Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees

Alessandro Moschitti

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

Abstract. In this paper, we provide a study on the use of tree kernels to encode syntactic parsing information in natural language learning. In particular, we propose a new convolution kernel, namely the Partial Tree (PT) kernel, to fully exploit dependency trees. We also propose an efficient algorithm for its computation which is futhermore sped-up by applying the selection of tree nodes with non-null kernel. The experiments with Support Vector Machines on the task of semantic role labeling and question classification show that (a) the kernel running time is linear on the average case and (b) the PT kernel improves on the other tree kernels when applied to the appropriate parsing paradigm.

1 Introduction

Literature work shows several attempts (e.g. [1]) to define linking theories between the syntax and semantics of natural languages. As no complete theory has yet been defined the design of syntactic features to learn semantic structures requires a remarkable research effort and intuition. Tree kernels have been applied to reduce such effort for several natural language tasks, e.g. syntactic parsing re-ranking [2], relation extraction [3], named entity recognition [4,5] and Semantic Role Labeling [6].

These studies show that the kernel ability to generate large feature sets is useful to quickly model new and not well understood linguistic phenomena in learning machines. However, it is often possible to manually design features for linear kernels that produce high accuracy and fast computation time whereas the complexity of tree kernels may prevent their application in real scenarios.

In general, the poor tree kernel results depend on the specific application but also on the absence of studies that suggest which tree kernel type should be applied. For example, the *subtree* (ST) kernel defined in [7] is characterized by structures that contain all the descendants of the target root node until the leaves whereas the *subset trees* (SSTs) defined in [2] may contain internal subtrees, with no leaves. How do such different spaces impact on natural language tasks? Does the parsing paradigm (constituent or dependency) affect the accuracy of different kernels?

Regarding the complexity problem, although the SST kernel computation time has been proven to be inherently quadratic in the number of tree nodes [2], we may design algorithms that run fast on the average case.

J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (Eds.): ECML 2006, LNAI 4212, pp. 318–329, 2006.

In this paper, we study the impact of the ST and SST kernels on the modeling of syntactic information in Support Vector Machines. To carry out a comprehensive investigation, we have defined a novel tree kernel based on a general form of substructures, namely, the partial tree (PT) kernel. Moreover, to solve the computation problems, we propose algorithms which, on the average case, evaluate the above kernels in a running time linear in the number of nodes of the two input parse trees.

We experimented with such kernels and Support Vector Machines (SVMs) on (a) the classification of semantic roles defined in PropBank [8] and FrameNet [9] and (b) the classification of questions from Question Answering scenarios. We used both gold standard trees from the Penn Treebank [10] and automatic trees derived with the Collins [11] and Stanford [12] parsers. The results show that: (1) the SST kernel is more appropriate to exploit syntactic information from constituent trees. (2) The new PT kernel is slightly less accurate than the SST one on constituent trees but much more accurate on dependency structures. (3) Our fast algorithms show a linear running time.

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 introduces the Semantic Role Labeling (SRL) and Question Classification (QC) problems and their solution along with the related work. Section 5 shows the comparative kernel performance in terms of execution time and accuracy. Finally, Section 6 summarizes the conclusions.

2 Tree Kernel Spaces

The kernels that we consider represent trees in terms of their substructures (fragments). The 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 desired fragments need to be described. We consider three important characterizations: the subtrees (STs), the subset trees (SSTs) and a new tree class, i.e. the partial trees (PTs).

As we consider syntactic parse trees, each node with its children is associated with a grammar production rule, where the symbol at the left-hand side corresponds to the parent and the symbols at the right-hand side are associated with the children. The terminal symbols of the grammar are always associated with the tree leaves.

We define as a **subtree** (ST) any node of a tree along with all its descendants. For example, Figure 1 shows the parse tree of the sentence `"Mary brought a cat"` together with its 6 STs. A **subset tree** (SST) is a more general structure since its leaves can be non-terminal symbols.

For example, Figure 2 shows 10 SSTs (out of 17) of the subtree of Figure 1 rooted in `VP`. The SSTs satisfy the constraint that grammatical rules cannot be broken. For example, `[VP [V NP]]` is an SST which has two non-terminal symbols, `V` and `NP`, as leaves whereas `[VP [V]]` is not an SST. If we relax the constraint over the SSTs, we obtain a more general form of substructures that we

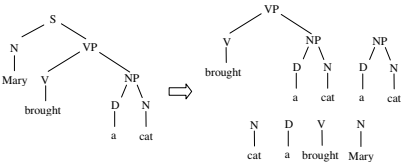

Fig. 1. A syntactic parse tree with its subtrees (STs)

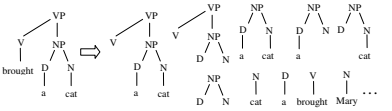

Fig. 2. A tree with some of its subset trees (SSTs)

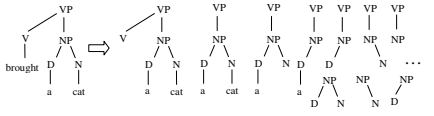

Fig. 3. A tree with some of its partial trees (PTs)

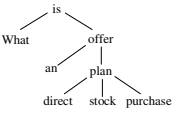

Fig. 4. A dependency tree of a question

call **partial trees** (PTs). These can be generated by the application of partial production rules of the grammar, consequently `[VP [V]]` and `[VP [NP]]` are valid PTs. Figure 3 shows that the number of PTs derived from the same tree as before is still higher (i.e. 30 PTs). These different substructure numbers provide an intuitive quantification of the different information levels among the tree-based representations.

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 have designed a general function to compute the ST, SST and PT kernels. Our fast evaluation of the PT kernel is inspired by the efficient evaluation of non-continuous subsequences (described in [13]). To increase the computation speed of the above tree kernels, we also apply the pre-selection of node pairs which have non-null kernel.

3.1 The Partial Tree Kernel

The evaluation of the common PTs rooted in nodes n_1 and n_2 requires the selection of the shared child subsets of the two nodes, e.g. `[S [DT JJ N]]` and `[S [DT N N]]` have `[S [N]]` (2 times) and `[S [DT N]]` in common. As the order of the children is important, we can use subsequence kernels for their generation. More in detail, let $\mathcal{F} = \{f_1, f_2, .., f_{|\mathcal{F}|}\}$ be a tree fragment space of type PTs and let the indicator function $I_i(n)$ be equal to 1 if the target f_i is rooted at node n and 0 otherwise, we define the PT kernel as:

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

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

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

where $\boldsymbol{J}_1 = \langle J_{11}, J_{12}, J_{13}, .. \rangle$ and $\boldsymbol{J}_2 = \langle J_{21}, J_{22}, J_{23}, .. \rangle$ are index sequences associated with the ordered child sequences c_{n_1} of n_1 and c_{n_2} of n_2, respectively, $\boldsymbol{J}_{1i}$ and $\boldsymbol{J}_{2i}$ 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 [14]. (2) Such kernel generates a richer feature space than those defined in [7, 2, 3, 5, 13]. Additionally, we add two decay factors: μ for the height of the tree and λ for the length of the child sequences. It follows that

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

where $d(\boldsymbol{J}_1) = \boldsymbol{J}_{1l(\boldsymbol{J}_1)} - \boldsymbol{J}_{11}$ and $d(\boldsymbol{J}_2) = \boldsymbol{J}_{2l(\boldsymbol{J}_2)} - \boldsymbol{J}_{21}$. In this way, we penalize both larger trees and 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. $K'(T_1, T_2) = \frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1) \times K(T_2, T_2)}}$.

3.2 Efficient Tree Kernel Computation

Clearly, the naïve approach to evaluate Eq. 3 requires exponential time. We can efficiently compute it by considering that 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 $\Delta(n_1, n_2) = \mu\big(\lambda^2 + \sum_{p=1}^{lm} \Delta_p(c_{n_1}, c_{n_2})\big)$, (4)

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

Given the two child sequences $s_1 a = c_{n_1}$ and $s_2 b = c_{n_2}$ (a and b are the last children),

$$\Delta_p(s_1 a, s_2 b) = \Delta(a, b) \times \sum_{i=1}^{|s_1|} \sum_{r=1}^{|s_2|} \lambda^{|s_1| - i + |s_2| - r} \times \Delta_{p-1}(s_1[1:i], s_2[1:r]),$$

where $s_1[1:i]$ and $s_2[1:r]$ are the child subsequences from 1 to i and from 1 to r of s_1 and s_2. If we name the double summation term as D_p, we can rewrite the relation as:

$$\Delta_p(s_1a, s_2b) = \begin{cases} \Delta(a,b)D_p(|s_1|,|s_2|) \text{ if } a = b; \\ 0 \qquad\qquad\qquad\qquad otherwise. \end{cases}$$

Note that D_p satisfies the recursive relation: $D_p(k,l) =$

$$\Delta_{p-1}(s_1[1:k], s_2[1:l]) + \lambda D_p(k, l-1) + \lambda D_p(k-1, l) + \lambda^2 D_p(k-1, l-1) \quad (5)$$

By means of the above relation, we can compute the child subsequences of two sequences s_1 and s_2 in $O(p|s_1||s_2|)$. 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.

Table 1. Pseudo-code for fast evaluation of the node pairs with non-null kernel (FTK)

```
function Evaluate_Pair_Set(Tree T1, T2)
LIST L1,L2;
NODE_PAIR_SET Np;
begin
   L1 = T1.ordered_list;
   L2 = T2.ordered_list; // lists sorted at loading time
   n1 = extract(L1); // get the head element and remove it from the list
   n2 = extract(L2);
   while (n1 and n2 are not NULL)
      if (label(n1) > label(n2))
         then n2 = extract(L2);
         else if (label(n1) < label(n2))
            then n1 = extract(L1);
            else
               while (label(n1) == label(n2))
                  while (label(n1) == label(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
```

3.3 Fast Non-null Node Pair Computation

To compute the tree kernels, 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 $N_p = \{\langle n_1, n_2\rangle \in N_{T_1} \times N_{T_2} : label(n_1) = label(n_2)\}$. N_p can be evaluated by (i) extracting the L_1 and L_2 lists of nodes from T_1 and T_2, (ii) sorting them in alphanumeric order and (iii) scanning them to derive the node intersection. Step (iii) may require only $O(|N_{T_1}| + |N_{T_2}|)$ time, but, if $label(n_1)$=$label(n_2)$ appears r_1 times in T_1 and r_2 times in T_2, the number of pairs will be $r_1 \times r_2$. The formal algorithm (FTK) is shown in Table 1.

Note that the list sorting can be done only once at data preparation time (i.e. before training) in $O(|N_{T_1}| \times log(|N_{T_1}|))$. The worst case occurs when the

two parse trees are both generated by only one production rule since the two internal *while* cycles generate $|N_{T_1}| \times |N_{T_2}|$ pairs. Moreover, the probability of two identical production rules is lower than that of two identical nodes, thus, we can furthermore speed up the SST (and ST) kernel by (a) sorting the node list with respect to production rules and (b) replacing the `label(`n`)` function with `production_at(`n`)`.

3.4 Partial Tree Kernel Remarks

In order to model a very fast PT kernel computation, we have defined the algorithm in Section 3.2 to evaluate it efficiently and we apply the selection of non-null node pairs (algorithm in Table 1) which can be also applied to the ST and SST kernels.

Our algorithm in Section 3.2 allows us to evaluate PT kernel in $O(\rho^3|N_{T_1}||N_{T_2}|)$, where ρ is the maximum branching factor of the two trees T_1 and T_2. It should be emphasized that the naïve approach for the evaluation of the PT function is exponential. Therefore, a fairer comparison of our approach should be carried out against the efficient algorithm proposed in [3] for the evaluation of relation extraction kernels (REKs). These are not convolution kernels and produce a much lower number of substructures than the PT kernel. The complexity of REK was $O(\rho^4)$ when applied to only two nodes. If we applied it to all the node pairs of two trees (as we do with the PT kernel), we would obtain a complexity of $O(\rho^4|N_{T_1}||N_{T_2}|)$ which is higher than the one produced by our method. Consequently, our solution is very efficient and produces larger substructure spaces.

Moreover, to further speed up the kernel computation, we apply Eq. 4 to node pairs for which the output is not null. A similar approach was suggested in [2,13] for the computation of the SST kernel. However, its impact on such kernel has not been clearly shown by an extensive experimentation and the effect on the new PT kernel should also be measured. For this purpose, in sections 5.1 and 5.2 we report the running time experiments for the evaluation of the SST and PT kernels and the training time that they generate in SVMs.

4 Semantic Applications of Parse Tree Kernels

Semantic Role Labeling (SRL) and Question Classification (QC) are two interesting natural language tasks in which the impact of tree kernels can be measured. The former relates to the classification of the predicate argument structures defined in PropBank [8] or FrameNet [9]. For example, Figure 5 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 Arg5. Additionally, adjuncts are labeled with several ArgM labels, e.g. ArgM-TMP or ArgM-LOC.

In FrameNet predicate/argument information is described by means of rich semantic structures called Frames. These are schematic representations of situations involving various participants, properties and roles in which a word may

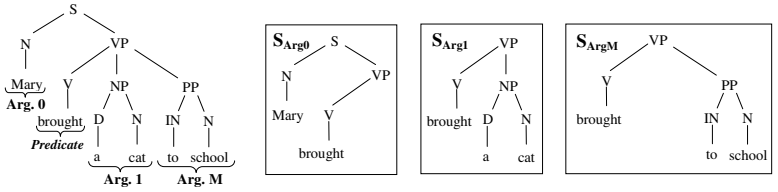

Fig. 5. Tree substructure space for predicate argument classification

typically be used. Frame elements or semantic roles are arguments of target words, i.e. the predicates. 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 semantic roles *Suspect* and *Authorities* are specific to this Frame.

The common approach to learn the classification of predicate arguments relates to the extraction of features from syntactic parse trees of the training sentences [15]. An alternative representation based on tree kernels selects the minimal partial tree that includes a predicate with only one of its arguments [6]. For example, in Figure 5, the semantic/syntactic substructures associated with the three arguments of the verb *to bring*, i.e. S_{Arg0}, S_{Arg1} and S_{ArgM}, are shown inside the three boxes. Note that such representation is quite intuitive.

Another interesting task is the classification of questions in the context of Question Answering (QA) systems. Detecting the type of a question, e.g. whether it asks for a person or for an organization, is critical to locate and extract the right answer from the available documents. The long tradition of QA in TREC has produced a large question set used in several researches. These are categorized according to different taxonomies of different *grains*. We consider the *coarse grained* classification scheme described in [16, 17]: Abbreviations, Descriptions (e.g. *definition* and *manner*), Entity (e.g. *animal*, *body* and *color*), Human (e.g. *group* and *individual*), Location (e.g. *city* and *country*) and Numeric (e.g. *code* and *date*).

The idea of using tree kernels for Question Classification is to encode questions by means of their whole syntactic parse tree. This is simpler than tailoring the subtree around the semantic information provided by PropBank or FrameNet for the SRL task. Additionally, we can easily experiment with other kind of parsing paradigms, e.g. the dependency parsing. A dependency tree of a sentence is a syntactic representation that denotes grammatical relations between words. For example, Figure 4 shows a dependency tree of the question "`What is an offer of direct stock purchase plan?`".

We note that (1) the father-children node relationship encodes the dependency between the head, e.g. *plan*, and its modifiers, e.g. *direct*, *stock* and *purchase*. In our approximation, we only consider the dependency structure by removing the link labels, i.e. we do not use either "of" between *offer* and *plan* or the other labels like "object" and "subject". (2) It is clear that the SST and ST

kernels cannot fully exploit the representational power of a dependency tree since from subtrees like `[plan [direct stock purchase]]`, they cannot generate substructures like `[plan [stock purchase]]` or `[plan [direct purchase]]`. In contrast, the PT kernel can generate all of these subsequences allowing SVMs to better generalize on dependency structures although the strong specialization of the SST kernel may be superior in some tasks. The experiments of Section 5 confirm our observations.

4.1 Related Work

In [2], the SST kernel was experimented with the Voted Perceptron for the parse-tree re-ranking task. The combination with the original PCFG model improved the syntactic parsing. In [18], an interesting algorithm that speeds up the average running time is presented. Such algorithm uses the explicit fragment space to compute the kernel between small trees. The results show an increase of the speed similar to the one produced by our methods. In [3], 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* were used. In [5] such kernels were slightly generalized by providing a matching function for the node pairs. The time complexity for their computation limited the experiments on a data set of just 200 news items. In [4], a feature description language was used to extract structural features from the syntactic shallow parse trees associated with named entities. The experiments on named entity categorization showed that too many irrelevant tree fragments may cause overfitting. In [6] the SST kernel was firstly proposed for semantic role classification. The combination between such kernel and a polynomial kernel of standard features improved the state-of-the-art. To complete such work, an analysis of different tree kernel spaces as carried out here was required. In [19], the computational complexity problem is addressed by considering only selected trees and the RankBoost algorithm.

5 The Experiments

In these experiments, we study tree kernels in terms of (a) average running time, (b) accuracy on the classification of predicate argument structures of PropBank (gold trees) and FrameNet (automatic trees) and (c) accuracy of QC on automatic question trees.

The experiments were carried out with the SVM-light-TK software available at `http://ai-nlp.info.uniroma2.it/moschitti/` which encodes ST, SST and PT kernels in the SVM-light software [21]. We adopted the default regularization parameter and we tried a few cost-factor values (i.e., $\{1, 3, 7, 10, 30, 100\}$) to adjust the rate between Precision and Recall on the development set. We modeled the multiclassifiers by training an SVM for each class according to the ONE-vs-ALL scheme and by selecting the class associated with the maximum 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. We measured the performance by using the F_1 measure[1] for the single arguments and the accuracy for the final multiclassifiers.

5.1 Kernel Running Time Experiments

To study the FTK running time, we extracted from the Penn Treebank 2 [10] several samples of 500 trees containing exactly n nodes. Each point of Figure 6 shows the average computation time[2] of the kernel function applied to the 250,000 pairs of trees of size n. It clearly appears that the FTK and FTK-PT (i.e. FTK applied to the PT kernel) average running time has linear behavior whereas, as expected, the algorithm (QTK) which does not use non-null pair selection shows a quadratic curve.

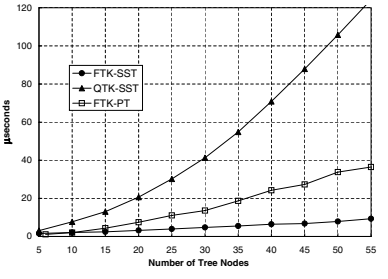

Fig. 6. Average time in μseconds for the QTK, FTK and FTK-PT evaluations

5.2 Experiments on ProbBank

The aim of these experiments is to measure the impact of kernels on the semantic role classification accuracy. We used PropBank (`www.cis.upenn.edu/~ace`) along with the gold standard parses of the Penn Treebank.

The corpus contains about 53,700 sentences and a fixed split between training and testing used in other researches, e.g. [22]. Sections from 02 to 21 are used for training, Section 23 for testing and Section 22 as development set for a total of 122,774 and 7,359 arguments in training and testing, respectively. We considered arguments from Arg0 to Arg5, ArgA and ArgM. This latter refers to all adjuncts collapsed together, e.g. *adverb*, *manner*, *negation*, *location* and so on (13 different types).

Figure 7 illustrates the learning curves associated with the above kernels for the SVM multiclassifiers. We note that: (a) the SST and linear kernels show the highest accuracy, (b) the richest kernel in terms of substructures, i.e. the one based on PTs, shows lower accuracy than the SST and linear kernels but higher than the ST kernel and (c) the results using all training data are comparable with those obtained in [22], i.e. 87.1% (role classification) but we should take into account the different treatment of ArgMs.

Regarding the convergence complexity, Figure 8 shows the learning time of SVMs using QTK, FTK and FTK-PT for the classification of one large argument (Arg0), according to different sizes of training data. With 70% of the data, FTK is about 10 times faster than QTK. With all the data FTK terminated in 6 hours

[1] F_1 assigns equal importance to Precision P and Recall R, i.e. $f_1 = \frac{2P \times R}{P+R}$.
[2] We run the experiments on a Pentium 4, 2GHz, with 1 Gb ram.

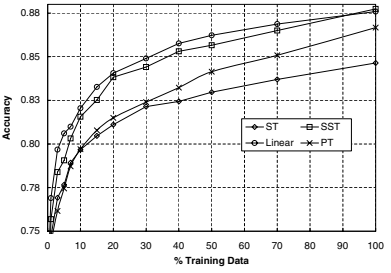

Fig. 7. Multiclassifier accuracy according to different training set percentages

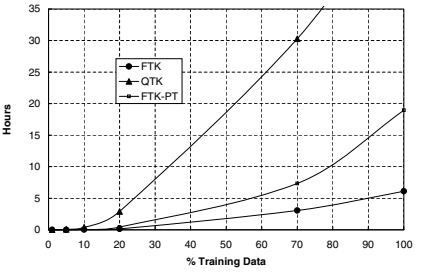

Fig. 8. Arg0 classifier learning time according to different training percentages and kernel algorithms

whereas QTK required more than 1 week. However, the real *complexity burden* relates to working in the dual space. To alleviate such problem interesting and effective approaches have been proposed [23, 24].

5.3 Classification Accuracy with Automatic Trees on FrameNet

As PropBank arguments are defined with respect to syntactic considerations, we should verify that the syntactic information provided by tree kernels is also effective to detect other forms of semantic structures. For this purpose, we experimented with our models and FrameNet data (`www.icsi.berkeley.edu/~framenet`) which is mainly produced based on semantic considerations. We extracted all 24,558 sentences from 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 [11] to generate automatic parse trees.

Table 2 reports the F_1 measure of some argument classifiers and the accuracy of the multiclassifier using all available training data for linear, ST, SST and PT kernels. We note that: (1) the F_1 of the single arguments across the different kernels follows a behavior similar to the accuracy of the global multiclassifier. (2) The high F_1 measures of tree kernels on automatic trees of FrameNet show that they are robust with respect to parsing errors.

5.4 Experiments on Question Classification

We used the data set available at `http://l2r.cs.uiuc.edu/~cogcomp/Data/QA/QC/`. This contains 5,500 training and 500 test questions from the TREC 10 QA competition. As we adopted the question taxonomy known as coarse grained introduced in Section 4, we can compare with literature results, e.g. [16, 17].

These experiments show that the PT kernel can be superior to the SST kernel when the source of syntactic information is expressed by dependency rather than constituent trees. For this purpose, we run the Stanford Parser (available

Table 2. Evaluation of kernels on 18 FrameNet semantic roles

Roles	Linear	ST	SST	PT
agent	89.8	86.9	87.8	86.2
theme	82.9	76.1	79.2	79.4
manner	70.8	79.9	82.0	81.7
source	86.5	85.6	87.7	86.6
Acc.	82.3	80.0	81.2	79.9

Table 3. Kernel evaluation on Question Classification according to different parsing approaches

Parsers	Const.		Depend.		BOW
Kernels	SST	PT	SST	PT	Linear
Acc.	88.2	87.2	82.1	90.4	87.3

at `http://www-nlp.stanford.edu/software/lex-parser.shtml`) to generate both parse types. Moreover, we used an SVM with the linear kernel over the bag-of-words (BOW) as baseline. Columns 2 and 3 of Table 3 show the accuracy of the SST and PT kernels over the constituent trees, columns 4 and 5 report the accuracy on the dependency data and Column 6 presents the BOW kernel accuracy.

We note that (1) the SST kernel is again superior to the PT kernel when using constituent trees. If we apply the SST kernel on the dependency trees the resulting accuracy is rather lower than the one of the PT kernel (82.1% vs. 90.4%). This is quite intuitive as the SST kernel cannot generate the features needed to represent all the possible n-ary relations derivable from father-children relations. Overall, the accuracy produced by the dependency trees is higher than the one attainable with the constituent trees. Nevertheless, when the SST kernel applied to the dependency structures is combined with BOW, the SVM accuracy reaches 90% as well [16].

6 Conclusions

In this paper, we have studied the impact of diverse tree kernels for the learning of syntactic/semantic linguistic structures. We used the subtree (ST) and the subset tree (SST) kernels defined in previous work, and we designed a novel general tree kernel, i.e. the partial tree (PT) kernel. Moreover, we improved the kernel usability by designing fast algorithms which process syntactic structures in linear average time.

The experiments with Support Vector Machines on the PropBank and FrameNet predicate argument structures show that richer kernel spaces are more accurate, e.g. SSTs and PTs produce higher accuracy than STs. However, if such structures are not relevant for the representation of the target linguistic objects improvement does not occur, e.g. PTs are not better than SSTs to describe constituent trees. On the contrary, as suggested by the experiments on Question Classification, the richer space provided by PTs produces a much higher accuracy than SSTs when applied to dependency trees. This because the SST kernel seems not adequate to process such data.

Finally, the running time experiments show that our fast tree kernels can be efficiently applied to hundreds of thousands of instances.

References

1. Jackendoff, R.: Semantic Structures, Current Studies in Linguistics series. Cambridge, Massachusetts: The MIT Press (1990)
2. Collins, M., Duffy, N.: New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In: ACL. (2002)
3. Zelenko, D., Aone, C., Richardella, A.: Kernel methods for relation extraction. JMLR (2003)
4. Cumby, C., Roth, D.: Kernel methods for relational learning. In: ICML. (2003)
5. Culotta, A., Sorensen, J.: Dependency tree kernels for relation extraction. In: Proceedings ACL, Barcelona, Spain (2004)
6. Moschitti, A.: A study on convolution kernels for shallow semantic parsing. In: proceedings of ACL, Barcelona, Spain (2004)
7. Vishwanathan, S., Smola, A.: Fast kernels on strings and trees. In: Proceedings of NIPS. (2002)
8. Kingsbury, P., Palmer, M.: From Treebank to PropBank. In: Proceedings of LREC, Las Palmas, Spain (2002)
9. Fillmore, C.J.: Frame semantics. In: Linguistics in the Morning Calm. (1982)
10. Marcus, M.P., Santorini, B., Marcinkiewicz, M.A.: Building a large annotated corpus of english: The Penn Treebank. CLJ. (1993).
11. Collins, M.: Three generative, lexicalized models for statistical parsing. In: Proceedings of the ACL, Somerset, New Jersey. (1997).
12. Klein, D., Manning, C.D.: Fast exact inference with a factored model for natural language parsing. In: NIPS. (2002)
13. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)
14. Haussler, D.: Convolution kernels on discrete structures. Technical report ucs-crl-99-10, University of California Santa Cruz (1999)
15. Gildea, D., Jurasfky, D.: Automatic labeling of semantic roles. CLJ. (2002)
16. Zhang, D., Lee, W.S.: Question classification using support vector machines. In: Proceedings of SIGIR. (2003).
17. Li, X., Roth, D.: Learning question classifiers: The role of semantic information. JNLE. (2005).
18. Kazama, J., Torisawa, K.: Speeding up training with tree kernels for node relation labeling. In: Proceedings of EMNLP, Toronto, Canada (2005)
19. Kudo, T., Suzuki, J., Isozaki, H.: Boosting-based parse reranking with subtree features. In: Proceedings ACL'05, (2005).
20. Carreras, X., Màrquez, L.: Introduction to the CoNLL-2005 shared task: Semantic role labeling. In: Proceedings of CoNLL-2005. (2005).
21. Joachims, T.: Making large-scale SVM learning practical. In Schölkopf, B., Burges, C., Smola, A., eds.: Advances in Kernel Methods - Support Vector Learning. (1999)
22. Pradhan, S., Hacioglu, K., Krugler, V., Ward, W., Martin, J.H., Jurafsky, D.: Support vector learning for semantic argument classification. MLJ. (2005).
23. Kudo, T., Matsumoto, Y.: Fast methods for kernel-based text analysis. In: Proceedings of ACL. (2003).
24. Suzuki, J., Isozaki, H., Maeda, E.: Convolution kernels with feature selection for natural language processing tasks. In: Proceedings of ACL, Spain (2004).