A Composite Kernel to Extract Relations between Entities with both Flat and Structured Features

Min Zhang Jie Zhang Jian Su Guodong Zhou Institute for Infocomm Research 21 Heng Mui Keng Terrace, Singapore 119613 {mzhang, zhangjie, sujian, zhougd}@i2r.a-star.edu.sg

Abstract

This paper proposes a novel composite kernel for relation extraction. The composite kernel consists of two individual kernels: an entity kernel that allows for entity-related features and a convolution parse tree kernel that models syntactic information of relation examples. The motivation of our method is to fully utilize the nice properties of kernel methods to explore diverse knowledge for relation extraction. Our study illustrates that the composite kernel can effectively capture both flat and structured features without the need for extensive feature engineering, and can also easily scale to include more features. Evaluation on the ACE corpus shows that our method outperforms the previous best-reported methods and significantly outperforms previous two dependency tree kernels for relation extraction.

1 Introduction

The goal of *relation extraction* is to find various predefined semantic relations between pairs of entities in text. The research on relation extraction has been promoted by the Message Understanding Conferences (MUCs) (MUC, 1987-1998) and Automatic Content Extraction (ACE) program (ACE, 2002-2005). According to the ACE Program, an entity is an object or set of objects in the world and a *relation* is an explicitly or implicitly stated relationship among entities. For example, the sentence "Bill Gates is chairman and chief software architect of Microsoft Corporation." conveys the ACE-style relation "EMPLOYMENT.exec" between the entities "Bill Gates" (PERSON.Name) and "Microsoft Corporation" (ORGANIZATION. Commercial).

In this paper, we address the problem of relation extraction using kernel methods (Schölkopf and Smola, 2001). Many feature-based learning algorithms involve only the dot-product between feature vectors. Kernel methods can be regarded as a generalization of the feature-based methods by replacing the dot-product with a kernel function between two vectors, or even between two objects. A kernel function is a similarity function satisfying the properties of being symmetric and positive-definite. Recently, kernel methods are attracting more interests in the NLP study due to their ability of implicitly exploring huge amounts of structured features using the original representation of objects. For example, the kernels for structured natural language data, such as parse tree kernel (Collins and Duffy, 2001), string kernel (Lodhi et al., 2002) and graph kernel (Suzuki et al., 2003) are example instances of the wellknown *convolution kernels*¹ in NLP. In relation extraction, typical work on kernel methods includes: Zelenko et al. (2003), Culotta and Sorensen (2004) and Bunescu and Mooney (2005).

This paper presents a novel composite kernel to explore diverse knowledge for relation extraction. The composite kernel consists of an entity kernel and a convolution parse tree kernel. Our study demonstrates that the composite kernel is very effective for relation extraction. It also shows without the need for extensive feature engineering the composite kernel can not only capture most of the flat features used in the previous work but also exploit the useful syntactic structure features effectively. An advantage of our method is that the composite kernel can easily cover more knowledge by introducing more kernels. Evaluation on the ACE corpus shows that our method outperforms the previous bestreported methods and significantly outperforms the previous kernel methods due to its effective exploration of various syntactic features.

The rest of the paper is organized as follows. In Section 2, we review the previous work. Section 3 discusses our composite kernel. Section 4 reports the experimental results and our observations. Section 5 compares our method with the

¹ Convolution kernels were proposed for a discrete structure by Haussler (1999) in the machine learning field. This framework defines a kernel between input objects by applying convolution "sub-kernels" that are the kernels for the decompositions (parts) of the objects.

previous work from the viewpoint of feature exploration. We conclude our work and indicate the future work in Section 6.

2 Related Work

Many techniques on relation extraction, such as rule-based (MUC, 1987-1998; Miller et al., 2000), feature-based (Kambhatla 2004; Zhou et al., 2005) and kernel-based (Zelenko et al., 2003; Culotta and Sorensen, 2004; Bunescu and Mooney, 2005), have been proposed in the literature.

Rule-based methods for this task employ a number of linguistic rules to capture various relation patterns. Miller et al. (2000) addressed the task from the syntactic parsing viewpoint and integrated various tasks such as POS tagging, NE tagging, syntactic parsing, template extraction and relation extraction using a generative model.

Feature-based methods (Kambhatla, 2004; Zhou et al., 2005; Zhao and Grishman, 2005^2) for this task employ a large amount of diverse linguistic features, such as lexical, syntactic and semantic features. These methods are very effective for relation extraction and show the bestreported performance on the ACE corpus. However, the problems are that these diverse features have to be manually calibrated and the hierarchical structured information in a parse tree is not well preserved in their parse tree-related features, which only represent simple flat *path* information connecting two entities in the parse tree through a *path* of non-terminals and a *list* of base phrase chunks.

Prior kernel-based methods for this task focus on using individual tree kernels to exploit tree structure-related features. Zelenko et al. (2003) developed a kernel over parse trees for relation extraction. The kernel matches nodes from roots to leaf nodes recursively layer by layer in a topdown manner. Culotta and Sorensen (2004) generalized it to estimate similarity between dependency trees. Their tree kernels require the matchable nodes to be at the same layer counting from the root and to have an identical path of ascending nodes from the roots to the current nodes. The two constraints make their kernel high precision but very low recall on the ACE 2003 corpus. Bunescu and Mooney (2005) proposed another dependency tree kernel for relation extraction.

Their kernel simply counts the number of common word classes at each position in the shortest paths between two entities in dependency trees. The kernel requires the two paths to have the same length; otherwise the kernel value is zero. Therefore, although this kernel shows performance improvement over the previous one (Culotta and Sorensen, 2004), the constraint makes the two dependency kernels share the similar behavior: good precision but much lower recall on the ACE corpus.

The above discussion shows that, although kernel methods can explore the huge amounts of implicit (structured) features, until now the feature-based methods enjoy more success. One may ask: how can we make full use of the nice properties of kernel methods and define an effective kernel for relation extraction?

In this paper, we study how relation extraction can benefit from the elegant properties of kernel methods: 1) implicitly exploring (structured) features in a high dimensional space; and 2) the nice mathematical properties, for example, the sum, product, normalization and polynomial expansion of existing kernels is a valid kernel (Schölkopf and Smola, 2001). We also demonstrate how our composite kernel effectively captures the diverse knowledge for relation extraction.

3 Composite Kernel for Relation Extraction

In this section, we define the composite kernel and study the effective representation of a relation instance.

3.1 Composite Kernel

Our composite kernel consists of an entity kernel and a convolution parse tree kernel. To our knowledge, convolution kernels have not been explored for relation extraction.

(1) Entity Kernel: The ACE 2003 data defines four entity features: entity headword, entity type and subtype (only for GPE), and mention type while the ACE 2004 data makes some modifications and introduces a new feature "LDC mention type". Our statistics on the ACE data reveals that the entity features impose a strong constraint on relation types. Therefore, we design a linear kernel to explicitly capture such features:

$$K_{L}(R_{1}, R_{2}) = \sum_{i=1,2} K_{E}(R_{1}.E_{i}, R_{2}.E_{i})$$
(1)

where R_1 and R_2 stands for two relation instances, E_i means the *i*th entity of a relation instance, and

 $^{^{2}}$ We classify the feature-based kernel defined in (Zhao and Grishman, 2005) into the feature-based methods since their kernels can be easily represented by the dot-products between explicit feature vectors.

 $K_E(\cdot, \cdot)$ is a simple kernel function over the features of entities:

$$K_{E}(E_{1}, E_{2}) = \sum_{i} C(E_{1}.f_{i}, E_{2}.f_{i})$$
(2)

where f_i represents the i^{th} entity feature, and the function $C(\cdot, \cdot)$ returns 1 if the two feature values are identical and 0 otherwise. $K_E(\cdot, \cdot)$ returns the number of feature values in common of two entities.

(2) Convolution Parse Tree Kernel: A convolution kernel aims to capture structured information in terms of substructures. Here we use the same convolution parse tree kernel as described in Collins and Duffy (2001) for syntactic parsing and Moschitti (2004) for semantic role labeling. Generally, we can represent a parse tree T by a vector of integer counts of each sub-tree type (regardless of its ancestors):

$$\phi(T) = (\# subtree_{l}(T), ..., \# subtree_{i}(T), ..., \# subtree_{n}(T))$$

where # subtree_i(T) is the occurrence number of the *i*th sub-tree type (subtree_i) in T. Since the number of different sub-trees is exponential with the parse tree size, it is computationally infeasible to directly use the feature vector $\phi(T)$. To solve this computational issue, Collins and Duffy (2001) proposed the following parse tree kernel to calculate the dot product between the above high dimensional vectors implicitly.

$$K(T_{1},T_{2}) = \langle \phi(T_{1}), \phi(T_{2}) \rangle$$

$$= \sum_{i} \# subtree_{i}(T_{1}) \cdot \# subtree_{i}(T_{2})$$

$$= \sum_{i} \left(\sum_{n_{i} \in N_{1}} I_{subtree_{i}}(n_{1}) \right) \cdot \left(\sum_{n_{2} \in N_{2}} I_{subtree_{i}}(n_{2}) \right)$$

$$= \sum_{n_{i} \in N_{1}} \sum_{n_{2} \in N_{2}} \Delta(n_{1},n_{2})$$
(3)

where N_1 and N_2 are the sets of nodes in trees T_1 and T_2 , respectively, and $I_{subtree_i}(n)$ is a function that is 1 iff the *subtree_i* occurs with root at node *n* and zero otherwise, and $\Delta(n_1, n_2)$ is the number of the common *subtrees* rooted at n_1 and n_2 , i.e.

$$\Delta(n_1, n_2) = \sum_i I_{subtree_i}(n_1) \cdot I_{subtree_i}(n_2)$$

 $\Delta(n_1, n_2)$ can be computed by the following recursive rules:

- (1) if the productions (CFP rules) at n_1 and n_2 are different, $\Delta(n_1, n_2) = 0$;
- (2) else if both n_1 and n_2 are pre-terminals (POS tags), $\Delta(n_1, n_2) = 1 \times \lambda$;
- (3) else, $\Delta(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + \Delta(ch(n_1, j), ch(n_2, j))),$

where $nc(n_1)$ is the child number of n_1 , $ch(n_j)$ is the j^{th} child of node n and λ ($0 < \lambda < 1$) is the decay factor in order to make the kernel value less variable with respect to the *subtree* sizes. In addition, the recursive rule (3) holds because given two nodes with the same children, one can construct common sub-trees using these children and common sub-trees of further offspring.

The parse tree kernel counts the number of common sub-trees as the syntactic similarity measure between two relation instances. The time complexity for computing this kernel is $O(|N_1| \cdot |N_2|)$.

In this paper, two composite kernels are defined by combing the above two individual kernels in the following ways:

1) Linear combination:

$$K_1(R_1, R_2) = \alpha \cdot \hat{K}_L(R_1, R_2) + (1 - \alpha) \cdot \hat{K}(T_1, T_2)$$
(4)

Here, $\hat{K}(\bullet, \bullet)$ is the normalized ${}^{3}K(\bullet, \bullet)$ and α is the coefficient. Evaluation on the development set shows that this composite kernel yields the best performance when α is set to 0.4.

2) Polynomial expansion:

$$K_2(R_1, R_2) = \alpha \cdot \hat{K}_L^P(R_1, R_2) + (1 - \alpha) \cdot \hat{K}(T_1, T_2)$$
(5)

Here, $\hat{K}(\bullet, \bullet)$ is the normalized $K(\bullet, \bullet)$, $K^{p}(\bullet, \bullet)$ is the polynomial expansion of $K(\bullet, \bullet)$ with degree d=2, i.e. $K^{p}(\bullet, \bullet) = (K(\bullet, \bullet)+1)^{2}$, and α is the coefficient. Evaluation on the development set shows that this composite kernel yields the best performance when α is set to 0.23.

The polynomial expansion aims to explore the entity bi-gram features, esp. the combined features from the first and second entities, respectively. In addition, due to the different scales of the values of the two individual kernels, they are normalized before combination. This can avoid one kernel value being overwhelmed by that of another one.

The entity kernel formulated by *eqn*. (1) is a proper kernel since it simply calculates the dot product of the entity feature vectors. The tree kernel formulated by *eqn*. (3) is proven to be a proper kernel (Collins and Duffy, 2001). Since kernel function set is closed under normalization, polynomial expansion and linear combination (Schölkopf and Smola, 2001), the two composite kernels are also proper kernels.

³ A kernel K(x, y) can be normalized by dividing it by $\sqrt{K(x, x) \cdot K(y, y)}$.

3.2 Relation Instance Spaces

A relation instance is encapsulated by a parse tree. Thus, it is critical to understand which portion of a parse tree is important in the kernel calculation. We study five cases as shown in Fig.1.

(1) Minimum Complete Tree (MCT): the complete sub-tree rooted by the nearest common ancestor of the two entities under consideration.

(2) Path-enclosed Tree (PT): the smallest common sub-tree including the two entities. In other words, the sub-tree is enclosed by the shortest path linking the two entities in the parse tree (this path is also commonly-used as the path tree feature in the feature-based methods).

(3) Context-Sensitive Path Tree (CPT): the PT extended with the 1^{st} left word of entity 1 and the 1^{st} right word of entity 2.

(4) Flattened Path-enclosed Tree (FPT): the **PT** with the single in and out arcs of non-terminal nodes (except POS nodes) removed.

(5) Flattened CPT (FCPT): the CPT with the single in and out arcs of non-terminal nodes (except POS nodes) removed.

Fig. 1 illustrates different representations of an example relation instance. T_1 is MCT for the relation instance, where the sub-tree circled by a dashed line is **PT**, which is also shown in T_2 for clarity. The only difference between MCT and **PT** lies in that **MCT** does not allow partial production rules (for example, NP \rightarrow PP is a partial production rule while NP→NP+PP is an entire production rule in the top of T_2). For instance, only the most-right child in the most-left sub-tree [NP [CD 200] [JJ domestic] [E1-PER ...]] of T_1 is kept in T_2 . By comparing the performance of T_1 and T_2 , we can evaluate the effect of sub-trees with partial production rules as shown in T_2 and the necessity of keeping the whole left and right context sub-trees as shown in T_1 in relation extraction. T_3 is **CPT**, where the two sub-trees circled by dashed lines are included as the context to T_2 and make T_3 context-sensitive. This is to evaluate whether the limited context information in CPT can boost performance. FPT in T_4 is formed by removing the two circled nodes in T_2 . This is to study whether and how the elimination of single non-terminal nodes affects the performance of relation extraction.



Figure 1. Different representations of a relation instance in the example sentence "... provide benefits to 200 domestic partners of their own workers in New York", where the phrase type "E1-PER" denotes that the current node is the 1st entity with type "PERSON", and likewise for the others. The relation instance is excerpted from the ACE 2003 corpus, where a relation "SOCIAL.Other-Personal" exists between entities "partners" (PER) and "workers" (PER). We use Charniak's parser (Charniak, 2001) to parse the example sentence. To save space, the FCPT is not shown here.

4 **Experiments**

4.1 Experimental Setting

Data: We use the English portion of both the ACE 2003 and 2004 corpora from LDC in our experiments. In the ACE 2003 data, the training set consists of 674 documents and 9683 relation instances while the test set consists of 97 documents and 1386 relation instances. The ACE 2003 data defines 5 entity types, 5 major relation types and 24 relation subtypes. The ACE 2004 data contains 451 documents and 5702 relation instances. It redefines 7 entity types, 7 major relation types and 23 subtypes. Since Zhao and Grishman (2005) use a 5-fold cross-validation on a subset of the 2004 data (newswire and broadcast news domains, containing 348 documents and 4400 relation instances), for comparison, we use the same setting (5-fold cross-validation on the same subset of the 2004 data, but the 5 partitions may not be the same) for the ACE 2004 data. Both corpora are parsed using Charniak's parser (Charniak, 2001). We iterate over all pairs of entity mentions occurring in the same sentence to generate potential relation instances. In this paper, we only measure the performance of relation extraction models on "true" mentions with "true" chaining of coreference (i.e. as annotated by LDC annotators).

Implementation: We formalize relation extraction as a multi-class classification problem. SVM is selected as our classifier. We adopt the *one vs. others* strategy and select the one with the largest margin as the final answer. The training parameters are chosen using cross-validation (C=2.4 (SVM); $\lambda = 0.4$ (tree kernel)). In our implementation, we use the binary SVMLight (Joachims, 1998) and Tree Kernel Tools (Moschitti, 2004). Precision (**P**), Recall (**R**) and F-measure (**F**) are adopted to measure the performance.

4.2 Experimental Results

In this subsection, we report the experiments of different kernel setups for different purposes.

(1) Tree Kernel only over Different Relation Instance Spaces: In order to better study the impact of the syntactic structure information in a parse tree on relation extraction, we remove the entity-related information from parse trees by replacing the entity-related phrase types ("E1-PER" and so on as shown in Fig. 1) with "NP". Table 1 compares the performance of 5 tree kernel setups on the ACE 2003 data using the tree structure information only. It shows that: • Overall the five different relation instance spaces are all somewhat effective for relation extraction. This suggests that structured syntactic information has good predication power for relation extraction and the structured syntactic information can be well captured by the tree kernel.

- MCT performs much worse than the others. The reasons may be that MCT includes too much left and right context information, which may introduce many noisy features and cause over-fitting (high precision and very low recall as shown in Table 1). This suggests that only keeping the complete (not partial) production rules in MCT does harm performance.
- **PT** achieves the best performance. This means that only keeping the portion of a parse tree enclosed by the shortest path between entities can model relations better than all others. This may be due to that most significant information is with **PT** and including context information may introduce too much noise. Although context may include some useful information, it is still a problem to correctly utilize such useful information.
- **CPT** performs a bit worse than **PT**. In some cases (e.g. in sentence "the merge of company A and company B....", "merge" is a critical context word), the context information is helpful. However, the effective scope of context is hard to determine given the complexity and variability of natural languages.
- The two flattened trees perform worse than the original trees. This suggests that the single non-terminal nodes are useful for relation extraction.

Evaluation on the ACE 2004 data also shows that **PT** achieves the best performance (72.5/56.7 /63.6 in **P/R/F**). More evaluations with the entity type and order information incorporated into tree nodes ("E1-PER", "E2-PER" and "E-GPE" as shown in Fig. 1) also show that **PT** performs best with 76.1/62.6/68.7 in **P/R/F** on the 2003 data and 74.1/62.4/67.7 in **P/R/F** on the 2004 data.

Instance Spaces	P (%)	R (%) F
Minimum Complete Tree (MCT)	77.5	38.4 51.3
Path-enclosed Tree (PT)	72.8	53.8 61.9
Context-Sensitive PT(CPT)	75.9	48.6 59.2
Flattened PT	72.7	51.7 60.4
Flattened CPT	76.1	47.2 58.2

Table 1. five different tree kernel setups on the ACE 2003 five major types using the parse tree structure information only (regardless of any entity-related information)

PTs (with Tree Struc- ture Information only)	P (%)	R (%)	F
Entity kernel only	75.1	42.7	54.4
	(79.5)	(34.6)	(48.2)
Tree kernel only	72.5	56.7	63.6
	(72.8)	(53.8)	(61.9)
Composite kernel 1	73.5	67.0	70.1
(linear combination)	(76.3)	(63.0)	(69.1)
Composite kernel 2	76.1	68.4	72.1
(polynomial expansion)	(77.3)	(65.6)	(70.9)

Table 2. Performance comparison of different kernel setups over the ACE major types of both the 2003 data (the numbers in parentheses) and the 2004 data (the numbers outside parentheses)

(2) **Composite Kernels**: Table 2 compares the performance of different kernel setups on the ACE major types. It clearly shows that:

- The composite kernels achieve significant performance improvement over the two individual kernels. This indicates that the flat and the structured features are complementary and the composite kernels can well integrate them: 1) the flat entity information captured by the entity kernel; 2) the structured syntactic connection information between the two entities captured by the tree kernel.
- The composite kernel via the polynomial expansion outperforms the one via the linear combination by ~2 in F-measure. It suggests that the bi-gram entity features are very useful.
- The entity features are quite useful, which can achieve F-measures of 54.4/48.2 alone and can boost the performance largely by ~7 (70.1-63.2/69.1-61.9) in F-measure when combining with the tree kernel.
- It is interesting that the ACE 2004 data shows consistent better performance on all setups than the 2003 data although the ACE 2003 data is two times larger than the ACE 2004 data. This may be due to two reasons: 1) The ACE 2004 data defines two new entity types and re-defines the relation types and subtypes in order to reduce the inconsistency between LDC annotators. 2) More importantly, the ACE 2004 data defines 43 entity subtypes while there are only 3 subtypes in the 2003 data. The detailed classification in the 2004 data leads to significant performance improvement of 6.2 (54.4-48.2) in F-measure over that on the 2003 data.

Our composite kernel can achieve 77.3/65.6/70.9 and 76.1/68.4/72.1 in **P/R/F** over the ACE 2003/2004 major types, respectively.

Methods (2002/2003 data)	P (%)	R (%)	F
Ours: composite kernel 2	77.3	65.6	70.9
(polynomial expansion)	(64.9)	(51.2)	(57.2)
Zhou et al. (2005):	77.2	60.7	68.0
feature-based SVM	(63.1)	(49.5)	(55.5)
Kambhatla (2004):	(-)	(-)	(-)
feature-based ME	(63.5)	(45.2)	(52.8)
Ours: tree kernel with en-	76.1	62.6	68.7
tity information at node	(62.4)	(48.5)	(54.6)
Bunescu and Mooney	65.5	43.8	52.5
(2005): shortest path de- pendency kernel	(-)	(-)	(-)
Culotta and Sorensen (2004): dependency kernel	67.1	35.0	45.8

Table 3. Performance comparison on the ACE 2003/2003 data over both 5 major types (the numbers outside parentheses) and 24 subtypes (the numbers in parentheses)

Methods (2004 data)	P (%)	R (%)	F
Ours: composite kernel 2	76.1	68.4	72.1
(polynomial expansion)	(68.6)	(59.3)	(63.6)
Zhao and Grishman (2005):	69.2	70.5	70.4
feature-based kernel	(-)	(-)	(-)

Table 4. Performance comparison on the ACE 2004 data over both 7 major types (the numbers outside parentheses) and 23 subtypes (the numbers in parentheses)

(3) **Performance Comparison**: Tables 3 and 4 compare our method with previous work on the ACE 2002/2003/2004 data, respectively. They show that our method outperforms the previous methods and significantly outperforms the previous two dependency kernels⁴. This may be due to two reasons: 1) the dependency tree (Culotta and Sorensen, 2004) and the shortest path (Bunescu and Mooney, 2005) lack the internal hierarchical phrase structure information, so their corresponding kernels can only carry out node-matching directly over the nodes with word tokens; 2) the parse tree kernel has less constraints. That is, it is

⁴ Bunescu and Mooney (2005) used the ACE 2002 corpus, including 422 documents, which is known to have many inconsistencies than the 2003 version. Culotta and Sorensen (2004) used a generic ACE corpus including about 800 documents (no corpus version is specified). Since the testing corpora are in different sizes and versions, strictly speaking, it is not ready to compare these methods exactly and fairly. Therefore Table 3 is only for reference purpose. We just hope that we can get a few clues from this table.

not restricted by the two constraints of the two dependency kernels (identical layer and ancestors for the matchable nodes and identical length of two shortest paths, as discussed in Section 2).

The above experiments verify the effectiveness of our composite kernels for relation extraction. They suggest that the parse tree kernel can effectively explore the syntactic features which are critical for relation extraction.

Error Type	# of error instances	
	2004 data	2003 data
False Negative	198	416
False Positive	115	171
Cross Type	62	96

Table 5. Error distribution of major types on both the 2003 and 2004 data for the composite kernel by polynomial expansion

(4) Error Analysis: Table 5 reports the error distribution of the polynomial composite kernel over the major types on the ACE data. It shows that 83.5%(198+115/198+115+62) / 85.8%(416 +171/416+171+96) of the errors result from relation detection and only 16.5%/14.2% of the errors result from relation characterization. This may be due to data imbalance and sparseness issues since we find that the negative samples are 8 times more than the positive samples in the training set. Nevertheless, it clearly directs our future work.

5 Discussion

In this section, we compare our method with the previous work from the feature engineering viewpoint and report some other observations and issues in our experiments.

5.1 Comparison with Previous Work

This is to explain more about why our method performs better and significantly outperforms the previous two dependency tree kernels from the theoretical viewpoint.

(1) Compared with Feature-based Methods: The basic difference lies in the relation instance representation (parse tree vs. feature vector) and the similarity calculation mechanism (kernel function vs. dot-product). The main difference is the different feature spaces. Regarding the parse tree features, our method implicitly represents a parse tree by a vector of integer counts of each sub-tree type, i.e., we consider the entire sub-tree types and their occurring frequencies. In this way, the parse tree-related features (the *path* features and the *chunking* features) used in the featurebased methods are embedded (as a subset) in our feature space. Moreover, the in-between word features and the entity-related features used in the feature-based methods are also captured by the tree kernel and the entity kernel, respectively. Therefore our method has the potential of effectively capturing not only most of the previous flat features but also the useful syntactic structure features.

(2) Compared with Previous Kernels: Since our method only counts the occurrence of each sub-tree without considering the layer and the ancestors of the root node of the sub-tree, our method is not limited by the constraints (identical layer and ancestors for the matchable nodes, as discussed in Section 2) in Culotta and Sorensen (2004). Moreover, the difference between our method and Bunescu and Mooney (2005) is that their kernel is defined on the shortest *path* between two entities instead of the entire subtrees. However, the *path* does not maintain the tree structure information. In addition, their kernel requires the two paths to have the same length. Such constraint is too strict.

5.2 Other Issues

(1) Speed Issue: The recursively-defined convolution kernel is much slower compared to feature-based classifiers. In this paper, the speed issue is solved in three ways. First, the inclusion of the entity kernel makes the composite kernel converge fast. Furthermore, we find that the small portion (PT) of a full parse tree can effectively represent a relation instance. This significantly improves the speed. Finally, the parse tree kernel requires exact match between two subtrees, which normally does not occur very frequently. Collins and Duffy (2001) report that in practice, running time for the parse tree kernel is more close to linear $(O(|N_1|+|N_2|))$, rather than $O(|N_1|^*|N_2|)$). As a result, using the PC with Intel P4 3.0G CPU and 2G RAM, our system only takes about 110 minutes and 30 minutes to do training on the ACE 2003 (~77k training instances) and 2004 (~33k training instances) data, respectively.

(2) **Further Improvement:** One of the potential problems in the parse tree kernel is that it carries out exact matches between sub-trees, so that this kernel fails to handle sparse phrases (i.e. "a car" *vs.* "a red car") and near-synonymic grammar tags (for example, the variations of a verb (i.e. go, went, gone)). To some degree, it could possibly lead to over-fitting and compromise the per-

formance. However, the above issues can be handled by allowing grammar-driven partial rule matching and other approximate matching mechanisms in the parse tree kernel calculation.

Finally, it is worth noting that by introducing more individual kernels our method can easily scale to cover more features from a multitude of sources (e.g. Wordnet, gazetteers, etc) that can be brought to bear on the task of relation extraction. In addition, we can also easily implement the feature weighting scheme by adjusting the eqn.(2) and the rule (2) in calculating $\Delta(n_1, n_2)$ (see subsection 3.1).

6 Conclusion and Future Work

Kernel functions have nice properties. In this paper, we have designed a composite kernel for relation extraction. Benefiting from the nice properties of the kernel methods, the composite kernel could well explore and combine the flat entity features and the structured syntactic features, and therefore outperforms previous bestreported feature-based methods on the ACE corpus. To our knowledge, this is the first research to demonstrate that, without the need for extensive feature engineering, an individual tree kernel achieves comparable performance with the feature-based methods. This shows that the syntactic features embedded in a parse tree are particularly useful for relation extraction and which can be well captured by the parse tree kernel. In addition, we find that the relation instance representation (selecting effective portions of parse trees for kernel calculations) is very important for relation extraction.

The most immediate extension of our work is to improve the accuracy of relation detection. This can be done by capturing more features by including more individual kernels, such as the WordNet-based semantic kernel (Basili et al., 2005) and other feature-based kernels. We can also benefit from machine learning algorithms to study how to solve the data imbalance and sparseness issues from the learning algorithm viewpoint. In the future work, we will design a more flexible tree kernel for more accurate similarity measure.

Acknowledgements: We would like to thank Dr. Alessandro Moschitti for his great help in using his Tree Kernel Toolkits and fine-tuning the system. We also would like to thank the three anonymous reviewers for their invaluable suggestions.

References

- ACE. 2002-2005. The Automatic Content Extraction Projects. <u>http://www.ldc.upenn.edu/Projects /ACE/</u>
- Basili R., Cammisa M. and Moschitti A. 2005. A Semantic Kernel to classify text with very few training examples. ICML-2005
- Bunescu R. C. and Mooney R. J. 2005. A Shortest Path Dependency Kernel for Relation Extraction. EMNLP-2005
- Charniak E. 2001. Immediate-head Parsing for Language Models. ACL-2001
- Collins M. and Duffy N. 2001. Convolution Kernels for Natural Language. NIPS-2001
- Culotta A. and Sorensen J. 2004. Dependency Tree Kernel for Relation Extraction. ACL-2004
- Haussler D. 1999. *Convolution Kernels on Discrete Structures.* Technical Report UCS-CRL-99-10, University of California, Santa Cruz.
- Joachims T. 1998. Text Categorization with Support Vecor Machine: learning with many relevant features. ECML-1998
- Kambhatla N. 2004. Combining lexical, syntactic and semantic features with Maximum Entropy models for extracting relations. ACL-2004 (poster)
- Lodhi H., Saunders C., Shawe-Taylor J., Cristianini N. and Watkins C. 2002. *Text classification using string kernel*. Journal of Machine Learning Research, 2002(2):419-444
- Miller S., Fox H., Ramshaw L. and Weischedel R. 2000. A novel use of statistical parsing to extract information from text. NAACL-2000
- Moschitti A. 2004. A Study on Convolution Kernels for Shallow Semantic Parsing. ACL-2004
- MUC. 1987-1998. <u>http://www.itl.nist.gov/iaui/894.02/</u> related_projects/muc/
- Schölkopf B. and Smola A. J. 2001. *Learning with Kernels: SVM, Regularization, Optimization and Beyond.* MIT Press, Cambridge, MA 407-423
- Suzuki J., Hirao T., Sasaki Y. and Maeda E. 2003. Hierarchical Directed Acyclic Graph Kernel: Methods for Structured Natural Language Data. ACL-2003
- Zelenko D., Aone C. and Richardella A. 2003. *Kernel Methods for Relation Extraction*. Journal of Machine Learning Research. 2003(2):1083-1106
- Zhao S.B. and Grishman R. 2005. Extracting Relations with Integrated Information Using Kernel Methods. ACL-2005
- Zhou G.D., Su J, Zhang J. and Zhang M. 2005. *Exploring Various Knowledge in Relation Extraction*. ACL-2005