Making Tree Kernels practical for Natural Language Learning

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

Abstract

In recent years tree kernels have been proposed for the automatic learning of natural language applications. Unfortunately, they show (a) an inherent super linear complexity and (b) a lower accuracy than traditional attribute/value methods.

In this paper, we show that tree kernels are very helpful in the processing of natural language as (a) we provide a simple algorithm to compute tree kernels in linear average running time and (b) our study on the classification properties of diverse tree kernels show that kernel combinations always improve the traditional methods. Experiments with Support Vector Machines on the predicate argument classification task provide empirical support to our thesis.

1 Introduction

In recent years tree kernels have been shown to be interesting approaches for the modeling of syntactic information in natural language tasks, e.g. syntactic parsing (Collins and Duffy, 2002), relation extraction (Zelenko et al., 2003), Named Entity recognition (Cumby and Roth, 2003; Culotta and Sorensen, 2004) and Semantic Parsing (Moschitti, 2004).

The main tree kernel advantage is the possibility to generate a high number of syntactic features and let the learning algorithm to select those most relevant for a specific application. In contrast, their major drawback are (a) the computational time complexity which is superlinear in the number of tree nodes and (b) the accuracy that they produce is often lower than the one provided by linear models on manually designed features.

To solve problem (a), a linear complexity algorithm for the *subtree* (ST) kernel computation, was designed in (Vishwanathan and Smola, 2002). Unfortunately, the ST set is rather poorer than the one generated by the subset tree (SST) kernel designed in (Collins and Duffy, 2002). Intuitively, an ST rooted in a node n of the target tree always contains all n's descendants until the leaves. This does not hold for the SSTs whose leaves can be internal nodes.

To solve the problem (b), a study on different tree substructure spaces should be carried out to derive the tree kernel that provide the highest accuracy. On the one hand, SSTs provide learning algorithms with richer information which may be critical to capture syntactic properties of parse trees as shown, for example, in (Zelenko et al., 2003; Moschitti, 2004). On the other hand, if the SST space contains too many irrelevant features, overfitting may occur and decrease the classification accuracy (Cumby and Roth, 2003). As a consequence, the fewer features of the ST approach may be more appropriate.

In this paper, we aim to solve the above problems. We present (a) an algorithm for the evaluation of the ST and SST kernels which runs in linear average time and (b) a study of the impact of diverse tree kernels on the accuracy of Support Vector Machines (SVMs).

Our fast algorithm computes the kernels between two syntactic parse trees in $O(m + n)$ average time, where m and n are the number of nodes in the two trees. This low complexity allows SVMs to carry out experiments on hundreds of thousands of training instances since it is not higher than the complexity of the polynomial ker-

nel, widely used on large experimentation e.g. (Pradhan et al., 2004). To confirm such hypothesis, we measured the impact of the algorithm on the time required by SVMs for the learning of about 122,774 predicate argument examples annotated in PropBank (Kingsbury and Palmer, 2002) and 37,948 instances annotated in FrameNet (Fillmore, 1982).

Regarding the classification properties, we studied the argument labeling accuracy of ST and SST kernels and their combinations with the *standard features* (Gildea and Jurafsky, 2002). The results show that, on both PropBank and FrameNet datasets, the SST-based kernel, i.e. the richest in terms of substructures, produces the highest SVM accuracy. When SSTs are combined with the manual designed features, we always obtain the best figure classifier. This suggests that the many fragments included in the SST space are relevant and, since their manual design may be problematic (requiring a higher programming effort and deeper knowledge of the linguistic phenomenon), tree kernels provide a remarkable help in feature engineering.

In the remainder of this paper, Section 2 describes the parse tree kernels and our fast algorithm. Section 3 introduces the predicate argument classification problem and its solution. Section 4 shows the comparative performance in term of the execution time and accuracy. Finally, Section 5 discusses the related work whereas Section 6 summarizes the conclusions.

2 Fast Parse Tree Kernels

The kernels that we consider represent trees in terms of their substructures (fragments). These latter define feature spaces which, in turn, are mapped into vector spaces, e.g. $\Re^n$. The associated kernel function measures the similarity between two trees by counting the number of their common fragments. More precisely, a kernel function detects if a tree subpart (common to both trees) belongs to the feature space that we intend to generate. For such purpose, the fragment types need to be described. We consider two important characterizations: the subtrees (STs) and the subset trees (SSTs).

2.1 Subtrees and Subset Trees

In our study, we consider syntactic parse trees, consequently, each node with its children is associated with a grammar production rule, where the symbol at left-hand side corresponds to the parent node and the symbols at right-hand side are associated with its children. The terminal symbols of the grammar are always associated with the leaves of the tree. For example, Figure 1 illustrates the syntactic parse of the sentence `"Mary brought a cat to school"`.

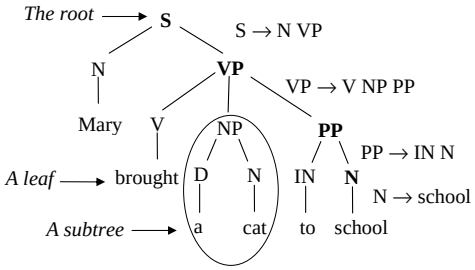

Figure 1: A syntactic parse tree.

We define as a **subtree** (ST) any node of a tree along with all its descendants. For example, the line in Figure 1 circles the subtree rooted in the `NP` node. A **subset tree** (SST) is a more general structure. The difference with the subtrees is that the leaves can be associated with non-terminal symbols. The SSTs satisfy the constraint that they are generated by applying the same grammatical rule set which generated the original tree. For example, `[S [N VP]]` is a SST of the tree in Figure 1 which has two non-terminal symbols, `N` and `VP`, as leaves.

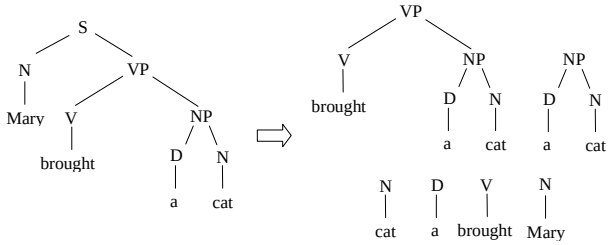

Figure 2: A syntactic parse tree with its subtrees (STs).

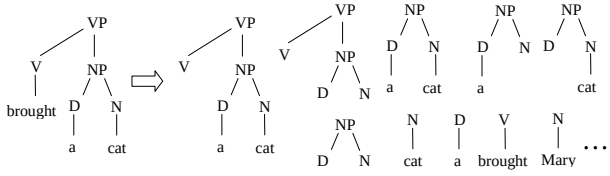

Figure 3: A tree with some of its subset trees (SSTs).

Given a syntactic tree we can use as feature representation the set of all its STs or SSTs. For example, Figure 2 shows the parse tree of the sentence `"Mary brought a cat"` together with its 6 STs, whereas Figure 3 shows 10 SSTs (out of 17) of the subtree of Figure 2 rooted in `VP`. The

high different number of substructures gives an intuitive quantification of the different information level between the two tree-based representations.

2.2 The Tree Kernel Functions

The main idea of tree kernels is to compute the number of the common substructures between two trees T_1 and T_2 without explicitly considering the whole fragment space. For this purpose, we slightly modified the kernel function proposed in (Collins and Duffy, 2002) by introducing a parameter σ which enables the ST or the SST evaluation.

Given the set of fragments $\{f_1, f_2, ..\} = \mathcal{F}$, we defined the indicator function $I_i(n)$ which is equal 1 if the target f_i is rooted at node n and 0 otherwise. We define

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

where N_{T_1} and N_{T_2} are the sets of the T_1's and T_2's nodes, respectively and $\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} I_i(n_1) I_i(n_2)$. This latter is equal to the number of common fragments rooted in the n_1 and n_2 nodes. We can compute Δ as follows:

1. if the productions at n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
2. if the productions at n_1 and n_2 are the same, and n_1 and n_2 have only leaf children (i.e. they are pre-terminals symbols) then $\Delta(n_1, n_2) = 1$;
3. if the productions at n_1 and n_2 are the same, and n_1 and n_2 are not pre-terminals then

$$\Delta(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (\sigma + \Delta(c_{n_1}^j, c_{n_2}^j)) \quad (2)$$

where $\sigma \in \{0, 1\}$, $nc(n_1)$ is the number of the children of n_1 and c_n^j is the j-th child of the node n. Note that, since the productions are the same, $nc(n_1) = nc(n_2)$.

When $\sigma = 0$, $\Delta(n_1, n_2)$ is equal 1 only if $\forall j \quad \Delta(c_{n_1}^j, c_{n_2}^j) = 1$, i.e. all the productions associated with the children are identical. By recursively applying this property, it follows that the subtrees in n_1 and n_2 are identical. Thus, Eq. 1 evaluates the subtree (ST) kernel. When $\sigma = 1$, $\Delta(n_1, n_2)$ evaluates the number of SSTs common to n_1 and n_2 as proved in (Collins and Duffy, 2002).

Additionally, we study some variations of the above kernels which include the leaves in the fragment space. For this purpose, it is enough to add the condition:

0. if n_1 and n_2 are leaves and their associated symbols are equal then $\Delta(n_1, n_2) = 1$,

to the recursive rule set for the Δ evaluation (Zhang and Lee, 2003). We will refer to such extended kernels as ST+bow and SST+bow (*bag-of-words*).

Moreover, we add the decay factor λ by modifying steps (2) and (3) as follows[1]:

2. $\Delta(n_1, n_2) = \lambda$,
3. $\Delta(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (\sigma + \Delta(c_{n_1}^j, c_{n_2}^j))$.

The computational complexity of Eq. 1 is $O(|N_{T_1}| \times |N_{T_2}|)$. We will refer to this basic implementation as the Quadratic Tree Kernel (QTK). However, as observed in (Collins and Duffy, 2002) this worst case is quite unlikely for the syntactic trees of natural language sentences, thus, we can design algorithms that run in linear time on average.

```
function Evaluate_Pair_Set(Tree T1, T2) returns NODE_PAIR_SET;
LIST L1,L2;
NODE_PAIR_SET Np;
begin
   L1 = T1.ordered_list;
   L2 = T2.ordered_list; /*the lists were sorted at loading time*/
   n1 = extract(L1); /*get the head element and*/
   n2 = extract(L2); /*remove it from the list*/
   while (n1 and n2 are not NULL)
      if (production_of(n1) > production_of(n2))
         then n2 = extract(L2);
         else if (production_of(n1) < production_of(n2))
            then n1 = extract(L1);
            else
               while (production_of(n1) == production_of(n2))
                  while (production_of(n1) == production_of(n2))
                     add(⟨n1, n2⟩, Np);
                     n2=get_next_elem(L2); /*get the head element
                     and move the pointer to the next element*/
                  end
                  n1 = extract(L1);
                  reset(L2); /*set the pointer at the first element*/
               end
   end
   return Np ;
end
```

Table 1: Pseudo-code for fast evaluation of the node pair sets used in the fast Tree Kernel.

2.3 A Fast Tree Kernel (FTK)

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 productions associated with n_1 and n_2 are different, we can avoid to evaluate $\Delta(n_1, n_2)$ since it is 0.

[1] To have a similarity score between 0 and 1, we also apply the normalization in the kernel space, i.e. $K'(T_1, T_2) = \frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1) \times K(T_2, T_2)}}$.

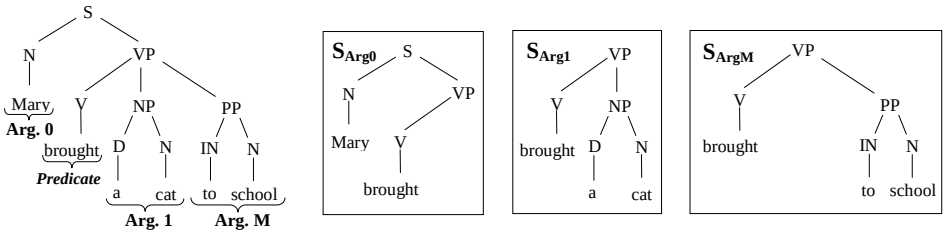

Figure 4: Tree substructure space for predicate argument classification.

Thus, we look for a node pair set N_p =$\{\langle n_1, n_2\rangle \in N_{T_1} \times N_{T_2} : p(n_1) = p(n_2)\}$, where $p(n)$ returns the production rule associated with n.

To efficiently build N_p, we (i) extract the L_1 and L_2 lists of the production rules from T_1 and T_2, (ii) sort them in the alphanumeric order and (iii) scan them to find the node pairs $\langle n_1, n_2\rangle$ such that $(p(n_1) = p(n_2)) \in L_1 \cap L_2$. Step (iii) may require only $O(|N_{T_1}| + |N_{T_2}|)$ time, but, if $p(n_1)$ appears r_1 times in T_1 and $p(n_2)$ is repeated r_2 times in T_2, we need to consider $r_1 \times r_2$ pairs. The formal algorithm is given in Table 1.

Note that:
(a) The list sorting can be done only once at the data preparation time (i.e. before training) in $O(|N_{T_1}| \times log(|N_{T_1}|))$.
(b) The algorithm shows that the worst case occurs when the parse trees are both generated using only one production rule, i.e. the two internal while cycles carry out $|N_{T_1}| \times |N_{T_2}|$ iterations. In contrast, two identical parse trees may generate a linear number of non-null pairs if there are few groups of nodes associated with the same production rule.
(c) Such approach is perfectly compatible with the dynamic programming algorithm which computes Δ. In fact, the only difference with the original approach is that the matrix entries corresponding to pairs of different production rules are not considered. Since such entries contain null values they do not affect the application of the original dynamic programming. Moreover, the order of the pair evaluation can be established at run time, starting from the root nodes towards the children.

3 A Semantic Application of Parse Tree Kernels

An interesting application of the SST kernel is the classification of the predicate argument structures defined in PropBank (Kingsbury and Palmer, 2002) or FrameNet (Fillmore, 1982). Figure 4 shows the parse tree of the sentence: "Mary brought a cat to school" along with the predicate argument annotation proposed in the PropBank project. Only verbs are considered as predicates whereas arguments are labeled sequentially from ARG0 to ARG9.

Also in FrameNet predicate/argument information is described but for this purpose richer semantic structures called Frames are used. The Frames are schematic representations of situations involving various participants, properties and roles in which a word may be typically used. Frame elements or semantic roles are arguments of predicates called target words. For example the following sentence is annotated according to the ARREST frame:
[$_{Time}$ One Saturday night] [$_{Authorities}$ police in Brooklyn] [$_{Target}$ apprehended] [$_{Suspect}$ sixteen teenagers].
The roles *Suspect* and *Authorities* are specific to the frame.

The common approach to learn the classification of predicate arguments relates to the extraction of features from the syntactic parse tree of the target sentence. In (Gildea and Jurafsky, 2002) seven different features[2], which aim to capture the relation between the predicate and its arguments, were proposed. For example, the *Parse Tree Path* of the pair $\langle$brought, ARG1$\rangle$ in the syntactic tree of Figure 4 is V $\uparrow$ VP $\downarrow$ NP. It encodes the dependency between the predicate and the argument as a sequence of nonterminal labels linked by direction symbols (up or down).

An alternative tree kernel representation, proposed in (Moschitti, 2004), is the selection of the minimal tree subset that includes a predicate with only one of its arguments. For example, in Figure 4, the substructures inside the three frames are the semantic/syntactic structures associated with the three arguments of the verb *to bring*, i.e. S_{ARG0}, S_{ARG1} and S_{ARGM}.

Given a feature representation of predicate ar-

[2]Namely, they are *Phrase Type*, *Parse Tree Path*, *Predicate Word*, *Head Word*, *Governing Category*, *Position* and *Voice*.

guments, we can build an individual ONE-vs-ALL (OVA) classifier C_i for each argument i. As a final decision of the multiclassifier, we select the argument type ARG_t associated with the maximum value among the scores provided by the C_i, i.e. $t = argmax_{i \in S} \quad score(C_i)$, where S is the set of argument types. We adopted the OVA approach as it is simple and effective as showed in (Pradhan et al., 2004).

Note that the representation in Figure 4 is quite intuitive and, to conceive it, the designer requires much less linguistic knowledge about semantic roles than those necessary to define relevant features manually. To understand such point, we should make a step back before Gildea and Jurafsky defined the first set of features for Semantic Role Labeling (SRL). The idea that syntax may have been useful to derive semantic information was already inspired by linguists, but from a machine learning point of view, to decide which tree fragments may have been useful for semantic role labeling was not an easy task. In principle, the designer should have had to select and experiment all possible tree subparts. This is exactly what the tree kernels can automatically do: the designer just need to roughly select the interesting whole subtree (correlated with the linguistic phenomenon) and the tree kernel will generate all possible syntactic features from it. The task of selecting the most relevant substructures is carried out by the kernel machines themselves.

4 The Experiments

The aim of the experiments is twofold. On the one hand, we show that the FTK running time is linear on the average case and is much faster than QTK. This is accomplished by measuring the learning time and the average kernel computation time. On the other hand, we study the impact of the different tree based kernels on the predicate argument classification accuracy.

4.1 Experimental Set-up

We used two different corpora: PropBank (`www.cis.upenn.edu/~ace`) along with PennTree bank 2 (Marcus et al., 1993) and FrameNet.

PropBank contains about 53,700 sentences and a fixed split between training and testing which has been used in other researches, e.g. (Gildea and Palmer, 2002; Pradhan et al., 2004). In this split, sections from 02 to 21 are used for training, section 23 for testing and sections 1 and 22 as developing set. We considered a total of 122,774 and 7,359 arguments (from ARG0 to ARG9, ARGA and ARGM) in training and testing, respectively. Their tree structures were extracted from the Penn Treebank. It should be noted that the main contribution to the global accuracy is given by ARG0, ARG1 and ARGM.

From the FrameNet corpus (`http://www.icsi.berkeley.edu/~framenet`), we extracted all 24,558 sentences of the 40 Frames selected for the *Automatic Labeling of Semantic Roles* task of Senseval 3 (`www.senseval.org`). We mapped together the semantic roles having the same name and we considered only the 18 most frequent roles associated with verbal predicates, for a total of 37,948 arguments. We randomly selected 30% of sentences for testing and 70% for training. Additionally, 30% of training was used as a *validation-set*. Note that, since the FrameNet data does not include deep syntactic tree annotation, we processed the FrameNet data with Collins' parser (Collins, 1997), consequently, the experiments on FrameNet relate to automatic syntactic parse trees.

The classifier evaluations were carried out with the SVM-light-TK software available at `http://ai-nlp.info.uniroma2.it/moschitti/` which encodes ST and SST kernels in the SVM-light software (Joachims, 1999). We used the default linear (Linear) and polynomial (Poly) kernels for the evaluations with the standard features defined in (Gildea and Jurafsky, 2002). We adopted the default regularization parameter (i.e., the average of $1/||\vec{x}||$) and we tried a few cost-factor values (i.e., $j \in \{1, 3, 7, 10, 30, 100\}$) to adjust the rate between Precision and Recall on the *validation-set*.

For the ST and SST kernels, we derived that the best λ (see Section 2.2) were 1 and 0.4, respectively. The classification performance was evaluated using the F_1 measure[3] for the single arguments and the accuracy for the final multiclassifier. This latter choice allows us to compare our results with previous literature work, e.g. (Gildea and Jurafsky, 2002; Pradhan et al., 2004).

4.2 Time Complexity Experiments

In this section we compare our Fast Tree Kernel (FTK) approach with the Quadratic Tree Kernel (QTK) algorithm. The latter refers to the naive evaluation of Eq. 1 as presented in (Collins and Duffy, 2002).

[3] F_1 assigns equal importance to Precision P and Recall R, i.e. $f_1 = \frac{2P \times R}{P+R}$.

Figure 5 shows the learning time[4] of the SVMs using QTK and FTK (over the SST structures) for the classification of one large argument (i.e. ARG0), according to different percentages of training data. We note that, with 70% of the training data, FTK is about 10 times faster than QTK. With all the training data FTK terminated in 6 hours whereas QTK required more than 1 week.

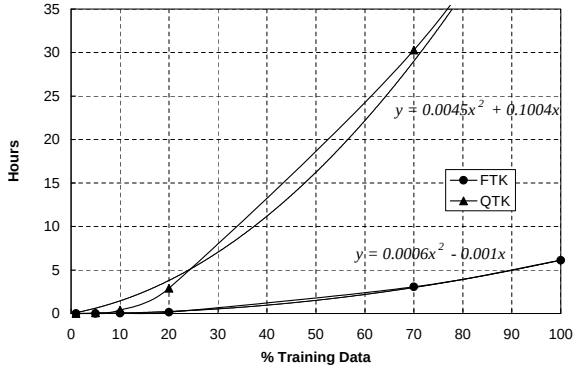

Figure 5: ARG0 classifier learning time according to different training percentages.

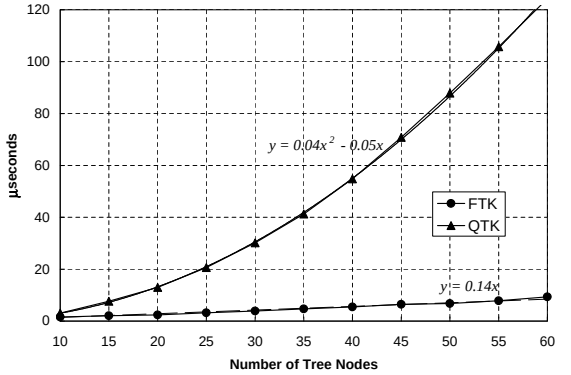

Figure 6: Average time in seconds for the QTK and FTK evaluations.

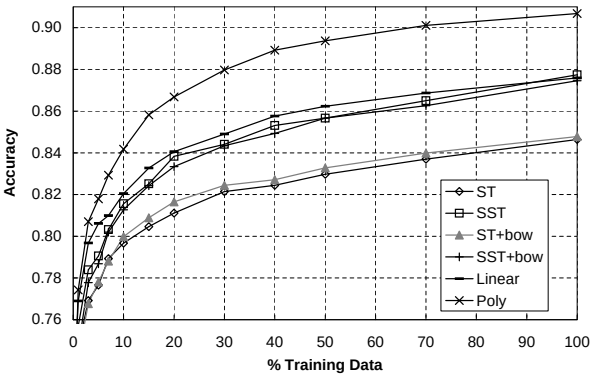

Figure 7: Multiclassifier accuracy according to different training set percentages.

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

The above results are quite interesting because they show that (1) we can use tree kernels with SVMs on huge training sets, e.g. on 122,774 instances and (2) the time needed to converge is approximately the one required by SVMs when using polynomial kernel. This latter shows the minimal complexity needed to work in the dual space.

To study the FTK running time, we extracted from PennTree bank the first 500 trees[5] containing exactly n nodes, then, we evaluated all 25,000 possible tree pairs. Each point of the Figure 6 shows the average computation time on all the tree pairs of a fixed size n.

In the figures, the trend lines which best interpolates the experimental values are also shown. It clearly appears that the training time is quadratic as SVMs have quadratic learning time complexity (see Figure 5) whereas the FTK running time has a linear behavior (Figure 6). The QTK algorithm shows a quadratic running time complexity, as expected.

4.3 Accuracy of the Tree Kernels

In these experiments, we investigate which kernel is the most accurate for the predicate argument classification.

First, we run ST, SST, ST+bow, SST+bow, Linear and Poly kernels over different training-set size of PropBank. Figure 7 shows the learning curves associated with the above kernels for the SVM-based multiclassifier. We note that (a) SSTs have a higher accuracy than STs, (b) *bow* does not improve either ST or SST kernels and (c) in the final part of the plot SST shows a higher gradient than ST, Linear and Poly. This latter produces the best accuracy 90.5% in line with the literature findings using standard features and polynomial SVMs, e.g. 87.1%[6] in (Pradhan et al., 2004).

Second, in tables 2 and 3, we report the results using all available training data, on PropBank and FrameNet test sets, respectively. Each row of the two tables shows the F_1 measure of the individual classifiers using different kernels whereas the last column illustrates the global accuracy of the multiclassifier.

[5]We measured also the computation time for the incomplete trees associated with the predicate argument structures (see Section 3); we obtained the same results.

[6]The small difference (2.4%) is mainly due to the different treatment of ARGMs: we built a single ARGM class for all subclasses, e.g. ARGM-LOC and ARGM-TMP, whereas in (Pradhan et al., 2004), the ARGMs, were evaluated separately.

We note that, the F_1 of the single arguments across the different kernels follows the same behavior of the global multiclassifier accuracy. On FrameNet, the *bow* impact on the ST and SST accuracy is higher than on PropBank as it produces an improvement of about 1.5%. This suggests that (1) to detect semantic roles, lexical information is very important, (2) *bow* give a higher contribution as errors in POS-tagging make the *word + POS* fragments less reliable and (3) as the FrameNet trees are obtained with the Collins' syntactic parser, tree kernels seem robust to incorrect parse trees.

Third, we point out that the polynomial kernel on flat features is more accurate than tree kernels but the design of such effective features required noticeable knowledge and effort (Gildea and Jurafsky, 2002). On the contrary, the choice of subtrees suitable to syntactically characterize a target phenomenon seems a easier task (see Section 3 for the predicate argument case). Moreover, by combining polynomial and SST kernels, we can improve the classification accuracy (Moschitti, 2004), i.e. tree kernels provide the learning algorithm with many relevant fragments which hardly can be designed by hand. In fact, as many predicate argument structures are quite large (up to 100 nodes) they contain many fragments.

ARGs	ST	SST	ST+bow	SST+bow	Linear	$Poly$
ARG0	86.5	88.0	86.9	88.4	88.6	90.6
ARG1	83.1	87.4	82.8	86.7	85.9	90.8
ARG2	58.0	67.6	58.9	66.7	65.5	80.4
ARG3	35.7	37.5	39.3	41.2	51.9	60.4
ARG4	62.7	65.6	63.3	63.9	66.2	70.0
ARGM	92.0	94.2	92.0	93.7	94.9	95.3
Acc.	84.6	87.7	84.8	87.5	87.6	90.7

Table 2: Evaluation of Kernels on PropBank.

Roles	ST	SST	ST+bow	SST+bow	Linear	$Poly$
agent	86.9	87.8	89.2	90.2	89.8	91.7
theme	76.1	79.2	78.5	80.7	82.9	90.4
goal	77.9	78.9	78.2	80.1	80.2	85.8
path	82.8	84.4	83.7	85.1	81.3	85.5
manner	79.9	82.0	81.3	82.5	70.8	80.5
source	85.6	87.7	86.9	87.8	86.5	89.8
time	76.3	78.3	77.0	79.1	61.8	68.3
reason	75.9	77.3	78.9	81.4	82.9	86.4
Acc. 18 roles	80.0	81.2	81.3	82.9	82.3	85.6

Table 3: Evaluation of the Kernels on FrameNet semantic roles.

Finally, to study the combined kernels, we applied the $K_1 + \gamma K_2$ formula, where K_1 is either the Linear or the Poly kernel and K_2 is the ST or the SST kernel. Table 4 shows the results of four kernel combinations. We note that, (a) STs and SSTs improve Poly (about 0.5 and 2 percent points on PropBank and FrameNet, respectively) and (b) the linear kernel, which uses fewer features than Poly, is more enhanced by the SSTs than STs (for example on PropBank we have 89.4% and 88.6% vs. 87.6%), i.e. Linear takes advantage by the richer feature set of the SSTs. It should be noted that our results of kernel combinations on FrameNet are in contrast with (Moschitti, 2004), where no improvement was obtained. Our explanation is that, thanks to the fast evaluation of FTK, we could carry out an adequate parameterization.

Corpus	Poly	ST+Linear	SST+Linear	ST+Poly	SST+Poly
PropBank	90.7	88.6	89.4	91.1	91.3
FrameNet	85.6	85.3	85.8	87.5	87.2

Table 4: Multiclassifier accuracy using Kernel Combinations.

5 Related Work

Recently, several tree kernels have been designed. In the following, we highlight their differences and properties.

In (Collins and Duffy, 2002), the SST tree kernel was experimented with the Voted Perceptron for the parse-tree reranking task. The combination with the original PCFG model improved the syntactic parsing. Additionally, it was alluded that the average execution time depends on the number of repeated productions.

In (Vishwanathan and Smola, 2002), a linear complexity algorithm for the computation of the ST kernel is provided (in the worst case). The main idea is the use of the suffix trees to store partial matches for the evaluation of the string kernel (Lodhi et al., 2000). This can be used to compute the ST fragments once the tree is converted into a string. To our knowledge, ours is the first application of the ST kernel for a natural language task.

In (Kazama and Torisawa, 2005), an interesting algorithm that speeds up the average running time is presented. Such algorithm looks for node pairs that have in common a large number of trees (*malicious nodes*) and applies a transformation to the trees rooted in such nodes to make faster the kernel computation. The results show an increase of the speed similar to the one produced by our method.

In (Zelenko et al., 2003), two kernels over syntactic shallow parser structures were devised for the extraction of linguistic relations, e.g. *person-affiliation*. To measure the similarity between two

nodes, the *contiguous string kernel* and the *sparse string kernel* (Lodhi et al., 2000) were used. In (Culotta and Sorensen, 2004) such kernels were slightly generalized by providing a matching function for the node pairs. The time complexity for their computation limited the experiments on data set of just 200 news items. Moreover, we note that the above tree kernels are not convolution kernels as those proposed in this article.

In (Shen et al., 2003), a tree-kernel based on Lexicalized Tree Adjoining Grammar (LTAG) for the parse-reranking task was proposed. Since QTK was used for the kernel computation, the high learning complexity forced the authors to train different SVMs on different slices of training data. Our FTK, adapted for the LTAG tree kernel, would have allowed SVMs to be trained on the whole data.

In (Cumby and Roth, 2003), a feature description language was used to extract structural features from the syntactic shallow parse trees associated with named entities. The experiments on the named entity categorization showed that when the description language selects an adequate set of tree fragments the Voted Perceptron algorithm increases its classification accuracy. The explanation was that the complete tree fragment set contains many irrelevant features and may cause overfitting.

6 Conclusions

In this paper, we have shown that tree kernels can effectively be adopted in practical natural language applications. The main arguments against their use are their efficiency and accuracy lower than traditional feature based approaches. We have shown that a fast algorithm (FTK) can evaluate tree kernels in a linear average running time and also that the overall converging time required by SVMs is compatible with very large data sets. Regarding the accuracy, the experiments with Support Vector Machines on the PropBank and FrameNet predicate argument structures show that: (a) the richer the kernel is in term of substructures (e.g. SST), the higher the accuracy is, (b) tree kernels are effective also in case of automatic parse trees and (c) as kernel combinations always improve traditional feature models, the best approach is to combine scalar-based and structured based kernels.

Acknowledgments

I would like to thank the AI group at the University of Rome "Tor Vergata". Many thanks to the EACL 2006 anonymous reviewers, Roberto Basili and Giorgio Satta who provided me with valuable suggestions. This research is partially supported by the Presto Space EU Project#: FP6-507336.

References

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*, Madrid, Spain.

Aron Culotta and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *proceedings of ACL04*, Barcelona, Spain.

Chad Cumby and Dan Roth. 2003. Kernel methods for relational learning. In *proceedings of ICML 2003*. Washington, US.

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

Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistic*, 28(3):496–530.

Daniel Gildea and Martha Palmer. 2002. The necessity of parsing for predicate argument recognition. In *proceedings of ACL02, Philadelphia, PA*.

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*.

Junichi Kazama and Kentaro Torisawa. 2005. Speeding up training with tree kernels for node relation labeling. In *proceedings of EMNLP 2005*, Toronto, Canada.

Paul Kingsbury and Martha Palmer. 2002. From Treebank to PropBank. In *proceedings of LREC-2002*, Spain.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Christopher Watkins. 2000. Text classification using string kernels. In *NIPS02*, Vancouver, Canada.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19:313–330.

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

Sameer Pradhan, Kadri Hacioglu, Valeri Krugler, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2005. Support vector learning for semantic argument classification. *Machine Learning Journal*.

Libin Shen, Anoop Sarkar, and Aravind Joshi. 2003. Using LTAG based features in parse reranking. In *proceedings of EMNLP 2003*, Sapporo, Japan.

Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In *proceedings of EMNLP 2004* Barcelona, Spain.

S.V.N. Vishwanathan and A.J. Smola. 2002. Fast kernels on strings and trees. In *proceedings of Neural Information Processing Systems*.

D. Zelenko, C. Aone, and A. Richardella. 2003. Kernel methods for relation extraction. *Journal of Machine Learning Research*.

Dell Zhang and Wee Sun Lee. 2003. Question classification using support vector machines. In *proceedings of SIGIR'03*, ACM Press.