A Systematic Exploration of the Feature Space for Relation Extraction

Jing Jiang and **ChengXiang Zhai**
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{jiang4,czhai}@cs.uiuc.edu

Abstract

Relation extraction is the task of finding semantic relations between entities from text. The state-of-the-art methods for relation extraction are mostly based on statistical learning, and thus all have to deal with feature selection, which can significantly affect the classification performance. In this paper, we systematically explore a large space of features for relation extraction and evaluate the effectiveness of different feature subspaces. We present a general definition of feature spaces based on a graphic representation of relation instances, and explore three different representations of relation instances and features of different complexities within this framework. Our experiments show that using only basic unit features is generally sufficient to achieve state-of-the-art performance, while over-inclusion of complex features may hurt the performance. A combination of features of different levels of complexity and from different sentence representations, coupled with task-oriented feature pruning, gives the best performance.

1 Introduction

An important information extraction task is relation extraction, whose goal is to detect and characterize semantic relations between entities in text. For example, the text fragment "hundreds of Palestinians converged on the square" contains the *located* relation between the *Person* entity "hundreds of Palestinians" and the *Bounded-Area* entity "the square". Relation extraction has applications in many domains, including finding affiliation relations from web pages and finding protein-protein interactions from biomedical literature.

Recent studies on relation extraction have shown the advantages of discriminative model-based statistical machine learning approach to this problem. There are generally two lines of work following this approach. The first utilizes a set of carefully selected features obtained from different levels of text analysis, from part-of-speech (POS) tagging to full parsing and dependency parsing (Kambhatla, 2004; Zhao and Grishman, 2005; Zhou et al., 2005)[1]. The second line of work designs kernel functions on some structured representation (sequences or trees) of the relation instances to capture the similarity between two relation instances (Zelenko et al., 2003; Culotta and Sorensen, 2004; Bunescu and Mooney, 2005a; Bunescu and Mooney, 2005b; Zhang et al., 2006a; Zhang et al., 2006b). Of particular interest among the various kernels proposed are the convolution kernels (Bunescu and Mooney, 2005b; Zhang et al., 2006a), because they can efficiently compute the similarity between two instances in a huge feature space due to their recursive nature. Apart from their computational efficiency, convolution kernels also implicitly correspond to some feature space. Therefore, both lines of work rely on an appropriately de-

[1] Although Zhao and Grishman (2005) defined a number of kernels for relation extraction, the method is essentially similar to feature-based methods.

Proceedings of NAACL HLT 2007, pages 113–120,
Rochester, NY, April 2007.

fined set of features. As in any learning problem, the choice of features can affect the performance significantly.

Despite the importance of feature selection, there has not been any systematic exploration of the feature space for relation extraction, and the choices of features in existing work are somewhat arbitrary. In this paper, we conduct a systematic study of the feature space for relation extraction, and evaluate the effectiveness of different feature subspaces. Our motivations are twofold. First, based on previous studies, we want to identify and characterize the types of features that are potentially useful for relation extraction, and define a relatively complete and structured feature space that can be systematically explored. Second, we want to compare the effectiveness of different features. Such a study can guide us to choose the most effective feature set for relation extraction, or to design convolution kernels in the most effective way.

We propose and define a unified graphic representation of the feature space, and experiment with three feature subspaces, corresponding to sequences, syntactic parse trees and dependency parse trees. Experiment results show that each subspace is effective by itself, with the syntactic parse tree subspace being the most effective. Combining the three subspaces does not generate much improvement. Within each feature subspace, using only the basic unit features can already give reasonably good performance. Adding more complex features may not improve the performance much or may even hurt the performance. Task-oriented heuristics can be used to prune the feature space, and when appropriately done, can improve the performance. A combination of features of different levels of complexity and from different sentence representations, coupled with task-oriented feature pruning, gives the best performance.

2 Related Work

Zhao and Grishman (2005) and Zhou et al. (2005) explored a large set of features that are potentially useful for relation extraction. However, the feature space was defined and explored in a somewhat ad hoc manner. We study a broader scope of features and perform a more systematic study of different feature subspaces. Zelenko et al. (2003) and Culotta and Sorensen (2004) used tree kernels for relation extraction. These kernels can achieve high precision but low recall because of the relatively strict matching criteria. Bunescu and Mooney (2005a) proposed a dependency path kernel for relation extraction. This kernel also suffers from low recall for the same reason. Bunescu and Mooney (2005b) and Zhang et. al. (2006a; 2006b) applied convolution string kernels and tree kernels, respectively, to relation extraction. The convolution tree kernels achieved state-of-the-art performance. Since convolution kernels correspond to some explicit large feature spaces, the feature selection problem still remains.

General structural representations of natural language data have been studied in (Suzuki et al., 2003; Cumby and Roth, 2003), but these models were not designed specifically for relation extraction. Our feature definition is similar to these models, but more specifically designed for relation extraction and systematic exploration of the feature space. Compared with (Cumby and Roth, 2003), our feature space is more compact and provides more guidance on selecting meaningful subspaces.

3 Task Definition

Given a small piece of text that contains two entity mentions, the task of relation extraction is to decide whether the text states some semantic relation between the two entities, and if so, classify the relation into one of a set of predefined semantic relation types. Formally, let $r = (s, arg_1, arg_2)$ denote a relation instance, where s is a sentence, arg_1 and arg_2 are two entity mentions contained in s, and arg_1 precedes arg_2 in the text. Given a set of relation instances $\{r_i\}$, each labeled with a type $t_i \in \mathcal{T}$, where $\mathcal{T}$ is the set of predefined relation types plus the type *nil*, our goal is to learn a function that maps a relation instance r to a type $t \in \mathcal{T}$. Note that we do not specify the representation of s here. Indeed, s can contain more structured information in addition to merely the sequence of tokens in the sentence.

4 Feature Space for Relation Extraction

Ideally, we would like to define a feature space with at least two properties: (1) It should be *complete* in the sense that all features potentially useful for the

classification problem are included. (2) It should have a good *structure* so that a systematic search in the space is possible. Below we show how a unified graph-based feature space can be defined to satisfy these two properties.

4.1 A Unified View of Features for Relation Extraction

Before we introduce our definition of the feature space, let us first look at some typical features used for relation extraction. Consider the relation instance *"hundreds of Palestinians converged on the square"* with arg_1 = *"hundreds of Palestinians"* and arg_2 = *"the square"*. Various types of information can be useful for classifying this relation instance. For example, knowing that arg_1 is an entity of type *Person* can be useful. This feature involves the single token *"Palestinians"*. Another feature, "the head word of arg_1 (*Palestinians*) is followed by a verb (*converged*)", can also be useful. This feature involves two tokens, *"Palestinians"* and *"converged"*, with a sequence relation. It also involves the knowledge that *"Palestinians"* is part of arg_1 and *"converged"* is a verb. If we have the syntactic parse tree of the sentence, we can obtain even more complex and discriminative features. For example, the syntactic parse tree of the same relation instance contains the following subtree: [VP → VBD [PP → [IN → *on*] NP]]. If we know that arg_2 is contained in the NP in this subtree, then this subtree states that arg_2 is in a PP that is attached to a VP, and the proposition is *"on"*. This subtree therefore may also a useful feature. Similarly, if we have the dependency parse tree of the relation instance, then the dependency link *"square ⇝ on"* states that the token *"square"* is dependent on the token *"on"*, which may also be a useful feature.

Given that useful features are of various forms, in order to systematically search the feature space, we need to first have a unified view of features. This problem is not trivial because it is not immediately clear how different types of features can be unified. We observe, however, that in general features fall into two categories: (1) properties of a single token, and (2) relations between tokens. Features that involve attributes of a single token, such as bag-of-word features and entity attribute features, belong to the first category, while features that involve sequence, syntactic or dependency relations between tokens belong to the second category. Motivated by this observation, we can represent relation instances as graphs, with nodes denoting single tokens or syntactic categories such as NPs and VPs, and edges denoting various types of relations between the nodes.

4.2 Relation Instance Graphs

We represent a relation instance as a labeled, directed graph $G = (V, E, A, B)$, where V is the set of nodes in the graph, E is the set of directed edges in the graph, and A, B are functions that assign labels to the nodes.

First, for each node $v \in V$, $A(v) = \{a_1, a_2, \ldots, a_{|A(v)|}\}$ is a set of attributes associated with node v, where $a_i \in \Sigma$, and Σ is an alphabet that contains all possible attribute values. The attributes are introduced to help generalize the node. For example, if node v represents a token, then $A(v)$ can include the token itself, its morphological base form, its POS, its semantic class (e.g. WordNet synset), etc. If v also happens to be the head word of arg_1 or arg_2, then $A(v)$ can also include the entity type and other entity attributes. If node v represents a syntactic category such as an NP or VP, $A(v)$ can simply contain only the syntactic tag.

Next, function $B : V \rightarrow \{0, 1, 2, 3\}$ is introduced to distinguish argument nodes from non-argument nodes. For each node $v \in V$, $B(v)$ indicates how node v is related to arg_1 and arg_2. 0 indicates that v does not cover any argument, 1 or 2 indicates that v covers arg_1 or arg_2, respectively, and 3 indicates that v covers both arguments. We will see shortly that only nodes that represent syntactic categories in a syntactic parse tree can possibly be assigned 3. We refer to $B(v)$ as the *argument tag* of v.

We now consider three special instantiations of this general definition of relation instance graphs. See Figures 1, 2 and 3 for examples of each of the three representations.

Sequence: Without introducing any additional structured information, a sequence representation preserves the order of the tokens as they occur in the original sentence. Each node in this graph is a token augmented with its relevant attributes. For example, head words of arg_1 and arg_2 are augmented with the corresponding entity types. A token is assigned the argument tag 1 or 2 if it is the head word of arg_1 or

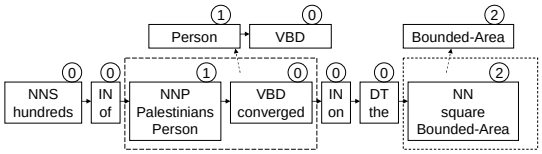

Figure 1: An example sequence representation. The subgraph on the left represents a bigram feature. The subgraph on the right represents a unigram feature that states the entity type of arg_2.

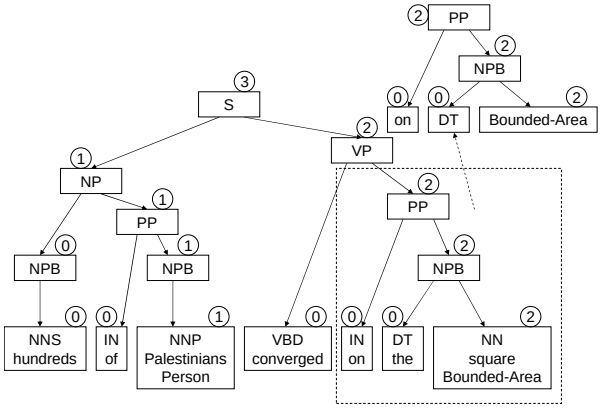

Figure 2: An example syntactic parse tree representation. The subgraph represents a subtree feature (grammar production feature).

arg_2. Otherwise, it is assigned the argument tag 0. There is a directed edge from u to v if and only if the token represented by v immediately follows that represented by u in the sentence.

Syntactic Parse Tree: The syntactic parse tree of the relation instance sentence can be augmented to represent the relation instance. First, we modify the tree slightly by conflating each leaf node in the original parse tree with its parent, which is a preterminal node labeled with a POS tag. Then, each node is augmented with relevant attributes if necessary. Argument tags are assigned to the leaf nodes in the same way as in the sequence representation. For an internal node v, argument tag 1 or 2 is assigned if either arg_1 or arg_2 is inside the subtree rooted at v, and 3 is assigned if both arguments are inside the subtree. Otherwise, 0 is assigned to v.

Dependency Parse Tree: Similarly, the dependency parse tree can also be modified to represent the relation instance. Assignment of attributes and argument tags is the same as for the sequence representation. To simplify the representation, we ignore

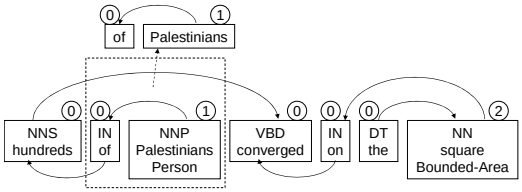

Figure 3: An example dependency parse tree representation. The subgraph represents a dependency relation feature between arg_1 *"Palestinians"* and *"of"*.

the dependency relation types.

4.3 Features

Given the above definition of relation instance graphs, we are now ready to define features. Intuitively, a feature of a relation instance captures part of the attributive and/or structural properties of the relation instance graph. Therefore, it is natural to define a feature as a subgraph of the relation instance graph. Formally, given a graph $G = (V, E, A, B)$, which represents a single relation instance, a feature that exists in this relation instance is a subgraph $G' = (V', E', A', B')$ that satisfies the following conditions: $V' \subseteq V$, $E' \subseteq E$, and $\forall v \in V', A'(v) \subseteq A(v), B'(v) = B(v)$.

We now show that many features that have been explored in previous work on relation extraction can be transformed into this graphic representation. See Figures 1, 2 and 3 for some examples.

Entity Attributes: Previous studies have shown that entity types and entity mention types of arg_1 and arg_2 are very useful (Zhao and Grishman, 2005; Zhou et al., 2005; Zhang et al., 2006b). To represent a single entity attribute, we can take a subgraph that contains only the node representing the head word of the argument, labeled with the entity type or entity mention type. A particularly useful type of features are *conjunctive entity features*, which are conjunctions of two entity attributes, one for each argument. To represent a conjunctive feature such as "arg_1 is a *Person* entity and arg_2 is a *Bounded-Area* entity", we can take a subgraph that contains two nodes, one for each argument, and each labeled with an entity attribute. Note that in this case, the subgraph contains two disconnected components, which is allowed by our definition.

Bag-of-Words: These features have also been

explore by Zhao and Grishman (2005) and Zhou et. al. (2005). To represent a bag-of-word feature, we can simply take a subgraph that contains a single node labeled with the token. Because the node also has an argument tag, we can distinguish between argument word and non-argument word.

Bigrams: A bigram feature (Zhao and Grishman, 2005) can be represented by a subgraph consisting of two connected nodes from the sequence representation, where each node is labeled with the token.

Grammar Productions: The features in convolution tree kernels for relation extraction (Zhang et al., 2006a; Zhang et al., 2006b) are sequences of grammar productions, that is, complete subtrees of the syntactic parse tree. Therefore, these features can naturally be represented by subgraphs of the relation instance graphs.

Dependency Relations and Dependency Paths: These features have been explored by Bunescu and Mooney (2005a), Zhao and Grishman (2005), and Zhou et. al. (2005). A dependency relation can be represented as an edge connecting two nodes from the dependency tree. The dependency path between the two arguments can also be easily represented as a path in the dependency tree connecting the two nodes that represent the two arguments.

There are some features that are not covered by our current definition, but can be included if we modify our relation instance graphs. For example, gapped subsequence features in subsequence kernels (Bunescu and Mooney, 2005b) can be represented as subgraphs of the sequence representation if we add more edges to connect any pair of nodes u and v provided that the token represented by u occurs somewhere before that represented by v in the sentence. Since our feature definition is very general, our feature space also includes many features that have not been explored before.

4.4 Searching the Feature Space

Although the feature space we have defined is relatively complete and has a clear structure, it is still too expensive to exhaustively search the space because the number of features is exponential in terms of the size of the relation instance graph. We thus propose to search the feature space in the following bottom-up manner: We start with the conjunctive entity features (defined in Section 4.3), which have been found effective in previous studies and are intuitively necessary for relation extraction. We then systematically add unit features with different granularities. We first consider the minimum (i.e. most basic) unit features. We then gradually include more complex features. The motivations for this strategy are the following: (1) Using the smallest features to represent a relation instance graph presumably covers all unit characteristics of the graph. (2) Using small subgraphs allows fuzzy matching, which is good for our task because relation instances of the same type may vary in their relation instance graphs, especially with the noise introduced by adjectives, adverbs, or irrelevant propositional phrases. (3) The number of features of a fixed small size is polynomial in terms of the size of the relation instance graph. It is therefore feasible to generate all the small unit features and use any classifier such as a maximum entropy classifier or an SVM.

In our experiments, we consider three levels of small unit features in increasing order of their complexity. First, we consider *unigram* features $G_{uni} = (\{u\}, \emptyset, A_{uni}, B)$, where $A_{uni}(u) = \{a_i\} \subseteq A(u)$. In another word, unigram features consist of a single node labeled with a single attribute. Examples of unigram features include bag-of-word features and non-conjunctive entity attribute features. At the second level, we consider *bigram* features $G_{bi} = (\{u, v\}, \{(u, v)\}, A_{uni}, B)$. *Bigram* features are therefore single edges connecting two nodes, where each node is labeled with a single attribute. The third level of attributes we consider are *trigram* features $G_{tri} = (\{u, v, w\}, \{(u, v), (u, w)\}, A_{uni}, B)$ or $G_{tri} = (\{u, v, w\}, \{(u, v), (v, w)\}, A_{uni}, B)$. Thus *trigram* features consist of two connected edges and three nodes, where each node is also labeled with a single attribute.

We treat the three relation instance graphs (sequences, syntactic parse trees, and dependency parse trees) as three feature subspaces, and search in each subspace. For each feature subspace, we incrementally add the unigram, bigram and trigram features to the working feature set. For the syntactic parse tree representation, we also consider a fourth level of small unit features, which are single grammar productions such as [VP $\rightarrow$ VBD PP], because these are the smallest features in convolution tree kernels. After we explore each feature subspace, we try to

combine the features from the three subspaces to see whether the performance can be improved, that is, we test whether the sequence, syntactic and dependency relations can complement each other.

5 Experiments

5.1 Data Set and Experiment Setup

We used the data set from ACE (Automatic Content Extraction) 2004 evaluation to conduct our experiments. This corpus defines 7 types of relations: *Physical*, *Personal / Social*, *Empolyment / Memebership / Subsidiary*, *Agent-Artifact*, *PER / ORG Affiliation*, *GPE Affiliation* and *Discourse*.

We used Collins parser to parse the sentences in the corpus because Collins parser gives us the head of each syntactic category, which allows us to transform the syntactic parse trees into dependency trees. We discarded sentences that could not be parsed by Collins parser. The candidate relation instances were generated by considering all pairs of entities that occur in the same sentence. We obtained 48625 candidate relation instances in total, among which 4296 instances were positive.

As in most existing work, instead of using the entire sentence, we used only the sequence of tokens that are inside the minimum complete subtree covering the two arguments. Presumably, tokens outside of this subtree are not so relevant to the task. In our graphic representation of relation instances, the attribute set for a token node includes the token itself, its POS tag, and entity type, entity subtype and entity mention type when applicable. The attribute set for a syntactic category node includes only the syntactic tag. We used both maximum entropy classifier and SVM for all experiments. We adopted one vs. others strategy for the multi-class classification problem. In all experiments, the performance shown was based on 5-fold cross validation.

5.2 General Search in the Feature Subspaces

Following the general search strategy, we conducted the following experiments. For each feature subspace, we started with the conjunctive entity features plus the unigram features. We then incrementally added bigram and trigram features. For the syntactic parse tree feature space, we conducted an additional experiment: We added basic grammar production features on top of the unigram, bigram and trigram features. Adding production features allows us to study the effect of adding more complex and presumably more specific and discriminative features.

Table 1 shows the precision (P), recall (R) and F1 measure (F) from the experiments with the maximum entropy classifier (ME) and the SVM classifier (SVM). We can compare the results in two dimensions. First, within each feature subspace, while bigram features improved the performance significantly over unigrams, trigrams did not improve the performance very much. This trend is observed for both classifiers. In the case of the syntactic parse tree subspace, adding production features even hurt the performance. This suggests that inclusion of complex features is not guaranteed to improve the performance.

Second, if we compare the best performance achieved in each feature subspace, we can see that for both classifiers, syntactic parse tree is the most effective feature space, while sequence and dependency tree are similar. However, the difference in performance between the syntactic parse tree subspace and the other two subspaces is not very large. This suggests that each feature subspace alone already captures most of the useful structural information between tokens for relation extraction. The reason why the sequence feature subspace gave good performance although it contained the least structural information is probably that many relations defined in the ACE corpus are short-range relations, some within single noun phrases. For such kind of relations, sequence information may be even more reliable than syntactic or dependency information, which may not be accurate due to parsing errors.

Next, we conducted experiments to combine the features from the three subspaces to see whether this could further improve the performance. For sequence subspace and dependency tree subspace, we used up to bigram features, and for syntactic parse tree subspace, we used up to trigram features. In Table 2, we show the experiment results. We can see that for both classifiers, adding features from the sequence subspace or from the dependency tree subspace to the syntactic parse tree subspace can improve the performance slightly. But combining sequence subspace and dependency tree subspace does not generate any performance improvement. Again,

			Uni	+Bi	+Tri	+Prod
ME	Seq	P	0.647	0.662	0.717	N/A
		R	0.614	0.701	0.653	
		F	0.630	0.681	0.683	
	Syn	P	0.651	0.695	0.726	0.702
		R	0.645	0.698	0.688	0.691
		F	0.648	0.697	**0.707**	0.696
	Dep	P	0.647	0.673	0.718	N/A
		R	0.614	0.676	0.652	
		F	0.630	0.674	0.683	
SVM	Seq	P	0.583	0.666	0.684	N/A
		R	0.586	0.650	0.648	
		F	0.585	0.658	0.665	
	Syn	P	0.598	0.645	0.679	0.674
		R	0.611	0.663	0.681	0.672
		F	0.604	0.654	**0.680**	0.673
	Dep	P	0.583	0.644	0.682	N/A
		R	0.586	0.638	0.645	
		F	0.585	0.641	0.663	

Table 1: Comparison among the three feature subspaces and the effect of including larger features.

		Seq+Syn	Seq+Dep	Syn+Dep	All
ME	P	0.737	0.687	0.695	0.724
	R	0.694	0.682	0.731	0.702
	F	**0.715**	0.684	0.712	0.713
SVM	P	0.689	0.669	0.687	0.691
	R	0.686	0.653	0.682	0.686
	F	**0.688**	0.661	0.684	**0.688**

Table 2: The effect of combining the three feature subspaces.

this suggests that since many of the ACE relations are local, there is likely much overlap between sequence information and dependency information.

We also tried the convolution tree kernel method (Zhang et al., 2006a), using an SVM tree kernel package[2]. The performance we obtained was P = 0.705, R = 0.685, and F = 0.695[3]. This F measure is higher than the best SVM performance in Table 1. The convolution tree kernel uses large subtree features, but such features are deemphasized with an exponentially decaying weight. We found that the performance was sensitive to this decaying factor, suggesting that complex features can be useful if they are weighted appropriately, and further study of how to optimize the weights of such complex features is needed.

[2]http://ai-nlp.info.uniroma2.it/moschitti/Tree-Kernel.htm

[3]The performance we achieved is lower than that reported in (Zhang et al., 2006b), due to different data preprocessing, data partition, and parameter setting.

5.3 Task-Oriented Feature Pruning

Apart from the general bottom-up search strategy we have proposed, we can also introduce some task-oriented heuristics based on intuition or domain knowledge to prune the feature space. In our experiments, we tried the following heuristics.

H1: Zhang et al. (2006a) found that using *path-enclosed tree* performed better than using *minimum complete tree*, when convolution tree kernels were applied. In path-enclosed trees, tokens before arg_1 and after arg_2 as well as their links with other nodes in the tree are removed. Based on this previous finding, our first heuristic was to change the syntactic parse tree representation of the relation instances into path-enclosed trees.

H2: We hypothesize that words such as articles, adjectives and adverbs are not very useful for relation extraction. We thus removed sequence unigram features and bigram features that contain an article, adjective or adverb.

H3: Similar to H2, we can remove bigrams in the syntactic parse tree subspace if the bigram contains an article, adjective or adverb.

H4: Similar to H1, we can also remove the tokens before arg_1 and after arg_2 from the sequence representation of a relation instance.

In Table 3, we show the performance after applying these heuristics. We started with the best configuration from our previous experiments, that is, combing up to bigram features in the sequence subspace and up to trigram features in the syntactic tree subspace. We then applied heuristics H1 to H4 incrementally unless we saw that a heuristic was not effective. We found that H1, H2 and H4 slightly improved the performance, but H3 hurt the performance. On the one hand, the improvement suggests that our original feature configuration included some irrelevant features, and in turn confirmed that over-inclusion of features could hurt the performance. On the other hand, since the improvement brought by H1, H2 and H4 was rather small, and H3 even hurt the performance, we could see that it is in general very hard to find good feature pruning heuristics.

6 Conclusions and Future Work

In this paper, we conducted a systematic study of the feature space for relation extraction. We pro-

	ME			SVM		
	P	R	F	P	R	F
Best	0.737	0.694	0.715	0.689	0.686	0.688
+H1	0.714	0.729	0.721	0.698	0.699	0.699
+H2	0.730	0.723	0.726	0.704	0.704	**0.704**
+H3	0.739	0.704	0.721	0.701	0.696	0.698
-H3+H4	0.746	0.713	**0.729**	0.702	0.701	0.702

Table 3: The effect of various heuristic feature pruning methods.

posed and defined a unified graphic representation of features for relation extraction, which serves as a general framework for systematically exploring features defined on natural language sentences. With this framework, we explored three different representations of sentences—sequences, syntactic parse trees, and dependency trees—which lead to three feature subspaces. In each subspace, starting with the basic unit features, we systematically explored features of different levels of complexity. The studied feature space includes not only most of the effective features explored in previous work, but also some features that have not been considered before.

Our experiment results showed that using a set of basic unit features from each feature subspace, we can achieve reasonably good performance. When the three subspaces are combined, the performance can improve only slightly, which suggests that the sequence, syntactic and dependency relations have much overlap for the task of relation extraction. We also found that adding more complex features may not improve the performance much, and may even hurt the performance. A combination of features of different levels of complexity and from different sentence representations, coupled with task-oriented feature pruning, gives the best performance.

In our future work, we will study how to automatically conduct task-oriented feature search, feature pruning and feature weighting using statistical methods instead of heuristics. In this study, we only considered features from the local context, i.e. the sentence that contains the two arguments. Some existing studies use corpus-based statistics for relation extraction (Hasegawa et al., 2004). In the future, we will study the effectiveness of these global features.

Acknowledgments

This work was in part supported by the National Science Foundation under award numbers 0425852 and 0428472. We thank Alessandro Moschitti for providing the SVM tree kernel package. We also thank Min Zhang for providing the implementation details of the convolution tree kernel for relation extraction.

References

Razvan C. Bunescu and Raymond J. Mooney. 2005a. A shortest path dependency kenrel for relation extraction. In *Proceedings of HLT/EMNLP*.

Razvan C. Bunescu and Raymond J. Mooney. 2005b. Subsequence kernels for relation extraction. In *Proceedings of NIPS*.

Aron Culotta and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of ACL*.

Chad Cumby and Dan Roth. 2003. On kernel methods for relational learning. In *Proceedings of ICML*.

Takaaki Hasegawa, Satoshi Sekine, and Ralph Grishman. 2004. Discovering relations among named entities from large corpora. In *Proceedings ACL*.

Nanda Kambhatla. 2004. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In *Proceedings of ACL*.

Jun Suzuki, Tsutomu Hirao, Yutaka Sasaki, and Eisaku Maeda. 2003. Hierarchical directed acyclic graph kernel: Methods for structured natural language data. In *Proceedings of ACL*.

Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. 2003. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:1083–1106.

Min Zhang, Jie Zhang, and Jian Su. 2006a. Exploring syntactic features for relation extraction using a convolution tree kernel. In *Proceedings of HLT/NAACL*.

Min Zhang, Jie Zhang, Jian Su, and Guodong Zhou. 2006b. A composite kernel to extract relations between entities with both flat and structured features. In *Proceedings of ACL*.

Shubin Zhao and Ralph Grishman. 2005. Extracting relations with integrated information using kernel methods. In *Proceedings of ACL*.

GuoDong Zhou, Jian Su, Jie Zhang, and Min Zhang. 2005. Exploring various knowledge in relation extraction. In *Proceedings of ACL*.