Tree Kernel Engineering in Semantic Role Labeling Systems

Alessandro Moschitti and **Daniele Pighin** and **Roberto Basili**
University of Rome, Tor Vergata
{moschitti,basili}@info.uniroma2.it
daniele.pighin@gmail.com

Abstract

Recent work on the design of automatic systems for semantic role labeling has shown that feature engineering is a complex task from a modeling and implementation point of view. Tree kernels alleviate such complexity as kernel functions generate features automatically and require less software development for data extraction.

In this paper, we study several tree kernel approaches for both boundary detection and argument classification. The comparative experiments on Support Vector Machines with such kernels on the CoNLL 2005 dataset show that very simple tree manipulations trigger automatic feature engineering that highly improves accuracy and efficiency in both phases. Moreover, the use of different classifiers for internal and pre-terminal nodes maintains the same accuracy and highly improves efficiency.

1 Introduction

A lot of attention has been recently devoted to the design of systems for the automatic labeling of semantic roles (SRL) as defined in two important projects: FrameNet (Johnson and Fillmore, 2000), inspired by Frame Semantics, and PropBank (Kingsbury and Palmer, 2002) based on Levin's verb classes. In general, given a sentence in natural language, the annotation of a predicate's semantic roles requires (1) the detection of the target word that embodies the predicate and (2) the detection and classification of the word sequences constituting the predicate's arguments. In particular, step (2) can be divided into two different phases: (a) boundary detection, in which the words of the sequence are detected and (b) argument classification, in which the type of the argument is selected.

Most machine learning models adopted for the SRL task have shown that (shallow or deep) syntactic information is necessary to achieve a good labeling accuracy. This research brings a wide empirical evidence in favor of the linking theories between semantics and syntax, e.g. (Jackendoff, 1990). However, as no theory provides a sound and complete treatment of such issue, the choice and design of syntactic features for the automatic learning of semantic structures requires remarkable research efforts and intuition.

For example, the earlier studies concerning linguistic features suitable for semantic role labeling were carried out in (Gildea and Jurasfky, 2002). Since then, researchers have proposed diverse syntactic feature sets that only slightly enhance the previous ones, e.g. (Xue and Palmer, 2004) or (Carreras and Màrquez, 2005). A careful analysis of such features reveals that most of them are syntactic tree fragments of training sentences, thus a natural way to represent them is the adoption of tree kernels as described in (Moschitti, 2004). The idea is to associate with each argument the minimal subtree that includes the target predicate with one of its arguments, and to use a tree kernel function to evaluate the number of common substructures between two such trees. Such approach is in line with current research on the use of tree kernels for natural language learning, e.g. syntactic parsing re-ranking (Collins and Duffy, 2002), relation extraction (Zelenko et al., 2003) and named entity recognition (Cumby and Roth, 2003; Culotta and Sorensen, 2004).

Regarding the use of tree kernels for SRL, in (Moschitti, 2004) two main drawbacks have been

pointed out:

- Highly accurate boundary detection cannot be carried out by a tree kernel model since correct and incorrect arguments may share a large portion of the encoding trees, i.e. they may share many substructures.
- Manually derived features (extended with a polynomial kernel) have been shown to be superior to tree kernel approaches.

Nevertheless, we believe that modeling a completely kernelized SRL system is useful for the following reasons:

- We can implement it very quickly as the feature extractor module only requires the writing of the subtree extraction procedure. Traditional SRL systems are, in contrast, based on the extraction of more than thirty features (Pradhan et al., 2005), which require the writing of at least thirty different procedures.
- Combining it with a traditional attribute-value SRL system allows us to obtain a more accurate system. Usually the combination of two traditional systems (based on the same machine learning model) does not result in an improvement as their features are more or less equivalent as shown in (Carreras and Màrquez, 2005).
- The study of the effective structural features can inspire the design of novel linear features which can be used with a more efficient model (i.e. linear SVMs).

In this paper, we carry out tree kernel engineering (Moschitti et al., 2005) to increase both accuracy and speed of the boundary detection and argument classification phases. The engineering approach relates to marking the nodes of the encoding subtrees in order to generate substructures more strictly correlated with a particular argument, boundary or predicate. For example, marking the node that exactly covers the target argument helps tree kernels to generate different substructures for correct and incorrect argument boundaries.

The other technique that we applied to engineer different kernels is the subdivision of internal and pre-terminal nodes. We show that designing different classifiers for these two different node types slightly increases the accuracy and remarkably decreases the learning and classification time.

An extensive experimentation of our tree kernels with Support Vector Machines on the CoNLL 2005 data set provides interesting insights on the design of performant SRL systems entirely based on tree kernels.

In the remainder of this paper, Section 2 introduces basic notions on SRL systems and tree kernels. Section 3 illustrates our new kernels for both boundary and classification tasks. Section 4 shows the experiments of SVMs with the above tree kernel based classifiers.

2 Preliminary Concepts

In this section we briefly define the SRL model that we intend to design and the kernel function that we use to evaluate the similarity between subtrees.

2.1 Basic SRL approach

The SRL approach that we adopt is based on the deep syntactic parse (Charniak, 2000) of the sentence that we intend to annotate semantically. The standard algorithm is to classify the tree node pair $\langle p, a\rangle$, where p and a are the nodes that exactly cover the target predicate and a potential argument, respectively. If $\langle p, a\rangle$ is labeled with an argument, then the terminal nodes dominated by a will be considered as the words constituting such argument. The number of pairs for each sentence can be hundreds, thus, if we consider training corpora of thousands of sentences, we have to deal with millions of training instances.

The usual solution to limit such complexity is to divide the labeling task in two subtasks:

- Boundary detection, in which a single classifier is trained on many instances to detect if a node is an argument or not, i.e. if the sequence of words dominated by the target node constitutes a correct boundary.
- Argument classification: only the set of nodes corresponding to correct boundaries are considered. These can be used to train a multiclassifier that, for such nodes, only decides the type of the argument. For example, we can train n classifiers in the style One-vs-All. At classification time, for each argument node, we can select the argument type associated with the maximum among the n scores provided by the single classifiers.

We adopt this solution as it enables us to use only one computationally expensive classifier, i.e. the boundary detection one. This, as well as the argument classifiers, requires a feature representation of the predicate-argument pair. Such features are mainly extracted from the parse trees of the target sentence, e.g. *Phrase Type*, *Predicate Word*, *Head Word*, *Governing Category*, *Position* and *Voice* proposed in (Gildea and Jurasfky, 2002).

As most of the features proposed in literature are subsumed by tree fragments, tree-kernel functions are a natural way to produce them automatically.

2.2 Tree kernel functions

Tree-kernel functions simply evaluate the number of substructures shared between two trees T_1 and T_2. Such functions can be seen as a scalar product in the huge vector space constituted by all possible substructures of the training set. Thus, kernel functions implicitly define a large feature space.

Formally, given a tree fragment space $\{f_1, f_2, ..\} = \mathcal{F}$, we can define an indicator function $I_i(n)$, which is equal to 1 if the target f_i is rooted at node n and equal to 0 otherwise. Therefore, a tree-kernel function K over T_1 and T_2 can be defined as $K(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2)$, 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 at nodes n_1 and n_2 and, according to (Collins and Duffy, 2002), it can be computed 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-terminal symbols) then $\Delta(n_1, n_2) = \lambda$;
3. if the productions at n_1 and n_2 are the same, and n_1 and n_2 are not pre-terminal then $\Delta(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + \Delta(c_{n_1}^j, c_{n_2}^j))$.

where λ is the decay factor to scale down the impact of large structures, $nc(n_1)$ is the number of the children of n_1 and c_n^j is the j-th child of the node n. Note that, as the productions are the same, $nc(n_1) = nc(n_2)$. Additionally, to map similarity scores in the [0,1] range, we applied a normalization in the kernel space, i.e. $K'(T_1, T_2) = \frac{K(T_1,T_2)}{\sqrt{K(T_1,T_1) \times K(T_2,T_2)}}$.

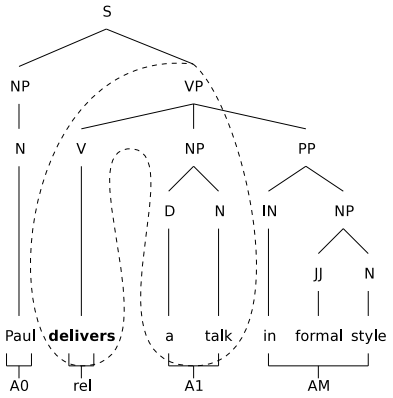

Figure 1: The PAF subtree associated with A1.

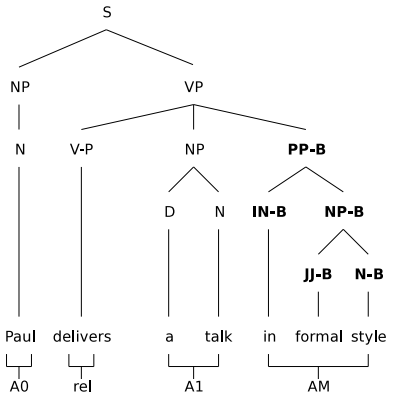

Figure 2: Example of CMST.

Once a kernel function is defined, we need to characterize the predicate-argument pair with a subtree. This allows kernel machines to generate a large number of syntactic features related to such pair. The approach proposed in (Moschitti, 2004) selects the minimal subtree that includes a predicate with its argument. We follow such approach by studying and proposing novel, interesting solutions.

3 Novel Kernels for SRL

The basic structure used to characterize the predicate argument relation is the smallest subtree that includes a predicate with one of its argument. For example, in Figure 1, the dashed line encloses a predicate argument feature (PAF) over the parse tree of the sentence: "Paul delivers a talk in formal style". This PAF is a subtree that characterizes the predicate *to deliver* with its argument *a talk*. In this section, we improve PAFs, propose different kernels for internal and pre-terminal nodes and new kernels based on complete predicate ar-

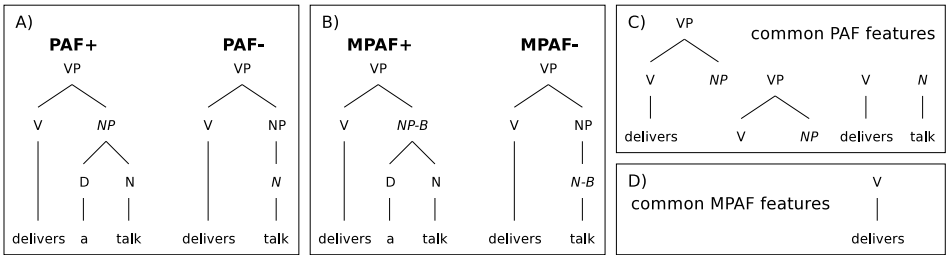

Figure 3: Differences between PAF (a) and MPAF (b) structures.

gument structures.

3.1 Improving PAF

PAFs have shown to be very effective for argument classification but not for boundary detection. The reason is that two nodes that encode correct and incorrect boundaries may generate very similar PAFs. For example, Figure 3.A shows two PAFs corresponding to a correct (PAF+) and an incorrect (PAF-) choice of the boundary for A1: PAF+ from the NP vs. PAF- from the N nodes. The number of their common substructures is high, i.e. the four subtrees shown in Frame C. This prevents the algorithm from making different decisions for such cases.

To solve this problem, we specify which is the node that exactly covers the argument (also called argument node) by simply marking it with the label B denoting the boundary property. Figure 3.B shows the two new marked PAFs (MPAFs). The features generated from the two subtrees are now very different so that there is only one substructure in common (see Frame D). Note that, each markup strategy impacts on the output of a kernel function in terms of the number of structures common to two trees. The same output can be obtained using unmarked trees and redefining consistently the kernel function, e.g. the algorithm described in Section 2.2.

An alternative way to partially solve the structure overlapping problem is the use of two different classifiers, one for the internal nodes and one for the pre-terminal nodes, and combining their decisions. In this way, the negative example of Figure 3 would not be used to train the same classifier that uses PAF+. Of course, similar structures can both be rooted on internal nodes, therefore they can belong to the training data of the same classifier. However, the use of different classifiers is motivated also by the fact that many argument types can be found mostly in pre-terminal nodes, e.g. modifier or negation arguments, and do not necessitate training data extracted from internal nodes. Consequently, it is more convenient (at least from a computational point of view) to use two different boundary classifiers, hereinafter referred to as combined classifier.

3.2 Kernels on complete predicate argument structures

The type of a target argument strongly depends on the type and number of the predicate's arguments[1] (Punyakanok et al., 2005; Toutanova et al., 2005). Consequently, to correctly label an argument, we should extract features from the complete predicate argument structure it belongs to. In contrast, PAFs completely neglect the information (i.e. the tree portions) related to non-target arguments.

One way to use this further information with tree kernels is to use the minimum subtree that spans all the predicate's arguments. The whole parse tree in Figure 1 is an example of such Minimum Spanning Tree (MST) as it includes all and only the argument structures of the predicate "to deliver". However, MSTs pose some problems:

- We cannot use them for the boundary detection task since we do not know the predicate's argument structure yet. However, we can derive the MST (its approximation) from the nodes selected by a boundary classifier, i.e. the nodes that correspond to potential arguments. Such approximated MSTs can be easily used in the argument type classification phase. They can also be used to re-rank the most probable m sequences of arguments for both labeling phases.

- Obviously, an MST is the same for all the arguments it includes, thus we need a way to differentiate it for each target argument.

[1] This is true at least for core arguments.

Again, we can mark the node that exactly covers the target argument as shown in the previous section. We refer to this subtree as marked MST (MMST). However, for large arguments (i.e. spread on a large part of the sentence tree) the substructures' likelihood of being part of other arguments is quite high.

To address this latter problem, we can mark all nodes that descend from the target argument node. Figure 2 shows a MST in which the subtree associated with the target argument (AM) has the nodes marked. We refer to this structure as a completely marked MST (CMST). CMSTs may be seen as PAFs enriched with new information coming from the other arguments (i.e. the non-marked subtrees). Note that if we consider only the PAF subtree from a CMST we obtain a differently marked subtree which we refer to as CPAF.

In the next section we study the impact of the proposed kernels on the boundary detection and argument classification performance.

4 Experiments

In these experiments we evaluate the impact of our proposed kernels in terms of accuracy and efficiency. The accuracy improvement confirms that the node marking approach enables the automatic engineering of effective SRL features. The efficiency improvement depends on (a) the less training data used when applying two distinct type classifiers for internal and pre-terminal nodes and (b) a more adequate feature space which allows SVMs to converge faster to a model containing a smaller number of support vectors, i.e. faster training and classification.

4.1 Experimental set up

The empirical evaluations were carried out within the setting defined in the CoNLL-2005 Shared Task (Carreras and Màrquez, 2005). We used as a target dataset the PropBank corpus available at `www.cis.upenn.edu/~ace`, along with the Penn TreeBank 2 for the gold trees (`www.cis.upenn.edu/~treebank`) (Marcus et al., 1993), which includes about 53,700 sentences. Since the aim of this study was to design a real SRL system we adopted the Charniak parse trees from the CoNLL 2005 Shared Task data (available at `www.lsi.upc.edu/~srlconll/`).

We used Section 02, 03 and 24 from the Penn TreeBank in most of the experiments. Their characteristics are shown in Table 1. *Pos* and *Neg* indicate the number of nodes corresponding or not to a correct argument boundary. Rows 3 and 4 report such number for the internal and pre-terminal nodes separately. We note that the latter are much fewer than the former; this results in a very fast pre-terminal classifier.

As the automatic parse trees contain errors, some arguments cannot be associated with any covering node. This prevents us to extract a tree representation for them. Consequently, we do not consider them in our evaluation. In sections 2, 3 and 24 there are 454, 347 and 731 such cases, respectively.

The experiments were carried out with the SVM-light-TK software available at `http://ai-nlp.info.uniroma2.it/moschitti/` which encodes fast tree kernel evaluation (Moschitti, 2006) in the SVM-light software (Joachims, 1999). We used a regularization parameter (option `-c`) equal to 1 and $\lambda = 0.4$ (see (Moschitti, 2004)).

4.2 Boundary Detection Results

In these experiments, we used Section 02 for training and Section 24 for testing. The results using the PAF and the MPAF based kernels are reported in Table 2 in rows 2 and 3, respectively. Columns 3 and 4 show the CPU testing time (in seconds) and the F_1 of the monolithic boundary classifier. The next 3 columns show the CPU time for the internal (Int) and pre-terminal (Pre) node classifiers, as well as their total (All). The F_1 measures are reported in the 3 rightmost columns. In particular, the third column refers to the F_1 of the combined classifier. This has been computed by summing correct, incorrect and not retrieved examples of the two distinct classifiers.

We note that: first, the monolithic classifier applied to MPAF improves both the efficiency, i.e. about 3,131 seconds vs. 5,179, of PAF and the F_1, i.e. 82.07 vs. 75.24. This suggests that marking the argument node simplifies the generalization process.

Second, by dividing the boundary classification in two tasks, internal and pre-terminal nodes, we furthermore improve the classification time for both PAF and MPAF kernels, i.e. 5,179 vs. 1,851 (PAF) and 3,131 vs. 1,471 (MPAF). The separated classifiers are much faster, especially the pre-terminal one (about 61 seconds to classify 81,075 nodes).

	Section 2			Section 3			Section 24		
Nodes	pos	neg	tot	pos	neg	tot	pos	neg	tot
Internal	11,847	71,126	82,973	6,403	53,591	59,994	7,525	50,123	57,648
Pre-terminal	894	114,052	114,946	620	86,232	86,852	709	80,366	81,075
Both	12,741	185,178	197,919	7,023	139,823	146,846	8,234	130,489	138,723

Table 1: Tree nodes of the sentences from sections 2, 3 and 24 of the PropBank. pos and neg are the nodes that exactly cover arguments and all the other nodes, respectively.

Tagging strategy	Monolithic		Combined					
	CPU_{time}	F1	CPU_{time}			F1		
			Int	Pre	All	Int	Pre	All
PAF	5,179.18	75.24	1,794.92	56.72	1,851.64	79.93	79.39	79.89
MPAF	3,131.56	82.07	1,410.10	60.99	1,471.09	82.20	79.14	81.96

Table 2: F1 comparison between PAF and MPAF based kernels using different classification strategies. Int, Pre and ALL are the internal, pre-terminal and combined classifiers. The CPU time refers to the classification time in seconds of all Section 24.

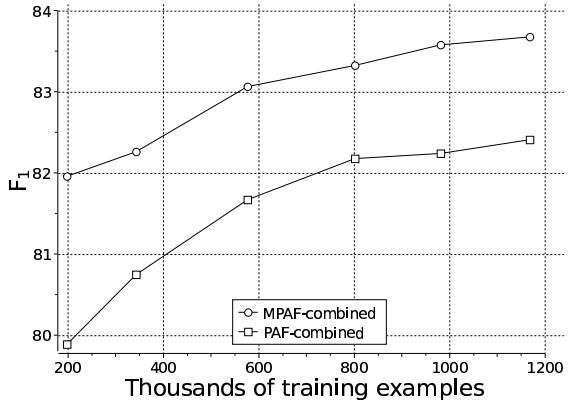

Figure 4: Learning curve comparison between the PAF and MPAF F1 measures using the combined classifier.

Third, the combined classifier approach seems quite feasible as its F_1 is almost equal to the monolithic one (81.96 vs. 82.07) in case of MPAF and even superior when using PAF (79.89 vs. 75.34). This result confirms the observation given in Section 3.1 about the importance of reducing the number of substructures common to PAFs associated with correct and incorrect boundaries.

Finally, we trained the combined boundary classifiers with sets of increasing size to derive the learning curves of the PAF and MPAF models. To have more significant results, we increased the training set by using also sections from 03 to 07. Figure 4 shows that the MPAF approach is constantly over the PAF. Consider also that the marking strategy has a lesser impact on the combined classifier.

4.3 Argument Classification Results

In these experiments we tested different kernels on the argument classification task. As some arguments have a very small number of training instances in a single section, we also used Section 03 for training and we continued to test on only Section 24.

The results of the multiclassifiers on 59 argument types[2] (e.g. constituted by 59 binary classifiers in the monolithic approach) are reported in Table 3. The rows from 3 to 5 report the accuracy when using the PAF, MPAF and CPAF whereas the rows from 6 to 8 show the accuracy for the complete argument structure approaches, i.e. MST, MMST and CMST.

More in detail, Column 2 shows the accuracy of the monolithic multi-argument classifiers whereas Columns 3, 4 and 5 report the accuracy of the internal, pre-terminal and combined multi-argument classifiers, respectively.

We note that:

First, the two classifier approach does not improve the monolithic approach accuracy. Indeed, the subtrees describing different argument types are quite different and this property holds also for the pre-terminal nodes. However, we still measured a remarkable improvement in efficiency.

Second, MPAF is the best kernel. This confirms the outcome on boundary detection experiments. The fact that it is more accurate than CPAF reveals that we need to distin-

[2] 7 for the core arguments (A0...AA), 13 for the adjunct arguments (AM-*), 19 for the argument references (R-*) and 20 for the continuations (C-*).

Tagging strategy	Monolithic	Combined		
		Internal nodes	Pre-terminals	Overall
PAF	75.06	74.16	85.61	75.15
MPAF	77.17	76.25	85.76	77.07
CPAF	76.79	75.68	85.76	76.54
MST	34.80	36.52	78.14	40.10
MMST	72.55	71.59	86.32	72.86
CMST	73.21	71.93	86.32	73.17

Table 3: Accuracy produced by different tree kernels on argument classification. We trained on sections 02 and 03 and tested on Section 24.

guish the argument node from the other nodes. To explain this, suppose that two argument nodes, $\texttt{NP}_1$ and $\texttt{NP}_2$, dominate the following structures: `[`$\texttt{NP}_1$ `[NP [DT NN]][PP]]` and `[`$\texttt{NP}_2$ `[DT NN]]`. If we mark only the argument node we obtain `[NP-B [NP [DT NN]][PP]]` and `[NP-B [DT NN]]` which have no structure in common. In contrast, if we mark them completely, i.e. `[NP-B [NP-B [DT-B NN-B]][PP-B]]` and `[NP-B [DT-B NN-B]]`, they will share the subtree `[NP-B [DT-B NN-B]]`. Thus, although it may seem counterintuitive, by marking only one node, we obtain more specific substructures. Of course, if we use different labels for the argument nodes and their descendants, we obtain the same specialization effect.

Finally, if we do not mark the target argument in the MSTs, we obtain a very low result (i.e. 40.10%) as expected. When we mark the covering node or the complete argument subtree we obtain an acceptable accuracy. Unfortunately, such accuracy is lower than the one produced by PAFs, e.g. 73.17% vs. 77.07%, thus it may seem that the additional information provided by the whole argument structure is not effective. A more careful analysis can be carried out by considering a CMST as composed by a PAF and the rest of the argument structure. We observe that some pieces of information provided by a PAF are not derivable by a CMST (or a MMST). For example, Figure 1 shows that the PAF contains the subtree `[VP [V NP]]` while the associated CMST (see Figure 2) contains `[VP [V NP PP]]`. The latter structure is larger and more sparse and consequently, the learning machine applied to CMSTs (or MMSTs) performs a more difficult generalization task. This problem is emphasized by our use of the adjuncts in the design of MSTs. As adjuncts tend to be the same for many predicates they do not provide a very discriminative information.

5 Discussions and Conclusions

The design of automatic systems for the labeling of semantic roles requires the solution of complex problems. Among others, feature engineering is made difficult by the structural nature of the data, i.e. features should represent information contained in automatic parse trees. This raises two problems: (1) the modeling of effective features, partially solved in the literature work and (2) the implementation of the software for the extraction of a large number of such features.

A system completely based on tree kernels alleviate both problems as (1) kernel functions automatically generate features and (2) only a procedure for subtree extraction is needed. Although some of the manual designed features seem to be superior to those derived with tree kernels, their combination seems still worth applying.

In this paper, we have improved tree kernels by studying different strategies: MPAF and the combined classifier (for internal and pre-terminal nodes) highly improve efficiency and accuracy in both the boundary detection and argument classification tasks. In particular, MPAF improves the old PAF-based tree kernel of about 8 absolute percent points in the boundary classification task, and when used along the combined classifier approach the speed of the model increases of 3.5 times. In case of argument classification the improvement is less evident but still consistent, about 2%.

We have also studied tree representations based on complete argument structures (MSTs). Our preliminary results seem to suggest that additional information extracted from other arguments is not effective. However, such findings are affected by two main problems: (1) We used adjuncts in the tree representation. They are likely to add more noise than useful information for the recognition of the argument type. (2) The traditional PAF contains subtrees that cannot be derived by the

MMSTs, thus we should combine these structures rather than substituting one with the other.

In the future, we plan to extend this study as follows:

First, our results are computed individually for boundary and classification tasks. Moreover, in our experiments, we removed arguments whose PAF or MST could not be extracted due to errors in parse trees. Thus, we provided only indicative accuracy to compare the different tree kernels. A final evaluation of the most promising structures using the CoNLL 2005 evaluator should be carried out to obtain a sound evaluation.

Second, as PAFs and MSTs should be combined to generate more information, we are going to carry out a set of experiments that combine different kernels associated with different subtrees. Moreover, as shown in (Basili and Moschitti, 2005; Moschitti, 2006), there are other tree kernel functions that generate different fragment types. The combination of such functions with the marking strategies may provide more general and effective kernels.

Third, once the final set of the most promising kernels is established, we would like to use all the available CoNLL 2005 data. This would allow us to study the potentiality of our approach by exactly comparing with literature work.

Next, our fast tree kernel function along with the combined classification approach and the improved tree representation make the learning and classification much faster so that the overall running time is comparable with polynomial kernels. However, when these latter are used with SVMs the running time is prohibitive when very large datasets (e.g. millions of instances) are targeted. Exploiting tree kernel derived features in a more efficient way is thus an interesting line of future research.

Finally, as CoNLL 2005 has shown that the most important contribution relates on re-ranking predicate argument structures based on one single tree (Toutanova et al., 2005) or several trees (Punyakanok et al., 2005), we would like to use tree kernels for the re-ranking task.

Acknowledgments

This research is partially supported by the European project, PrestoSpace (FP6-IST-507336).

References

Roberto Basili and Alessandro Moschitti. 2005. *Automatic Text Categorization: from Information Retrieval to Support Vector Learning*. Aracne Press, Rome, Italy.

Xavier Carreras and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of CoNLL'05*.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the NACL'00*.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL'02*.

Aron Culotta and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of ACL'04*.

Chad Cumby and Dan Roth. 2003. Kernel methods for relational learning. In *Proceedings of ICML'03*.

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

R. Jackendoff. 1990. *Semantic Structures, Current Studies in Linguistics series*. Cambridge, Massachusetts: The MIT Press.

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*.

Christopher R. Johnson and Charles J. Fillmore. 2000. The framenet tagset for frame-semantic and syntactic coding of predicate-argument structure. In *In the Proceedings ANLP-NAACL*.

Paul Kingsbury and Martha Palmer. 2002. From Treebank to PropBank. In *Proceedings of LREC'02*.

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 of ACL'04*, Barcelona, Spain.

Alessandro Moschitti, Bonaventura Coppola, Daniele Pighin, and Roberto Basili. 2005. Engineering of syntactic features for shallow semantic parsing. In *of the ACL05 Workshop on Feature Engineering for Machine Learning in Natural Language Processing*, USA.

Alessandro Moschitti. 2006. Making tree kernels practical for natural language learning. In *Proceedings of EACL'06*, Trento, Italy.

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*.

V. Punyakanok, D. Roth, and W. Yih. 2005. The necessity of syntactic parsing for semantic role labeling. In *Proceedings of IJCAI'05*.

Kristina Toutanova, Aria Haghighi, and Christopher Manning. 2005. Joint learning improves semantic role labeling. In *Proceedings of ACL'05*.

Nianwen Xue and Martha Palmer. 2004. Calibrating features for semantic role labeling. In *Proceedings of EMNLP 2004*.

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