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

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 and combine diverse features for relation extraction. Our study illustrates that the composite kernel can capture both flat and structured features effectively, and can also easily scale to include more features. Evaluation on the ACE corpus shows that our method outperforms the previous best-reported method. It also shows that due to the effective exploration of the syntactic features the sole parse tree kernel significantly outperforms the previous two dependency kernels by 16 in F-measure on the ACE 2003 corpus.

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 prompted 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 seen 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 interesting in the NLP study due to the ability of exploring huge amounts of structured features implicitly using the original representation of objects. For example, the parse tree kernel (Collins and Duffy, 2001), string kernel (Lodhi et al., 2002) and graph kernel (Suzuki et al., 2003) are good example instances of the well-known convolution kernels 1 in the NLP field. Moschitti (2004) studied the parse tree kernel for semantic role labeling. In relation extraction, the typical work using kernel methods include: Zelenko et al. (2003), Culotta and Sorensen (2004) and Bunescu and Mooney (2005).

This paper presents a novel composite kernel to explore diverse features 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, which can not only capture most of the flat features used in the previous work but also exploit the useful syntactic structure features effectively. Our method can also easily cover more features by introducing more kernels. Evaluation on the ACE corpus shows that our method outperforms the previous best-reported method and significantly outperforms the previous kernels. It also shows that using the parse tree kernel alone could achieve the state-of-the-art performance due to the effective exploration of the 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 previous work from the viewpoint of feature exploration. We conclude our work and indicate the future work in Section 6.

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

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.

Prior rule-based methods for this task employ a number of linguistic rules to capture the various relation patterns. Miller et al. (2000) address the task of relation extraction from the syntactic parsing viewpoint. They integrate various tasks such as POS tagging, NE tagging, syntactic parsing, template extraction and relation extraction into a generative model. Their results essentially depend on the entire full parse trees.

Prior feature-based methods (Kambhatla, 2004; Zhou et al., 2005) for this task employ a large amount of diverse linguistic features, such as lexical, syntactic and semantic features. These methods are quite effective for relation extraction and show the best-reported performance on the ACE 2003 corpus. However, the problems are that these diverse features have to be manually calibrated and the structured information in a parse tree is not well preserved in their parse tree feature, which is represented as a *path* of non-terminals connecting two entities in a parse tree.

Prior kernel-based methods for this task focus on using individual tree kernels to exploit tree structure features. Zelenko et al. (2003) develop a relation 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) generalize it to over dependency trees. Their tree kernels require the matchable nodes to be at the same depth 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 show high precision but very low recall on the ACE 2003 corpus. Bunescu and Mooney (2005) propose a dependency 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 that of the previous one (Culotta and Sorensen, 2004), their reported performance is much lower than that of the feature-based method (Zhou et al., 2005) by 16 in F-measure on the ACE 2003 corpus.

Zhao and Grishman (2005) define a feature-based composite kernel and achieve very good performance on the ACE 2004 corpus. All the features used in it have to be explicitly enumerated. Therefore, we wonder whether the performance improvement of this method is mainly due to the explicit combination of diverse flat features instead of the kernel method itself.

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 in relation extraction and significantly outperform the kernel methods on the ACE corpus. One may ask: Have kernel methods been well studied for relation extraction? Can kernel methods capture more useful features for relation extraction? Are tree structure features useful for this task?

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

3 The Composite Kernel for Relation Extraction

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

3.1 The Composite Kernel

(1) The 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 re-defines these features and introduces a new feature "LDC mention type". Our statistics on the ACE data reveals that the entity features impose a big constrain 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 $K_E(\cdot, \cdot)$ 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} feature of an entity, 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) The Convolution Parse Tree Kernel: A convolution kernel aims to capture structural information in terms of substructures. Here we use the same convolution kernel as the parse tree kernel (Collins and Duffy, 2001) and the semantic kernel (Moschitti, 2004). First, we can represent a parse tree T by a vector of integer counts of each sub-tree type (regardless of its ancestors):

$$\phi(T) = (\# subtree_i(T), ..., \# subtree_i(T), ..., \# subtree_i(T))$$

where # subtree $_i(T)$ is the number of appearances of the i^{th} type of sub-tree (subtree $_i$) in T. Since the number of different sub-trees is exponential in its size, it is computational infeasible to directly use the feature vector $\phi(T)$. To solve the computational issue, Collins and Duffy (2001) propose the following parse tree kernel to calculate the dot product between the above high dimensional vectors implicitly.

$$K(T_{1},T_{2}) = \langle \phi(T_{1}),\phi(T_{2}) \rangle = \sum_{i} \phi(T_{1})[i] \cdot \phi(T_{2})[i]$$

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

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

$$= \sum_{n_{i} \in N_{i}} \sum_{n_{i} \in N_{2}} \Delta(n_{1},n_{2})$$
(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 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 number of the children of n_1 ,

ch(n,j) is the j^{th} child of node n and λ (0< λ <1) is the decay factor in order to make the kernel value less variable with respect to the *subtree* sizes. 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 of two relation instances. The time complexity of computing this kernel is $O(|N_1| \cdot |N_2|)$.

(3) The Composite Kernels: Let K(x, y) be a proper kernel, and $\hat{K}(x, y)$ be the normalized² K(x, y) and $K^p(x, y)$ be the polynomial extension of K(x, y) with degree d=2, i.e. $K^p(x, y) = (K(x, y) + 1)^2$. Using the above notation, the composite kernels are defined as follows:

1) Liner combination:

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

where α is set to 0.4 to yield the best performance on the developing set.

2) Polynomial extension:

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

where α is set to 0.23 to generate the best performance on the developing set.

The polynomial extension aims to explore the bi-gram of entity features, esp. the combined features from the first and second entities respectively. In addition, since the size of an input tree is not constant and the value range of each individual kernel is also different, we normalize the kernel values to range from 0 to 1 before combining. This can avoid one kernel value is overwhelmed by another one in the combinations.

The two individual kernels are proper kernels since they calculate dot-product implicitly. It can be proven that the polynomial extension and their combinations are also proper kernels.

3.2 Relation Instance Spaces

A relation instance is represented by a parse tree. The performance of the tree kernel depends on which portion of a parse tree is involved in the kernel calculation. We study five different relation instance spaces as follows (see Figure 1 in the next page):

² A kernel K(x, y) can be normalized by dividing it by $\sqrt{K(x,x) \cdot K(y,y)}$.

(1) Minimum Complete Tree (MCT):

It is the complete sub-tree rooted by the node of the nearest common ancestor of the two entities under consideration.

(2) Path-enclosed Tree (PT):

It is the least common sub-tree comprising 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 used as the path tree feature in the feature-based methods).

(3) Context-Sensitive Path Tree (CPT):

It is the **PT** extending with the 1st left word of entity 1 and the 1st right word of entity 2.

(4) Flattened Path-enclosed Tree (FPT):

We define the following criterion to flatten the **PT** in order to generate the **FPT**: if the in and out arcs of a non-terminal node (except POS node) are both single, the node is to be removed.

(5) Flattened CPT (FCPT):

We use the above-defined criterion to flatten the **CPT** tree to generate the **Flattened CPT**.

Figure 1 illustrates the different representations of an example relation instance. Tree T_1 is the **MCT** of the relation instance example, where the sub-structure circled by a dashed line is the **PT**. For clarity, we re-draw the **PT** as in T_2 . The only difference between the MCT and the PT lies in that the MCT does not allow the partial production rules. For instance, the most-left phrase [NP [CD JJ E1-PER]] in T_1 is broken apart in T_2 . By comparing the performance of T_1 and T_2 , we can test whether the sub-structures with partial production rules as in T_2 will decrease performance, and we can also study whether the rich left and right context structures are useful for relation extraction. T_3 is the **CPT**, where the two structures circled by dashed lines are the so-called context structures. We want to study whether the limited context information in the **CPT** can help boost performance. The two circled nodes in T_2 are removed to form the **FPT** in T_4 . We want to study whether the eliminated small structures are noisy features for relation extraction.

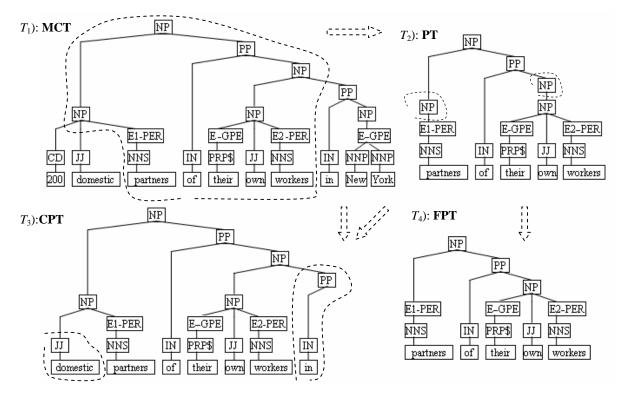


Figure 1. Different representations of a relation instance in the example sentence "...provide benefits to 200 domestic partners of their own workers in New York", where the phrase type "E1-PER" denotes that the current node is the 1st entity with type "PERSON", and likewise for the others. The relation instance is excerpted from the ACE 2003 corpus, where an ACE relation "SOCIAL.Other-Personal" exists between the 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

The aim of our experiments is to verify the effectiveness of our composite kernel and study the optimal relation instance representation for relation extraction.

4.1 Experimental Setting

Data: We use the ACE 2003 and 2004 corpora from LDC as our experimental corpus. In the ACE 2003 data, the training set consists of 674 documents and 9683 relation instances. The testing set consists of 97 documents and 1386 relation instances. The ACE 2003 defined 5 types of entities, 5 major relation types and 24 subtypes. The ACE 2004 data contains 451 documents and 5702 relation instances. They redefined **7** types of entities, 7 major relation types and 23 subtypes³. The two corpora are parsed using Charniak's parser (Charniak, 2001). We iterate over all pair of entity mentions occurring in the same sentence to generate potential relation instances. We find the negative samples are 8 times more than the positive samples. Thus data imbalance and sparseness are potential problems.

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. In our implementation, we use the binary SVMLight developed by Joachims (1998) and Tree Kernel Tools developed by Moschitti (2004). Precision (**P**), Recall (**R**) and F-measure (**F**) are adopted as the main performance measure.

4.2 Experimental Results

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

(1) Tree Kernel only over Different Relation Instance Spaces: In order to study the impact of the syntactic structure information in a parse tree on relation extraction, we remove the entity information from parse trees by replacing the entity-related phrase types ("E1-PER" and so on in Figure 1) with "NP". Then we carry out a couple

of preliminary experiments on the ACE 2003 data using the parse tree kernel only.

Instance Spaces	P (%)	R (%) F
Minimum Complete Tree	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. Tree kernel over different instance spaces on the ACE 2003 five major types using parse tree information only

Table 1 reports the performance of different tree kernel setups. It shows that:

- Overall the tree kernel over different instance spaces is effective for relation extraction in terms of using parse tree information only. It suggests that the syntactic structure information has good predicative power for relation detection and classification, and the structured information can be well captured by the tree kernel.
- Using the **PT**s achieves the best performance. This means the portion of a parse tree enclosed by the shortest path between entities can model relations better than other sub-trees.
- Using the MCTs gets the worst performance. There may be two reasons for this: 1) the MCTs introduce too much left and right context information, which may be noisy features, as shown in Figure 1; 2) the big sub-structures suffer from the over-fitting issue (high precision and very low recall as shown in Table 1.). It also suggests that constraint of complete (not partial) production rules in the MCTs do harm performance.
- The **CPT**s show a little lower performance than the **PT**s. In some cases (e.g. in sentence "the merge of company A and company B....", "merge" is a context word), the context information is helpful. But the effective scope of context is hard to determine.
- The two flattened trees perform worse than the original trees. It suggests that the structures removed by the flattened trees are still useful.

The same experiment on the ACE 2004 data shows the similar observations: using the PTs achieves the best performance (72.5/56.7/63.6 in P/R/F). We also carry out the similar experiments of using tree kernel on ACE 2003/2004 data with incorporating the entity type and order information into tree nodes as shown in Figure 1. The observations are the same, but the perform-

³ Please refer to http://www.ldc.upenn.edu/Projects/ACE/ for details about the ACE 2003 and 2004 data. Zhao and Grishman (2005) use a 5-fold cross-validation on a subset of the 2004 data (newswire and broadcast news domains, containing 384 documents and 4400 relations). For comparison, we use the same subset data and setting in this paper for the ACE 2004 data.

ance is improved significantly (the **PT**s achieve the best performance: 76.1/62.6/68.7 (2003 data) and 74.1/62.4/67.7 (2004 data) in **P/R/F**).

PTs (with Tree Structure 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 (linear combination)	73.5	67.0	70.1
	(76.3)	(63.0)	(69.1)
Composite kernel 2 (polynomial extension)	76.1	68.4	72.1
	(77.3)	(65.6)	(70.9)

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

- **(2) Composite Kernels**: Table 2 reports the performance of different kernel setups. It clearly shows that:
- The composite kernels achieve significant performance improvement over the two individual kernels. This indicates that the composite kernels can well integrate both the flat and the structured features (which are complementary to each other): 1) the argument information of a relation instance captured by the entity kernel; 2) the syntactic connection between the two arguments introduced by the tree kernel.
- The composite kernel via the polynomial extension outperforms the one via the linear combination by ~2 in F-measure. It suggests that the bi-gram of entity features is 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 that on the 2003 data although the ACE 2003 provides more training data. This is due to two reasons: 1) The ACE 2004 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 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. It suggests that the ACE 2004 data could more accurately represent the nature of the task of relation extraction.

The final best performance of our composite kernel for relation extraction is 77.3/65.6/70.9 and 76.1/68.4/72.1 in P/R/F over the ACE 2003/2004 major types.

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

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

Methods (on 2004 data)	P (%)	R (%)	F
Ours: composite kernel 2	76.1	68.4	72.1
(polynomial extension)	(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 7 major types (the numbers outside parentheses) and 23 subtypes (the numbers in parentheses)

(3) **Performance Comparison**: Table 3 and Table 4 compare our method with the previous work. They show that our method outperforms the previous best-reported method on the ACE 2003/ 2004 data. Table 3 (the 3rd, 6th and 7th rows) also shows that using the same kind of information (tree structures with entity information at node) the parse tree kernel significantly outperforms the previous two dependency kernels by 16 (68.7-52.5) in F-measure on the ACE 2003 major types. This may be due to two reasons: 1) the dependency tree and the shortest path lack of the internal phrase structure information, so their kernels carry out only hard-matching directly

over the nodes with word tokens; 2) the parse tree kernel is not restricted by the two constraints of the two dependency kernels as discussed in Section 2.

The above experiments verify the effectiveness of our composite kernels for relation extraction. They also suggest that the parse tree kernel can fully explore the syntactic features, and which are very useful 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 the 2003 and 2004 data

(4) Error Analysis: Table 5 reports the error distribution of the 2nd 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 is mainly due to the sparseness of some relation types and possibly due to the imbalance of the positive/negative instances. 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

(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 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 can effectively capture

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, our method is not limited by the two constraints (as discussed in Section 2) in Zelenko et al. (2003) and 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 sub-tree. However, the *path* does not maintain the tree structure information. In addition, their kernel requires the two paths to have the same length. This is a very strong constraint.

The above discussion is expected to answer the question why our method performs better on the ACE corpus and significantly outperforms the previous two dependency kernels.

5.2 Other Issues

(1) The Speed Issue: One practical problem in applying kernel methods to NLP is their speed. Kernel classifiers are relatively slow compared to feature classifiers. Therefore, there is a pressing need to develop fast algorithms for kernel calculation (Vishwanathan and Smola, 2002). But fortunately, in our experiments the speed of our kernel is not an issue from the practical application perspective. Using the PC with Intel P4 3.0G CPU and 2G RAM, it 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. Not surprising, this is because two reasons: 1) we use the small portion (the **PT**) of a full parse tree to represent a relation instance. It can significantly improve the speed. Our testing shows that the MCT consumes more than twice times than the PT in training and it takes almost two days to do training if using the full parse tree on the 2003 data. 2) The parse tree kernel requires exact match between two sub-trees, but it is normally not occurring very frequently. Collins and Duffy (2001) report that in practice, running time is more close to linear $(|N_1|+|N_2|)$, rather than $O(|N_1| \cdot |N_2|)$, please see subsection 3.1).

(2) Further Improvement: One of the potential problems of the parse tree kernel is that it carries out exact matches between sub-trees. But exact matches are normally not very frequent and may fail 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 the over-fitting and less accurate similarity estimation issues. In the future, we will design a more flexible kernel to handle such issues.

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

In this paper, we have designed the 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 syntactic structure features, and therefore outperforms the previously best-reported method on the ACE corpus. To our knowledge, this is the first research to have demonstrated that: 1) the syntactic features embedded in a parse tree are particularly effective for relation extraction; 2) due to the extensive exploration of the implicit syntactic features, the parse tree kernel significantly outperforms the previous two dependency kernels. In addition, we find that the relation instance representation (selecting effective portions of parse trees) is important for relation extraction.

The most immediate extension of our work is to improve the accuracy of relation detection. We can capture more features by including more individual kernels, such as the WordNet-based semantic kernel (Baslli et al., 2005) and other feature-based kernels. We can also benefit from the 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 to handle the sparse phrases and the near-synonymic grammar tags by the allowing grammar-driven partial rule matching and other approximate matching mechanisms.

References

ACE. 2002-2005. The Automatic Content Extraction Projects. http://www.ldc.upenn.edu/Projects/ACE/

- Baslli 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. *The Nist MUC website*: http://www.itl.nist.gov/iaui/894.02/related_projects/muc/
- Schölkopf B. and Smola A. J. 2001. *Learning with Kernels: Support Vector Machines, 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
- Vishwanathan S.V.N. and Smola A.J. 2002. Fast kernels for String and Tree Matching. NIPS-2002
- 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