Engineering of Syntactic Features for Shallow Semantic Parsing

Alessandro Moschitti[◇] **Bonaventura Coppola**[†‡] **Daniele Pighin**[◇] **Roberto Basili**[◇]

[◇] DISP - University of Rome "Tor Vergata", Rome, Italy
{moschitti, pighin, basili}@info.uniroma2.it

[†] ITC-Irst, [‡] DIT - University of Trento, Povo-Trento, Italy
coppolab@itc.it

Abstract

Recent natural language learning research has shown that structural kernels can be effectively used to induce accurate models of linguistic phenomena.

In this paper, we show that the above properties hold on a novel task related to predicate argument classification. A tree kernel for selecting the subtrees which encodes argument structures is applied. Experiments with Support Vector Machines on large data sets (i.e. the PropBank collection) show that such kernel improves the recognition of argument boundaries.

1 Introduction

The design of features for natural language processing tasks is, in general, a critical problem. The inherent complexity of linguistic phenomena, often characterized by structured data, makes difficult to find effective linear feature representations for the target learning models.

In many cases, the traditional feature selection techniques (Kohavi and Sommerfield, 1995) are not so useful since the critical problem relates to feature generation rather than selection. For example, the design of features for a natural language syntactic parse-tree re-ranking problem (Collins, 2000) cannot be carried out without a deep knowledge about automatic syntactic parsing. The modeling of syntactic/semantic based features should take into account linguistic aspects to detect the interesting context, e.g. the ancestor nodes or the semantic dependencies (Toutanova et al., 2004).

A viable alternative 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. Another interesting model for parsing re-ranking based on tree kernel is presented in (Taskar et al., 2004). The good results show that tree kernels are very promising for automatic feature engineering, especially when the available knowledge about the phenomenon is limited.

Along the same line, automatic learning tasks that rely on syntactic information may take advantage of a tree kernel approach. One of such tasks is the automatic boundary detection of predicate arguments of the kind defined in PropBank (Kingsbury and Palmer, 2002). For this purpose, given a predicate p in a sentence s, we can define the notion of *predicate argument spanning trees* ($PAST$s) as those syntactic subtrees of s which *exactly cover* all and only the p's arguments (see Section 4.1). The set of non-spanning trees can be then associated with all the remaining subtrees of s.

An automatic classifier which recognizes the spanning trees can potentially be used to detect the predicate argument boundaries. Unfortunately, the application of such classifier to all possible sentence subtrees would require an exponential execution time. As a consequence, we can use it only to decide for a reduced set of subtrees associated with a corresponding set of candidate boundaries. Notice how these can be detected by previous approaches

Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in NLP, pages 48–56,
Ann Arbor, June 2005.

(e.g. (Pradhan et al., 2004)) in which a traditional boundary classifier (tbc) labels the parse-tree nodes as potential arguments ($\mathcal{PA}$). Such classifiers, generally, are not sensitive to the overall argument structure. On the contrary, a $PAST$ classifier ($past_c$) can consider the overall argument structure encoded in the associated subtree. This is induced by the $\mathcal{PA}$ subsets.

The feature design for the $PAST$ representation is not simple. Tree kernels are a viable alternative that allows the learning algorithm to measure the similarity between two $PAST$s in term of all possible tree substructures.

In this paper, we designed and experimented a boundary classifier for predicate argument labeling based on two phases: (1) a first annotation of potential arguments by using a high recall tbc and (2) a $PAST$ classification step aiming to select the correct substructures associated with potential arguments. Both classifiers are based on Support Vector Machines learning. The $past_c$ uses the tree kernel function defined in (Collins and Duffy, 2002). The results show that the $PAST$ classification can be learned with high accuracy (the f-measure is about 89%) and the impact on the overall boundary detection accuracy is good.

In the remainder of this paper, Section 2 introduces the Semantic Role Labeling problem along with the boundary detection subtask. Section 3 defines the SVMs using the linear kernel and the parse tree kernel for boundary detection. Section 4 describes our boundary detection algorithm. Section 5 shows the preliminary comparative results between the traditional and the two-step boundary detection. Finally, Section 7 summarizes the conclusions.

2 Automated Semantic Role Labeling

One of the largest resources of manually annotated predicate argument structures has been developed in the PropBank (PB) project. The PB corpus contains 300,000 words annotated with predicative information on top of the Penn Treebank 2 Wall Street Journal texts. For any given predicate, the expected arguments are labeled sequentially from *Arg0* to *Arg9*, *ArgA* and *ArgM*. Figure 1 shows an example of the PB predicate annotation of the sentence: `John rented a room in Boston.`

Predicates in PB are only embodied by verbs whereas most of the times *Arg0* is the *subject*, *Arg1* is the *direct object* and *ArgM* indicates *locations*, as in our example.

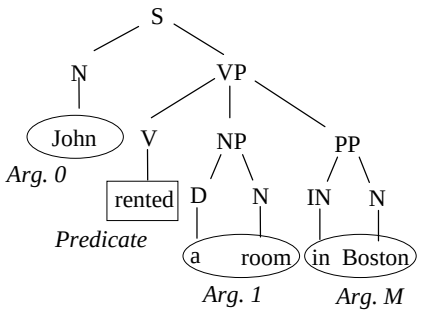

Figure 1: A predicate argument structure in a parse-tree representation.

Several machine learning approaches for automatic predicate argument extraction have been developed, e.g. (Gildea and Jurasfky, 2002; Gildea and Palmer, 2002; Gildea and Hockenmaier, 2003; Pradhan et al., 2004). Their common characteristic is the adoption of feature spaces that model predicate-argument structures in a flat feature representation. In the next section, we present the common parse tree-based approach to this problem.

2.1 Predicate Argument Extraction

Given a sentence in natural language, all the predicates associated with the verbs have to be identified along with their arguments. This problem is usually divided in two subtasks: (a) the detection of the target argument boundaries, i.e. the span of its words in the sentence, and (b) the classification of the argument type, e.g. *Arg0* or *ArgM* in PropBank or *Agent* and *Goal* in FrameNet.

The standard approach to learn both the detection and the classification of predicate arguments is summarized by the following steps:

1. Given a sentence from the *training-set*, generate a full syntactic parse-tree;
2. let $\mathcal{P}$ and $\mathcal{A}$ be the set of predicates and the set of parse-tree nodes (i.e. the potential arguments), respectively;
3. for each pair $< p, a > \in \mathcal{P} \times \mathcal{A}$:
   - 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 T^+ (positive examples), otherwise put it in T^- (negative examples).

For instance, in Figure 1, for each combination of the predicate *rent* with the nodes N, S, VP, V, NP, PP, D or IN the instances $F_{rent,a}$ are generated. In case the node a exactly covers "John", "a room" or "in Boston", it will be a positive instance otherwise it will be a negative one, e.g. $F_{rent,IN}$.

The T^+ and T^- sets are used to train the boundary classifier. To train the multi-class classifier T^+ can be reorganized as positive $T^+_{arg_i}$ and negative $T^-_{arg_i}$ examples for each argument i. In this way, an individual ONE-vs-ALL classifier for each argument i can be trained. We adopted this solution, according to (Pradhan et al., 2004), since it is simple and effective. In the classification phase, given an unseen sentence, all its $F_{p,a}$ are generated and classified by each individual classifier C_i. The argument associated with the maximum among the scores provided by the individual classifiers is eventually selected.

2.2 Standard feature space

The discovery of relevant features is, as usual, a complex task. However, there is a common consensus on the set of basic features. These standard features, firstly proposed in (Gildea and Jurasfky, 2002), refer to unstructured information derived from parse trees, i.e. *Phrase Type*, *Predicate Word*, *Head Word*, *Governing Category*, *Position* and *Voice*. For example, the *Phrase Type* indicates the syntactic type of the phrase labeled as a predicate argument, e.g. NP for *Arg1* in Figure 1. 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 for *Arg1* in Figure 1. The *Predicate Word* is the surface form of the verbal predicate, e.g. *rent* for all arguments.

In the next section we describe the SVM approach and the basic kernel theory for the predicate argument classification.

3 Learning predicate structures via Support Vector Machines

Given a vector space in $\Re^n$ and a set of positive and negative points, SVMs classify vectors according to a separating hyperplane, $H(\vec{x}) = \vec{w} \times \vec{x} + b = 0$, where $\vec{w} \in \Re^n$ and $b \in \Re$ are learned by applying the *Structural Risk Minimization principle* (Vapnik, 1995).

To apply the SVM algorithm to Predicate Argument Classification, we need a function $\phi : \mathcal{F} \rightarrow \Re^n$ to map our features space $\mathcal{F} = \{f_1, .., f_{|\mathcal{F}|}\}$ and our predicate/argument pair representation, $F_{p,a} = F_z$, into $\Re^n$, such that:

$$F_z \rightarrow \phi(F_z) = (\phi_1(F_z), .., \phi_n(F_z))$$

From the kernel theory we have that:

$$H(\vec{x}) = \Big(\sum_{i=1..l} \alpha_i \vec{x}_i \Big) \cdot \vec{x} + b =$$

$$\sum_{i=1..l} \alpha_i \vec{x}_i \cdot \vec{x} + b = \sum_{i=1..l} \alpha_i \phi(F_i) \cdot \phi(F_z) + b.$$

where, $F_i \; \forall i \in \{1, .., l\}$ are the training instances and the product $K(F_i, F_z) = <\phi(F_i) \cdot \phi(F_z)>$ is the kernel function associated with the mapping ϕ.

The simplest mapping that we can apply is $\phi(F_z) = \vec{z} = (z_1, ..., z_n)$ where $z_i = 1$ *if* $f_i \in F_z$ and $z_i = 0$ otherwise, i.e. the characteristic vector of the set F_z with respect to $\mathcal{F}$. If we choose the scalar product as a kernel function we obtain the linear kernel $K_L(F_x, F_z) = \vec{x} \cdot \vec{z}$.

An interesting property is that we do not need to evaluate the ϕ function to compute the above vector. Only the $K(\vec{x}, \vec{z})$ values are in fact required. This allows us to derive efficient classifiers in a huge (possible infinite) feature space, provided that the kernel is processed in an efficient way. This property is also exploited to design convolution kernel like those based on tree structures.

3.1 The tree kernel function

The main idea of the tree kernels is the modeling of a $K_T(T_1, T_2)$ function which computes the number of common substructures between two trees T_1 and T_2.

Given the set of substructures (fragments) $\{f_1, f_2, ..\} = \mathcal{F}$ extracted from all the trees of the training set, we define the indicator function $I_i(n)$

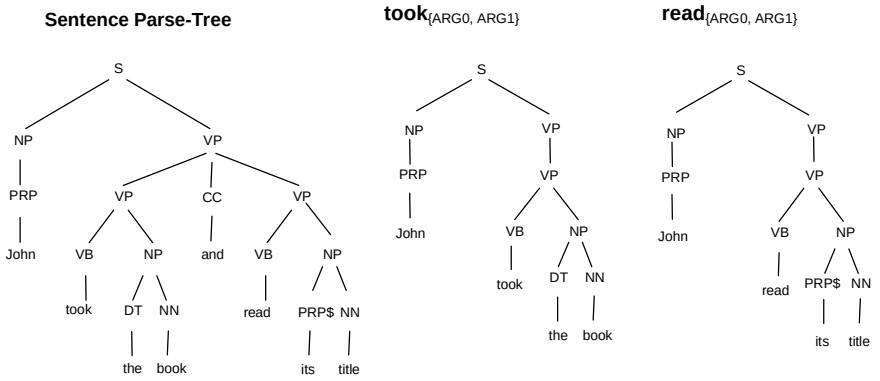

Figure 2: A sentence parse tree with two predicative tree structures ($PAST$s)

which is equal 1 if the target f_i is rooted at node n and 0 otherwise. It follows that:

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

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

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

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

where $nc(n_1)$ 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, $nc(n_1) = nc(n_2)$.

The above kernel has the drawback of assigning higher weights to larger structures[1]. In order to overcome this problem we scale the relative importance of the tree fragments imposing a parameter λ in conditions 2 and 3 as follows: $\Delta(n_x, n_z) = \lambda$ and $\Delta(n_x, n_z) = \lambda \prod_{j=1}^{nc(n_x)} (1 + \Delta(c_{n_1}^j, c_{n_2}^j))$.

[1]In order to approach this problem and to map similarity scores in the [0,1] range, a normalization in the kernel space, i.e. $K'_T(T_1, T_2) = \frac{K_T(T_1, T_2)}{\sqrt{K_T(T_1, T_1) \times K_T(T_2, T_2)}}$. is always applied

4 Boundary detection via argument spanning

Section 2 has shown that traditional argument boundary classifiers rely only on features extracted from the current potential argument node. In order to take into account a complete argument structure information, the classifier should select a set of parse-tree nodes and consider them as potential arguments of the target predicate. The number of all possible subsets is exponential in the number of the parse-tree nodes of the sentence, thus, we need to cut the search space. For such purpose, a traditional boundary classifier can be applied to select the set of potential arguments $\mathcal{PA}$. The reduced number of $\mathcal{PA}$ subsets can be associated with sentence subtrees which in turn can be classified by using tree kernel functions. These measure if a subtree is *compatible* or not with the subtree of a correct predicate argument structure.

4.1 The Predicate Argument Spanning Trees ($PAST$s)

We consider the predicate argument structures annotated in PropBank along with the corresponding TreeBank data as our object space. Given the target predicate p in a sentence parse tree T and a subset $s = \{n_1, .., n_k\}$ of the T's nodes, N_T, we define as the spanning tree root r the lowest common ancestor of $n_1, .., n_k$. The node spanning tree (NST), p_s is the subtree rooted in r, from which the nodes that are neither ancestors nor descendants of any n_i are removed.

Since predicate arguments are associated with tree nodes, we can define the *predicate argu-*

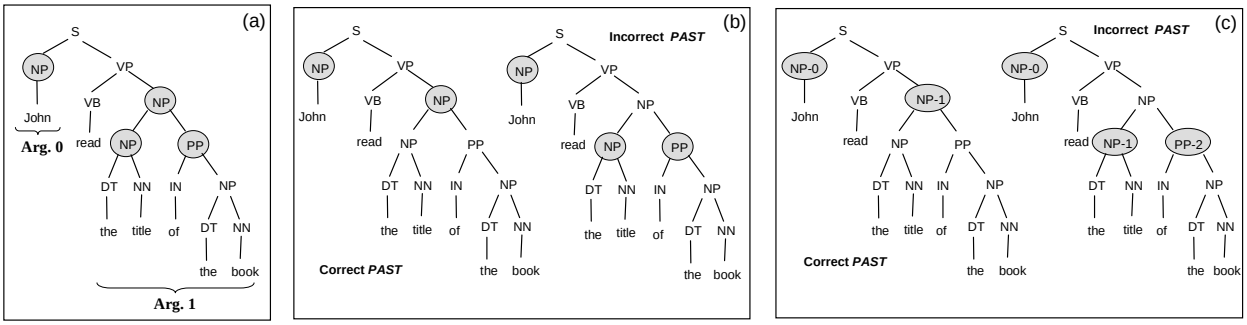

Figure 3: Two-step boundary classifier.

ment spanning tree ($PAST$) of a predicate argument set, $\{a_1, .., a_n\}$, as the NST over such nodes, i.e. $p_{\{a_1,..,a_n\}}$. A $PAST$ corresponds to the *minimal* subparse tree whose leaves are all and only the word sequence compounding the arguments. For example, Figure 2 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 $PAST$ structures associated with the two predicates *took* and *read*, respectively. All the other NSTs are not valid $PAST$s.

Notice that, labeling $p_s, \forall s \subseteq N_T$ with a $PAST$ classifier ($past_c$) corresponds to solve the boundary problem. The critical points for the application of this strategy are: (1) how to design suitable features for the $PAST$ characterization. This new problem requires a careful linguistic investigation about the significant properties of the argument spanning trees and (2) how to deal with the exponential number of NSTs.

For the first problem, the use of tree kernels over the $PAST$s can be an alternative to the manual features design as the learning machine, (e.g. SVMs) can select the most relevant features from a high dimensional feature space. In other words, we can use Eq. 1 to estimate the similarity between two $PAST$s avoiding to define explicit features. The same idea has been successfully applied to the parse-tree reranking task (Taskar et al., 2004; Collins and Duffy, 2002) and predicate argument classification (Moschitti, 2004).

For the second problem, i.e. the high computational complexity, we can cut the search space by using a traditional boundary classifier (tbc), e.g. (Pradhan et al., 2004), which provides a small set of potential argument nodes. Let $\mathcal{PA}$ be the set of nodes located by tbc as arguments. We may consider the set $\mathcal{P}$ of the NSTs associated with any subset of $\mathcal{PA}$, i.e. $\mathcal{P} = \{p_s : s \subseteq \mathcal{PA}\}$. However, also the classification of $\mathcal{P}$ may be computationally problematic since theoretically there are $|\mathcal{P}| = 2^{|\mathcal{PA}|}$ members.

In order to have a very efficient procedure, we applied $past_c$ to only the $\mathcal{PA}$ sets associated with incorrect $PAST$s. A way to detect such incorrect NSTs is to look for a node pair $<n_1, n_2> \in \mathcal{PA} \times \mathcal{PA}$ of *overlapping nodes*, i.e. n_1 is ancestor of n_2 or viceversa. After we have detected such nodes, we create two node sets $PA_1 = \mathcal{PA} - \{n_1\}$ and $PA_2 = \mathcal{PA} - \{n_2\}$ and classify them with the $past_c$ to select the correct set of argument boundaries. This procedure can be generalized to a set of overlapping nodes O greater than 2 as reported in Appendix 1.

Note that the algorithm selects a maximal set of non-overlapping nodes, i.e. the first that is generated. Additionally, the worst case is rather rare thus the algorithm is very fast on average.

The Figure 3 shows a working example of the multi-stage classifier. In Frame (a), tbc labels as potential arguments (gray color) three overlapping nodes (in Arg.1). The overlap resolution algorithm proposes two solutions (Frame (b)) of which only one is correct. In fact, according to the second solution the propositional phrase "*of the book*" would incorrectly be attached to the verbal predicate, i.e. in contrast with the parse tree. The $past_c$, applied

to the two NSTs, should detect this inconsistency and provide the correct output. Note that, during the learning, we generate the non-overlapping structures in the same way to derive the positive and negative examples.

4.2 Engineering Tree Fragment Features

In the Frame (b) of Figure 3, we show one of the possible cases which $past_c$ should deal with. The critical problem is that the two NSTs are perfectly identical, thus, it is not possible to discern between them using only their parse-tree fragments.

The solution to engineer novel features is to simply add the boundary information provided by the tbc to the NSTs. We mark with a progressive number the phrase type corresponding to an argument node, 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 have an hypothesis of three arguments on the `NP`, `NP` and `PP` nodes. We trasform them in `NP-0`, `NP-1` and `PP-2`.

This simple modification enables the tree kernel to generate features useful to distinguish between two identical parse trees associated with different argument structures. For example, for the first NST 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 of the second NST.

In order to verify the relevance of our model, the next section provides empirical evidence about the effectiveness of our approach.

5 The Experiments

The experiments were carried out with the SVM-light-TK software available at `http://ai-nlp.info.uniroma2.it/moschitti/` which encodes the tree kernels in the SVM-light software (Joachims, 1999). For tbc, we used the linear kernel with a regularization parameter (option `-c`) equal to 1 and a cost-factor (option `-j`) of 10 to have a higher Recall. For the $past_c$ we used $\lambda = 0.4$ (see (Moschitti, 2004)).

As referring dataset, we used the PropBank corpora available at `www.cis.upenn.edu/~ace`, along with the Penn TreeBank 2 (`www.cis.upenn.edu/~treebank`) (Marcus et al., 1993). This corpus contains about 53,700 sentences and a fixed split between training and testing which has been used in other researches, e.g. (Pradhan et al., 2004; Gildea and Palmer, 2002). We did not include continuation and co-referring arguments in our experiments.

We used sections from 02 to 07 (54,443 argument nodes and 1,343,046 non-argument nodes) to train the traditional boundary classifier (tbc). Then, we applied it to classify the sections from 08 to 21 (125,443 argument nodes vs. 3,010,673 non-argument nodes). As results we obtained 2,988 NSTs containing at least an overlapping node pair out of the total 65,212 predicate structures (according to the tbc decisions). From the 2,988 overlapping structures we extracted 3,624 positive and 4,461 negative NSTs, that we used to train the $past_c$.

The performance was evaluated with the F_1 measure[2] over the section 23. This contains 10,406 argument nodes out of 249,879 parse tree nodes. By applying the tbc classifier we derived 235 overlapping $NSTs$, from which we extracted 204 $PAST$s and 385 incorrect predicate argument structures. On such test data, the performance of $past_c$ was very high, i.e. 87.08% in Precision and 89.22% in Recall.

Using the $past_c$ we removed from the tbc the $\mathcal{PA}$ that cause overlaps. To measure the impact on the boundary identification performance, we compared it with three different boundary classification baselines:

- tbc: overlaps are ignored and no decision is taken. This provides an upper bound for the recall as no potential argument is rejected for later labeling. Notice that, in presence of overlapping nodes, the sentence cannot be annotated correctly.

- RND: one among the non-overlapping structures with maximal number of arguments is randomly selected.

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

	tbc			tbc+RND			tbc+Heu			tbc+$past_c$		
	P	R	F	P	R	F	P	R	F	P	R	F
All Struct.	92.21	98.76	95.37	93.55	97.31	95.39	92.96	97.32	95.10	94.40	98.42	96.36
Overl. Struct.	98.29	65.8	78.83	74.00	72.27	73.13	68.12	75.23	71.50	89.61	92.68	91.11

Table 1: Two-steps boundary classification performance using the traditional boundary classifier tbc, the random selection of non-overlapping structures (RND), the heuristic to select the most suitable non-overlapping node set (Heu) and the predicate argument spanning tree classifier ($past_c$).

- Heu (heuristic): one of the NSTs which contain the nodes with the lowest overlapping score is chosen. This score counts the number of overlapping node pairs in the NST. For example, in Figure 3.(a) we have a NP that overlaps with two nodes NP and PP, thus it is assigned a score of 2.

The third row of Table 1 shows the results of tbc, $tbc + RND$, $tbc + Heu$ and $tbc + past_c$ in the columns 2,3,4 and 5, respectively. We note that:

- The tbc F_1 is slightly higher than the result obtained in (Pradhan et al., 2004), i.e. 95.37% vs. 93.8% on same training/testing conditions, i.e. (same PropBank version, same training and testing split and same machine learning algorithm). This is explained by the fact that we did not include the continuations and the co-referring arguments that are more difficult to detect.

- Both RND and Heu do not improve the tbc result. This can be explained by observing that in the 50% of the cases a correct node is removed.

- When, to select the correct node, the $past_c$ is used, the F_1 increases of 1.49%, i.e. (96.86 vs. 95.37). This is a very good result considering that to increase the very high baseline of tbc is hard.

In order to give a fairer evaluation of our approach we tested the above classifiers on the overlapping structures only, i.e. we measured the $past_c$ improvement on all and only the structures that required its application. Such reduced test set contains 642 argument nodes and 15,408 non-argument nodes. The fourth row of Table 1 reports the classifier performance on such task. We note that the $past_c$ improves the other heuristics of about 20%.

6 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 reranking. It was experimented with the Voted Perceptron and was shown to improve the syntactic parsing. A refinement of such technique was presented in (Taskar et al., 2004). The substructures produced by the proposed tree kernel were bound to local properties of the target parse tree and more lexical information was added to the overall kernel function.

In (Zelenko et al., 2003), two kernels over syntactic shallow parser structures were devised for the extraction of linguistic relations, e.g. *person-affiliation*. To measure the similarity between two nodes, the *contiguous string kernel* and the *sparse string kernel* (Lodhi et al., 2000) were used. The former can be reduced to the contiguous substring kernel whereas the latter can be transformed in the non-contiguous string kernel. The high running time complexity, caused by the general form of the fragments, limited the experiments on data-set of just 200 news items.

In (Cumby and Roth, 2003), it is proposed a description language that models feature descriptors to generate different feature type. The descriptors, which are quantified logical prepositions, are instantiated by means of a *concept graph* which encodes the structural data. In the case of relation extraction the *concept graph* is associated with a syntactic shallow parse and the extracted propositional features express fragments of a such syntactic structure. The experiments over the named entity class categorization show that when the description language selects an adequate set of tree fragments the Voted Perceptron algorithm increases its classification accuracy.

In (Culotta and Sorensen, 2004) a dependency

tree kernel is used to detect the Named Entity classes in natural language texts. The major novelty was the combination of the contiguous and sparse kernels with the word kernel. The results show that the contiguous outperforms the sparse kernel and the *bag-of-words*.

7 Conclusions

The feature design for new natural language learning tasks is difficult. We can take advantage from the kernel methods to model our intuitive knowledge about the target linguistic phenomenon. In this paper we have shown that we can exploit the properties of tree kernels to engineer syntactic features for the predicate argument boundary detection task.

Preliminary results on gold standard trees suggest that (1) the information related to the whole predicate argument structure is important and (2) tree kernel can be used to generate syntactic features.

In the future, we would like to use an approach similar to the $PAST$ classifier on parses provided by different parsing models to detect boundary and to classify semantic role more accurately .

Acknowledgements

We wish to thank Ana-Maria Giuglea for her help in the design and implementation of the basic Semantic Role Labeling system that we used in the experiments.

References

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

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of ICML 2000*.

Aron Culotta and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 423–429, Barcelona, Spain, July.

Chad Cumby and Dan Roth. 2003. Kernel methods for relational learning. In *Proceedings of the Twentieth International Conference (ICML 2003)*, Washington, DC, USA.

Daniel Gildea and Julia Hockenmaier. 2003. Identifying semantic roles using combinatory categorial grammar. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, Sapporo, Japan.

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

Daniel Gildea and Martha Palmer. 2002. The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th Annual Conference of the Association for Computational Linguistics (ACL-02)*, Philadelphia, PA, USA.

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 the 3rd International Conference on Language Resources and Evaluation (LREC-2002)*, Las Palmas, Spain.

Ron Kohavi and Dan Sommerfield. 1995. Feature subset selection using the wrapper model: Overfitting and dynamic search space topology. In *The First International Conference on Knowledge Discovery and Data Mining*, pages 192–197. AAAI Press, Menlo Park, California, August. Journal version in AIJ.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Christopher Watkins. 2000. Text classification using string kernels. In *NIPS*, pages 563–569.

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 the 42th Conference on Association for Computational Linguistic (ACL-2004)*, Barcelona, Spain.

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

Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 1–8, Barcelona, Spain, July. Association for Computational Linguistics.

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 *Proceedings of EMNLP 2004*.

V. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer.

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

Appendix 1: Generalized Boundary Selection Algorithm

Let O be the set of overlapping nodes of $\mathcal{PA}$, and NO the set of non overlapping nodes of $\mathcal{PA}$.
Let $subs^{(-1)}(A) = \{B | B \in 2^A, |B| = |A| - 1\}$.
Let $\hat{O} = subs^{(-1)}(O)$.

while($true$)
begin

1. $\mathcal{H} = \emptyset$
2. $\forall o \in \hat{O}$:
   (a) **If** o does not include any overlapping node pair
   then $\mathcal{H} = \mathcal{H} \cup \{o\}$
3. **If** $\mathcal{H} \neq \emptyset$ **then**:
   (a) Let $\hat{s}$ =$argmax_{o \in H}\ past_c(p_{NO \cup o})$, where $p_{NO \cup o}$ represents the node spanning tree compatible with o, and the $past_c(p_{NO \cup o})$ is the score provided by the $PAST$ SVM categorizer on it
   (b) **If** $past_c(\hat{s}) > 0$ **then RETURN**($\hat{s}$)
4. **If** $\hat{O} = \{\emptyset\}$ **then RETURN**(NO)
5. **Else**:
   (a) $\hat{O} = \hat{O} - \mathcal{H}$
   (b) $\hat{O} = \bigcup_{o \in \hat{O}} subs^{(-1)}(o)$

end