A Re-examination of Dependency Path Kernels for Relation Extraction

Mengqiu Wang
Computer Science Department
Stanford University
mengqiu@cs.stanford.edu

Abstract

Extracting semantic relations between entities from natural language text is an important step towards automatic knowledge extraction from large text collections and the Web. The state-of-the-art approach to relation extraction employs Support Vector Machines (SVM) and kernel methods for classification. Despite the diversity of kernels and the near exhaustive trial-and-error on kernel combination, there lacks a clear understanding of how these kernels relate to each other and why some are superior than others. In this paper, we provide an analysis of the relative strength and weakness of several kernels through systematic experimentation. We show that relation extraction can benefit from increasing the feature space through convolution kernel and introducing bias towards more syntactically meaningful feature space. Based on our analysis, we propose a new convolution dependency path kernel that combines the above two benefits. Our experimental results on the standard ACE 2003 datasets demonstrate that our new kernel gives consistent and significantly better performance than baseline methods, obtaining very competitive results to the state-of-the-art performance.

1 Introduction

There exists a large body of knowledge embedded in unstructured natural language text on the Web. The sheer volume and heterogeneity of such knowledge renders traditional rule-based and manually-crafted knowledge extraction systems unsuitable. Thus it calls for methods that automatically extract knowledge from natural language text. An important step towards automatic knowledge discovery is to extract semantic relations between entities.

Two types of collections are commonly studied for relation extraction. The first type is annotated newswire text made available by programs such as Message Understanding Conferences (MUC) and Automatic Content Extraction (ACE). The types of entities that are of interest to these programs include *person*, *organization*, *facilities*, *location* and *GPE (Geo-political entities)*. Given entities in a document, the relation extraction task is to identify explicit semantic relationship such as *Located-In* and *Citizen-Of* between pairs of entities. For example, in the sentence “The funeral was scheduled for Thursday in Paris at the Saint-Germain-des-Pres Church”, the organization *Saint-Germain-des-Pres Church* is “*Located-In*” GPE *Paris*. The second type of collection that has been widely studied is biomedical literature (Bunescu and Mooney, 2005b; Giuliano et al., 2006; McDonald et al., 2005b), promoted by evaluation programs such as BioCreAtIvE and JNLPBA 2004. In this particular domain, studies often focus on specific entities such as genes and proteins. And the kinds of relations to extract are usually gene-to-protein interactions.

The predominant approach to relation extraction treats the task as a multi-class classification problem, in which different relation types form different output classes. Early work employed a diverse range of features in a linear classifier (commonly referred to as “feature-based” approaches), including lexical features, syntactic parse features, dependency features and semantic features (Jiang and Zhai, 2007; Kambhatla, 2004; Zhou et al., 2005). These approaches were hindered by drawbacks such as limited feature space and excessive feature engineering. Kernel methods (Cortes and Vapnik, 1995; Cristianini and Shawe-Taylor, 2000) on the other hand can explore a much larger feature space very efficiently. Recent studies on relation extraction have shown that by combining kernels with Support-vector Machines (SVM), one can obtain results superior to feature-based methods (Bunescu and Mooney, 2005b; Bunescu and Mooney, 2005a; Culotta and Sorensen, 2004; Cumby and Roth, 2003; Zelenko et al., 2003; Zhang et al., 2006a; Zhang et al., 2006b; Zhao and Grishman, 2005).

Despite the large number of recently proposed kernels and their reported success, there lacks a clear understanding of their relative strength and weakness. In this study, we provide a systematic comparison and analysis of three such kernels — subsequence kernel (Bunescu and Mooney, 2005b), dependency tree kernel (Culotta and Sorensen, 2004) and dependency path kernel (Bunescu and Mooney, 2005a). We replicated these kernels and conducted experiments on the standard ACE 2003 newswire text evaluation set. We show that whereas some kernels are less effective than others, they exhibit properties that are complementary to each other. In particular, We found that relation extraction can benefit from increasing the feature space through convolution kernel and introducing bias towards more syntactically meaningful feature space.

Drawn from our analysis, we further propose a new convolution dependency path kernel which combines the benefits of the subsequence kernel and shortest path dependency kernel. Comparing to the previous kernels, our new kernel gives consistent and significantly better performance than all three previous kernels that we look at.

2 Related Work

Statistical methods for relation extraction can be roughly categorized into two categories: feature-based and kernel-based.

Feature-based methods (Jiang and Zhai, 2007; Kambhatla, 2004; Zhou et al., 2005) use pre-defined feature sets to extract features to train classification models. Zhou et al. (2005) manually crafted a wide range of features drawn from sources such as lexical, syntactic and semantic analyses. Combined with SVM, they reported the best results at the time on ACE corpus. Kambhatla (2004) took a similar approach but used multivariate logistic regression (Kambhatla, 2004). Jiang & Zhai (2007) gave a systematic examination of the efficacy of unigram, bigram and trigram features drawn from different representations — surface text, constituency parse tree and dependency parse tree.

One drawback of these feature-based methods is that the feature space that can be explored is often limited. On the other hand, kernel-based methods offer efficient solutions that allow us to explore a much larger (often exponential, or in some cases, infinite) feature space in polynomial time, without the need to explicitly represent the features.

Lodhi et al. (2002) described a convolution string kernel, which measures the similarity between two strings by recursively computing matching of all possible subsequences of the two strings. Bunescu & Mooney (2005b) generalized the string kernel to work with vectors of objects occurred in relation extraction. In a later work also done by Bunescu & Mooney (2005a), they proposed a kernel that computes similarities between nodes on the shortest dependency paths that connect the entities. Their kernel assigns no-match to paths that are of different length. And for paths that are of the same length, it simply computes the product of the similarity score of node pairs at each index. The dependency tree kernel proposed by Zelenko et al. (2003) was also inspired by the string kernel of Lodhi et al. (2002). Their kernel walks down the parse trees from the root and computes a similarity score for children nodes at each depth level using the same subsequence algorithm as the string kernel. Culotta & Sorensen (2004) worked on the same idea but applied it to dependency parse trees. Prior to these two tree kernels, Collins & Duffy (2001) proposed a convolution tree kernel for natural language tasks. Their kernel has since been applied to relation extraction by Zhang et al. (2006a). The tree kernel considers matching of all subtrees that share the same production rule at the root of the subtree. Zhang et al. (2006a) showed results that are significantly better than the previous two dependency tree kernels. They obtained further improvements in their later paper (2006b) by composing the tree kernel with a simple entity kernel and raising the composite kernel to polynomial degree 2. Another study on kernel composition is the work by Zhao & Grishman (2005).

It is worth noting that although there exist standard evaluation datasets such as ACE 2003 and 2004, many of the aforementioned work report results on non-standard datasets or splits, making it difficult to directly compare the performance. We feel that there is a sense of increasing confusion down this line of research. Although partly due to the lack of compatibility in evaluation results, we believe it is more due to the lack of understanding in the relative strength and weakness of these kernels. Therefore we focus on analyzing and understanding the pros and cons of different kernels, through systematic comparison and experimentation.

3 Kernel Methods for Relation Extraction

In this Section we first give a very brief introduction to kernel methods. We then present the algorithms behind three kernels that we are particularly interested in: subsequence kernel (Bunescu and Mooney, 2005b), dependency tree kernel (Culotta and Sorensen, 2004) and shortest path dependency kernel (Bunescu and Mooney, 2005a).

3.1 SVM and Kernels

Support-Vector Machines (Cortes and Vapnik, 1995; Cristianini and Shawe-Taylor, 2000) learn to find hyperplanes that separate the positive and negative data points so that the margin between the support-vector points and the hyperplane is maximized. The dual formulation of the optimization problem involves only computing the dot product of feature vectors. This is equivalent to mapping the data points into a high dimensional space. And the separating plane learnt in the high dimensional space can give non-linear decision boundaries. The dot product of data points can be computed using a kernel function $K(X, Y) = \langle \phi(X), \phi(Y) \rangle$ for any mapping function. A valid kernel function satisfies certain properties: it is symmetric and the *Gram* matrix G formed by $K(X, Y)$ is positive semi-definite.

3.2 Subsequence Kernel

The subsequence kernel introduced in (Bunescu and Mooney, 2005b) is a generalization of the string kernel first introduced by Lodhi et al. (2002). The feature space of the original string kernel Σ_{string_kernel} is defined as $\Sigma_{string_kernel} = \Sigma_{char}$, where Σ_{char} is simply a set of characters. Bunescu & Mooney (2005a) re-defined the feature space to be $\Sigma_x = \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_k$, where $\Sigma_1, \Sigma_2, \cdots, \Sigma_k$ can be some arbitray disjoint feature spaces, such as the set of words, part-of-speech (POS) tags, etc. We can measure the number of common features shared by two feature vectors $x, y \in \Sigma_x$ using function $c(x, y)$. Let s, t be two sequences over the feature set Σ_x, we use $|s|$ to denote the length of s. Thus s can be written out as $s_1 \cdots s_{|s|}$. We use $s[i : j]$ to denote a continuous subsequence $s_i \cdots s_j$ of s. Let $\mathbf{i} = (i_1, \cdots, i_{|\mathbf{i}|})$ be a sequence of $|\mathbf{i}|$ indices in s, we define the *length* of the index sequence $\mathbf{i}$ to be $l(\mathbf{i}) = i_{|\mathbf{i}|} - i_1 + 1$. Similarly we have index sequence $\mathbf{j}$ in t of length $l(\mathbf{j})$.

Let $\Sigma_\cup = \Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_k$ be the set of all possible features. A sequence $u \in \Sigma_\cup^*$ is a subsequence of feature vector sequence s if there exists a sequence of $|u|$ indices $\mathbf{i}$, such that $u_k \in s_{i_k}$, $\forall k \in \{1, \cdots, |u|\}$. Follow the notions in (Bunescu and Mooney, 2005b; Cumby and Roth, 2003), we use $u \prec s[\mathbf{i}]$ as a shorthand for the above component-wise '$\in$' relationship. Now we can define the kernel function $K_n(s, t)$ to be the total number of weighted common subsequence of length n between the two sequeneces s and t.

$$K_n(s,t) = \sum_{u \in \Sigma_\cup^n} \sum_{\mathbf{i}: u \prec s[\mathbf{i}]} \sum_{\mathbf{j}: u \prec t[\mathbf{j}]} \lambda^{l(\mathbf{i}) + l(\mathbf{j})} \quad (1)$$

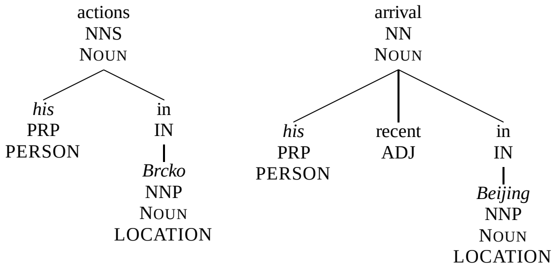
where λ is a decaying factor ≤ 1, penalizing long, sparse subsequence. We can re-write this kernel function as

$$K_n(s,t) = \sum_{\mathbf{i}: |\mathbf{i}| = n} \sum_{\mathbf{j}: |\mathbf{j}| = n} \prod_{k=1}^{n} c(s_{i_k}, t_{j_k}) \lambda^{l(\mathbf{i}) + l(\mathbf{j})} \quad (2)$$

(Bunescu and Mooney, 2005b) showed that using the recursive dynamic programming algorithm from (Cumby and Roth, 2003), the kernel $K_n(s, t)$ can be computed in $O(kn|s||t|)$ time.

3.3 From Subsequence to Tree Kernels

We will use an example to illustrate the relation between the dependency tree kernels proposed by (Culotta and Sorensen, 2004; Zelenko et al., 2003) and the subsequence kernel we introduced above. Consider two instances of the "Located-In" relations "*his* actions in *Brcko*" and "*his* recent arrival in *Beijing*". The dependency parse trees of these two sentences are shown below.



The entities in these two relations are the pronoun mentions of "*his*", and two locations "*Brcko*" and "*Beijing*", all shown in italic. The dependency tree kernel visits nodes in the two trees starting from the root. And at each depth level, it takes nodes that are at that level and form two sequences of nodes. For example, in the example instances, nodes at one level below the root forms vectors *s*=⟨{*his*, PRP, PERSON},{in, IN}⟩ and *t*=⟨{*his*,PRP,PERSON},{recent, ADJ},{in, IN}⟩. It then makes use of the subsequence kernel in the previous section to compute the total number of weighted subsequences between these two vectors. The kernel returns the sum of subsequence matching scores at each depth level as the final score.

3.4 Shortest Path Dependency Kernel

The shortest path dependency kernel proposed by Bunescu & Mooney (2005a) also works with dependency parse trees. Reuse our example in the previous section, the shortest dependency path between entity *his* and *Brcko* in the first sentence is *s*=⟨{*his*, PRP, PERSON}, {actions, NNS, NOUN}, {in, IN}, {*Brcko*, NNP, NOUN, LOCATION}⟩; and the path between *his* and *Beijing* in the second sentence is *t*=⟨{*his*, PRP, PERSON}, {arrival, NN, NOUN}, {in, IN}, {*Beijing*, NNP, NOUN, LOCATION}⟩. Since most dependency parser output connected *trees*, finding the shortest path between two nodes is trivial. Once the two paths are found, the kernel simply computes the product of the number of common features between a pair of nodes at each index along the path. If the two paths have different number of nodes, the kernel assigns 0 (no-match) to the pair. Formally, the kernel is defined as:

$$K(s,t) = \begin{cases} 0, & \text{if } |s| \neq |t| \\ \prod_{i=1}^{n} c(s_i,t_i), & \text{if } |s| = |t| \end{cases} \tag{3}$$

	5-fold CV on ACE 2003		
kernel method	Precision	Recall	F1
subsequence	0.703	**0.389**	0.546
dependency tree	0.681	0.290	0.485
shortest path	**0.747**	0.376	**0.562**

Table 1: Results of different kernels on ACE 2003 training set using 5-fold cross-validation.

4 Experiments and Analysis

We implemented the above three kernels and conducted a set of experiments to compare these kernels. By minimizing divergence in our experiment setup and implementation for these kernels, we hope to reveal intrinsic properties of different kernels.

4.1 Experiment setup

We conducted experiments using the ACE 2003 standard evaluation set. Training set of this collection contains 674 doc and 9683 relations. The test set contains 97 doc and 1386 relations. 5 entity types (Person, Organization, Location, Facilities and Geopolitical Entities) and 5 top-level relation types (At, Near, Part-of, Role and Social) are manually annotated in this collection. Since no development set is given, we report results in this section only on the training set, using 5-fold cross-validation, and defer the comparison of results on the test set till Section 6. Corpus preprocessing is done as the following: sentence segmentation was performed using the tool from CCG group at UIUC [1]; words are then tokenized and tagged with part-of-speech using MXPOST (Ratnaparkhi, 1996) and dependency parsing is performed using MSTParser (McDonald et al., 2005a). We used the SVM-light (Joachims, 2002) toolkit and augmented it with our custom kernels. SVM parameters are chosen using cross-validation (C=2.4), and the decaying factor in all kernels are uniformally set to be 0.75. We report precision (P), recall (R) and F-measure (F) on the training (5-fold cross-validation) and test set.

4.2 Comparison of Kernels

In table 1 we listed results of the above three kernels on the training set using 5-fold cross-validation. A

[1] http://l2r.cs.uiuc.edu/~cogcomp/atool.php?tkey=SS

first glimpse of the results tells us that the shortest path kernel performs the best in terms of F-measure, while the dependency tree kernel did the worst. The performance of subsequence kernel is not as good as the dependency path kernel, but the difference is small. In particular, the subsequence kernel gave the best recall, whereas the dependency path kernel gave the highest precision.

To understand why shortest path kernel performs better than the subsequence kernel, let us review the definition of these two kernels. The subsequence kernel considers all subsequences of feature vector sequences that are formed by all words occurred in-between two entities in a sentence; while the shortest path kernel only considers feature vector sequences formed by words that are connected through a dependency path. In general, the sequences considered in the dependency path kernel are more compact than the sequences used in the subsequence kernel. Actually, in most cases the dependency path sequence is indeed *one particular subsequence* of the entire subsequence used in subsequence kernel. Arguably, this particular subsequence is the one that captures the most important syntactic information. Although the feature spaces of the dependency path kernels are not subsets of the subsequence kernel, we can clearly see that we get higher precisions by introducing bias towards the syntactically more meaningful feature space.

However, the dependency path kernel is fairly rigid and imposes many hard constraints such as requiring the two paths to have exactly the same number of nodes. This restriction is counter-intuitive. To illustrate this, let us reconsider the example given in Section 3. In that example, it is obviously the case that the two instances of relations have very similar dependency path connecting the entities. However, the second path is one node longer than the first path, and therefore the dependency path kernel will declare no match for them. The subsequence kernel, on the other hand, considers subsequence matching and therefore inherently incorporates a notion of fuzzy matching. Furthermore, we have observed from the training data that many short word sequences carry strong relational information; hence only part of the entire dependency path is truly meaningful in most cases. It also helps to understand why subsequence kernel has better recall than dependency path kernel.

	ACE 2003 test set		
kernel method	Precision	Recall	F1
subsequence	0.673	0.499	0.586
dependency tree	0.621	0.362	0.492
shortest path	0.691	0.462	0.577
convolution dep. path	**0.725**	**0.541**	**0.633**
(Zhang et al., 2006b)	*0.773*	*0.656*	*0.709*

Table 2: Results on the ACE 2003 test set. We reference the best-reported score (in italic) on this test set, given by (Zhang et al., 2006b)

The disappointing performance of the dependency tree kernel can also be explained by our analysis. Although the dependency tree kernel performs subsequence matching for nodes at each depth level, it is unclear what the relative syntactic or semantic relation is among sibling nodes in the dependency tree. The sequence formed by sibling nodes is far less intuitive from a linguistic point of view than the sequence formed by nodes on a dependency path.

To summarize the above results, we found that dependency path kernel benefits from a reduction in feature space by using syntactic dependency information. But the subsequence kernel has an edge in recall by allowing fuzzy matching and expanding the feature space into convolution space. We will show in the following section that these two benefits are complementary and can be combined to give better performance.

5 Combining the Benefits – A New Kernel

It is a natural extension to combine the two benefits that we have identified in the previous section. The idea is simple: we want to allow subsequence matching in order to gain more flexibility and therefore higher recall, but constrain the sequence from which to deduce subsequences to be the dependency path sequence. We call the combined kernel a "convolution dependency path kernel".

6 Final Test Results

We obtained the final results on the test set of the ACE 2003 collection, using the same experimental setting as above. The results are listed in Table 2. From the table we can see that the performances of the previous three kernels hold up qualitatively on the test set as cross-validation on training set. There is one exception that the shortest path kernel's F-measure score is no longer better than the subsequence kernel on the test set, but the difference is small. And our new convolution dependency path kernel beats all above three kernels in precision, recall and F-measure, suggesting that our analysis is accurate and the benefits we outlined are truly complementary.

Comparing to the best reported results on the same test set from (Zhang et al., 2006b), our scores are not as high, but the results are quite competitive, given our minimum efforts on tuning kernel parameters and trying out kernel combinations.

7 Conclusion

We re-examined three existing kernel methods for relation extraction. We conducted experiments on the standard ACE 2003 evaluation set and showed that whereas some kernels are less effective than others, they exhibit properties that are complementary to each other. In particular, we found that relation extraction can benefit from increasing the feature space through convolution kernel and introducing bias towards more syntactically meaningful feature space. Drawn from our analysis, we proposed a new convolution dependency path kernel which combines the benefits of the subsequence kernel and shortest path dependency kernel. Comparing with previous kernels, our new kernel consistently and significantly outperforms all three previous kernels, suggesting that our analyses of the previously proposed kernels are correct.

References

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

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

M. Collins and N. Duffy. 2001. Convolution kernels for natural language. In *Proceedings of NIPS*.

C. Cortes and V. Vapnik. 1995. Support-vector networks. *Machine Learning*, 20(3):273–297.

N. Cristianini and J. Shawe-Taylor. 2000. *An Introduction to Support-vector Machines*. Cambridge University Press.

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

C. M. Cumby and D. Roth. 2003. On kernel methods for relation learning. In *Proceedings of ICML*.

C. Giuliano, A. Lavelli, and L. Romano. 2006. Exploiting shallow linguistic information for relation extraction from biomedical literature. In *Proceedings of EACL*.

J. Jiang and C. Zhai. 2007. A systematic exploration of the feature space for relation extraction. In *Proceedings of NAACL-HLT*.

T. Joachims. 2002. *Learning to Classify Text Using Support Vector Machines*. Ph.D. thesis, Universit¨at Dortmund.

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

H. Lodhi, C Saunders, J. Shawe-Taylor, N. Cristianini, and C Watkins. 2002. Text classification using string kernels. *JMLR*, 2:419–444.

R. McDonald, K. Crammer, and F. Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of ACL*.

R. McDonald, F. Pereira, S. Kulick, S. Winters, Y. Jin, and P. White. 2005b. Simple algorithms for complex relation extraction with applications to biomedical ie. In *Proceedings of ACL*.

A. Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of EMNLP*.

D. Zelenko, C. Aone, and A. Richardella. 2003. Kernel methods for relation extraction. *JMLR*, 3:1083–1106.

M. Zhang, J. Zhang, and J. Su. 2006a. Exploring syntactic features for relation extraction using a convolution tree kernel. In *Proceedings of NAACL-HLT*.

M. Zhang, J. Zhang, J. Su, and G. Zhou. 2006b. A composite kernel to extract relations between entities with both flat and structured features. In *Proceedings of ACL*.

S. Zhao and R. Grishman. 2005. Extraction relations with integrated information using kernel methods. In *Proceedings of ACL*.

G. Zhou, S. Jian, J. Zhang, and M. Zhang. 2005. Exploring various knowledge in relation extraction. In *Proceedings of ACL*.