Exploring Syntactic Features for Relation Extraction using a Convolution Tree Kernel

Min ZHANG **Jie ZHANG** **Jian SU**

Institute for Infocomm Research

21 Heng Mui Keng Terrace, Singapore 119613

{mzhang, zhangjie, sujian}@i2r.a-star.edu.sg

Abstract

This paper proposes to use a convolution kernel over parse trees to model syntactic structure information for relation extraction. Our study reveals that the syntactic structure features embedded in a parse tree are very effective for relation extraction and these features can be well captured by the convolution tree kernel. Evaluation on the ACE 2003 corpus shows that the convolution kernel over parse trees can achieve comparable performance with the previous best-reported feature-based methods on the 24 ACE relation subtypes. It also shows that our method significantly outperforms the previous two dependency tree kernels on the 5 ACE relation major types.

1 Introduction

Relation extraction is a subtask of information extraction that finds various predefined semantic relations, such as location, affiliation, rival, etc., between pairs of entities in text. For example, the sentence "George Bush is the president of the United States." conveys the semantic relation "President" between the entities "George Bush" (PER) and "the United States" (GPE: a Geo-Political Entity --- an entity with land and a government (ACE, 2004)).

Prior feature-based methods for this task (Kambhatla 2004; Zhou et al., 2005) employed a large amount of diverse linguistic features, varying from lexical knowledge, entity mention information to syntactic parse trees, dependency trees and semantic features. Since a parse tree contains rich syntactic structure information, in principle, the features extracted from a parse tree should contribute much more to performance improvement for relation extraction. However it is reported (Zhou et al., 2005; Kambhatla, 2004) that hierarchical structured syntactic features contributes less to performance improvement. This may be mainly due to the fact that the syntactic structure information in a parse tree is hard to explicitly describe by a vector of linear features. As an alternative, kernel methods (Collins and Duffy, 2001) provide an elegant solution to implicitly explore tree structure features by directly computing the similarity between two trees. But to our surprise, the sole two-reported dependency tree kernels for relation extraction on the ACE corpus (Bunescu and Mooney, 2005; Culotta and Sorensen, 2004) showed much lower performance than the feature-based methods. One may ask: are the syntactic tree features very useful for relation extraction? Can tree kernel methods effectively capture the syntactic tree features and other various features that have been proven useful in the feature-based methods?

In this paper, we demonstrate the effectiveness of the syntactic tree features for relation extraction and study how to capture such features via a convolution tree kernel. We also study how to select the optimal feature space (e.g. the set of sub-trees to represent relation instances) to optimize the system performance. The experimental results show that the convolution tree kernel plus entity features achieves slightly better performance than the previous best-reported feature-based methods. It also shows that our method significantly outperforms the two dependency tree kernels (Bunescu and Mooney, 2005; Culotta and Sorensen, 2004) on the 5 ACE relation types.

The rest of the paper is organized as follows. In Section 2, we review the previous work. Section 3 discusses our tree kernel based learning algorithm.

Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL, pages 288–295,
New York, June 2006.

Section 4 shows the experimental results and compares our work with the related work. We conclude our work in Section 5.

2 Related Work

The task of relation extraction was introduced as a part of the Template Element task in MUC6 and formulated as the Template Relation task in MUC7 (MUC, 1987-1998).

Miller et al. (2000) address the task of relation extraction from the statistical parsing viewpoint. They integrate various tasks such as POS tagging, NE tagging, template extraction and relation extraction into a generative model. Their results essentially depend on the entire full parse tree.

Kambhatla (2004) employs Maximum Entropy models to combine diverse lexical, syntactic and semantic features derived from the text for relation extraction. Zhou et al. (2005) explore various features in relation extraction using SVM. They conduct exhaustive experiments to investigate the incorporation and the individual contribution of diverse features. They report that chunking information contributes to most of the performance improvement from the syntactic aspect.

The features used in Kambhatla (2004) and Zhou et al. (2005) have to be selected and carefully calibrated manually. Kambhatla (2004) use the path of non-terminals connecting two mentions in a parse tree as the parse tree features. Besides, Zhou et al. (2005) introduce additional chunking features to enhance the parse tree features. However, the hierarchical structured information in the parse trees is not well preserved in their parse tree-related features.

As an alternative to the feature-based methods, kernel methods (Haussler, 1999) have been proposed to implicitly explore features in a high dimensional space by employing a kernel function to calculate the similarity between two objects directly. In particular, the kernel methods could be very effective at reducing the burden of feature engineering for structured objects in NLP research (Culotta and Sorensen, 2004). This is because a kernel can measure the similarity between two discrete structured objects directly using the original representation of the objects instead of explicitly enumerating their features.

Zelenko et al. (2003) develop a tree kernel for relation extraction. Their tree kernel is recursively defined in a top-down manner, matching nodes from roots to leaf nodes. For each pair of matching nodes, a subsequence kernel on their child nodes is invoked, which matches either contiguous or sparse subsequences of node. Culotta and Sorensen (2004) generalize this kernel to estimate similarity between dependency trees. One may note that their tree kernel requires the matchable nodes must be at the same depth counting from the root node. This is a strong constraint on the matching of syntax so it is not surprising that the model has good precision but very low recall on the ACE corpus (Zhao and Grishman, 2005). In addition, according to the top-down node matching mechanism of the kernel, once a node is not matchable with any node in the same layer in another tree, all the sub-trees below this node are discarded even if some of them are matchable to their counterparts in another tree.

Bunescu and Mooney (2005) propose a shortest path dependency kernel for relation extraction. They argue that the information to model a relationship between entities is typically captured by the shortest path between the two entities in the dependency graph. Their kernel is very straightforward. It just sums up the number of common word classes at each position in the two paths. We notice that one issue of this kernel is that they limit the two paths must have the same length, otherwise the kernel similarity score is zero. Therefore, although this kernel shows non-trivial performance improvement than that of 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.

Zhao and Grishman (2005) define a feature-based composite kernel to integrate diverse features. Their kernel displays very good performance on the 2004 version of ACE corpus. Since this is a feature-based kernel, all the features used in the kernel have to be explicitly enumerated. Similar with the feature-based method, they also represent the tree feature as a link path between two entities. Therefore, we wonder whether their performance improvement is mainly due to the explicitly incorporation of diverse linguistic features instead of the kernel method itself.

The above discussion suggests that the syntactic features in a parse tree may not be fully utilized in the previous work, whether feature-based or kernel-based. We believe that the syntactic tree features could play a more important role than that

reported in the previous work. Since convolution kernels aim to capture structural information in terms of sub-structures, which providing a viable alternative to flat features, in this paper, we propose to use a convolution tree kernel to explore syntactic features for relation extraction. To our knowledge, convolution kernels have not been explored for relation extraction[1].

3 Tree Kernels for Relation Extraction

In this section, we discuss the convolution tree kernel associated with different relation feature spaces. In Subsection 3.1, we define seven different relation feature spaces over parse trees. In Subsection 3.2, we introduce a convolution tree kernel for relation extraction. Finally we compare our method with the previous work in Subsection 3.3.

3.1 Relation Feature Spaces

In order to study which relation feature spaces (i.e., which portion of parse trees) are optimal for relation extraction, we define seven different relation feature spaces as follows (as shown in Figure 1):

(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 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 typically used as the path tree features in the feature-based methods).

(3) Chunking Tree (CT):

It is the base phrase list extracted from the **PT**. We prune out all the internal structures of the **PT** and only keep the root node and the base phrase list for generating the chunking tree.

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

It is the **PT** extending with the 1st left sibling of the node of entity 1 and the 1st right sibling of the node of entity 2. If the sibling is unavailable, then we move to the parent of current node and repeat the same process until the sibling is available or the root is reached.

(5) Context-Sensitive Chunking Tree (CCT):

It is the **CT** extending with the 1st left sibling of the node of entity 1 and the 1st right sibling of the node of entity 2. If the sibling is unavailable, the same process as generating the **CPT** is applied. Then we do a further pruning process to guarantee that the context structures of the **CCT** is still a list of base phrases.

(6) Flattened PT (FPT):

We define two criteria to flatten the **PT** in order to generate the **Flattened Parse tree**: if the in and out arcs of a non-terminal node (except POS node) are both single, the node is to be removed; if a node has the same phrase type with its father node, the node is also to be removed.

(7) Flattened CPT (FCPT):

We use the above two criteria to flatten the **CPT** tree to generate the **Flattened CPT**.

Figure 1 in the next page illustrates the different sub-tree structures for a relation instance in sentence "*Akyetsu testified he was powerless to stop the merger of an estimated 2000 ethnic **Tutsi's** in the **district** of Tawba.*". The relation instance is an example excerpted from the ACE corpus, where an ACE-defined relation "AT.LOCATED" exists between the entities "***Tutsi's***" (PER) and "***district***" (GPE).

We use Charniak's parser (Charniak, 2001) to parse the example sentence. Due to space limitation, we do not show the whole parse tree of the entire sentence here. Tree T_1 in Figure 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 two-layer sub-tree [NP [DT … E1-O-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. T_3 is the **CT**. By comparing the performance of T_2 and T_3, we want to study whether the chunking information or the parse tree is more effective

[1] Convolution kernels were proposed as a concept of kernels for a discrete structure by Haussler (1999) in machine learning study. This framework defines a kernel between input objects by applying convolution "sub-kernels" that are the kernels for the decompositions (parts) of the objects. Convolution kernels are abstract concepts, and the instances of them are determined by the definition of "sub-kernels". The *Tree Kernel* (Collins and Duffy, 2001), *String Subsequence Kernel* (SSK) (Lodhi et al., 2002) and *Graph Kernel* (HDAG Kernel) (Suzuki et al., 2003) are examples of convolution kernels instances in the NLP field.

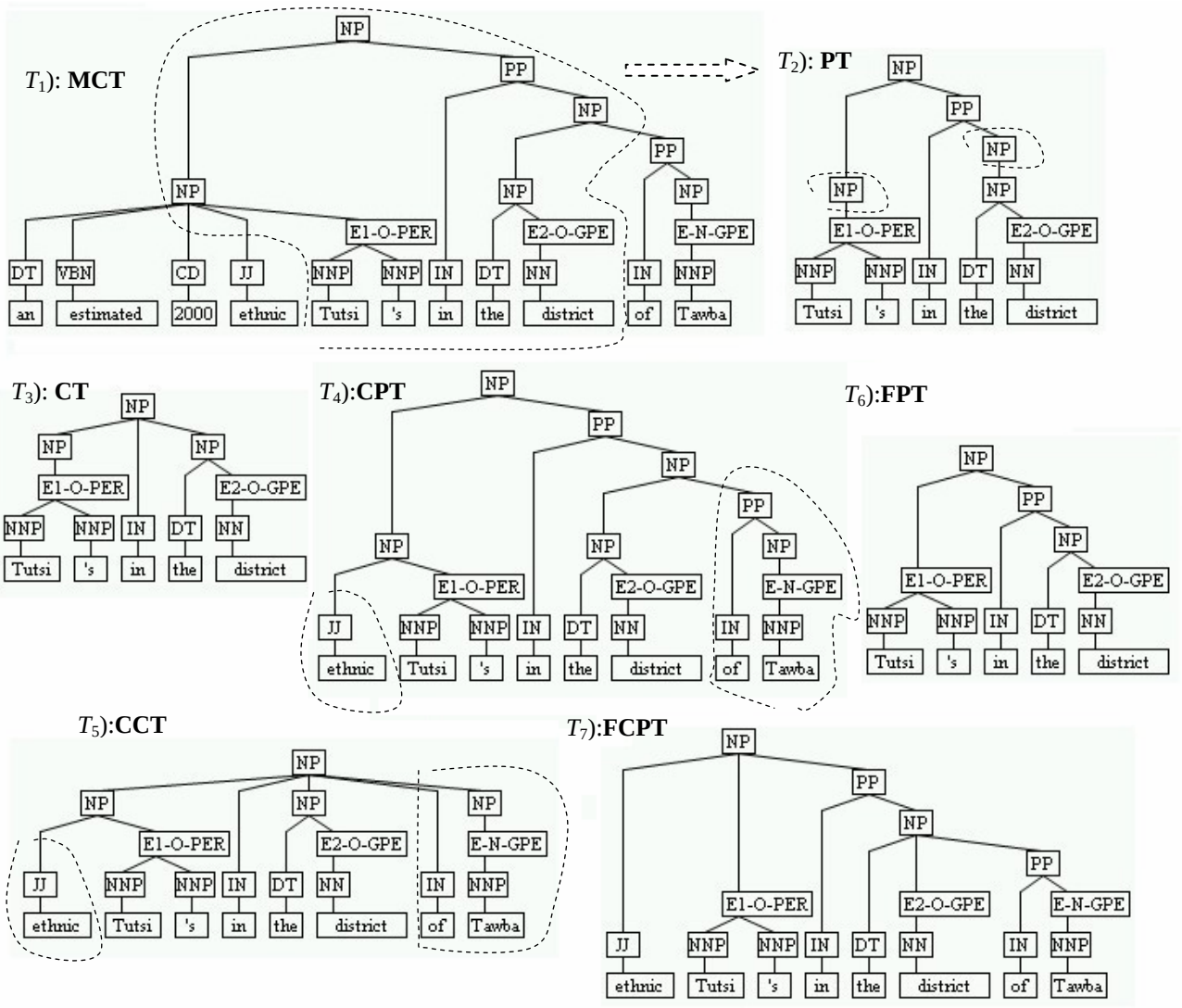

Figure 1. Relation Feature Spaces of the Example Sentence "…… *to stop the merger of an estimated 2000 ethnic* ***Tutsi's*** *in the* ***district*** *of Tawba.*", where the phrase type "E1-O-PER" denotes that the current phrase is the 1st entity, its entity type is "PERSON" and its mention level is "NOMIAL", and likewise for the other two phrase types "E2-O-GPE" and "E-N-GPE".

for relation extraction. T_4 is the **CPT**, where the two structures circled by dashed lines are the so-called context structures. T_5 is the **CCT**, where the additional context structures are also circled by dashed lines. We want to study if the limited context information in the **CPT** and the **CCT** can help boost performance. Moreover, we illustrate the other two flattened trees in T_6 and T_7. The two circled nodes in T_2 are removed in the flattened trees. We want to study if the eliminated small structures are noisy features for relation extraction.

3.2 The Convolution Tree Kernel

Given the relation instances defined in the previous section, we use the same convolution tree kernel as the parse tree kernel (Collins and Duffy, 2001) and the semantic kernel (Moschitti, 2004). Generally, we can represent a parse tree T by a vector of integer counts of each sub-tree type (regardless of its ancestors):

$\phi(T) = (\#\ of\ sub\text{-}trees\ of\ type\ 1, \ldots, \#\ of\ sub\text{-}trees\ of\ type\ i, \ldots, \#\ of\ sub\text{-}trees\ of\ type\ n)$

This results in a very high dimensionality since the number of different sub-trees is exponential in its size. Thus it is computational infeasible to directly use the feature vector $\phi(T)$. To solve the compu-

tational issue, we introduce the tree kernel function which is able to calculate the dot product between the above high dimensional vectors efficiently. The kernel function is defined as follows:

$$K(T_1,T_2) = <\phi(T_1),\phi(T_2)> = \sum_i \phi(T_1)[i],\phi(T_2)[i]$$
$$= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \sum_i I_i(n_1) * I_i(n_2)$$

where N_1 and N_2 are the sets of all nodes in trees T_1 and T_2, respectively, and I_i(n) is the indicator function that is 1 iff a sub-tree of type *i* occurs with root at node n and zero otherwise. Collins and Duffy (2002) show that $K(T_1,T_2)$ is an instance of convolution kernels over tree structures, and which can be computed in $O(|N_1| \times |N_2|)$ by the following recursive definitions (Let $\Delta(n_1,n_2) = \sum_i I_i(n_1) * I_i(n_2)$):

(1) if n_1 and n_2 do not have the same syntactic tag or their children are different then $\Delta(n_1,n_2) = 0$;

(2) else if their children are leaves (POS tags), then $\Delta(n_1,n_2) = 1 \times \lambda$;

(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 ($0 < \lambda < 1$) is the decay factor in order to make the kernel value less variable with respect to the tree sizes.

3.3 Comparison with Previous Work

It would be interesting to review the differences between our method and the feature-based methods. The basic difference between them lies in the relation instance representation and the similarity calculation mechanism. A relation instance in our method is represented as a parse tree while it is represented as a vector of features in the feature-based methods. Our method estimates the similarity between two relation instances by only counting the number of sub-structures that are in common while the feature methods calculate the dot-product between the feature vectors directly. The main difference between them is the different feature spaces. By the kernel method, we implicitly represent a parse tree by a vector of integer counts of each sub-structure type. That is to say, we consider the entire sub-structure types and their occurring frequencies. In this way, on the one hand, the parse tree-related features in the *flat feature set*[2] are embedded in the feature space of our method: "*Base Phrase Chunking*" and "*Parse Tree*" features explicitly appear as substructures of a parse tree. A few of entity-related features in the *flat feature set* are also captured by our feature space: "*entity type*" and "*mention level*" explicitly appear as phrase types in a parse tree. On the other hand, the other features in the *flat feature set*, such as "*word features*", "*bigram word features*", "*overlap*" and "*dependency tree*" are not contained in our feature space. From the syntactic viewpoint, the tree representation in our feature space is more robust than "*Parse Tree Path*" feature in the *flat feature set* since the *path* feature is very sensitive to the small changes of parse trees (Moschitti, 2004) and it also does not maintain the hierarchical information of a parse tree. Due to the extensive exploration of syntactic features by kernel, our method is expected to show better performance than the previous feature-based methods.

It is also worth comparing our method with the previous relation kernels. Since our method only counts the occurrence of each sub-tree without considering its ancestors, our method is not limited by the constraints in Culotta and Sorensen (2004) and that in Bunescu and Mooney (2005) as discussed in Section 2. Compared with Zhao and Grishman's kernel, our method directly uses the original representation of a parse tree while they flatten a parse tree into a *link* and a *path*. Given the above improvements, our method is expected to outperform the previous relation kernels.

4 Experiments

The aim of our experiment is to verify the effectiveness of using richer syntactic structures and the convolution tree kernel for relation extraction.

4.1 Experimental Setting

Corpus: we use the official ACE corpus for 2003 evaluation from LDC as our test corpus. The ACE corpus is gathered from various newspaper, newswire and broadcasts. The same as previous work

[2] For the convenience of discussion, without losing generality, we call the features used in Zhou et al. (2005) and Kambhatla (2004) *flat feature set*.

(Zhou et al., 2005), our experiments are carried out on explicit relations due to the poor inter-annotator agreement in annotation of implicit relations and their limited numbers. The training set consists of 674 annotated text documents and 9683 relation instances. The test set consists of 97 documents and 1386 relation instances. The 2003 evaluation defined 5 types of entities: Persons, Organizations, Locations, Facilities and GPE. Each mention of an entity is associated with a mention type: proper name, nominal or pronoun. They further defined **5** major relation types and **24** subtypes: AT (Base-In, Located…), NEAR (Relative-Location), PART (Part-of, Subsidiary …), ROLE (Member, Owner …) and SOCIAL (Associate, Parent…). As previous work, we explicitly model the argument order of the two mentions involved. We thus model relation extraction as a multi-class classification problem with 10 classes on the major types (2 for each relation major type and a "NONE" class for non-relation (except 1 symmetric type)) and 43 classes on the subtypes (2 for each relation subtype and a "NONE" class for non-relation (except 6 symmetric subtypes)). 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).

Classifier: we select SVM as the classifier used in this paper since SVM can naturally work with kernel methods and it also represents the state-of-the-art machine learning algorithm. We adopt the *one vs. others* strategy and select the one with largest margin as the final answer. The training parameters are chosen using cross-validation (C=2.4 (SVM); λ =0.4(tree kernel)). In our implementation, we use the binary SVMLight developed by Joachims (1998) and Tree Kernel Toolkits developed by Moschitti (2004).

Kernel Normalization: since the size of a parse tree is not constant, we normalize $K(T_1, T_2)$ by dividing it by $\sqrt{K(T_1, T_1) \bullet K(T_2, T_2)}$.

Evaluation Method: we parse the sentence using Charniak parser and iterate over all pair of mentions occurring in the same sentence to generate potential instances. We find the negative samples are 10 times more than the positive samples. Thus data imbalance and sparseness are potential problems. Recall (**R**), Precision (**P**) and F-measure (**F**) are adopted as the performance measure.

4.2 Experimental Results

In order to study the impact of the sole syntactic structure information embedded in parse trees on relation extraction, we remove the entity information from parse trees by replacing the entity-related phrase type ("E1-O-PER", etc., in Figure 1) with "NP". Then we carry out a couple of preliminary experiments on the test set using parse trees regardless of entity information.

Feature Spaces	P	R	F
Minimum Complete Tree	77.45	38.39	51.34
Path-enclosed Tree (PT)	**72.77**	**53.80**	**61.87**
Chunking Tree (**CT**)	75.18	44.75	56.11
Context-Sensitive PT(**CPT**)	77.87	42.80	55.23
Context-Sensitive **CT**	78.33	40.84	53.69
Flattened **PT**	76.86	45.69	57.31
Flattened **CPT**	80.60	41.20	54.53

Table 1. Performance of seven relation feature spaces over the 5 ACE major types using parse tree information only

Table 1 reports the performance of our defined seven relation feature spaces over the 5 ACE major types using parse tree information regardless of any entity information. This preliminary experiments show that:

- Overall the tree kernel over different relation feature spaces is effective for relation extraction since we use the parse tree information only. We will report the detailed performance comparison results between our method and previous work later in this section.
- 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 **MCT**s get the worst performance. This is because the **MCT**s introduce too much left and right context information, which may be noisy features, as shown in Figure 1. It suggests that only allowing complete (not partial) production rules in the **MCT**s does harm performance.
- The performance of using **CT**s drops by 5 in F-measure compared with that of using the **PT**s. This suggests that the middle and high-level structures beyond chunking is also very useful for relation extraction.

- The context-sensitive trees show lower performance than the corresponding original **PT**s and **CT**s. In some cases (e.g. in sentence "the merge of company A and company B…."; "merge" is the context word), the context information is helpful. However the effective scope of context is hard to determine.
- The two flattened trees perform worse than the original trees, but better than the corresponding context-sensitive trees. This suggests that the removed structures by the flattened trees contribute non-trivial performance improvement.

In the above experiments, the path-enclosed tree displays the best performance among the seven feature spaces when using the parse tree structural information only. In the following incremental experiments, we incorporate more features into the path-enclosed parse trees and it shows significant performance improvement.

Path-enclosed Tree (**PT**)	**P**	**R**	**F**
Parse tree structure information only	72.77	53.80	61.87
+Entity information	76.14	62.85	68.86
+Semantic features	76.32	62.99	69.02

Table 2. Performance of Path-enclosed Trees with different setups over the 5 ACE major types

Table 2 reports the performance over the 5 ACE major types using Path-enclosed trees enhanced with more features in nodes. The 1st row is the baseline performance using structural information only. We then integrate entity information, including Entity type and Mention level features, into the corresponding nodes as shown in Figure 1. The 2nd row in Table 2 reports the performance of this setup. Besides the entity information, we further incorporate the semantic features used in Zhou et al. (2005) into the corresponding leaf nodes. The 3rd row in Table 2 reports the performance of this setup. Please note that in the 2nd and 3rd setups, we still use the same tree kernel function with slight modification on the rule (2) in calculating $\Delta(n_1, n_2)$ (see subsection 3.2) to make it consider more features associated with each individual node: $\Delta(n_1, n_2) = \textit{feature weight} \times \lambda$. From Table 2, we can see that the basic feature of entity information is quite useful, which largely boosts performance by 7 in F-measure. The final performance of our tree kernel method for relation extraction is 76.32/62.99/69.02 in precision/recall/F-measure over the 5 ACE major types.

Methods	**P**	**R**	**F**
Ours: convolution kernel over parse trees	76.32 (64.6)	62.99 (50.76)	69.02 (56.83)
Kambhatla (2004): feature-based ME	- (63.5)	- (45.2)	- (52.8)
Zhou et al. (2005): feature-based SVM	77.2 (63.1)	60.7 (49.5)	68.0 (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, the numbers in parentheses report the performance over the 24 ACE subtypes while the numbers outside parentheses is for the 5 ACE major types

Table 3 compares the performance of different methods on the ACE corpus[3]. It shows that our method achieves the best-reported performance on both the 24 ACE subtypes and the 5 ACE major types. It also shows that our tree kernel method significantly outperform the previous two dependency kernel algorithms by 16 in F-measure on the 5 ACE relation types[4]. This may be due to two reasons: one reason is that the dependency tree lacks the hierarchical syntactic information, and another reason is due to the two constraints of the two dependency kernels as discussed in Section 2 and Subsection 3.3. The performance improvement by our method suggests that the convolution tree kernel can explore the syntactic features (e.g. parse tree structures and entity information) very effectively and the syntactic features are also particu-

[3] Zhao and Grishman (2005) also evaluated their algorithm on the ACE corpus and got good performance. But their experimental data is for 2004 evaluation, which defined 7 entity types with 44 entity subtypes, and 7 relation major types with 27 subtypes, so we are not ready to compare with each other.

[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 an ACE corpus including about 800 documents, and they did not specify the corpus version. Since the testing corpora are in different sizes and versions, strictly speaking, it is not ready to compare these methods exactly and fairly. Thus Table 3 is only for reference purpose. We just hope that we can get a few clues from this table.

larly effective for the task of relation extraction. In addition, we observe from Table 1 that the feature space selection (the effective portion of a parse tree) is also critical to relation extraction.

Error Type	# of error instance
False Negative	414
False Positive	173
Cross Type	97

Table 4. Error Distribution

Finally, Table 4 reports the error distribution in the case of the 3rd experiment in Table 2. It shows that 85.9% (587/684) of the errors result from relation detection and only 14.1% (97/684) of the errors result from relation characterization. This is mainly due to the imbalance of the positive/negative instances and the sparseness of some relation types on the ACE corpus.

5 Conclusion and Future Work

In this paper, we explore the syntactic features using convolution tree kernels for relation extraction. We conclude that: 1) the relations between entities can be well represented by parse trees with carefully calibrating effective portions of parse trees; 2) the syntactic features embedded in a parse tree are particularly effective for relation extraction; 3) the convolution tree kernel can effectively capture the syntactic features for relation extraction.

The most immediate extension of our work is to improve the accuracy of relation detection. We may adopt a two-step method (Culotta and Sorensen, 2004) to separately model the relation detection and characterization issues. We may integrate more features (such as head words or WordNet semantics) into nodes of parse trees. We can also benefit from the learning algorithm to study how to solve the data imbalance and sparseness issues from the learning algorithm viewpoint. In the future, we would like to test our algorithm on the other version of the ACE corpus and to develop fast algorithm (Vishwanathan and Smola, 2002) to speed up the training and testing process of convolution kernels.

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. 2004. *The Automatic Content Extraction (ACE) Projects*. http://www.ldc.upenn.edu/Projects/ACE/

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 Nanda. 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 Alessandro. 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/

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 Shubin and Grishman Ralph. 2005. *Extracting Relations with Integrated Information Using Kernel Methods*. ACL-2005

Zhou Guodong, Su Jian, Zhang Jie and Zhang Min. 2005. *Exploring Various Knowledge in Relation Extraction*. ACL-2005