled samples to train the SVM model in advance for each entity.

We view each instance as a 'document', and convert the 'document' into a bag of words. We apply stop-word filtering and word stemming on the bag of words. After that, we construct an attribute-value representation of the 'document'. Each distinct word w_i corresponds to a feature with its value. For the feature value, we use $TF(w_i, x)*IDF(w_i)$, a typical method to estimate the word weight in that document. $TF(w_i, x)$

represents the frequency of word w_i occurs in the document x. $IDF(w_i)$ represents the inverse document frequency of a word. It is defined by:

$$IDF(w_i) = \log \frac{n}{DF(w_i)}$$

here n is the total number of instances. $DF(w_i)$ is the number of documents the word w_i occurs in.

Finally, for multiple instances, we obtain a ranked list, and then we do the selection according to the following rules:

(1) For entity whose cardinality is unique, the top ranked instance is selected as the correct one.

(2) For entity whose cardinality a is greater than 1, the a top ranked instances that do not overlap with each other are selected as correct ones. No overlap is very important. Considering the examples above, instances "a MIS Manager" and "MIS Manager" might both obtain a higher scores compared to other instances. No overlap rule can further remove "a MIS Manager".

(3) For entity whose cardinality is multiple, instances that are classified as positive samples and do not overlap with each other are selected as correct ones.

5.4. Missing Instance Prediction

In missing instance prediction, we aim to predict the missing instances that have enumerative values (we judge whether an entity has enumerative values in terms of its definition in ontology). We view the problem of missing instance prediction as that of multi-class classification. Here, values of the missing instances correspond to the classes in the classification. It also consists of two stages: training and prediction.

In prediction, we predict the missing instances by using the extracted instances. Specifically, for an entity with enumerative value type, we view the annotated document as a sample in SVM, and then predict the value which the entity should have. We use the value that has the highest score output by SVM as the missing instance.

In training, we construct the SVM models that can be used to predict the value. In SVM model, for an enumerative valued entity, we view each annotated document as a sample. For each sample, we define a set of features and assign a label. The label represents all possible values of the entity. For each label of an entity, we use the annotated documents with this label as the positive samples for SVM and the others as negative samples. We use the labeled samples to train the SVM models in advance for each enumerative valued entity.

To represent the features for each sample, we view each annotated document as a sample. For an annotated document, we first extract all the annotated instances from the annotated document. Next, the instances are converted into a bag of words. After that we use the same method as that in the correct instance selection to prepare the attribute-value representation for each sample.

Finally, for a missing instance of the enumerative valued entity, we obtain scores from SVM for each possible value. We complete the missing instances by the following rules:

(1) For entity whose cardinality is unique, the value with the highest score is selected as the missing instance.

(2) For entity whose cardinality *a* is greater than 1, *a* values that have higher scores are selected as the missing instances.

(3) For entity whose cardinality is multiple, only the value with the highest score is selected as the missing instance.

6. Explanation

In iASA, we try to provide an environment for user to quickly annotate the web pages according to an ontology. Practically, the user needs to inspect the learned rules and the annotation results produced by the system, modify them and provide feedbacks to the system.

As an annotation system relies on complex algorithms, there is a requirement for the system to explain the nature of the generated rules to the user. This idea is derived from [16]. Explanations can greatly help user gain insights into the rule induction and annotation process. In this way, the user can easily focus on the correct rules and the annotation results.

Figure 10 is an example of the scenario. It gives a learned rule for entity *etime* by iASA.

```
<rule name="etime" no="12">
 <leftpattern>
  <tag indicator="word" kind="word" name="Time" orth="O: Capitalized" type="token" />
  <tag indicator="word" kind="punctuation" name=":" orth="O: OtherPunct" pos=":" type="token" />
  <tag indicator="unknown" />
  <tag indicator="word" kind="punctuation" name="-" orth="O: OneHyphen" pos=":" type="token" />
 </leftpattern>
 <bodypattern>
  <tag indicator="nameentity" name="date" />
 </bodypattern>
 <rightpattern>
  <tag indicator="word" name="" type="spacetoken" />
  <tag indicator="word" kind="word" orth="O: Capitalized" type="token" />
  <tag indicator="word" kind="punctuation" name=":" orth="O: OtherPunct" pos=":" type="token" />
 </rightpattern>
</rule>
```

Fig. 10. An example rule

This rule produces well accepted results on training corpus but low recall on the test corpus. The user is uncertain about how to improve this rule or whether or not the rule should be removed from the rule set. He wants iASA to explain how the rule is generalized and how this rule is evaluated.

iASA induces rules from three initial rules and also shows how the tokens are generalized. For example, the third token in Figure 10 (<tag indicator="unknown"/> indicates a placeholder) in the left pattern of the rule is generalized from the tokens: a name entity (date: "Jan, 15th, 2004"), a word ("12") and a gazetteer entry (also called lookup) (time: "4:30"). This rule is tested on the initial rule set by score: 0.987 and on

training corpus by score 0.87. The final score is 0.86, which is higher than the minimum rule threshold.

In the rest of this section, we will describe the data structure in explanation module and introduce what kinds of questions can be answered in the explanation.

6.1. Data Structure in Explanation

The key data structure underlying the explanation component of iASA is the dependency graph, which is constructed during the inducing process and the annotation process. The dependency graph records the flows of induction, data, and input and output of each system component. The nodes of the graph are: annotated documents, initial rules, similar rules, selected similar rule pairs, generalized pattern collections, all possible rules by the three generalized patterns, scores on the initial rule set and the training corpus.

Two nodes in the graph are connected by a directed edge if one of them is the successor of the other in the induction process. The label of the edge is the system process.

In explanation, we define an abstract node, which can be written as a tuple:

Abstract_Node=<super_edges, *node_type*, *node_data*, *sub_edges>*

The four elements in the tuple respectively represent a set of link-in edges, type of current node, data stored in the node, and a set of link-out edges. By link-in edge, we mean the directed edge that one predecessor of the current node links to the current node. By link-out edge, we mean the directed edge that the current node links to one successor. The type of current node can be one of rule, rule pair, pattern, tag, and annotated instance. The data corresponding to the *node_type* is stored in *node_data*.

We also define an abstract edge, which can be written as a tuple:

Abstract_Edge=<subject_node, *edge_type*, *edge_data*, *object_node>*

The four elements in the tuple respectively represent a subject node, type of the edge, data stored in the edge, and an object node. By subject node, we mean a node that the directed edge comes from. By object node, we mean a node that the edge directed to. The type of current edge can be one kind of processes in iASA, e.g. rule similarity computing, similar rule selection, rule induction, pattern generalization, rule scoring on initial rule sets, rule scoring on original training corpus, annotation, etc. The data corresponding to the type of the edge is stored in *edge_data*.

We make implementations for the nodes and the edges. In the implementation of each type of node, we extend the abstract node and define the corresponding data structure for storing information that is required in the explanation. In the implementation of each type of edge, we extend the abstract edge and define the corresponding data structure.

We take rule-node (implementation of node for rule) as the example to describe how we define the data structure for explanation. A rule has three patterns: left pattern, body pattern, and right pattern. The rule links to the three patterns by three link-out edges. A rule has three scores: score on initial rule set, score on original training corpus, and the final score. The three scores are defined as attributes in the rule-node. A rule has a semantic tag indicating which entity the rule is used to annotate. The semantic tag is also defined as an attribute. A rule has an attribute

"number-of-covered-instance" indicating how many instances the rule can annotate in the training documents. Moreover, a rule has a global unique ID. Through link-in edges, a rule can have a set of predecessor. We can find one kind of predecessor by looking up the corresponding edge type in the set of edges. For example, if we want to find the pair of rules that are used to directly generalize into the current rule, we can lookup the link-in edges by "rule induction", the system returns a sub set of edges. We next query the subject nodes of the returned edges and can get the pair of rules. Traversing up in this way, we can get all rules that are used to generalize the current rule.

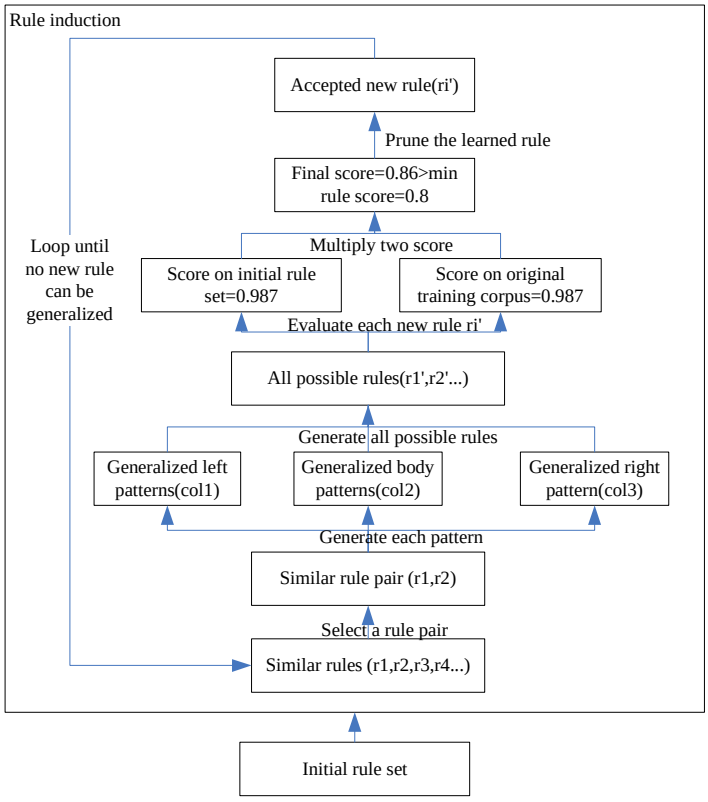


Fig. 11. An example snippet of dependency graph recorded in iASA

Figure 11 shows a dependency graph fragment that records the generalization of a new rule. This procedure is the visualization of rule induction to give the user an insight look of the rule induction. SRL firstly uses MLRM algorithm to search for the most similar rules (r1, r2, r3, r4), and obtains a rule pair (r1,r2) for induction. Then, the rule generalization induces left patterns, body patterns and right patterns respectively. Each generalization returns a collection of possible patterns. After that, SRL combines the three pattern collections, and generates all possible rules. Next, SRL evaluates each rule on the initial rule set and the training corpus, and obtains the scores. For rule r1', the scores are 0.987 and 0.87 respectively. The final score 0.86 exceed the threshold. Therefore, SRL add the new rule (r1') into the learned rule set.

The dependency graph is constructed while the system is running. Each of the components contributes nodes and edges during the execution of the system. When the system generates rules or annotates new documents, the dependency graph is created at the same time.

6.2. Answer the Questions

In principle, a user may ask iASA many questions of following categories:

(1) Explain existing rule: "why is a rule induced as the final rule?" In essence, the user wants to know how it was induced and survived from the evaluation and pruning process.

(2) Explain absent rule: this is to answer the question: "Why is a certain rule not present in the output?"

(3) Explain rule score: "why is rule *x* scored higher than rule *y* in the output?". For such question, iASA gives the details of the two scores (i.e. *rulescore* and *realscore*), including their coverage, error count, missing count, precision, recall and F-measure. The score is the key point in deciding where the rule should be put in the output.

(4) Explain annotation: "which rule is an instance annotated by?" For a wrong annotation, the user wants to know which rule brings out the error. He wants to know the reason so as to modify the rule.

We now briefly describe how iASA generates explanations for the four kinds of predefined queries described above.

To answer the question "why is rule *x* present", iASA selects the slice of dependency graph that records the generation and processing of rule *x*.

To answer the question "why is rule *x* scored higher than rule *y*", iASA first searches for the two rules in the dependency graph. Then iASA compares the two slices of the dependency graph corresponding to *x* and *y*. When comparing the slices, it focuses on the places where the two rules are evaluated and scored. iASA outputs the details of the scores on the two rules. The details of the scores indicate to the user that the difference between the two rules. For example, the user can find from the scores in which step the rule *x* scored higher than the rule *y* so that it survives in the final rules.

To answer the question "why is rule *x* not present", iASA first examines the dependency graph to check whether rule *x* has been generated before. If it has, then iASA finds out where it has been eliminated, and searches for the places where the rule is scored. iASA outputs the scores of the rule on initial rule set, original training corpus, the final score, and the threshold. By comparing the scores with each other and with the threshold, the user can know why the rule is not present. For example, the reason might be the score of the rule is below the threshold.

If rule *x* has not been generated, then iASA checks whether or not the rule can be generalized from the initial rule set. We conduct the check by searching for whether there are rules that are covered by the rule. Success of the check indicates that the rule can be generalized (but did not). Then iASA checks why it was not generated. We conduct the check by tracing all rules that are covered by the rule. Those rules may be selected in different pair of similar rules for induction. iASA outputs all covered rules

and presents the similar rule pairs that include the covered rules, and their similarity scores.

To answer the question "which rule is an instance annotated by?", iASA searches for the rule in the predecessors of annotated instance. At last, iASA outputs the rule.

By the explanation module, the user can take corresponding actions for improving the learned rules or the annotation results. For example, the user finds that a rule is not present in the learned rule set. He can first ask the explanation module the question, and the explanation returns the answer (e.g. the answer is: the rule is generated; however, it is pruned because its score is below the threshold). And then the user can choose to accept the rule.

7. Experiments and Discussion

7.1. Experimental Setup

In existing annotation systems, some are manual, some are based on GATE (its rules need to be predefined manually), and most of the semi-automatic semantic annotation systems exploit existing IE algorithms. Table 5 lists the relationships between IE algorithms and some semantic annotation systems.

Table 5. Relationships between semantic annotation systems and IE algorithms

SA Systems	IE algorithms
S-CREAM	LP^2
MnM	LP^2, Badger, Marmot, Crystal
SHOE	Manual
AeroDAML	AeroText, NLP
Annotea	Manual
KIM	GATE
SEAN	Syntactic and semantic structure learning
Protégé 2000	Manual
OntoMat-Annotizer	LP^2
Melita	LP^2
Artequakt	GATE
SemTag	TBD
SCORE	Name entity and relation learning

For the semi-automatic semantic annotation system, its performance naturally depends on the IE algorithms that it exploits. Therefore, to evaluate iASA, we compare iASA with the IE algorithms used in the semantic annotation systems, e.g. LP^2 (LP^2 is an algorithm for adaptive Information Extraction from Web-related text that induces symbolic rules learning from a corpus tagged with SGML tags). We conducted the comparison between iASA and some other popular algorithms (e.g. Rapier, SRV, HMM, BWI, Whisk, etc).

Our experiments are performed on two standard tasks for adaptive IE: the CMU seminar announcements [20] and Austin job announcements [7][2].

CMU seminar task consists of 485 seminar announcements from Carnegie Mellon University. The announcements contain details of the upcoming seminars. Each seminar is annotated with unique *starting time*, *ending time*, *location* and possible multiple *speaker name*. We denote CMU as the data set of CMU seminar announcements.

The Austin job task consists of 300 newsgroup messages on job details in Austin area. The task has been required to annotate 17 elements (see table 7). We denote JOBS as the data set of Austin job announcements.

In experiments, we used a random 50:50 split of the two datasets and repeated 10 times. We used 50:50 split for facilitating the comparison because the results for BWI, RAPIER and LP^2, etc [33] use the same splits for each system.

All experiments use the strategy of dynamic window size and machine learning methods to conduct instance selection and missing instance prediction.

7.2. Evaluation Measures

A truly comprehensive comparison should compare each algorithm on the same dataset, using the same splits, and the same scoring system. Unfortunately, it is impossible to end up with a conclusive comparison of different algorithms using current published results. Different algorithms have been evaluated by slightly different methodologies [33].

In semantic annotation, two kinds of issues would be considered with respect to the evaluation: how to decide an instance is correct and how to count the correct/wrong instances.

For the first issue, we take a compromised approach of combining exact matches and partial matches. Exact match contributes a full score and partial match contributes a half score. For example, if the correct speaker is “Dr Jim Boshears, PhD” and iASA annotates “Dr Jim Boshears” as a speaker, this would be viewed as partial match and count as half a correct instance (0.5).

For the second issue, there exist two approaches: instance exact matching, value exact matching. The former one requires the system to annotate all possible instances. Thus if a document contains a *stime*’s instance which has two occurrences “1:00 PM” and “1 p.m”, then the system is required to annotate them both. The second approach only compares the output annotations. In this case, it is sufficient to annotate *stime* either by “1:00 PM” or “1 p.m” as they refer to the same meaning. In this paper, we adopt the later approach to count the correct/wrong instances. As for the multiple instances referring to the same meaning, we only count one time.

In all the experiments, we conducted evaluations in terms of F-measure (β=1). The evaluation measure has been introduced in section 3.3.

[2] http://www.isi.edu/info-agents/RISE/repository.html

7.3. Experimental Results

Table 6 shows the comparison between iASA and several algorithms on CMU. The columns respectively represent the Algorithm, F1-score on entity *starting time*, *ending time*, *location* and *speaker name*, and the average F1-score.

Table 6. Comparison of the seven methods on CMU (%)

Algorithm	stime	etime	speaker	location	Average
BWI	99.6	93.9	67.7	76.7	83.9
HMM	98.5	62.1	76.6	**78.6**	82.0
SRV	98.5	77.9	56.3	72.3	77.1
Rapier	93.4	**96.2**	53.0	72.7	77.3
Whisk	92.6	86.0	18.3	66.4	64.9
LP^2	99.0	95.5	**77.6**	75.0	**86.0**
iASA	**99.8**	95.2	75.7	76.5	85.8

As shown in table 6, we see that iASA outperforms most of the other algorithms (averagely +2.26% wrt BWI, +4.63% wrt HMM, +11.28% wrt SRV, +11.0% wrt Rapier, +32.2% wrt to Whisk), and is competitive with $(LP)^2$ (-0.2%).

In JOBS, It requires to annotate the seventeen kinds of information related to computer job (some are unique and some are not). We used half of the corpus to train, and the rest to test the learned rules. Table 7 shows the experimental results. The columns respectively represent the entity that is required to annotate, three algorithms (Rapier, BWI, and $(LP)^2$), and iASA.

Table 7. Comparison of the four methods on JOBS (%)

Entity	Rapier	BWI	(LP)2	iASA
Id	97.5	**100.0**	**100.0**	**100.0**
title	40.5	50.1	43.9	**89.1**
company	69.5	**78.2**	71.9	73.6
salary	67.4	-	62.8	**80.0**
recruiter	68.4	-	80.6	**91.3**
state	90.2	-	84.7	**91.5**
city	90.4	-	93.0	**95.6**
country	93.2	-	81.0	**96.6**
language	80.6	-	**91.0**	83.2
platform	72.5	-	80.5	**82.4**
application	69.3	-	**78.4**	73.8
area	42.4	-	**66.9**	55.3
req-years-e	67.1	-	68.8	**73.7**
des-years-e	**87.5**	-	60.4	66.7
req-degree	81.5	-	**84.7**	65.9
des-degree	72.2	-	65.1	**80.0**
post date	99.5	-	99.5	**100.0**
Average	75.1	-	84.1	**89.4**

We see averagely iASA significantly outperforms Rapier (by +19%) and $(LP)^2$ (by +6.3%) in terms of F1-measure. On the three entities that are available for BWI, iASA significantly outperforms it on *title* (+77.8%), and underperforms it on *company* (-5.9%).

7.4. Discussion

Intuitively, dynamic window size can optimize the learning scenario, instance selection can improve the precision by pruning the potential wrong annotations, and missing instance prediction can improve the recall of the annotation. We performed several special experiments for confirming the idea.

1. Dynamic window size vs. Fixed window size

In our experiments, some elements are not affected by the window size, while the others are sensitive to it. For the entities that are sensitive to window size, some prefer small window size while the others prefer large window size.

We give an experimental comparison of dynamic window size and fixed window size. We conducted the comparison on CMU. The result is shown in figure 12.

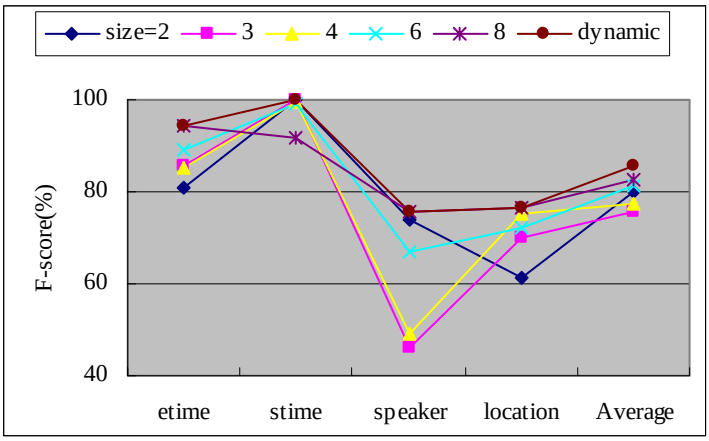


Fig. 12. Comparison between different fixed window size and dynamic window size on CMU

By fixed-window size, the best average result is 82.8% obtained by using fixed-window size 8, and the second is 81.4% when the window size is set to 6. The result of dynamic window size outperforms that of all the fixed window size (+3.6% than that of size=8, +5.4% than that of size=6).

We also see that for the entities that are sensitive to the window size, the improvements by dynamic strategy are significantly (e.g. for *speaker*, the improvements are respectively +54.2% and +63.5% compared to the results of size=4 and size=3).

We note that optimizing dynamic window size is a time consuming process, which limits its applications with large scale ontology.

2. Correct instance selection vs. High scored selection vs. Without selection

We conducted the experiment to test the performance of the proposed method for correct instance selection. We conducted the comparison of results by correct instance selection, high scored selection, and result without any selection. By high scored selection, we mean selecting the annotated instances by the highest scored rules. We conducted the comparison on JOBS. (On CMU, the two selection methods do not affect the results of the four entities.)

Table 8 shows the comparison. The columns respectively represent entity, F1-scores (F) without selection, with high scored selection and with correct instance selection. For short, we use ML to denote correct instance selection by machine learning; we use HS to denote high scored selection. In table 8, we also present the recall (R) and error rate (E) of the selection. Recall indicates how many correct instances are selected. Error rate indicates the accuracy of instance selection. The selection methods work on six entities in JOBS, and do not affect the other eleven entities. Therefore, in table 8, we list only the six entities and omit details of the other eleven entities.

Table 8. Correct instance selection vs. High scored (HS) selection in JOBS (%)

Entity	Without selection	HS selection			ML selection		
		R	E	F	R	E	F
title	77.1	33.3	0	82.1	74.2	0	89.1
platform	79.2	0	0	79.2	72.0	0	80.9
city	88.2	42.9	33.3	92.3	52.5	0	95.6
area	50.6	0	0	50.6	72.9	33.3	53.1
application	67.7	0	0	67.7	77.0	5.0	71.2
req-degree	57.2	0	0	57.2	50.0	12.5	65.9
Average	70.0	-		71.5	-		76.0

We see that ML based instance selection significantly outperforms high scored selection (+6.3% on average). The improvement over the result without selection is also significant (+8.6% on average). By high score selection, only two entities obtained improvements: title and city. By ML selection, all the six entities obtained improvements.

We also note that the recall of the ML based selection is still low (ranging from 50% to 77%) and errors are also induced by the wrong selection (e.g. *area*, *application*, and *req-degree*). This also means that we need further improve it.

3. ML based prediction vs. No prediction

We exploit machine learning methods to improve the recall of iASA. We conducted the comparison between results by machine learning based prediction and that without prediction. We conducted the comparison on JOBS. (On CMU, the prediction methods do not affect the results of the four entities.)

Table 9 shows the experiment results on JOBS. ML denotes the machine learning based method for prediction; E denotes the error rate of ML based prediction; F denotes the F1-score. The prediction methods work on four entities in JOBS, and do not affect the other thirteen entities. Therefore, in table 9, we list only the four entities and omit details of the other thirteen entities.

Table 9. ML based prediction vs. No prediction on JOBS (%)

Entity	No prediction	ML	
		E	**F**
language	75.7	0	83.2
platform	80.9	33.3	82.4
application	71.2	3.8	73.8
area	53.1	15	55.3
Average	70.1	-	73.7

By ML based prediction, we have observed improvement on the four entities (by +5.1% on average in terms of F1-measure). We have also observed that the prediction might slightly decrease the precision because of the wrong prediction.

On the other hand, we see that the prediction method only works on four entities in the two data sets. Because the instances of other entities (four entities in CMU and thirteen entities in JOBS) are not enumerative type and their instance values are too sparse. Moreover, the prediction introduces errors (ranging from 3.8% to 33.3%). Some errors (about 50%) imported from the fact that instance value of entity *platform* and *area* are similar. Such similarity confuses the prediction method.

4. More analysis

(1) From the experiments, we see that for the majority of the elements in the two tasks, iASA outperforms the other algorithms. On CMU, iASA averagely outperforms them (averagely +2.26% wrt BWI, +4.63% wrt HMM, +11.28% wrt SRV, +11.0% wrt Rapier, +32.2% wrt Whisk,), and is competitive with $(LP)^2$ (-0.2%). On JOBS, iASA averagely outperforms Rapier (by +19%) and $(LP)^2$ (by +6.3%).

(2) We conducted the analysis on each entity. On entities *stime*, *etime* in the CMU and *id*, *posting_date*, *platform* in JOBS, almost all algorithms perform well. These entities often have clear common linguistic information either in their contexts or in their body patterns. On the other hand, *location* and *speaker* are somewhat difficult for iASA. There may be two reasons. One is that these entities have little common linguistic information and their contexts are always inconsistent. The other lies in that several entities often appear in a document in a particular relationship which makes the situation suitable for learning them together. For example, the *stime* and *etime* in the CMU, and *title* with other elements (*area*, *country*, *state*, etc.) in the JOBS. This is the part of our future work for iASA.

(3) Considering the six elements that underperform LP^2 or Rapier, they can be classified into two groups: *des_years_e*, *req_degree* and *area*, *language*, *application*, *company*.

For the former two entities, we found that the two entities are difficult distinguished even by human. The context and body pattern for them are very similar. Analysis of syntactic structure and semantic structure can help to construct long distant context and semantic context (e.g. *subject predicate object*) and may correct the errors.

For the later four elements, the precisions are acceptable but the recalls are low. By experimental analysis, we found that many learned rules only are comprised of body patterns (i.e. left pattern and right pattern are null, e.g. "C++" for *language*), which means that such annotations heavily depend on whether the body patterns appear in the training samples. Two approaches may be useful to deal with such problems:

increasing training samples and creating domain thesaurus. More training samples can induce the rules covering more body patterns. But more training samples also means more manual works. Creating domain thesaurus means to construct a word list for the entity (e.g. *language*) to assist the annotation.

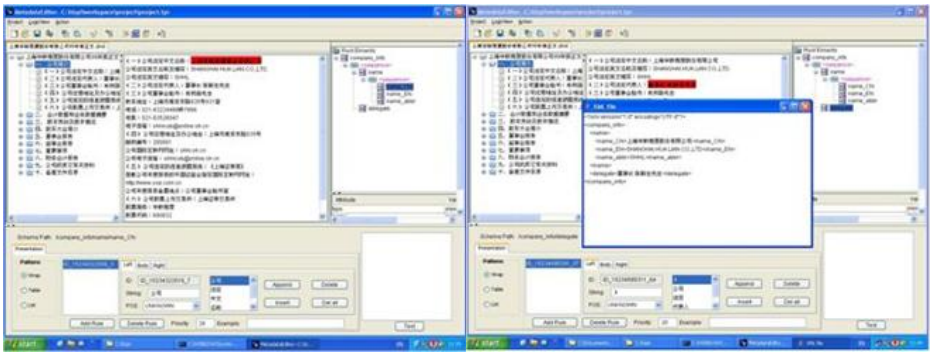
7.5. Applications

We have applied iASA to a practical application: TIPSI.

TIPSI is a project from Tsinghua-ITF Co-Lab. In TIPSI, we aim to extract and annotate the information in company annual reports for Stock Exchange.

The company annual report is a semi-structured document. In TIPSI, we first extract the logic structure from annual report. We conducted the logic structure extraction by a logic structure extractor. See [50] for details. Then, the document is organized into a tree structure. It is similar to the 'Document Map' in Office Word. Next, for each node in the logic structure, we applied iASA to annotate it according to the predefined ontology of company annual report.

For example, figure 13 (a) shows the user interface of TIPSI and figure 13 (b) shows the output of annotation. There are four main fields on the user interface. The top-left window shows the document logic structure. The mid window shows the text content of the selected node in logic structure. The right-top window shows the predefined ontology. The bottom window is a rule management tool.



(a) (b)

Fig. 13. Screenshots of TIPSI

In training stage, when user selects the text on the mid window, iASA automatically carries out tokenization and NLP processing, and then generates an initial rule. The rule can be incrementally added into the learned rule set and can be added to the initial rule set to retrain the whole rule set. If the rule is already covered by some other rules in the learned rule set, the system will prompt the user about that. In this way, the rule learning is a user interaction stage. User can also manually create a new rule or correct learned rules.

In annotation stage, iASA annotates documents by using the learned rules. It also records the occurring position of each instance. And the user can be navigated to

instances by selecting the concept or property in the ontology. When the user selects a concept/property in the ontology, the system will highlight the annotated instance(s). Figure 13 (a) shows the scenario. In the mid window, the highlighted text is an instance of company Chinese name.

After annotation, the user can choose to output the annotation results. Figure 13 (b) shows the annotation results by a popup window. The results are presented in XML format according to the requirement of the project.

We are also trying to apply the tool into Contact Search on internet.

In Contact Search, we aim to annotate the contact information for a given person. The user inputs a person name. The system submits the person's name to Google. Then we use a text classifier to identify the web page that contains the contact information of the person. After that, we apply the iASA to annotate the contact information. The contact information includes: person name, email, telephone, fax, homepage, address, and job title. The project is still ongoing and the preliminary results show that the tool is promising.

8. Related works

In this section, we introduce the related works from four aspects: Knowledge Acquisition Frameworks, Annotation Framework, (Semi-)Automated Annotation with Support from Information Extraction, and Information Extraction. There are a number of available systems that address these four aspects. A complete review of this subject is therefore outside the scope of this paper. We present some of them through their principles and availabilities.

8.1. Knowledge Acquisition Frameworks

Several systems are designed to allow for knowledge acquisition and to use knowledge markups in semantic web, for example: Protégé-2000 [18], WebKB [35], SHOE [26] and Artequakt [1]. These four systems all start from providing manual mark-up by editors.

Protégé-2000 is a tool which supports knowledge acquisition. But it doesn't support managing and annotating the web pages.

WebKB uses conceptual graphs to represent the semantic content of Web documents. It embeds conceptual graph statements into HTML pages. Essentially they offer a web template-based knowledge acquisition framework.

SHOE is one of the earliest systems for adding semantic annotations to web pages. SHOE Knowledge Annotator allows users to markup pages manually in SHOE guided by ontologies available locally or via a URL. These marked pages can be reasoned about by SHOE-aware tools such as SHOE Search. Such tools are described in [30, 48].

The Artequakt project links a knowledge extraction tool with ontology to achieve knowledge support and to guide information extraction. The extraction tool searches for online documents and extracts knowledge that matches the given classification

structure. Knowledge extraction is further enhanced by using a lexicon-based term expansion mechanism that provides extended ontology terminology [1].

8.2. Annotation Frameworks

There are a number of systems designed particularly for annotation, for example: Annotea [27], Ontobroker [19], OntoMat-Annotizer, SEAN [37], etc.

Annotea is a Web-based shared annotation system based on a general-purpose open resource description framework (RDF) infrastructure. In Annotea, the annotations are modeled as a class of metadata. Annotations are viewed as statements made by an author about a Web document.

Ontobroker facilitates manual annotation of HTML documents with semantic markups.

SEAN automatically discovers and labels concept instances in template-based, content-rich HTML documents according to an ontology. It combines structural and semantic analysis for annotation. SEAN focuses on well-organized documents, for example documents generated from databases.

8.3. (Semi-)Automated Annotation with Support from Information Extraction

Recently, efforts have been put into automating the annotation task by using machine learning methods. The principal tool is "wrapper" (see [11, 29, 31]). The Semantic Annotation systems which use IE algorithms can be referred to Table 5.

For example, S-CREAM [24], MnM [49] and Melita [10] are three systems exploiting IE algorithm LP^2 to automate the procedure of annotation.

S-CREAM is a comprehensive framework for creating annotations, relational metadata in the semantic web, including tools for both manual and semi-automatic annotation of pages. It also comprises inference services, crawler, document management system, ontology guidance/fact browser, document editors/viewers, and a meta ontology.

MnM produces semantic markups with the support from IE algorithm. Besides LP^2, it also integrates other IE components (Marmot, Badger, Crystal) from the University of Massachusetts at Amherst (UMass). It allows the semi-automatic population of ontology with metadata.

Melita is a tool for defining and developing automatic ontology-based annotation services that provides different views over the task. It provides manual and semi-automatic annotation, as well as a rule editor for IE experts to edit the annotation rules.

AeroDAML [28] is a tool which takes ontology as metadata and automatically produces a semantic annotation using NLP techniques. It supports only DAML language.

The KIM platform provides semantic annotation, indexing, retrieval services and infrastructure. It performs information extraction based on ontology and a massive knowledge base [42]. The information extraction process in KIM is based on the GATE platform. GATE's pattern-matching grammars have been modified so as to

handle entity class information and to allow generalization of the rules. But, GATE does not provide the feature for learning the annotation rules.

SCORE Enhancement Engine (SEE) supports heterogeneous contents, followed by an automatic classification with extraction of context relevant, domain-specific metadata. Extraction of semantic metadata includes not only the identification of the relevant entities, but also the relationships within the context of relevant ontology. It also presents an approach to automatic semantic annotation [22].

Li et al propose to combine natural language understanding with learning to automatically generate annotations for specific domains [34]. They aim to learn the syntactic structures from the sentences.

SemTag aims to annotate very large number of pages with terms from a standard ontology in an automated fashion based on disambiguation annotation [17]. SemTag operates as a centralized application with access to the entire database and associated metadata. SemTag manipulates the text linking in web page to its correct resource by disambiguation technology.

Esperonto has an annotation service that helps content providers bridge the gap between the current Web and the Semantic Web. It uses wrapper technology to upgrade content to Semantic Web content [6, 25].

So far, existing systems focus on different aspects that are concerned with semantic annotation. Comparing with the above methods, three features make iASA different: (1) similarity based rule induction; (2) using machine learning to refine the annotation; (2) explanation method by visualizing the main stages in rule induction and annotation procedure. Existing works improve the recall of annotation by combining data mining and IE techniques. They are similar to the missing instance prediction in iASA. In the experiments of [39], F1-measure could be improved about 3% by using soft matching mined rules when tested on 150 documents. It is difficult to conduct the comparison of the method and our method. The reason lies in that, their experiments were based on BWI algorithm (an IE algorithm), and their best F1-measure result was only 45% that is below the average in our experiments.

8.4. Information Extraction Technologies

In information extraction, given a sequence of instances, we identify and pull out a sub sequence of the input that represents the information we are interested in. Hidden Markov Model [21, 44], Maximum Entropy Model [3, 8], Maximum Entropy Markov Model [36], Support Vector Machines [13], Conditional Random Field [32], and Voted Perceptron [12] are widely used information extraction models.

Information extraction has been applied, for instance, to named entity recognition [51], table extraction [41], metadata extraction from research paper [23, 40].

9. Conclusions

In this paper, we have investigated the problem of semantic annotation. We have proposed a tool, called iASA, which learns to automatically annotate web documents according to an ontology. By using similarity based rule induction, we have been able

to improve the rule learning procedure. We have tried to improve the annotation results by making use of machine learning methods. Finally, we have developed an explanation module to express the nature of the learner and annotator to users. Experimental results show that our approach can significantly outperform most of the existing wrapper methods. By the analysis of the experimental results, we observed that the proposed methods (including: correct instance selection and missing instance prediction) work well.

As the future work, we plan to make further improvement on the annotation accuracy. We also want to apply the annotation method to other annotation applications. Apart from that, several challenges for semantic annotation, also being our research interests, including: (1) Using active learning to make iASA more adaptive to new documents. By active learning, we can prepare training documents more efficiently and more effectively. (2) Making use of the annotation. The goal of semantic annotation is to improve the efficiency of obtaining information in web environment. Therefore, a friendly and flexible mechanism for using the annotation is necessary. (3) Combining ontology mapping with semantic annotation. In order to reach interoperability in the heterogeneous environment of semantic web, a system for integrating ontology mapping and semantic annotation is required. (4) Multilingual annotation. Multilingual annotation is also important given the fact that Chinese websites are increasing exponentially, which could be a future direction to go as well.

Acknowledge

This work is supported by the National Natural Science Foundation of China under Grant No. 60443002 and by the Tsinghua-ITF co-laboratory.

We offer thanks to anonymous reviewers for their valuable comments.

We express our great sadness and sorrow regarding Professor Hongjun Lu, who passed away while this paper was in preparation.

References

1. H. Alani, S. Kim, D. Millard, M. Weal, W. Hall, P. Lewis, and N. Shadbolt. Automatic Ontology-Based Knowledge Extraction from Web Documents. IEEE Intelligent Systems. 2003, 18(1):14-21.
2. R. Benjamins, J. Contreras. White Paper Six Challenges for the Semantic Web. Intelligent Software Components. Intelligent software for the networked economy (isoco). April, 2002.
3. A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra, A Maximum Entropy Approach to Natural Language Processing. In Computational Linguistics. 22, 1996:39-71
4. T. Berners-Lee, M. Fischetti, and M. L. Dertouzos. Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web. 1999.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American, vol. 284, May 2001:34-43

6. P. Buitelaar and T. Declerck. Linguistic Annotation for the Semantic Web. In Annotation for the Semantic Web, Frontiers in Artificial Intelligence and Applications Series, Vol. 96, IOS Press, 2003.
7. M. E. Califf. Relational Learning Techniques for Natural Language Information Extraction. Ph.D. thesis. University of Texas, Austin. 1998
8. H. L. Chieu and H. T. Ng. A Maximum Entropy Approach to Information Extraction from Semi-Structured and Free Text. In Eighteenth national conference on Artificial intelligence, 2002.
9. F. Ciravegna. $(LP)^2$, 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), Seattle, Usa, August 2001.
10. F. Ciravegna, A. Dingli, J. Iria, and Y. Wilks. Multi-strategy Definition of Annotation Services in Melita. In ISWC'03 Workshop on Human Language Technology for the Semantic Web and Web Services, 2003:97-107
11. W. Cohen, L. Jensen, A Structured Wrapper Induction System for Extracting Information from Semi-structured Documents, In Proceedings of the Workshop on Adaptive Text Extraction and Mining (IJCAI'01), 2001.
12. M. Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In Proceedings of the Conference on Empirical Methods in NLP, 2002.
13. C. Cortes and V. Vapnik. Support-vector networks. Machine Learning 20, 1995:273-297
14. H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics, 2002.
15. M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. Andrea Stein. OWL Web Ontology Language Reference. W3C Recommendation. Available at http://www.w3.org/TR/owl-ref/. 10 February 2004.
16. R. Dhamankar, Y. Lee, A.H. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. SIGMOD 2004 June 1318, 2004, Paris, France.
17. S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, K. S. McCurley, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. A Case for Automated Large-scale Semantic Annotation. Journal of Web Semantics: Science, Services and Agents on the World Wide Web. Published by Elsevier B.V. July, 2003:115-132
18. H. Eriksson, R. Fergerson, Y. Shahar, and M. Musen. Automatic Generation of Ontology Editors. In Proceedings of the 12th Banff Knowledge Acquisition Workshop, Banff Alberta, Canada, 1999.
19. D. Fensel, S. Decker, M. Erdmann, and R. Studer. Ontobroker: Or how to enable intelligent access to the WWW. In Proceedings of 11th Banff Knowledge Acquisition for Knowledge-Based SystemsWorkshop, Banff, Canada, 1998.
20. D. Freitag and N. Kushmerick. Boosted Wrapper Induction. In Proceedings of 17th National Conference on Artificial Intelligence. 2000.
21. Z. Ghahramani and M. I. Jordan. Factorial Hidden Markov Models. Machine Learning, 1997, 29:245-273
22. B. Hammond, A. Sheth, and K. Kochut, Semantic Enhancement Engine: A Modular Document Enhancement Platform for Semantic Applications over Heterogeneous Content, in Real World Semantic Web Applications, V. Kashyap and L. Shklar, Eds., IOS Press. December 2002:29-49

23. H. Han, L. Giles, E. Manavoglu, H. Zha, Z. Zhang, and E. Fox. Automatic Document Metadata Extraction Using Support Vector Machine. In Proceedings of Joint Conference on Digital Libraries (JCDL 2003). 2003:37-48
24. S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM—Semi-automatic Creation of Metadata, In Proceedings of the 13th International Conference on Knowledge Engineering and Management (EKAW 2002), Siguenza, Spain, 2002:358-372.
25. S. Handschuh and S. Staab. Annotation for the Semantic Web. Volume 96 Frontiers in Artificial Intelligence and Applications. New IOS Publication. 2003
26. J. Heflin and J. Hendler. Searching the Web with SHOE. In Proceedings of AAAI-2000 Workshop on AI for Web Search, Austin, Texas, 2000.
27. J. Kahan and M. R. Koivunen. Annotea: an Open RDF Infrastructure for Shared Web Annotations. In Proceedings of World Wide Web, 2001:623-632.
28. P. Kogut and W. Holmes. AeroDAML: Applying Information Extraction to Generate DAML Annotations from Web Pages. 2001.
29. N. Kushmerick, D.S. Weld, and R.B. Doorenbos. Wrapper Induction for Information Extraction. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). Nagoya, Japan. 1997: 729-737.
30. T. Leonard and H. Glaser. Large Scale Acquisition and Maintenance from the Web without Source Access. http://www.semannot2001.aifb.uni-karlsruhe.de/positionpapers/Leonard.pdf. 2001.
31. K. Lerman, C. Knoblock, S. Minton, Automatic data extraction from lists and tables in web sources, in: IJCAI-2001 Workshop on Adaptive Text Extraction and Mining, Seattle, WA, August 2001.
32. J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In ICML 01, 2001.
33. A. Lavelli, M. Califf, F. Ciravegna, F. Freitag, D. Giuliano, C. Kushmerick, and N. Romano. A Critical Survey of the Methodology for IE Evaluation. In Proceedings of the 4th International Conference on Language Resources and Evaluation. 2004
34. J. Li and Y. Yu. Learning to Generate Semantic Annotation for Domain Specific Sentences. In Proceedings of the Knowledge Markup and Semantic Annotation Workshop in K-CAP 2001, Victoria, BC, 2001.
35. P. Martin and P. Eklund. Embedding Knowledge in Web Documents. In Proceedings of the 8th International World Wide Web Conf. (WWW'8), Toronto, Elsevier Science B.V. May 1999:1403-1419
36. A. McCallum, D. Freitag, and F. Pereira. Maximum Entropy Markov Models for Information Extraction and Segmentation, In Proceedings of the ICML Coference, 2000.
37. S. Mukherjee, G. Yang, and I. V. Ramakrishnan. Automatic Annotation of Content-Rich HTML Documents: Structural and Semantic Analysis. In Second International Semantic Web Conference (ISWC), Sanibel Island, Florida, October 2003.
38. I. Muslea. Active Learning with Multiple Views. Ph.D. dissertation. (USC, 2002)
39. U. Y. Nahm and R. J. Mooney. Using Soft-Matching Mined Rules to Improve Information Extraction. In Proceedings of the AAAI-2004 Workshop on Adaptive Text Extraction and Mining (ATEM-2004), San Jose, CA, July, 2004:27-32.
40. F. Peng and A. McCallum. Accurate Information Extraction from Research Papers using Conditional Random Fields. In Proceedings of Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL), 2004.
41. D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table Extraction Using Conditional Random Fields. In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval, 2003.

42. B. Popov, A. Kiryakov, D. Manov, A. Kirilov, D. Ognyanoff, and M. Goranov. Towards Semantic Web Information Extraction. In Proceedings of the ISWC'03 Workshop on Human Language Technology for the Semantic Web and Web Services, 2003.1-21
43. C. Schaffer. Selecting a Classification method by Cross-Validation. Machine Learning, 13(1), 1993:135-143
44. K. Seymore, A. McCallum, and R. Rosenfeld. Learning Hidden Markov Model Structure for Information Extraction. In Proceedings of AAAI'99 Workshop on Machine Learning for Information Extraction. 1999.
45. S. Soderland. Learning Information Extraction Rules for Semi-structured and Free Text. Machine Learning. Jan, 1999:1-44
46. V. W. Soo, C. Y. Lee, C. –C. Li, S. L. Chen, and C. Chen. Automated Semantic Annotation and Retrieval Based on Sharable Ontology and Case-based Learning Techniques. In Proceedings of the 2003 Joint Conference on Digital Libraries. 2003 IEEE.
47. V. Vapnik. Statistical Learning Theroy. Springer Verlage, New York, 1998.
48. M. Vargas-Vera, E. Motta, J. Domingue, S. Buckingham Shum, and M. Lanzoni. Knowledge Extraction by Using an Ontology-based Annotation Tool. In Proceedings of K-CAP 2001 Workshop on Knowledge Markup and Semantic Annotation, Victoria, BC, Canada, October 2001.
49. M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology Driven Semiautomatic and Automatic Support for Semantic Markup, In Proceedings of the 13th International Conference on Knowledge Engineering and Management (EKAW 2002), Siguenza, Spain, 2002.
50. K. Zhang, P. Xu, and J. Li. Optimal Hierarchical Clustering based Logic Structure Extraction. Journal of Tsinghua Science and Technology. 2005.
51. L. Zhang, Y. Pan, and T. Zhang. Recognising and using named entities: Focused named entity recognition using machine learning. In Proceedings of the SIGIR'04, 2004.