A Study on Convolution Kernels for Shallow Semantic Parsing

Alessandro Moschitti
University of Texas at Dallas
Human Language Technology Research Institute
Richardson, TX 75083-0688, USA
alessandro.moschitti@utdallas.edu

Abstract

In this paper we have designed and experimented novel convolution kernels for automatic classification of predicate arguments. Their main property is the ability to process structured representations. Support Vector Machines (SVMs), using a combination of such kernels and the flat feature kernel, classify PropBank predicate arguments with accuracy higher than the current argument classification *state-of-the-art*.

Additionally, experiments on FrameNet data have shown that SVMs are appealing for the classification of semantic roles even if the proposed kernels do not produce any improvement.

1 Introduction

Several linguistic theories, e.g. (Jackendoff, 1990) claim that semantic information in natural language texts is connected to syntactic structures. Hence, to deal with natural language semantics, the learning algorithm should be able to represent and process structured data. The classical solution adopted for such tasks is to convert syntax structures into flat feature representations which are suitable for a given learning model. The main drawback is that structures may not be properly represented by flat features.

In particular, these problems affect the processing of predicate argument structures annotated in PropBank (Kingsbury and Palmer, 2002) or FrameNet (Fillmore, 1982). Figure 1 shows an example of a predicate annotation in PropBank for the sentence: `"Paul gives a lecture in Rome"`. A predicate may be a verb or a noun or an adjective and most of the time Arg 0 is the *logical subject*, Arg 1 is the *logical object* and ArgM may indicate *locations*, as in our example.

FrameNet also describes predicate/argument structures but for this purpose it uses richer semantic structures called frames. These latter are schematic representations of situations involving various participants, properties and roles in which a word may be typically used. Frame elements or semantic roles are arguments of predicates called target words. In FrameNet, the argument names are local to a particular frame.

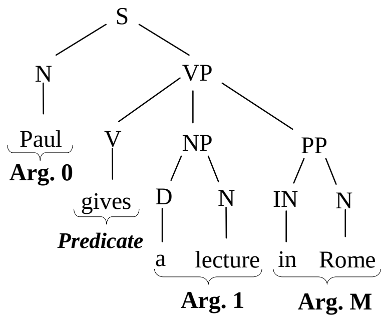

Figure 1: A predicate argument structure in a parse-tree representation.

Several machine learning approaches for argument identification and classification have been developed (Gildea and Jurasfky, 2002; Gildea and Palmer, 2002; Surdeanu et al., 2003; Hacioglu et al., 2003). Their common characteristic is the adoption of feature spaces that model predicate-argument structures in a flat representation. On the contrary, convolution kernels aim to capture structural information in term of sub-structures, providing a viable alternative to flat features.

In this paper, we select portions of syntactic trees, which include predicate/argument salient sub-structures, to define convolution kernels for the task of predicate argument classification. In particular, our kernels aim to (a) represent the relation between predicate and one of its arguments and (b) to capture the overall argument structure of the target predicate. Additionally, we define novel kernels as combinations of the above two with the polynomial kernel of standard flat features.

Experiments on Support Vector Machines using the above kernels show an improvement of the *state-of-the-art* for PropBank argument classification. On the contrary, FrameNet semantic parsing seems to not take advantage of the structural information provided by our kernels.

The remainder of this paper is organized as follows: Section 2 defines the Predicate Argument Extraction problem and the standard solution to solve it. In Section 3 we present our kernels whereas in Section 4 we show comparative results among SVMs using standard features and the proposed kernels. Finally, Section 5 summarizes the conclusions.

2 Predicate Argument Extraction: a standard approach

Given a sentence in natural language and the target predicates, all arguments have to be recognized. This problem can be divided into two subtasks: (a) the detection of the argument boundaries, i.e. all its compounding words 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 detection and 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 example, in Figure 1, for each combination of the predicate *give* with the nodes `N`, `S`, `VP`, `V`, `NP`, `PP`, `D` or `IN` the instances $F_{"give",a}$ are generated. In case the node a exactly covers *Paul*, *a lecture* or *in Rome*, it will be a positive instance otherwise it will be a negative one, e.g. $F_{"give","IN"}$.

To learn the argument classifiers the T^+ set can be re-organized 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 as it is simple and effective (Hacioglu et al., 2003). In the classification phase, given a sentence of the *test-set*, all its $F_{p,a}$ are generated and classified by each individual classifier. As a final decision, we select the argument associated with the maximum value among the scores provided by the SVMs, i.e. $argmax_{i \in S} \quad C_i$, where S is the target set of arguments.

- *Phrase Type*: This feature indicates the syntactic type of the phrase labeled as a predicate argument, e.g. NP for Arg_1.
- *Parse Tree Path*: This feature 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` $\uparrow$ `VP` $\downarrow$ `NP` for Arg_1.
- *Position*: Indicates if the constituent, i.e. the potential argument, appears before or after the predicate in the sentence, e.g. *after* for Arg_1 and *before* for Arg_0.
- *Voice*: This feature distinguishes between active or passive voice for the predicate phrase, e.g. *active* for every argument.
- *Head Word*: This feature contains the headword of the evaluated phrase. Case and morphological information are preserved, e.g. *lecture* for Arg_1.
- *Governing Category* indicates if an `NP` is dominated by a sentence phrase or by a verb phrase, e.g. the `NP` associated with Arg_1 is dominated by a `VP`.
- *Predicate Word*: This feature consists of two components: (1) the word itself, e.g. *gives* for all arguments; and (2) the lemma which represents the verb normalized to lower case and infinitive form, e.g. *give* for all arguments.

Table 1: Standard features extracted from the parse-tree in Figure 1.

2.1 Standard feature space

The discovery of relevant features is, as usual, a complex task, nevertheless, there is a common consensus on the basic features that should be adopted. These standard features, firstly proposed in (Gildea and Jurasfky, 2002), refer to a flat information derived from parse trees, i.e. *Phrase Type*, *Predicate Word*, *Head Word*, *Governing Category*, *Position* and *Voice*. Table 1 presents the standard features and exemplifies how they are extracted from the parse tree in Figure 1.

For example, the *Parse Tree Path* feature represents the path in the parse-tree between a predicate node and one of its argument nodes. It is expressed as a sequence of nonterminal labels linked by direction symbols (up or down), e.g. in Figure 1, `V`$\uparrow$`VP`$\downarrow$`NP` is the path between the predicate *to give* and the argument 1, *a lecture*. Two pairs $<p_1, a_1>$ and $<p_2, a_2>$ have two different *Path* features even if the paths differ only for a node in the parse-tree. This pre-

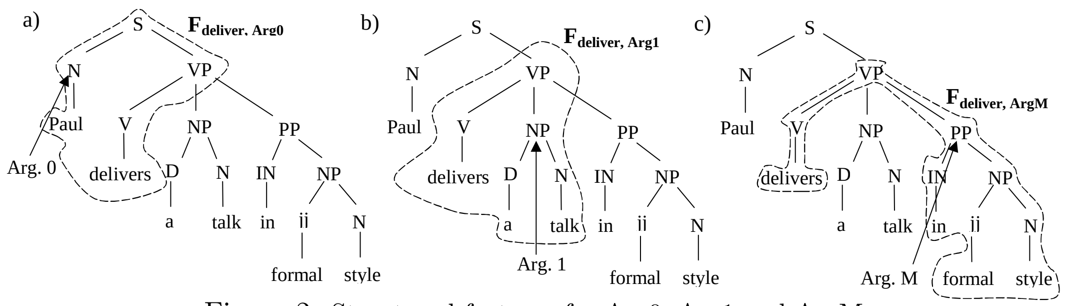

Figure 2: Structured features for Arg0, Arg1 and ArgM.

vents the learning algorithm to generalize well on unseen data. In order to address this problem, the next section describes a novel kernel space for predicate argument classification.

2.2 Support Vector Machine approach

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* $f_i \in F_z$ otherwise $z_i = 0$, i.e. the characteristic vector of the set F_z with respect to $\mathcal{F}$. If we choose as a kernel function the scalar product we obtain the linear kernel $K_L(F_x, F_z) = \vec{x} \cdot \vec{z}$.

Another function which is the current state-of-the-art of predicate argument classification is the polynomial kernel: $K_p(F_x, F_z) = (c + \vec{x} \cdot \vec{z})^d$, where c is a constant and d is the degree of the polynom.

3 Convolution Kernels for Semantic Parsing

We propose two different convolution kernels associated with two different predicate argument sub-structures: the first includes the target predicate with one of its arguments. We will show that it contains almost all the standard feature information. The second relates to the sub-categorization frame of verbs. In this case, the kernel function aims to cluster together verbal predicates which have the same syntactic realizations. This provides the classification algorithm with important clues about the possible set of arguments suited for the target syntactic structure.

3.1 Predicate/Argument Feature (PAF)

We consider the predicate argument structures annotated in PropBank or FrameNet as our semantic space. The smallest sub-structure which includes one predicate with only one of its arguments defines our structural feature. For example, Figure 2 illustrates the parse-tree of the sentence `"Paul delivers a talk in formal style"`. The circled substructures in (a), (b) and (c) are our semantic objects associated with the three arguments of the verb *to deliver*, i.e. <*deliver, Arg0*>, <*deliver, Arg1*> and <*deliver, ArgM*>. Note that each predicate/argument pair is associated with only one structure, i.e. $F_{p,a}$ contain only one of the circled sub-trees. Other important properties are the followings:

(1) The overall semantic feature space $\mathcal{F}$ contains sub-structures composed of syntactic information embodied by parse-tree dependencies and semantic information under the form of predicate/argument annotation.

(2) This solution is efficient as we have to classify as many nodes as the number of predicate arguments.

(3) A constituent cannot be part of two different arguments of the target predicate, i.e. there is no overlapping between the words of two arguments. Thus, two semantic structures F_{p_1,a_1} and F_{p_2,a_2}[1], associated with two different ar-

[1] $F_{p,a}$ was defined as the set of features of the object $<p, a>$. Since in our representations we have only one

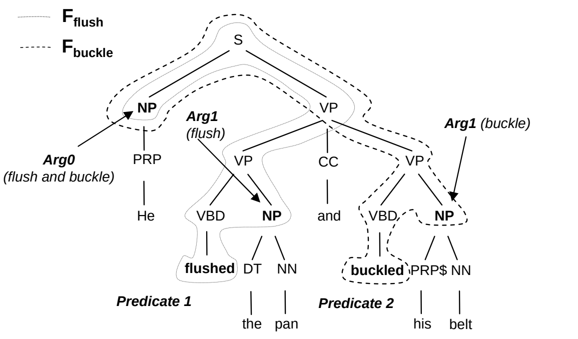

Figure 3: Sub-Categorization Features for two predicate argument structures.

guments, cannot be included one in the other. This property is important because a convolution kernel would not be effective to distinguish between an object and its sub-parts.

3.2 Sub-Categorization Feature (SCF)

The above object space aims to capture all the information between a predicate and one of its arguments. Its main drawback is that important structural information related to inter-argument dependencies is neglected. In order to solve this problem we define the Sub-Categorization Feature (SCF). This is the sub-parse tree which includes the sub-categorization frame of the target verbal predicate. For example, Figure 3 shows the parse tree of the sentence "`He flushed the pan and buckled his belt`". The solid line describes the SCF of the predicate *flush*, i.e. F_{flush} whereas the dashed line tailors the SCF of the predicate *buckle*, i.e. F_{buckle}. Note that SCFs are features for predicates, (i.e. they describe predicates) whereas PAF characterizes predicate/argument pairs.

Once semantic representations are defined, we need to design a kernel function to estimate the similarity between our objects. As suggested in Section 2 we can map them into vectors in $\Re^n$ and evaluate implicitly the scalar product among them.

3.3 Predicate/Argument structure Kernel (PAK)

Given the semantic objects defined in the previous section, we design a convolution kernel in a way similar to the parse-tree kernel proposed in (Collins and Duffy, 2002). We divide our mapping ϕ in two steps: (1) from the semantic structure space $\mathcal{F}$ (i.e. PAF or SCF objects) to the set of all their possible sub-structures

element in $F_{p,a}$ with an abuse of notation we use it to indicate the objects themselves.

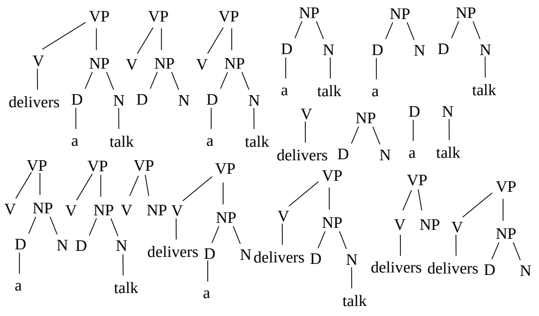

Figure 4: All 17 valid fragments of the semantic structure associated with Arg 1 of Figure 2.

$\mathcal{F}' = \{f'_1, .., f'_{|\mathcal{F}'|}\}$ and (2) from $\mathcal{F}'$ to $\Re^{|\mathcal{F}'|}$.

An example of features in $\mathcal{F}'$ is given in Figure 4 where the whole set of fragments, $F'_{deliver,Arg1}$, of the argument structure $F_{deliver,Arg1}$, is shown (see also Figure 2).

It is worth noting that the allowed sub-trees contain the entire (not partial) production rules. For instance, the sub-tree [`NP` [`D a`]] is excluded from the set of the Figure 4 since only a part of the production `NP` $\rightarrow$ `D N` is used in its generation. However, this constraint does not apply to the production `VP` $\rightarrow$ `V NP PP` along with the fragment [`VP` [`V NP`]] as the subtree [`VP` [`PP` [...]]] is not considered part of the semantic structure.

Thus, in step 1, an argument structure $F_{p,a}$ is mapped in a fragment set $F'_{p,a}$. In step 2, this latter is mapped into $\vec{x} = (x_1, .., x_{|\mathcal{F}'|}) \in \Re^{|\mathcal{F}'|}$, where x_i is equal to the number of times that f'_i occurs in $F'_{p,a}$[2].

In order to evaluate $K(\phi(F_x), \phi(F_z))$ without evaluating the feature vector $\vec{x}$ and $\vec{z}$ we define the indicator function $I_i(n) = 1$ if the sub-structure i is rooted at node n and 0 otherwise. It follows that $\phi_i(F_x) = \sum_{n \in N_x} I_i(n)$, where N_x is the set of the F_x's nodes. Therefore, the kernel can be written as:

$$K(\phi(F_x), \phi(F_z)) = \sum_{i=1}^{|\mathcal{F}'|} \Big(\sum_{n_x \in N_x} I_i(n_x)\Big)\Big(\sum_{n_z \in N_z} I_i(n_z)\Big)$$
$$= \sum_{n_x \in N_x} \sum_{n_z \in N_z} \sum_i I_i(n_x) I_i(n_z)$$

where N_x and N_z are the nodes in F_x and F_z, respectively. In (Collins and Duffy, 2002), it has been shown that $\sum_i I_i(n_x) I_i(n_z) = \Delta(n_x, n_z)$ can be computed in $O(|N_x| \times |N_z|)$ by the following recursive relation:

(1) if the productions at n_x and n_z are different then $\Delta(n_x, n_z) = 0$;

[2] A fragment can appear several times in a parse-tree, thus each fragment occurrence is considered as a different element in $F'_{p,a}$.

(2) if the productions at n_x and n_z are the same, and n_x and n_z are pre-terminals then $\Delta(n_x, n_z) = 1$;
(3) if the productions at n_x and n_z are the same, and n_x and n_z are not pre-terminals then

$$\Delta(n_x, n_z) = \prod_{j=1}^{nc(n_x)} (1 + \Delta(ch(n_x, j), ch(n_z, j))),$$

where $nc(n_x)$ is the number of the children of n_x and $ch(n, i)$ is the i-th child of the node n. Note that as the productions are the same $ch(n_x, i) = ch(n_z, i)$.

This kind of kernel has the drawback of assigning more weight to larger structures while the argument type does not strictly depend on the size of the argument (Moschitti and Bejan, 2004). To overcome this problem we can scale the relative importance of the tree fragments using a parameter λ for the cases (2) and (3), i.e. $\Delta(n_x, n_z) = \lambda$ and $\Delta(n_x, n_z) = \lambda \prod_{j=1}^{nc(n_x)} (1 + \Delta(ch(n_x, j), ch(n_z, j)))$ respectively.

It is worth noting that even if the above equations define a kernel function similar to the one proposed in (Collins and Duffy, 2002), the substructures on which it operates are different from the parse-tree kernel. For example, Figure 4 shows that structures such as [`VP` [`V`] [`NP`]], [`VP` [`V delivers`] [`NP`]] and [`VP` [`V`] [`NP` [`DT`] [`N`]]] are valid features, but these fragments (and many others) are not generated by a complete production, i.e. `VP` $\rightarrow$ `V NP PP`. As a consequence they would not be included in the parse-tree kernel of the sentence.

3.4 Comparison with Standard Features

In this section we compare standard features with the kernel based representation in order to derive useful indications for their use:

First, PAK estimates a similarity between two argument structures (i.e., PAF or SCF) by counting the number of sub-structures that are in common. As an example, the similarity between the two structures in Figure 2, $F_{\text{"}delivers\text{"},Arg0}$ and $F_{\text{"}delivers\text{"},Arg1}$, is equal to 1 since they have in common only the [`V delivers`] substructure. Such low value depends on the fact that different arguments tend to appear in different structures.

On the contrary, if two structures differ only for a few nodes (especially terminals or *near* terminal nodes) the similarity remains quite high. For example, if we change the tense of the verb *to deliver* (Figure 2) in *delivered*, the [`VP` [`V delivers`] [`NP`]] subtree will be transformed in [`VP` [`VBD delivered`] [`NP`]], where the `NP` is unchanged. Thus, the similarity with the previous structure will be quite high as: (1) the `NP` with all sub-parts will be matched and (2) the small difference will not highly affect the kernel norm and consequently the final score. The above property also holds for the SCF structures. For example, in Figure 3, $K_{PAK}(\phi(F_{flush}), \phi(F_{buckle}))$ is quite high as the two verbs have the same syntactic realization of their arguments. In general, flat features do not possess this conservative property. For example, the *Parse Tree Path* is very sensible to small changes of parse-trees, e.g. two predicates, expressed in different tenses, generate two different *Path* features.

Second, some information contained in the standard features is embedded in PAF: *Phrase Type*, *Predicate Word* and *Head Word* explicitly appear as structure fragments. For example, in Figure 4 are shown fragments like [`NP` [`DT`] [`N`]] or [`NP` [`DT` a] [`N` talk]] which explicitly encode the *Phrase Type* feature `NP` for the Arg 1 in Figure 2.b. The *Predicate Word* is represented by the fragment [`V delivers`] and the *Head Word* is encoded in [`N` talk]. The same is not true for SCF since it does not contain information about a specific argument. SCF, in fact, aims to characterize the predicate with respect to the overall argument structures rather than a specific pair $<p, a>$.

Third, *Governing Category*, *Position* and *Voice* features are not explicitly contained in both PAF and SCF. Nevertheless, SCF may allow the learning algorithm to detect the active/passive form of verbs.

Finally, from the above observations follows that the PAF representation may be used with PAK to classify arguments. On the contrary, SCF lacks important information, thus, alone it may be used only to classify verbs in syntactic categories. This suggests that SCF should be used in conjunction with standard features to boost their classification performance.

4 The Experiments

The aim of our experiments are twofold: On the one hand, we study if the PAF representation produces an accuracy higher than standard features. On the other hand, we study if SCF can be used to classify verbs according to their syntactic realization. Both the above aims can be carried out by combining PAF and SCF with the standard features. For this purpose we adopted two ways to combine kernels[3]: (1) $K = K_1 \cdot K_2$ and (2) $K = \gamma K_1 + K_2$. The resulting set of kernels used in the experiments is the following:

- K_{p^d} is the polynomial kernel with degree d over the standard features.
- K_{PAF} is obtained by using PAK function over the PAF structures.
- $K_{PAF+P} = \gamma \frac{K_{PAF}}{|K_{PAF}|} + \frac{K_{p^d}}{|K_{p^d}|}$, i.e. the sum between the normalized[4] PAF-based kernel and the normalized polynomial kernel.
- $K_{PAF \cdot P} = \frac{K_{PAF} \cdot K_{p^d}}{|K_{PAF}| \cdot |K_{p^d}|}$, i.e. the normalized product between the PAF-based kernel and the polynomial kernel.
- $K_{SCF+P} = \gamma \frac{K_{SCF}}{|K_{SCF}|} + \frac{K_{p^d}}{|K_{p^d}|}$, i.e. the summation between the normalized SCF-based kernel and the normalized polynomial kernel.
- $K_{SCF \cdot P} = \frac{K_{SCF} \cdot K_{p^d}}{|K_{SCF}| \cdot |K_{p^d}|}$, i.e. the normalized product between SCF-based kernel and the polynomial kernel.

4.1 Corpora set-up

The above kernels were experimented over two corpora: PropBank (`www.cis.upenn.edu/~ace`) along with Penn TreeBank[5] 2 (Marcus et al., 1993) and FrameNet.

PropBank contains about 53,700 sentences and a fixed split between training and testing which has been used in other researches e.g., (Gildea and Palmer, 2002; Surdeanu et al., 2003; Hacioglu et al., 2003). In this split, Sections from 02 to 21 are used for training, section 23 for testing and sections 1 and 22 as developing set. We considered all PropBank arguments[6] from *Arg0* to *Arg9*, *ArgA* and *ArgM* for a total of 122,774 and 7,359 arguments in training and testing respectively. It is worth noting that in the experiments we used the gold standard parsing from Penn TreeBank, thus our kernel structures are derived with high precision.

For the FrameNet corpus (`www.icsi.berkeley.edu/~framenet`) we extracted all 24,558 sentences from the 40 frames of Senseval 3 task (`www.senseval.org`) for the Automatic Labeling of Semantic Roles. We considered 18 of the most frequent roles and we mapped together those having the same name. Only verbs are selected to be predicates in our evaluations. Moreover, as it does not exist a fixed split between training and testing, we selected randomly 30% of sentences for testing and 70% for training. Additionally, 30% of training was used as a *validation-set*. The sentences were processed using Collins' parser (Collins, 1997) to generate parse-trees automatically.

4.2 Classification set-up

The classifier evaluations were carried out using the SVM-light software (Joachims, 1999) available at `svmlight.joachims.org` with the default polynomial kernel for standard feature evaluations. To process PAF and SCF, we implemented our own kernels and we used them inside SVM-light.

The classification performances were evaluated using the f_1 measure[7] for single arguments and the accuracy for the final multi-class classifier. This latter choice allows us to compare the results with previous literature works, e.g. (Gildea and Jurasfky, 2002; Surdeanu et al., 2003; Hacioglu et al., 2003).

For the evaluation of SVMs, we used the default regularization parameter (e.g., $C = 1$ for normalized kernels) and we tried a few cost-factor values (i.e., $j \in \{0.1, 1, 2, 3, 4, 5\}$) to adjust the rate between Precision and Recall. We chose parameters by evaluating SVM using K_{p^3} kernel over the *validation-set*. Both λ (see Section 3.3) and γ parameters were evaluated in a similar way by maximizing the performance of SVM using K_{PAF} and $\gamma \frac{K_{SCF}}{|K_{SCF}|} + \frac{K_{p^d}}{|K_{p^d}|}$ respectively. These parameters were adopted also for all the other kernels.

4.3 Kernel evaluations

To study the impact of our structural kernels we firstly derived the maximal accuracy reachable with standard features along with polynomial kernels. The multi-class accuracies, for PropBank and FrameNet using K_{p^d} with $d = 1, .., 5$, are shown in Figure 5. We note that (a) the highest performance is reached for $d = 3$, (b) for PropBank our maximal accuracy (90.5%)

[3]It can be proven that the resulting kernels still satisfy Mercer's conditions (Cristianini and Shawe-Taylor, 2000).

[4]To normalize a kernel $K(\vec{x}, \vec{z})$ we can divide it by $\sqrt{K(\vec{x}, \vec{x}) \cdot K(\vec{z}, \vec{z})}$.

[5]We point out that we removed from Penn TreeBank the function tags like *SBJ* and *TMP* as parsers usually are not able to provide this information.

[6]We noted that only *Arg0* to *Arg4* and *ArgM* contain enough training/testing data to affect the overall performance.

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

is substantially equal to the SVM performance (88%) obtained in (Hacioglu et al., 2003) with degree 2 and (c) the accuracy on FrameNet (85.2%) is higher than the best result obtained in literature, i.e. 82.0% in (Gildea and Palmer, 2002). This different outcome is due to a different task (we classify different roles) and a different classification algorithm. Moreover, we did not use the Frame information which is very important[8].

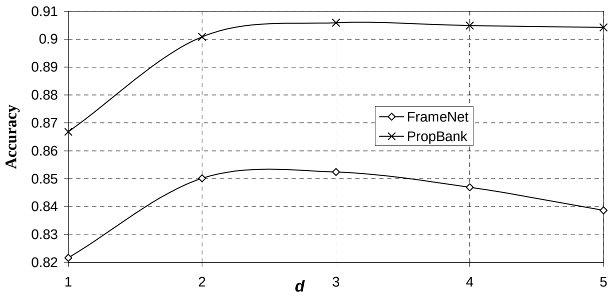

Figure 5: Multi-classifier accuracy according to different degrees of the polynomial kernel.

It is worth noting that the difference between linear and polynomial kernel is about 3-4 percent points for both PropBank and FrameNet. This remarkable difference can be easily explained by considering the meaning of standard features. For example, let us restrict the classification function C_{Arg0} to the two features *Voice* and *Position*. Without loss of generality we can assume: (a) *Voice*=1 if active and 0 if passive, and (b) *Position*=1 when the argument is after the predicate and 0 otherwise. To simplify the example, we also assume that if an argument precedes the target predicate it is a *subject*, otherwise it is an *object*[9]. It follows that a constituent is Arg0, i.e. $C_{Arg0} = 1$, *if* only one feature at a time is 1, otherwise it is not an $Arg0$, i.e. $C_{Arg0} = 0$. In other words, C_{Arg0} = *Position* XOR *Voice*, which is the classical example of a non-linear separable function that becomes separable in a superlinear space (Cristianini and Shawe-Taylor, 2000).

After it was established that the best kernel for standard features is K_{p^3}, we carried out all the other experiments using it in the kernel combinations. Table 2 and 3 show the single class (f_1 measure) as well as multi-class classifier (accuracy) performance for PropBank and FrameNet respectively. Each column of the two tables refers to a different kernel defined in the previous section. The overall meaning is discussed in the following points:

First, PAF alone has good performance, since in PropBank evaluation it outperforms the linear kernel (K_{p^1}), 88.7% vs. 86.7% whereas in FrameNet, it shows a similar performance 79.5% vs. 82.1% (compare tables with Figure 5). This suggests that PAF generates the same information as the standard features in a linear space. However, when a degree greater than 1 is used for standard features, PAF is outperformed[10].

Args	P^3	PAF	PAF+P	PAF·P	SCF+P	SCF·P
Arg0	90.8	88.3	90.6	90.5	94.6	94.7
Arg1	91.1	87.4	89.9	91.2	92.9	94.1
Arg2	80.0	68.5	77.5	74.7	77.4	82.0
Arg3	57.9	56.5	55.6	49.7	56.2	56.4
Arg4	70.5	68.7	71.2	62.7	69.6	71.1
ArgM	95.4	94.1	96.2	96.2	96.1	96.3
Acc.	90.5	88.7	90.2	90.4	92.4	93.2

Table 2: Evaluation of Kernels on PropBank.

Roles	P^3	PAF	PAF+P	PAF·P	SCF+P	SCF·P
agent	92.0	88.5	91.7	91.3	93.1	93.9
cause	59.7	16.1	41.6	27.7	42.6	57.3
degree	74.9	68.6	71.4	57.8	68.5	60.9
depict.	52.6	29.7	51.0	28.6	46.8	37.6
durat.	45.8	52.1	40.9	29.0	31.8	41.8
goal	85.9	78.6	85.3	82.8	84.0	85.3
instr.	67.9	46.8	62.8	55.8	59.6	64.1
mann.	81.0	81.9	81.2	78.6	77.8	77.8
Acc. 18 roles	85.2	79.5	84.6	81.6	83.8	84.2

Table 3: Evaluation of Kernels on FrameNet semantic roles.

Second, SCF improves the polynomial kernel (d = 3), i.e. the current *state-of-the-art*, of about 3 percent points on PropBank (column SCF·P). This suggests that (a) PAK can measure the similarity between two SCF structures and (b) the sub-categorization information provides effective clues about the expected argument type. The interesting consequence is that SCF together with PAK seems suitable to automatically cluster different verbs that have the same syntactic realization. We note also that to fully exploit the SCF information it is necessary to use a kernel product ($K_1 \cdot K_2$) combination rather than the sum ($K_1 + K_2$), e.g. column SCF+P.

Finally, the FrameNet results are completely different. No kernel combinations with both PAF and SCF produce an improvement. On

[8]Preliminary experiments indicate that SVMs can reach 90% by using the *frame* feature.

[9]Indeed, this is true in most part of the cases.

[10]Unfortunately the use of a polynomial kernel on top the tree fragments to generate the XOR functions seems not successful.

the contrary, the performance decreases, suggesting that the classifier is confused by this syntactic information. The main reason for the different outcomes is that PropBank arguments are different from semantic roles as they are an intermediate level between syntax and semantic, i.e. they are nearer to grammatical functions. In fact, in PropBank arguments are annotated consistently with syntactic alternations (see the *Annotation guidelines for PropBank* at `www.cis.upenn.edu/~ace`). On the contrary FrameNet roles represent the final semantic product and they are assigned according to semantic considerations rather than syntactic aspects. For example, *Cause* and *Agent* semantic roles have identical syntactic realizations. This prevents SCF to distinguish between them. Another minor reason may be the use of automatic parse-trees to extract PAF and SCF, even if preliminary experiments on automatic semantic shallow parsing of PropBank have shown no important differences versus semantic parsing which adopts Gold Standard parse-trees.

5 Conclusions

In this paper, we have experimented with SVMs using the two novel convolution kernels PAF and SCF which are designed for the semantic structures derived from PropBank and FrameNet corpora. Moreover, we have combined them with the polynomial kernel of standard features. The results have shown that:

First, SVMs using the above kernels are appealing for semantically parsing both corpora.

Second, PAF and SCF can be used to improve automatic classification of PropBank arguments as they provide clues about the predicate argument structure of the target verb. For example, SCF improves (a) the classification *state-of-the-art* (i.e. the polynomial kernel) of about 3 percent points and (b) the best literature result of about 5 percent points.

Third, additional work is needed to design kernels suitable to learn the deep semantic contained in FrameNet as it seems not sensible to both PAF and SCF information.

Finally, an analysis of SVMs using polynomial kernels over standard features has explained why they largely outperform linear classifiers based-on standard features.

In the future we plan to design other structures and combine them with SCF, PAF and standard features. In this vision the learning will be carried out on a set of structural features instead of a set of flat features. Other studies may relate to the use of SCF to generate verb clusters.

Acknowledgments

This research has been sponsored by the ARDA AQUAINT program. In addition, I would like to thank Professor Sanda Harabagiu for her advice, Adrian Cosmin Bejan for implementing the feature extractor and Paul Morărescu for processing the FrameNet data. Many thanks to the anonymous reviewers for their invaluable suggestions.

References

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

Michael Collins. 1997. Three generative, lexicalized models for statistical parsing. In *proceedings of the ACL-97*, pages 16–23, Somerset, New Jersey.

Nello Cristianini and John Shawe-Taylor. 2000. *An introduction to Support Vector Machines*. Cambridge University Press.

Charles J. Fillmore. 1982. Frame semantics. In *Linguistics in the Morning Calm*, pages 111–137.

Daniel Gildea and Daniel Jurasfky. 2002. Automatic labeling of semantic roles. *Computational Linguistic*.

Daniel Gildea and Martha Palmer. 2002. The necessity of parsing for predicate argument recognition. In *proceedings of ACL-02, Philadelphia, PA*.

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 *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*.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*.

Alessandro Moschitti and Cosmin Adrian Bejan. 2004. A semantic kernel for predicate argument classification. In *proceedings of CoNLL-04*, Boston, USA.

Kadri Hacioglu, Sameer Pradhan, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2003. Shallow Semantic Parsing Using Support Vector Machines. *TR-CSLR-2003-03, University of Colorado*.

Mihai Surdeanu, Sanda M. Harabagiu, John Williams, and John Aarseth. 2003. Using predicate-argument structures for information extraction. In *proceedings of ACL-03, Sapporo, Japan*.

V. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc.