Syntactic Kernels for Natural Language Learning: the Semantic Role Labeling Case

Alessandro Moschitti
Department of Computer Science
University of Rome ”Tor Vergata”
Rome, Italy
moschitti@info.uniroma2.it

Abstract

In this paper, we use tree kernels to exploit deep syntactic parsing information for natural language applications. We study the properties of different kernels and we provide algorithms for their computation in linear average time. The experiments with SVMs on the task of predicate argument classification provide empirical data that validates our methods.

1 Introduction

Recently, several tree kernels have been applied to natural language learning, e.g. (Collins and Duffy, 2002; Zelenko et al., 2003; Cumby and Roth, 2003; Culotta and Sorensen, 2004; Moschitti, 2004). Despite their promising results, three general objections against kernel methods are raised: (1) only a subset of the dual space features are relevant, thus, it may be possible to design features in the primal space that produce the same accuracy with a faster computation time; (2) in some cases the high number of features (substructures) of the dual space can produce overfitting with a consequent accuracy decrease (Cumby and Roth, 2003); and (3) the computation time of kernel functions may be too high and prevent their application in real scenarios.

In this paper, we study the impact of the subtree (ST) (Vishwanathan and Smola, 2002), subset tree (SST) (Collins and Duffy, 2002) and partial tree (PT) kernels on Semantic Role Labeling (SRL). The PT kernel is a new function that we have designed to generate larger substructure spaces. Moreover, to solve the computation problems, we propose algorithms which evaluate the above kernels in linear average running time.

We experimented such kernels with Support Vector Machines (SVMs) on the classification of semantic roles of PropBank (Kingsbury and Palmer, 2002) and FrameNet (Fillmore, 1982) data sets. The results show that: (1) the kernel approach provides the same accuracy of the manually designed features. (2) The overfitting problem does not occur although the richer space of PTs does not provide better accuracy than the one based on SST. (3) The average running time of our tree kernel computation is linear.

In the remainder of this paper, Section 2 introduces the different tree kernel spaces. Section 3 describes the kernel functions and our fast algorithms for their evaluation. Section 4 shows the comparative performance in terms of execution time and accuracy.

2 Tree kernel Spaces

We consider three different tree kernel spaces: the subtrees (STs), the subset trees (SSTs) and the novel partial trees (PTs).

An ST of a tree is rooted in any node and includes all its descendants. For example, Figure 1 shows the parse tree of the sentence `"Mary brought a cat"` together with its 6 STs. An SST is a more general structure since its leaves can be associated with non-terminal symbols. The SSTs satisfy the constraint that grammatical rules cannot be broken. For example, Figure 2 shows 10 SSTs out of 17 of the subtree of Figure 1 rooted in `VP`. If we relax the non-breaking rule constraint we obtain a more general form of substructures, i.e. the PTs. For example,

Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL, pages 97–100,
New York, June 2006. ©2006 Association for Computational Linguistics

Figure 3 shows 10 out of the total 30 PTs, derived from the same tree as before.

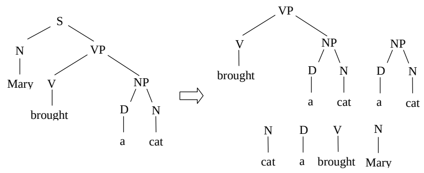

Figure 1: A syntactic parse tree with its subtrees (STs).

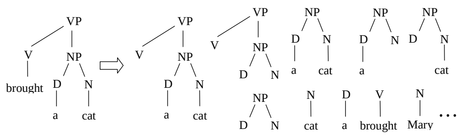

Figure 2: A tree with some of its subset trees (SSTs).

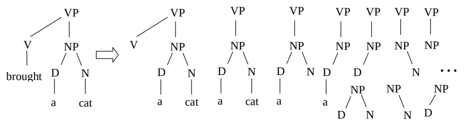

Figure 3: A tree with some of its partial trees (PTs).

3 Fast Tree Kernel Functions

The main idea of tree kernels is to compute the number of common substructures between two trees T_1 and T_2 without explicitly considering the whole fragment space. We designed a general function to compute the ST, SST and PT kernels. Our fast algorithm is inspired by the efficient evaluation of non-continuous subsequences (described in (Shawe-Taylor and Cristianini, 2004)). To further increase the computation speed, we also applied the pre-selection of node pairs which have non-null kernel.

3.1 Generalized Tree Kernel function

Given a tree fragment space $\mathcal{F} = \{f_1, f_2, .., f_{\mathcal{F}}\}$, we use the indicator function $I_i(n)$ which is equal to 1 if the target f_i is rooted at node n and 0 otherwise. We define the general kernel as:

$$K(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2), \qquad (1)$$

where N_{T_1} and N_{T_2} are the sets of nodes in T_1 and T_2, respectively and $\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} I_i(n_1) I_i(n_2)$, i.e. the number of common fragments rooted at the n_1 and n_2 nodes. We can compute it as follows:

- if the node labels of n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
- else:

$$\Delta(n_1, n_2) = 1 + \sum_{\vec{J}_1, \vec{J}_2, l(\vec{J}_1) = l(\vec{J}_2)} \prod_{i=1}^{l(\vec{J}_1)} \Delta(c_{n_1}[\vec{J}_{1i}], c_{n_2}[\vec{J}_{2i}]) \qquad (2)$$

where $\vec{J}_1 = \langle J_{11}, J_{12}, J_{13}, ..\rangle$ and $\vec{J}_2 = \langle J_{21}, J_{22}, J_{23}, ..\rangle$ are index sequences associated with the ordered child sequences c_{n_1} of n_1 and c_{n_2} of n_2, respectively, $\vec{J}_{1i}$ and $\vec{J}_{2i}$ point to the i-th children in the two sequences, and $l(\cdot)$ returns the sequence length. We note that (1) Eq. 2 is a convolution kernel according to the definition and the proof given in (Haussler, 1999). (2) Such kernel generates a feature space richer than those defined in (Vishwanathan and Smola, 2002; Collins and Duffy, 2002; Zelenko et al., 2003; Culotta and Sorensen, 2004; Shawe-Taylor and Cristianini, 2004). Additionally, we add the decay factor as follows: $\Delta(n_1, n_2) =$

$$\mu\Big(\lambda^2 + \sum_{\vec{J}_1, \vec{J}_2, l(\vec{J}_1) = l(\vec{J}_2)} \lambda^{d(\vec{J}_1) + d(\vec{J}_2)} \prod_{i=1}^{l(\vec{J}_1)} \Delta(c_{n_1}[\vec{J}_{1i}], c_{n_2}[\vec{J}_{2i}])\Big) \qquad (3)$$

where $d(\vec{J}_1) = \vec{J}_{1l(\vec{J}_1)} - \vec{J}_{11}$ and $d(\vec{J}_2) = \vec{J}_{2l(\vec{J}_2)} - \vec{J}_{21}$. In this way, we penalize subtrees built on child subsequences that contain gaps. Moreover, to have a similarity score between 0 and 1, we also apply the normalization in the kernel space, i.e. $K'(T_1, T_2) = \frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1) \times K(T_2, T_2)}}$. As the summation in Eq. 3 can be distributed with respect to different types of sequences, e.g. those composed by p children, it follows that

$$\Delta(n_1, n_2) = \mu\big(\lambda^2 + \textstyle\sum_{p=1}^{lm} \Delta_p(n_1, n_2)\big), \qquad (4)$$

where Δ_p evaluates the number of common subtrees rooted in subsequences of exactly p children (of n_1 and n_2) and $lm = min\{l(c_{n1}), l(c_{n2})\}$. Note also that if we consider only the contribution of the longest sequence of node pairs that have the same children, we implement the SST kernel. For the STs computation we need also to remove the λ^2 term from Eq. 4.

Given the two child sequences $c_1 a = c_{n_1}$ and $c_2 b = c_{n_2}$ (a and b are the last children), $\Delta_p(c_1 a, c_2 b) =$

$$\Delta(a, b) \times \sum_{i=1}^{|c_1|} \sum_{r=1}^{|c_2|} \lambda^{|c_1| - i + |c_2| - r} \times \Delta_{p-1}(c_1[1:i], c_2[1:r]),$$

where $c_1[1:i]$ and $c_2[1:r]$ are the child subsequences from 1 to i and from 1 to r of c_1 and c_2. If we name the double summation term as D_p, we can rewrite the relation as:

$$\Delta_p(c_1a, c_2b) = \begin{cases} \Delta(a,b)D_p(|c_1|,|c_2|) \text{ if } a = b; \\ 0 \qquad\qquad\qquad\qquad otherwise. \end{cases}$$

Note that D_p satisfies the recursive relation:

$$\begin{aligned} D_p(k,l) &= \Delta_{p-1}(s[1:k], t[1:l]) + \lambda D_p(k, l-1) \\ &+ \lambda D_p(k-1, l) + \lambda^2 D_p(k-1, l-1). \end{aligned}$$

By means of the above relation, we can compute the child subsequences of two sets c_1 and c_2 in $O(p|c_1||c_2|)$. This means that the worst case complexity of the PT kernel is $O(p\rho^2|N_{T_1}||N_{T_2}|)$, where ρ is the maximum branching factor of the two trees. Note that the average ρ in natural language parse trees is very small and the overall complexity can be reduced by avoiding the computation of node pairs with different labels. The next section shows our fast algorithm to find non-null node pairs.

3.2 Fast non-null node pair computation

To compute the kernels defined in the previous section, we sum the Δ function for each pair $\langle n_1, n_2\rangle \in N_{T_1} \times N_{T_2}$ (Eq. 1). When the labels associated with n_1 and n_2 are different, we can avoid evaluating $\Delta(n_1, n_2)$ since it is 0. Thus, we look for a node pair set $N_p = \{\langle n_1, n_2\rangle \in N_{T_1} \times N_{T_2} : label(n_1) = label(n_2)\}$.

To efficiently build N_p, we (i) extract the L_1 and L_2 lists of nodes from T_1 and T_2, (ii) sort them in alphanumeric order and (iii) scan them to find N_p. Step (iii) may require only $O(|N_{T_1}| + |N_{T_2}|)$ time, but, if $label(n_1)$ appears r_1 times in T_1 and $label(n_2)$ is repeated r_2 times in T_2, we need to consider $r_1 \times r_2$ pairs. The formal can be found in (Moschitti, 2006).

4 The Experiments

In these experiments, we study tree kernel performance in terms of average running time and accuracy on the classification of predicate arguments. As shown in (Moschitti, 2004), we can label semantic roles by classifying the smallest subtree that includes the predicate with one of its arguments, i.e. the so called PAF structure.

The experiments were carried out with the SVM-light-TK software available at `http://ai-nlp.info.uniroma2.it/moschitti/` which encodes the fast tree kernels in the SVM-light software (Joachims, 1999). The multiclassifiers were obtained by training an SVM for each class in the ONE-vs.-ALL fashion. In the testing phase, we selected the class associated with the maximum SVM score.

For the ST, SST and PT kernels, we found that the best λ values (see Section 3) on the development set were 1, 0.4 and 0.8, respectively, whereas the best μ was 0.4.

4.1 Kernel running time experiments

To study the FTK running time, we extracted from the Penn Treebank several samples of 500 trees containing exactly n nodes. Each point of Figure 4 shows the average computation time[1] of the kernel function applied to the 250,000 pairs of trees of size n. It clearly appears that the FTK-SST and FTK-PT (i.e. FTK applied to the SST and PT kernels) average running time has linear behavior whereas, as expected, the naïve SST algorithm shows a quadratic curve.

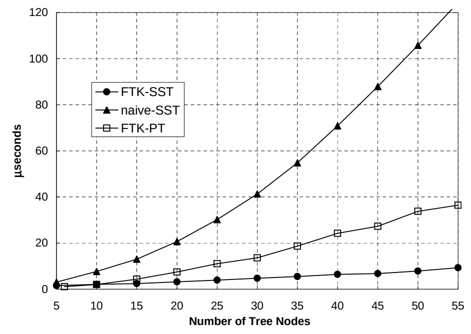

Figure 4: Average time in μseconds for the naïve SST kernel, FTK-SST and FTK-PT evaluations.

4.2 Experiments on SRL dataset

We used two different corpora: PropBank (`www.cis.upenn.edu/∼ace`) along with Penn Treebank 2 (Marcus et al., 1993) and FrameNet. PropBank contains about 53,700 sentences and a fixed split between training and testing used in other researches. In this split, sections from 02 to 21 are used for training, section 23 for testing and section 22 as development set. We considered a total of 122,774 and 7,359 arguments (from *Arg0* to *Arg5*, *ArgA* and *ArgM*) in training and testing, respectively. The tree structures were extracted from the Penn Treebank.

From the FrameNet corpus (`www.icsi.berkeley.edu/∼framenet`) we extracted all

[1]We run the experiments on a Pentium 4, 2GHz, with 1 Gb ram.

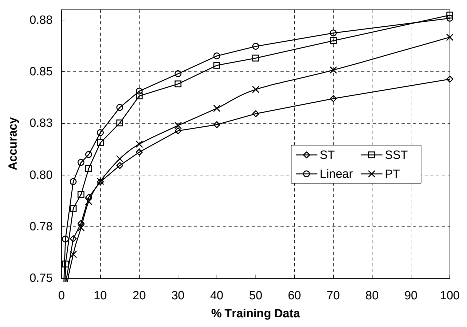

Figure 5: Multiclassifier accuracy according to different training set percentage.

24,558 sentences of the 40 Frames selected for the *Automatic Labeling of Semantic Roles* task of Senseval 3 (`www.senseval.org`). We considered the 18 most frequent roles, for a total of 37,948 examples (30% of the sentences for testing and 70% for training/validation). The sentences were processed with the Collins' parser (Collins, 1997) to generate automatic parse trees.

We run ST, SST and PT kernels along with the linear kernel of standard features (Carreras and Màrquez, 2005) on PropBank training sets of different size. Figure 5 illustrates the learning curves associated with the above kernels for the SVM multiclassifiers.

The tables 1 and 2 report the results, using all available training data, on PropBank and FrameNet test sets, respectively. We note that: (1) the accuracy of PTs is almost equal to the one produced by SSTs as the PT space is a hyperset of SSTs. The small difference is due to the poor relevance of the substructures in the PT $-$ SST set, which degrade the PT space. (2) The high F_1 measures of tree kernels on FrameNet suggest that they are robust with respect to automatic parse trees.

Moreover, the learning time of SVMs using FTK for the classification of one large argument (Arg 0) is much lower than the one required by naïve algorithm. With all the training data FTK terminated in 6 hours whereas the naïve approach required more than 1 week. However, the *complexity burden* of working in the dual space can be alleviated with recent approaches proposed in (Kudo and Matsumoto, 2003; Suzuki et al., 2004).

Finally, we carried out some experiments with the combination between linear and tree kernels and we found that tree kernels improve the models based on manually designed features by 2/3 percent points, thus they can be seen as a useful tactic to boost system accuracy.

Args	Linear	ST	SST	PT
Acc.	87.6	84.6	87.7	86.7

Table 1: Evaluation of kernels on PropBank data and gold parse trees.

Roles	Linear	ST	SST	PT
Acc.	82.3	80.0	81.2	79.9

Table 2: Evaluation of kernels on FrameNet data encoded in automatic parse trees.

References

Xavier Carreras and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of CoNLL05*.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL02*.

Michael Collins. 1997. Three generative, lexicalized models for statistical parsing. In *Proceedings of the ACL97*.

Aron Culotta and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of ACL04*.

Chad Cumby and Dan Roth. 2003. Kernel methods for relational learning. In *Proceedings of ICML03*.

Charles J. Fillmore. 1982. Frame semantics. In *Linguistics in the Morning Calm*.

D. Haussler. 1999. Convolution kernels on discrete structures. Technical report ucs-crl-99-10, University of California Santa Cruz.

T. Joachims. 1999. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*.

Paul Kingsbury and Martha Palmer. 2002. From Treebank to PropBank. In *Proceedings of LREC02*.

Taku Kudo and Yuji Matsumoto. 2003. Fast methods for kernel-based text analysis. In *Proceedings of ACL03*.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*.

Alessandro Moschitti. 2004. A study on convolution kernels for shallow semantic parsing. In *proceedings of ACL04*.

Alessandro Moschitti. 2006. Making tree kernels practical for natural language learning. In *Proceedings of EACL06*.

John Shawe-Taylor and Nello Cristianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Jun Suzuki, Hideki Isozaki, and Eisaku Maeda. 2004. Convolution kernels with feature selection for natural language processing tasks. In *Proceedings of ACL04*.

S.V.N. Vishwanathan and A.J. Smola. 2002. Fast kernels on strings and trees. In *Proceedings of NIPS02*.

D. Zelenko, C. Aone, and A. Richardella. 2003. Kernel methods for relation extraction. *JMLR*.