Syntactic Tree-based Relation Extraction Using a Generalization of Collins and Duffy Convolution Tree Kernel

Mahdy Khayyamian
Sharif University of Technology
khayyamian@ce.sharif.edu

Seyed Abolghasem Mirroshandel
Sharif University of Technology
mirroshandel@ce.sharif.edu

Hassan Abolhassani
Sharif University of Technology
abolhassani@sharif.edu

Abstract

Relation extraction is a challenging task in natural language processing. Syntactic features are recently shown to be quite effective for relation extraction. In this paper, we generalize the state of the art syntactic convolution tree kernel introduced by Collins and Duffy. The proposed generalized kernel is more flexible and customizable, and can be conveniently utilized for systematic generation of more effective application specific syntactic sub-kernels. Using the generalized kernel, we will also propose a number of novel syntactic sub-kernels for relation extraction. These kernels show a remarkable performance improvement over the original Collins and Duffy kernel in the extraction of ACE-2005 relation types.

1 Introduction

One of the contemporary demanding NLP tasks is information extraction, which is the procedure of extracting structured information such as entities, relations, and events from free text documents. As an information extraction sub-task, semantic relation extraction is the procedure of finding predefined semantic relations between textual entity mentions. For instance, assuming a semantic relation with type *Physical* and subtype *Located* between an entity of type *Person* and another entity of type *Location*, the sentence "*Police arrested Mark at the airport last week.*" conveys two mentions of this relation between "Mark" and "airport" and also between "police" and "airport" that can be shown in the following format.

Phys.Located(Mark, airport)
Phys.Located(police, airport)

Relation extraction is a key step towards question answering systems by which vital structured data is acquired from underlying free text resources. Detection of protein interactions in biomedical corpora (Li et al., 2008) is another valuable application of relation extraction.

Relation extraction can be approached by a standard classification learning method. We particularly use SVM (Boser et al., 1992; Cortes and Vapnik, 1995) and kernel functions as our classification method. A kernel is a function that calculates the inner product of two transformed vectors of a high dimensional feature space using the original feature vectors as shown in eq. 1.

$$K(X_i, X_j) = \phi(X_i).\phi(X_j) \quad (1)$$

Kernel functions can implicitly capture a large amount of features efficiently; thus, they have been widely used in various NLP tasks.

Various types of features have been exploited so far for relation extraction. In (Bunescu and Mooney, 2005b) *sequence of words* features are utilized using a sub-sequence kernel. In (Bunescu and Mooney, 2005a) *dependency graph* features are exploited, and in (Zhang et al., 2006a) *syntactic features* are employed for relation extraction. Although in order to achieve the best performance, it is necessary to use a proper combination of these features (Zhou et al., 2005), in this paper, we will concentrate on how to better capture the *syntactic features* for relation extraction.

Proceedings of the NAACL HLT Student Research Workshop and Doctoral Consortium, pages 66–71,
Boulder, Colorado, June 2009.

In CD'01 (Collins and Duffy, 2001) a convolution syntactic tree kernel is proposed that generally measures the syntactic similarity between parse trees. In this paper, a generalized version of CD'01 convolution tree kernel is proposed by associating generic weights to the nodes and sub-trees of the parse tree. These weights can be used to incorporate domain knowledge into the kernel and make it more flexible and customizable. The generalized kernel can be conveniently used to generate a variety of syntactic sub-kernels (including the original CD'01 kernel), by adopting appropriate weighting mechanisms.

As a result, in this paper, novel syntactic sub-kernels are generated from the generalized kernel for the task of relation extraction. Evaluations demonstrate that these kernels outperform the original CD'01 kernel in the extraction of ACE-2005 main relation types

The remainder of this paper is structured as follows. In section 2, the most related works are briefly reviewed. In section 3, CD'01 tree kernel is described. The proposed generalized convolution tree kernel is explained in section 4 and its produced sub-kernels for relation extraction are illustrated in section 5. The experimental results are discussed in section 6. Our work is concluded in section 7 and some possible future works are presented in section 8.

2 Related Work

In (Collins and Duffy, 2001), a convolution parse tree kernel has been introduced. This kernel is generally designed to measure syntactic similarity between parse trees and is especially exploited for parsing English sentences in their paper. Since then, the kernel has been widely used in different applications such as semantic role labeling (Moschitti, 2006b) and relation extraction (Zhang et al., 2006a; Zhang et al., 2006b; Zhou et al., 2007; Li et al. 2008).

For the first time, in (Zhang et al., 2006a), this convolution tree kernel was used for relation extraction. Since the whole syntactic parse tree of the sentence that holds the relation arguments contains a plenty of misleading features, several parse tree portions are studied to find the most feature-rich portion of the syntactic tree for relation extraction, and Path-Enclosed Tree (PT) is finally found to be the best performing tree portion. PT is a portion of parse tree that is enclosed by the shortest path between the two relation arguments. Moreover, this tree kernel is combined with an entity kernel to form a reportedly high quality composite kernel in (Zhang et al., 2006b).

3 CD'01 Convolution Tree Kernel

In (Collins and Duffy, 2001), a convolution tree kernel has been introduced that measures the syntactic similarity between parse trees. This kernel computes the inner products of the following feature vector.

$$H(T) = (\lambda^{\frac{size_1}{2}} \# subTree_1(T),..., \lambda^{\frac{size_i}{2}} \# subTree_i(T),..., \lambda^{\frac{size_n}{2}} \# subTree_n(T)) \quad (2)$$
$$0 < \lambda \le 1$$

Each feature of this vector is the occurrence count of a sub-tree type in the parse tree decayed exponentially by the parameter λ. Without this decaying mechanism used to retain the kernel values within a fairly small range, the value of the kernel for identical trees becomes far higher than its value for different trees. Term $size_i$ is defined to be the number of rules or internal nodes of the i^{th} sub-tree type. Samples of such sub-trees are shown in Fig. 1 for a simple parse tree. Since the number of sub-trees of a tree is exponential in its size (Collins and Duffy, 2001), direct inner product calculation is computationally infeasible. Consequently, Collins and Duffy (2001) proposed an ingenious kernel function that implicitly calculates the inner product in $O(N_1 \times N_2)$ time on the trees of size N_1 and N_2.

4 A Generalized Convolution Tree Kernel

In order to describe the kernel, a feature vector over the syntactic parse tree is firstly defined in eq. (3), in which the i^{th} feature equals the weighted sum of the number of instances of sub-tree type i^{th} in the tree.

Function $I_{subtree_i}(n)$ is an indicator function that returns 1 if the $subtree_i$ occurs with its root at node n and 0 otherwise. As described in eq. (4),

function *tw(T)* (which stands for "tree weight") assigns a weight to a tree T which is equal to the product of the weights of all its nodes.

$$H(T) = (\sum_{n\in T}[I_{subtree_1}(n)\times tw(subtree_1(n))],..., \sum_{n\in T}[I_{subtree_i}(n)\times tw(subtree_i(n))],..., \sum_{n\in T}[I_{subtree_m}(n)\times tw(subtree_m(n))]) \quad (3)$$

$$tw(T) = \prod_{n\in InternalNodes(T)} inw(n) \times \prod_{n\in ExternalNodes(T)} enw(n) \quad (4)$$

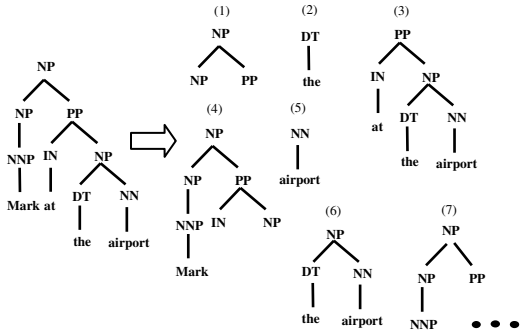

Figure 1. Samples of sub-trees used in convolution tree kernel calculation.

Since each node of the whole syntactic tree can either happen as an internal node or as an external node of a supposed sub-tree (presuming its existence in the sub-tree), two types of weights are respectively associated to each node by the functions $inw(n)$ and $enw(n)$ (which respectively stand for "internal node weight" and "external node weight"). For instance, in Fig. 1, the node with label PP is an external node for sub-trees (1) and (7) while it is an internal node of sub-trees (3) and (4).

$$\begin{aligned} K(T_1,T_2) &= <H(T_1),H(T_2)> \\ &= \sum_i [\ [\sum_{n_1\in T_1} I_{subtree_i}(n_1)\times tw(subTree_i(n_1))] \\ &\quad \times [\sum_{n_2\in T_2} I_{subtree_i}(n_2)\times tw(subTree_i(n_2))]] \\ &= \sum_{n_1\in T_1}\sum_{n_2\in T_2}[\sum_i I_{subtree_i}(n_1)\times I_{subtree_i}(n_2)\times \\ &\quad tw(subTree_i(n_1))\times tw(subTree_i(n_2))] \\ &= \sum_{n_1\in T_1}\sum_{n_2\in T_2} C_{gc}(n_1,n_2) \end{aligned} \quad (5)$$

As shown in eq. (5), A similar procedure to (Collins and Duffy, 2001) can be employed to develop a kernel function for the calculation of dot products on *H(T)* vectors. According to eq. (5) the calculation of the kernel finally leads to the sum of a $C_{gc}(n_1,n_2)$ function over all tree node pairs of T_1 and T_2. Function $C_{gc}(n_1,n_2)$ is the weighted sum of the common sub-trees rooted at n_1 and n_2, and can be recursively computed in a similar way to function $C(n_1,n_2)$ of (Collins and Duffy, 2001) as follows.

(1) if the production rules of nodes n_1 and n_2 are different then $C_{gc}(n_1,n_2) = 0$

(2) else if n_1 and n_2 are the same pre-terminals (the same part of speeches) then

$$C_{gc}(n_1,n_2) = inw(n_1)\times enw(child(n_1))\times inw(n_2)\times enw(child(n_2))$$

(3) else if both n_1 and n_2 have the same production rules then

$$C_{gc}(n_1,n_2) = inw(n_1)\times inw(n_2)\times \prod_i [enw(child_i(n_1))\times enw(child_i(n_2)) + C_{gc}(child_i(n_1),child_i(n_2))]$$

In the first case, when the two nodes represent different production rules they can't accordingly have any sub-trees in common. In the second case, there is exactly one common sub-tree of size two. It should be noted that all the leaf nodes of the tree (or words of the sentence) are considered identical in the calculation of the tree kernel. The value of the function in this case is the weight of this common sub-tree. In the third case, when the nodes generally represent the same production rules the weighted sum of the common sub-trees are calculated recursively. The equation holds because the existence of common sub-trees rooted at n_1 and n_2 implies the existence of common sub-trees rooted at their corresponding children, which can be combined multiplicatively to form their parents' common sub-trees.

Due to the equivalent procedure of kernel calculation, this generalized version of the tree kernel preserves the nice $O(N_1\times N_2)$ time complexity property of the original kernel. It is worthy of note that in (Moschitti, 2006b) a sorting based method is proposed for the fast implementation of such tree kernels that reduces the average running time to $O(N_1+N_2)$.

The generalized kernel can be converted to CD'01 kernel by defining $inw(n) = \sqrt{\lambda}$ and $enw(n) = 1$. Likewise, other definitions can be utilized to produce other useful sub-kernels.

5 Kernels for Relation Extraction

In this section, three sub-kernels of the generalized convolution tree kernel will be proposed for relation extraction. Using the embedded weights of the generalized kernel, these sub-kernels differentiate among sub-trees based on their expected relevance to semantic relations. More specifically, the sub-trees are weighted according to how their nodes interact to the arguments of the relation.

5.1 Argument Ancestor Path Kernel (AAP)

Definition of weighting functions is shown in eq. (6) and (7). Parameter $0 < \alpha \leq 1$ is a decaying parameter similar to λ.

$$inw(n) = \begin{cases} \alpha & \text{if } n \text{ is on the argument ancestor path} \\ & \text{or a direct child of a node on it} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

$$enw(n) = \begin{cases} 1 & \text{if } n \text{ is on the argument ancestor path} \\ & \text{or a direct child of a node on it} \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

This weighting method is equivalent to applying CD'01 tree kernel (by setting $\lambda = \alpha^2$) on a portion of the parse tree that exclusively includes the arguments ancestor nodes and their direct children.

5.2 Argument Ancestor Path Distance Kernel (AAPD)

$$inw(n) = \alpha^{\frac{Min(AAPDist(n,\arg_1), AAPDist(n,\arg_2))}{MAX\_DIST}} \quad (8)$$

$$enw(n) = \alpha^{\frac{Min(AAPDist(n,\arg_1), AAPDist(n,\arg_2))}{MAX\_DIST}} \quad (9)$$

Definition of weighting functions is shown in eq. (8) and (9). Both functions have identical definitions for this kernel.

Function *AAPDist(n,arg)* calculates the distance of the node *n* from the argument *arg* on the parse tree as illustrated by Fig. 2. MAX_DIST is used for normalization, and is the maximum of *AAPDist(n,arg)* in the whole tree. In this way, the closer a tree node is to one of the arguments ancestor path, the less it is decayed by this weighting method.

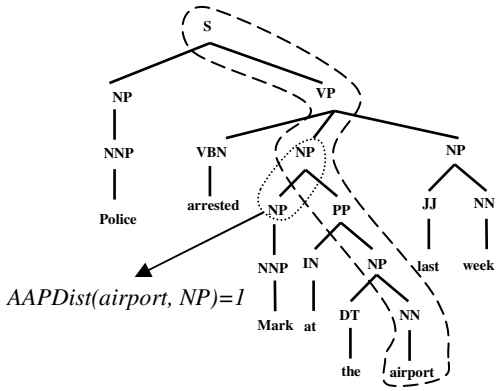

Figure 2. The syntactic parse tree of the sentence "Police arrested Mark at the airport last week" that conveys a Phys.Located(Mark, airport) relation. The ancestor path of the argument "airport" (dashed curve) and the distance of the node NP of "Mark" from it (dotted curve) is shown.

5.3 Threshold Sensitive Argument Ancestor Path Distance Kernel (TSAAPD)

This kernel is intuitively similar to the previous kernel but uses a rough threshold based decaying technique instead of a smooth one. The definition of weighting functions is shown in eq. (10) and (11). Both functions are again identical in this case.

$$inw(n) = \begin{cases} 1 & AAPDist(n) \leq Threshold \\ \alpha & AAPDist(n) \geq Threshold \end{cases} \quad (10)$$

$$enw(n) = \begin{cases} 1 & AAPDist(n) \leq Threshold \\ \alpha & AAPDist(n) \geq Threshold \end{cases} \quad (11)$$

6 Experiments

6.1 Experiments Setting

The proposed kernels are evaluated on ACE-2005 multilingual corpus (Walker et al., 2006). In order to avoid parsing problems, the more formal parts of the corpus in "news wire" and "broadcast news" sections are used for evaluation as in (Zhang et al., 2006b).

Relation / Kernel	PER-SOC	ART	GEN-AFF	ORG-AFF	PART-WHOLE	PHYS
CD'01	0.62	**0.51**	0.09	0.43	**0.30**	0.32
AAP	0.58	0.49	0.10	0.43	0.28	**0.36**
AAPD	0.70	0.50	**0.12**	0.43	0.29	0.29
TSAAPD-0	0.63	0.48	0.11	0.43	**0.30**	0.33
TSAAPD-1	**0.73**	0.47	0.11	**0.45**	0.28	0.33

Table 1: The F_1-Measure value is shown for every kernel on each ACE-2005 main relation type. For every relation type the best result is shown in bold font.

We have used LIBSVM (Chang and Lin 2001) java source for the SVM classification and Stanford NLP package[1] for tokenization, sentence segmentation and parsing.

Following [Bunescu and Mooney, 2007], every pair of entities within a sentence is regarded as a negative relation instance unless it is annotated as a positive relation in the corpus. The total number of negative training instances, constructed in this way, is about 20 times more than the number of annotated positive instances. Thus, we also imposed the restriction of maximum argument distance of 10 words. This constraint eliminates half of the negative constructed instances while slightly decreases positive instances. Nevertheless, since the resulted training set is still unbalanced, we used LIBSVM weighting mechanism. Precisely, if there are P positive and N negative instances in the training set, a weight value of N/P is used for positive instances while the default weight value of 1 is used for negative ones.

A binary SVM is trained for every relation type separately, and type compatible annotated and constructed relation instances are used to train it. For each relation type, only type compatible relation instances are exploited for training. For example to learn an ORG-AFF relation (which applies to (PER, ORG) or (ORG, ORG) argument types) it is meaningless to use a relation instance between two entities of type PERSON. Moreover, the total number of training instances used for training every relation type is restricted to 5000 instances to shorten the duration of the evaluation process. The reported results are achieved using a 5-fold cross validation method.

The kernels AAP, AAPD and TSAAPD-0 (TSAAPD with threshold = 0) and TSAAPD-1 (TSAAPD with threshold = 1) are compared with CD'01 convolution tree kernel. All the kernels except for AAP are computed on the PT portion described in section 2. AAP is computed over the MCT tree portion which is also proposed by (Zhang et al., 2006a) and is the sub-tree rooted at the first common ancestor of relation arguments.

For the proposed kernels α is set to 0.44 which is tuned on a development set that contained 5000 instances of type PHYS. The λ parameter of CD'01 kernel is set to 0.4 according to (Zhang et al., 2006a). The C parameter of SVM classification is set to 2.4 for all the kernels after tuning it individually for each kernel on the mentioned development set.

6.2 Experiments Results

The results of the experiments are shown in Table 1. The proposed kernels outperform the original CD'01 kernel in four of the six relation types. The performance of TSAAPD-1 is especially remarkable because it is the best kernel in ORG-AFF and PER-SOC relations. It particularly performs very well in the extraction of PER-SOC relation with an F_1-measure of 0.73. It should be noted that the general low performance of all the kernels on the GEN-AFF type is because of its extremely small number of annotated instances in the training set (40 in 5000). The AAPD kernel has the best performance with a remarkable improvement over the Collins kernel in GEN-AFF relation type.

The results clearly demonstrate that the nodes closer to the ancestor path of relation arguments contain the most useful syntactic features for relation extraction

7 Conclusion

In this paper, we proposed a generalized convolution tree kernel that can generate various syntactic sub-kernels including the CD'01 kernel.

1 http://nlp.stanford.edu/software/index.shtml

The kernel is generalized by assigning weights to the sub-trees. The weight of a sub-tree is the product of the weights assigned to its nodes by two types of weighting functions. In this way, impacts of the tree nodes on the kernel value can be discriminated purposely based on the application. Context information can also be injected to the kernel via context sensitive weighting mechanisms.

Using the generalized kernel, various sub-kernels can be produced by different definitions of the two weighting functions. We consequently used the generalized kernel for systematic generation of useful kernels in relation extraction. In these kernels, the closer a node is to the relation arguments ancestor paths, the less it is decayed by the weighting functions. Evaluation on the ACE-2005 main relation types demonstrates the effectiveness of the proposed kernels. They show remarkable performance improvement over CD'01 kernel.

8 Future Work

Although the path-enclosed tree portion (PT) (Zhang et al., 2006a) seems to be an appropriate portion of the syntactic tree for relation extraction, it only takes into account the syntactic information between the relation arguments, and discards many useful features (before and after the arguments features). It seems that the generalized kernel can be used with larger tree portions that contain syntactic features before and after the arguments, because it can be more easily targeted to related features.

Currently, the proposed weighting mechanisms are solely based on the location of the tree nodes in the parse tree; however other useful information such as labels of nodes can also be used in weighting.

Another future work can be utilizing the generalized kernel for other applicable NLP tasks such as co-reference resolution.

Acknowledgement

This work is supported by Iran Telecommunication Research Centre under contract No. 500-7725.

References

Boser B. E., Guyon I., and Vapnik V. 1992. *A training algorithm for optimal margin classifiers*. In Proceedings of the Fifth Annual Workshop on Computational Learning Theory, pages 144-152. ACM Press.

Bunescu R. C. and Mooney R. J. 2005a. *A Shortest Path Dependency Kernel for Relation Extraction*. EMNLP-2005

Bunescu R. C. and Mooney R. J. 2005b. *Subsequence kernels for relation extraction*. NIPS-2005.

Bunescu R. C. and Mooney R. J. 2007. *Learning for Information Extraction: From Named Entity Recognition and Disambiguation to Relation Extraction, Ph.D. Thesis*. Department of Computer Sciences, University of Texas at Austin.

Chang, C.-C. and C.-J. Lin 2001. *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Collins M. and Duffy N. 2001. *Convolution Kernels for Natural Language*. NIPS-2001

Cortes C. and Vapnik V. 1995. *Support-vector network*. Machine Learning. 20, 273-297.

Li J., Zhang Z., Li X. and Chen H. 2008. *Kernel-based learning for biomedical relation extraction*. J. Am. Soc. Inf. Sci. Technol. 59, 5, 756–769.

Moschitti A. 2006a. *Making tree kernels practical for natural language learning*. EACL-2006.

Moschitti A. 2006b. *Syntactic kernels for natural language learning: the semantic role labeling case*. HLT-NAACL-2006 (short paper)

Walker, C., Strassel, S., Medero J. and Maeda, K. 2006. *ACE 2005 Multilingual Training Corpus.* Linguistic Data Consortium, Philadelphia.

Zhang M., Zhang J. and SU j. 2006a. *Exploring syntactic features for relation extraction using a convolution tree kernel. HLT-NAACL-2006*.

Zhang M., Zhang J., Su J. and Zhou G.D. 2006b. *A Composite Kernel to Extract Relations between Entities with both Flat and Structured* COLINGACL-2006: 825-832.

Zhou G.D., Su J, Zhang J. and Zhang M. 2005. *Exploring Various Knowledge in Relation Extraction*. ACL-2005

Zhou G.D., Zhang M., Ji D.H. and Zhu Q.M. 2007. *Tree Kernel-based Relation Extraction with Context-Sensitive Structured Parse Tree Information.* EMNLP-CoNLL-2007