Semantic Role Labeling via Tree Kernel Joint Inference

Alessandro Moschitti, Daniele Pighin and **Roberto Basili**
Department of Computer Science
University of Rome "Tor Vergata"
00133 Rome, Italy
{moschitti,basili}@info.uniroma2.it
daniele.pighin@gmail.com

Abstract

Recent work on Semantic Role Labeling (SRL) has shown that to achieve high accuracy a joint inference on the whole predicate argument structure should be applied. In this paper, we used syntactic subtrees that span potential argument structures of the target predicate in tree kernel functions. This allows Support Vector Machines to discern between correct and incorrect predicate structures and to re-rank them based on the joint probability of their arguments. Experiments on the PropBank data show that both classification and re-ranking based on tree kernels can improve SRL systems.

1 Introduction

Recent work on Semantic Role Labeling (SRL) (Carreras and Màrquez, 2005) has shown that to achieve high labeling accuracy a joint inference on the whole predicate argument structure should be applied. For this purpose, we need to extract features from the sentence's syntactic parse tree that encodes the target semantic structure. This task is rather complex since we do not exactly know which are the syntactic clues that capture the relation between the predicate and its arguments. For example, to detect the interesting context, the modeling of syntax/semantics-based features should take into account linguistic aspects like ancestor nodes or semantic dependencies (Toutanova et al., 2004).

A viable approach to generate a large number of features has been proposed in (Collins and Duffy, 2002), where convolution kernels were used to implicitly define a tree substructure space. The selection of the relevant structural features was left to the Voted Perceptron learning algorithm. Such successful experimentation shows that tree kernels are very promising for automatic feature engineering, especially when the available knowledge about the phenomenon is limited.

In a similar way, we can model SRL systems with tree kernels to generate large feature spaces. More in detail, most SRL systems split the labeling process into two different steps: Boundary Detection (i.e. to determine the text boundaries of predicate arguments) and Role Classification (i.e. labeling such arguments with a semantic role, e.g. Arg0 or Arg1 as defined in (Kingsbury and Palmer, 2002)). The former relates to the detection of syntactic parse tree nodes associated with constituents that correspond to arguments, whereas the latter considers the boundary nodes for the assignment of the suitable label. Both steps require the design and extraction of features from parse trees. As capturing the tightly interdependent relations among a predicate and its arguments is a complex task, we can apply tree kernels on the subtrees that *span* the whole predicate argument structure to generate the feature space of all the possible subtrees.

In this paper, we apply the traditional boundary (TBC) and role (TRC) classifiers (Pradhan et al., 2005a), which are based on binary predicate/argument relations, to label all parse tree nodes corresponding to potential arguments. Then, we ex-

Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-X),
pages 61–68, New York City, June 2006.

tract the subtrees which span the predicate-argument dependencies of such arguments, i.e. Argument Spanning Trees (ASTs). These are used in a tree kernel function to generate all possible substructures that encode n-ary argument relations, i.e. we carry out an automatic feature engineering process.

To validate our approach, we experimented with our model and Support Vector Machines for the classification of valid and invalid ASTs. The results show that this classification problem can be learned with high accuracy. Moreover, we modeled SRL as a re-ranking task in line with (Toutanova et al., 2005). The large number of complex features provided by tree kernels for structured learning allows SVMs to reach the state-of-the-art accuracy.

The paper is organized as follows: Section 2 introduces the Semantic Role Labeling based on SVMs and the tree kernel spaces; Section 3 formally defines the ASTs and the algorithm for their classification and re-ranking; Section 4 shows the comparative results between our approach and the traditional one; Section 5 presents the related work; and finally, Section 6 summarizes the conclusions.

2 Semantic Role Labeling

In the last years, several machine learning approaches have been developed for automatic role labeling, e.g. (Gildea and Jurasfky, 2002; Pradhan et al., 2005a). Their common characteristic is the adoption of attribute-value representations for predicate-argument structures. Accordingly, our basic system is similar to the one proposed in (Pradhan et al., 2005a) and it is hereby described.

We use a boundary detection classifier (for any role type) to derive the words compounding an argument and a multiclassifier to assign the roles (e.g. `Arg0` or `ArgM`) described in PropBank (Kingsbury and Palmer, 2002)). To prepare the training data for both classifiers, we used the following algorithm:

1. Given a sentence from the *training-set*, generate a full syntactic parse tree;
2. Let $\mathcal{P}$ and $\mathcal{A}$ be respectively the set of predicates and the set of parse-tree nodes (i.e. the potential arguments);
3. For each pair $\langle p, a\rangle \in \mathcal{P} \times \mathcal{A}$:

   - extract the feature representation set, $F_{p,a}$;
   - if the subtree rooted in a covers exactly the words of one argument of p, put $F_{p,a}$ in the T^+ set (positive examples), otherwise put it in the T^- set (negative examples).

The outputs of the above algorithm are the T^+ and T^- sets. These sets can be directly used to train a boundary classifier (e.g. an SVM). Regarding the argument type classifier, a binary labeler for a role r (e.g. an SVM) can be trained on the T^+_r, i.e. its positive examples and T^-_r, i.e. its negative examples, where $T^+ = T^+_r \cup T^-_r$, according to the ONE-vs-ALL scheme. The binary classifiers are then used to build a general role multiclassifier by simply selecting the argument associated with the maximum among the SVM scores.

Regarding the design of features for predicate-argument pairs, we can use the attribute-values defined in (Gildea and Jurasfky, 2002) or tree structures (Moschitti, 2004). Although we focus on the latter approach, a short description of the former is still relevant as they are used by TBC and TRC. They include the *Phrase Type*, *Predicate Word*, *Head Word*, *Governing Category*, *Position* and *Voice* features. For example, the *Phrase Type* indicates the syntactic type of the phrase labeled as a predicate argument and the *Parse Tree Path* contains the path in the parse tree between the predicate and the argument phrase, expressed as a sequence of nonterminal labels linked by direction (up or down) symbols, e.g. `V` ↑ `VP` ↓ `NP`.

A viable alternative to manual design of syntactic features is the use of tree-kernel functions. These implicitly define a feature space based on all possible tree substructures. Given two trees T_1 and T_2, instead of representing them with the whole fragment space, we can apply the kernel function to evaluate the number of common fragments.

Formally, given a tree fragment space $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$, the indicator function $I_i(n)$ is equal to 1 if the target f_i is rooted at node n and equal to 0 otherwise. A tree-kernel function over t_1 and t_2 is $K_t(t_1, t_2) = \sum_{n_1 \in N_{t_1}} \sum_{n_2 \in N_{t_2}} \Delta(n_1, n_2)$, where N_{t_1} and N_{t_2} are the sets of the t_1's and t_2's nodes, respectively. In turn $\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} \lambda^{l(f_i)} I_i(n_1) I_i(n_2)$, where $0 \leq \lambda \leq 1$ and $l(f_i)$ is the height of the subtree f_i. Thus $\lambda^{l(f_i)}$ assigns a lower weight to larger frag-

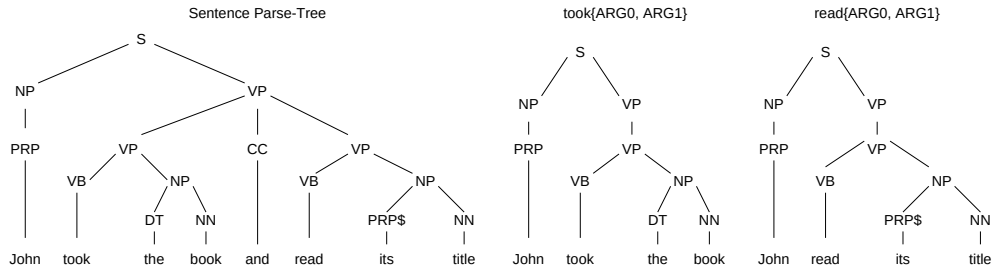

Figure 1: A sentence parse tree with two argument spanning trees (ASTs)

ments. When $\lambda = 1$, Δ is equal to the number of common fragments rooted at nodes n_1 and n_2. As described in (Collins and Duffy, 2002), Δ can be computed in $O(|N_{t_1}| \times |N_{t_2}|)$.

3 Tree kernel-based classification of Predicate Argument Structures

Traditional semantic role labeling systems extract features from pairs of nodes corresponding to a predicate and one of its argument, respectively. Thus, they focus on only binary relations to make classification decisions. This information is poorer than the one expressed by the whole predicate argument structure. As an alternative we can select the set of potential arguments (potential argument nodes) of a predicate and extract features from them. The number of the candidate argument sets is exponential, thus we should consider only those corresponding to the most probable correct argument structures.

The usual approach (Toutanova et al., 2005) uses a traditional boundary classifier (TBC) to select the set of potential argument nodes. Such set can be associated with a subtree which in turn can be classified by means of a tree kernel function. This function intuitively measures to what extent a given candidate subtree is *compatible* with the subtree of a correct predicate argument structure. We can use it to define two different learning problems: (a) the simple classification of correct and incorrect predicate argument structures and (b) given the best m structures, we can train a re-ranker algorithm able to exploit argument inter-dependencies.

3.1 The Argument Spanning Trees (ASTs)

We consider predicate argument structures annotated in PropBank along with the corresponding TreeBank data as our object space. Given the target predicate node p and a node subset $s = \{n_1, .., n_k\}$ of the parse tree t, we define as the spanning tree root r the lowest common ancestor of $n_1, .., n_k$ and p. The node set spanning tree (NST) p_s is the subtree of t rooted in r from which the nodes that are neither ancestors nor descendants of any n_i or p are removed.

Since predicate arguments are associated with tree nodes (i.e. they exactly fit into syntactic constituents), we can define the *Argument Spanning Tree* (AST) of a predicate argument set, $\{p, \{a_1, .., a_n\}\}$, as the NST over such nodes, i.e. $p_{\{a_1,..,a_n\}}$. An AST corresponds to the *minimal* subtree whose leaves are all and only the words compounding the arguments and the predicate. For example, Figure 1 shows the parse tree of the sentence `"John took the book and read its title"`. $took_{\{Arg_0,Arg_1\}}$ and $read_{\{Arg_0,Arg_1\}}$ are two AST structures associated with the two predicates *took* and *read*, respectively. All the other possible subtrees, i.e. NSTs, are not valid ASTs for these two predicates. Note that classifying p_s in AST or NST for each node subset s of t is equivalent to solve the boundary detection problem.

The critical points for the AST classification are: (1) how to design suitable features for the characterization of valid structures. This requires a careful linguistic investigation about their significant properties. (2) How to deal with the exponential number of NSTs.

The first problem can be addressed by means of tree kernels over the ASTs. Tree kernel spaces are an alternative to the manual feature design as the learning machine, (e.g. SVMs) can select the most relevant features from a high dimensional space. In other words, we can use a tree kernel function to estimate the similarity between two ASTs (see Sec-

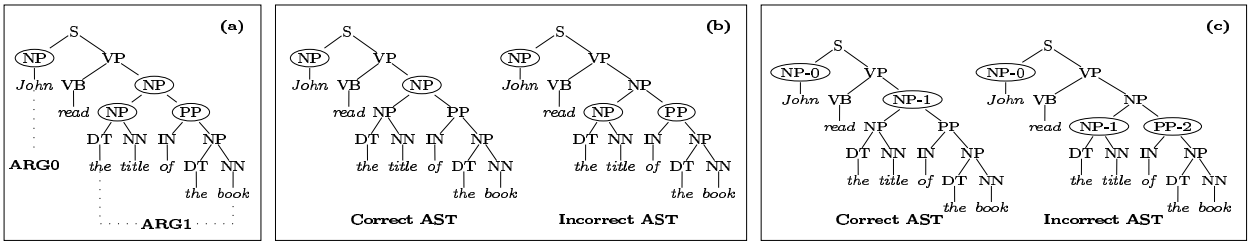

Figure 2: Two-step boundary classification. a) Sentence tree; b) Two candidate $ASTs$; c) Extended AST-Ord labeling

tion 2), hence avoiding to define explicit features.

The second problem can be approached in two ways:
(1) We can increase the recall of TBC to enlarge the set of candidate arguments. From such set, we can extract correct and incorrect argument structures. As the number of such structures will be rather small, we can apply the AST classifier to detect the correct ones.
(2) We can consider the classification probability provided by TBC and TRC (Pradhan et al., 2005a) and select the m most probable structures. Then, we can apply a re-ranking approach based on SVMs and tree kernels.

The re-ranking approach is the most promising one as suggested in (Toutanova et al., 2005) but it does not clearly reveal if tree kernels can be used to learn the difference between correct or incorrect argument structures. Thus it is interesting to study both the above approaches.

3.2 NST Classification

As we cannot classify all possible candidate argument structures, we apply the AST classifier just to detect the correct structures from a set of overlapping arguments. Given two nodes n_1 and n_2 of an NST, they overlap if either n_1 is ancestor of n_2 or vice versa. NSTs that contain overlapping nodes are not valid ASTs but subtrees of *NST*s may be valid $ASTs$. Assuming this, we define s as the set of potential argument nodes and we create two node sets $s_1 = s - \{n_1\}$ and $s_2 = s - \{n_2\}$. By classifying the two new NSTs p_{s_1} and p_{s_2} with the AST classifier, we can select the correct structures. Of course, this procedure can be generalized to a set of overlapping nodes greater than 2. However, considering that the Precision of TBC is generally high, the number of overlapping nodes is usually small.

Figure 2 shows a working example of the multi-stage classifier. In Frame (a), TBC labels as potential arguments (circled nodes) three overlapping nodes related to `Arg1`. This leads to two possible non-overlapping solutions (Frame (b)) but only the first one is correct. In fact, according to the second one the propositional phrase "*of the book*" would be incorrectly attached to the verbal predicate, i.e. in contrast with the parse tree. The AST classifier, applied to the two NSTs, is expected to detect this inconsistency and provide the correct output.

3.3 Re-ranking NSTs with Tree Kernels

To implement the re-ranking model, we follow the approach described in (Toutanova et al., 2005).

First, we use SVMs to implement the boundary TBC and role TRC *local* classifiers. As SVMs do not provide probabilistic output, we use the Platt's algorithm (Platt, 2000) and its revised version (Lin et al., 2003) to trasform scores into probabilities.

Second, we combine TBC and TRC probabilities to obtain the m most likely sequences s of tree nodes annotated with semantic roles. As argument constituents of the same verb cannot overlap, we generate sequences that respect such node constraint. We adopt the same algorithm described in (Toutanova et al., 2005). We start from the leaves and we select the m sequences that respect the constraints and at the same time have the highest joint probability of TBC and TRC.

Third, we extract the following feature representation:
(a) The ASTs associated with the predicate argument structures. To make faster the learning process and to try to only capture the most relevant features, we also experimented with a compact version of the

AST which is pruned at the level of argument nodes. (b) Attribute value features (standard features) related to the whole predicate structure. These include the features for each arguments (Gildea and Jurasfky, 2002) and global features like the sequence of argument labels, e.g. $\langle Arg0, Arg1, ArgM \rangle$.

Finally, we prepare the training examples for the re-ranker considering the m best annotations of each predicate structure. We use the approach adopted in (Shen et al., 2003), which generates all possible pairs from the m examples, i.e. $\binom{m}{2}$ pairs. Each pair is assigned to a positive example if the first member of the pair has a higher score than the second member. The score that we use is the F1 measure of the annotated structure with respect to the gold standard. More in detail, given training/testing examples $e_i = \langle t_i^1, t_i^2, v_i^1, v_i^2 \rangle$, where t_i^1 and t_i^2 are two ASTs and v_i^1 and v_i^2 are two feature vectors associated with two candidate predicate structures s_1 and s_2, we define the following kernels:

$$1) \quad K_{tr}(e_1, e_2) = K_t(t_1^1, t_2^1) + K_t(t_1^2, t_2^2) - K_t(t_1^1, t_2^2) - K_t(t_1^2, t_2^1),$$

where t_i^j is the j-th AST of the pair e_i, K_t is the tree kernel function defined in Section 2 and $i, j \in \{1, 2\}$.

$$2) \quad K_{pr}(e_1, e_2) = K_p(v_1^1, v_2^1) + K_p(v_1^2, v_2^2) - K_p(v_1^1, v_2^2) - K_p(v_1^2, v_2^1),$$

where v_i^j is the j-th feature vector of the pair e_i and K_p is the polynomial kernel applied to such vectors.

The final kernel that we use for re-ranking is the following:

$$K(e_1, e_2) = \frac{K_{tr}(e_1, e_2)}{|K_{tr}(e_1, e_2)|} + \frac{K_{pr}(e_1, e_2)}{|K_{pr}(e_1, e_2)|}$$

Regarding tree kernel feature engineering, the next section show how we can generate more effective features given an established kernel function.

3.4 Tree kernel feature engineering

Consider the Frame (b) of Figure 2, it shows two perfectly identical NSTs, consequently, their fragments will also be equal. This prevents the algorithm to learn something from such examples. To solve the problem, we can enrich the $NSTs$ by marking their argument nodes with a progressive number, starting from the leftmost argument. For example, in the first NST of Frame (c), we mark as `NP-0` and `NP-1` the first and second argument nodes whereas in the second NST we trasform the three argument node labels in `NP-0`, `NP-1` and `PP-2`. We will refer to the resulting structure as a AST-Ord (ordinal number). This simple modification allows the tree kernel to generate different argument structures for the above NSTs. For example, from the first NST in Figure 2.c, the fragments `[NP-1 [NP][PP]]`, `[NP [DT][NN]]` and `[PP [IN][NP]]` are generated. They do not match anymore with the `[NP-0 [NP][PP]]`, `[NP-1 [DT][NN]]` and `[PP-2 [IN][NP]]` fragments generated from the second NST in Figure 2.c.

Additionally, it should be noted that the semantic information provided by the role type can remarkably help the detection of correct or incorrect predicate argument structures. Thus, we can enrich the argument node label with the role type, e.g. the `NP-0` and `NP-1` of the correct AST of Figure 2.c become `NP-Arg0` and `NP-Arg1` (not shown in the figure). We refer to this structure as AST-Arg. Of course, to apply the AST-Arg classifier, we need that TRC labels the arguments detected by TBC.

4 The experiments

The experiments were carried out within the setting defined in the CoNLL-2005 Shared Task (Carreras and Màrquez, 2005). In particular, we adopted the Charniak parse trees available at `www.lsi.upc.edu/~srlconll/` along with the official performance evaluator.

All the experiments were performed with the SVM-light-TK software available at `http://ai-nlp.info.uniroma2.it/moschitti/` which encodes ST and SST kernels in SVM-light (Joachims, 1999). For TBC and TRC, we used the linear kernel with a regularization parameter (option `-c`) equal to 1. A cost factor (option `-j`) of 10 was adopted for TBC to have a higher Recall, whereas for TRC, the cost factor was parameterized according to the maximal accuracy of each argument class on the validation set. For the AST-based classifiers we used a λ equal to 0.4 (see (Moschitti, 2004)).

	Section 21			Section 23		
AST Class.	P.	R.	F_1	P.	R.	F_1
–	69.8	77.9	73.7	62.2	77.1	68.9
Ord	73.7	81.2	77.3	63.7	80.6	71.2
Arg	73.6	84.7	78.7	64.2	82.3	72.1

Table 1: AST, AST-Ord, and AST-Arg performance on sections 21 and 23.

4.1 Classification of whole predicate argument structures

In these experiments, we trained TBC on sections 02-08 whereas, to achieve a very accurate role classifier, we trained TRC on all sections 02-21. To train the AST, AST-Ord (AST with ordinal numbers in the argument nodes), and AST-Arg (AST with argument type in the argument nodes) classifiers, we applied the TBC and TRC over sections 09-20. Then, we considered all the structures whose automatic annotation showed at least an argument overlap. From these, we extracted 30,220 valid ASTs and 28,143 non-valid ASTs, for a total of 183,642 arguments.

First, we evaluate the accuracy of the AST-based classifiers by extracting 1,975 ASTs and 2,220 non-ASTs from Section 21 and the 2,159 ASTs and 3,461 non-ASTs from Section 23. The accuracy derived on Section 21 is an upperbound for our classifiers since it is obtained using an ideal syntactic parser (the Charniak's parser was trained also on Section 21) and an ideal role classifier.

Table 1 shows Precision, Recall and F_1 measures of the AST-based classifiers over the above NSTs. Rows 2, 3 and 4 report the performance of AST, AST-Ord, and AST-Arg classifiers, respectively. We note that: (a) The impact of parsing accuracy is shown by the gap of about 6% points between sections 21 and 23. (b) The ordinal numbering of arguments (Ord) and the role type information (Arg) provide tree kernels with more meaningful fragments since they improve the basic model of about 4%. (c) The deeper semantic information generated by the Arg labels provides useful clues to select correct predicate argument structures since it improves the Ord model on both sections.

Second, we measured the impact of the AST-based classifiers on the accuracy of both phases of semantic role labeling. Table 2 reports the results on sections 21 and 23. For each of them, Precision, Recall and F_1 of different approaches to boundary identification (bnd) and to the complete task, i.e. boundary and role classification (bnd+class) are shown. Such approaches are based on different strategies to remove the overlaps, i.e. with the AST, AST-Ord and AST-Arg classifiers and using the baseline (RND), i.e. a random selection of non-overlapping structures. The baseline corresponds to the system based on TBC and TRC[1].

We note that: (a) for any model, the boundary detection F_1 on Section 21 is about 10 points higher than the F_1 on Section 23 (e.g. 87.0% vs. 77.9% for RND). As expected the parse tree quality is very important to detect argument boundaries. (b) On the real test (Section 23) the classification introduces labeling errors which decrease the accuracy of about 5% (77.9 vs 72.9 for RND). (c) The Ord and Arg approaches constantly improve the baseline F_1 of about 1%. Such poor impact does not surprise as the overlapping structures are a small percentage of the test set, thus the overall improvement cannot be very high.

Third, the comparison with the CoNLL 2005 results (Carreras and Màrquez, 2005) can only be carried out with respect to the whole SRL task (bnd+class in table 2) since boundary detection versus role classification is generally not provided in CoNLL 2005. Moreover, our best global result, i.e. 73.9%, was obtained under two severe experimental factors: a) the use of just 1/3 of the available training set, and b) the usage of the linear SVM model for the TBC classifier, which is much faster than the polynomial SVMs but also less accurate. However, we note the promising results of the AST meta-classifier, which can be used with any of the best figure CoNLL systems.

Finally, the overall results suggest that the tree kernel model is robust to parse tree errors since preserves the same improvement across trees derived with different accuracy, i.e. the *semi-automatic* trees of Section 21 and the automatic tree of Section 23. Moreover, it shows a high accuracy for the classification of correct and incorrect ASTs. This last property is quite interesting as the best SRL systems

[1]We needed to remove the overlaps from the baseline outcome in order to apply the CoNLL evaluator.

(Punyakanok et al., 2005; Toutanova et al., 2005; Pradhan et al., 2005b) were obtained by exploiting the information on the whole predicate argument structure.

Next section shows our preliminary experiments on re-ranking using the AST kernel based approach.

4.2 Re-ranking based on Tree Kernels

In these experiments, we used the output of TBC and TRC[2] to provide an SVM tree kernel with a ranked list of predicate argument structures. More in detail, we applied a Viterbi-like algorithm to generate the 20 most likely annotations for each predicate structure, according to the joint probabilistic model of TBC and TRC. We sorted such structures based on their F_1 measure and used them to learn the SVM re-ranker described in 3.3.

For training, we used Sections 12, 14, 15, 16 and 24, which contain 24,729 predicate structures. For each of them, we considered the 5 annotations having the highest F1 score (i.e. 123,674 NSTs) on the span of the 20 best annotations provided by Viterbi algorithm. With such structures, we obtained 294,296 pairs used to train the SVM-based re-ranker. As the number of such structures is very large the SVM training time was very high. Thus, we sped up the learning process by using only the ASTs associated with the core arguments. From the test sentences (which contain 5,267 structures), we extracted the 20 best Viterbi annotated structures, i.e. 102,343 (for a total of 315.531 pairs), which were used for the following experiments:

First, we selected the best annotation (according to the F_1 provided by the gold standard annotations) out of the 20 provided by the Viterbi's algorithm. The resulting F_1 of 88.59% is the upperbound of our approach.

Second, we selected the top ranked annotation indicated by the Viterbi's algorithm. This provides our baseline F_1 measure, i.e. 75.91%. Such outcome is slightly higher than our official CoNLL result (Moschitti et al., 2005) obtained without converting SVM scores into probabilities.

Third, we applied the SVM re-ranker to select the best structures according to the core roles. We achieved 80.68% which is practically equal to the result obtained in (Punyakanok et al., 2005; Carreras and Màrquez, 2005) for core roles, i.e. 81%. Their overall F1 which includes all the arguments was 79.44%. This confirms that the classification of the non-core roles is more complex than the other arguments.

Finally, the high computation time of the re-ranker prevented us to use the larger structures which include all arguments. The major complexity issue was the slow training and classification time of SVMs. The time needed for tree kernel function was not so problematic as we could use the fast evaluation proposed in (Moschitti, 2006). This roughly reduces the computation time to the one required by a polynomial kernel. The real burden is therefore the learning time of SVMs that is quadratic in the number of training instances. For example, to carry out the re-ranking experiments required approximately one month of a 64 bits machine (2.4 GHz and 4Gb Ram). To solve this problem, we are going to study the impact on the accuracy of fast learning algorithms such as the Voted Perceptron.

5 Related Work

Recently, many kernels for natural language applications have been designed. In what follows, we highlight their difference and properties.

The tree kernel used in this article was proposed in (Collins and Duffy, 2002) for syntactic parsing re-ranking. It was experimented with the Voted Perceptron and was shown to improve the syntactic parsing. 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. In (Punyakanok et al., 2005), a set of different syntactic parse trees, e.g. the n best trees generated by the Charniak's parser, were used to improve the SRL accuracy. These different sources of syntactic information were used to generate a set of different SRL

[2]With the aim of improving the state-of-the-art, we applied the polynomial kernel for all basic classifiers, at this time. We used the models developed during our participation to the CoNLL 2005 shared task (Moschitti et al., 2005).

	Section 21								Section 23							
	bnd				bnd+class				bnd				bnd+class			
	AST Classifier			RND	AST Classifier			RND	AST Classifier			RND	AST Classifier			RND
	-	Ord	Arg		-	Ord	Arg		-	Ord	Arg		-	Ord	Arg	
P.	87.5	88.3	88.3	86.9	85.5	86.3	86.4	85.0	78.6	79.0	79.3	77.8	73.1	73.5	73.4	72.3
R.	87.3	88.1	88.3	87.1	85.7	86.5	86.8	85.6	78.1	78.4	78.7	77.9	73.8	74.1	74.4	73.6
F_1	87.4	88.2	88.3	87.0	85.6	86.4	86.6	85.3	78.3	78.7	79.0	77.9	73.4	73.8	73.9	72.9

Table 2: Semantic Role Labeling performance on automatic trees using AST-based classifiers.

outputs. A joint inference stage was applied to resolve the inconsistency of the different outputs. In (Toutanova et al., 2005), it was observed that there are strong dependencies among the labels of the semantic argument nodes of a verb. Thus, to approach the problem, a re-ranking method of role sequences labeled by a TRC is applied. In (Pradhan et al., 2005b), some experiments were conducted on SRL systems trained using different syntactic views.

6 Conclusions

Recent work on Semantic Role Labeling has shown that to achieve high labeling accuracy a joint inference on the whole predicate argument structure should be applied. As feature design for such task is complex, we can take advantage from kernel methods to model our intuitive knowledge about the n-ary predicate argument relations.

In this paper we have shown that we can exploit the properties of tree kernels to engineer syntactic features for the semantic role labeling task. The experiments suggest that (1) the information related to the whole predicate argument structure is important as it can improve the state-of-the-art and (2) tree kernels can be used in a joint model to generate relevant syntactic/semantic features. The real drawback is the computational complexity of working with SVMs, thus the design of fast algorithm is an interesting future work.

Acknowledgments

This research is partially supported by the PrestoSpace EU Project#: FP6-507336.

References

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

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

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

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

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

Paul Kingsbury and Martha Palmer. 2002. From Treebank to PropBank. In *Proceedings of LREC'02)*, Las Palmas, Spain.

H.T. Lin, C.J. Lin, and R.C. Weng. 2003. A note on platt's probabilistic outputs for support vector machines. Technical report, National Taiwan University.

Alessandro Moschitti, Bonaventura Coppola, Daniele Pighin, and Roberto Basili. 2005. Hierarchical semantic role labeling. In *Proceedings of CoNLL05 shared task*, Ann Arbor (MI), USA.

Alessandro Moschitti. 2004. A study on convolution kernels for shallow semantic parsing. In *Proceedings of ACL'04*, Barcelona, Spain.

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

J. Platt. 2000. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. MIT Press.

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

Sameer Pradhan, Wayne Ward, Kadri Hacioglu, James Martin, and Daniel Jurafsky. 2005b. Semantic role labeling using different syntactic views. In *Proceedings ACL'05*.

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

Libin Shen, Anoop Sarkar, and Aravind Joshi. 2003. Using ltag based features in parse reranking. In *Conference on EMNLP03*, Sapporo, Japan.

Kristina Toutanova, Penka Markova, and Christopher D. Manning. 2004. The leaf projection path view of parse trees: Exploring string kernels for hpsg parse selection. In *In Proceedings of EMNLP04*.

Kristina Toutanova, Aria Haghighi, and Christopher Manning. 2005. Joint learning improves semantic role labeling. In *Proceedings of ACL05*.