A Semantic Kernel for Predicate Argument Classification

Alessandro Moschitti and **Cosmin Adrian Bejan**
University of Texas at Dallas
Human Language Technology Research Institute
Richardson, TX 75083-0688, USA
alessandro.moschitti@utdallas.edu
ady@hlt.utdallas.edu

Abstract

Automatically deriving semantic structures from text is a challenging task for machine learning. The flat feature representations, usually used in learning models, can only partially describe structured data. This makes difficult the processing of the semantic information that is embedded into parse-trees.

In this paper a new kernel for automatic classification of predicate arguments has been designed and experimented. It is based on subparse-trees annotated with predicate argument information from PropBank corpus. This kernel, exploiting the convolution properties of the parse-tree kernel, enables us to learn which syntactic structures can be associated with the arguments defined in PropBank. Support Vector Machines (SVMs) using such a kernel classify arguments with a better accuracy than SVMs based on linear kernel.

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 in a flat feature representation, which is suitable for a given learning model. The main drawback is structures may not be properly represented by flat features as: (1) these latter may not be able to capture the required properties or (2) the feature designer may not know what structure properties enable the processing of semantic information.

In particular, these problems arise for semantic information represented via predicate argument structures defined on syntactic parse trees. For example, Figure 1 shows the parse tree of the sentence: `"Paul gives a lecture in Rome"` along with the annotation of predicate arguments.

A predicate may be a verb or a noun or an adjective whereas generally Arg 0 stands for *agent*, Arg 1 for *direct object* or *theme* or *patient* and ArgM may indicate *locations*, as in our example. A standard for predicate argument annotation is provided in the PropBank project (Kingsbury and Palmer, 2002). It has produced one million word corpus annotated with predicate-argument structures on top of the Penn Treebank 2 Wall Street Journal texts. In this way, for a large number of the Penn TreeBank parse-trees, there are available predicate annotations in a style similar to that shown in Figure 1.

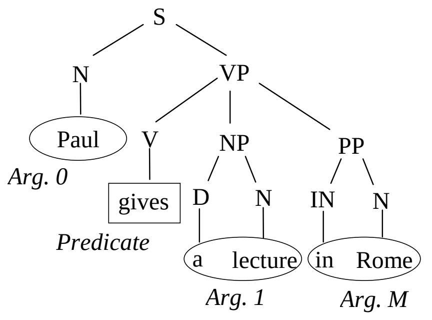

Figure 1: Predicate arguments in a parse-tree representation.

In PropBank only verbs are considered to be predicates whereas arguments are labeled sequentially from Arg 0 to Arg 9[1]. In addition to these core arguments, adjunctive arguments are marked up. They include functional tags, e.g. ArgM-DIR indicates a directional, ArgM-LOC indicates a locative and ArgM-TMP stands for a temporal. An example of PropBank markup is:

[1]Other arguments are: Arg 2 for *indirect object* or *benefactive* or *instrument* or *attribute* or *end state*, Arg 3 for *start point* or *benefactive* or *attribute*, Arg4 for *end point* and so on.

$[_{Arg_0^1}$ Analysts] have been $[_{predicate_1}$ expecting] $[_{Arg_1^1}$ a GM-Jaguar pact] that would $[_{predicate_2}$ give] $[_{Arg_2^2}$ the U.S. car maker] $[_{Arg_1^2}$ an eventual 30% state in the British Company].

Automatically recognizing the boundaries and classifying the type of arguments allows Natural Language Processing systems (e.g. Information Extraction, Question Answering or Summarization) to answer questions such as ”Who”, ”When”, ”What”, ”Where”, ”Why”, and so on.

Given the importance of this task for Natural Language Processing applications, several machine learning approaches for argument identification and classification have been developed (Gildea and Jurasky, 2002; Surdeanu et al., 2003; Hacioglu et al., 2003; Chen and Rambow, 2003; Gildea and Hockenmaier, 2003). Their common characteristic is the adoption of feature spaces that model predicate-argument structures in a flat representation. The major problem of this choice is that there is no linguistic theory that supports the selection of syntactic features to recognize semantic structures. As a consequence, researchers are still trying to extend the basic features with other ones, e.g. (Surdeanu et al., 2003), to improve the flat feature space.

Convolution kernels are a viable alternative to flat feature representation that aims to capture the structural information in term of sub-structures. The kernel functions can be used to measure *similarities* between two objects without explicitly evaluating the object features. That is, we do not need to understand which syntactic feature may be suited for representing semantic data. We need *only* to define the similarity function between two semantic structures. An example of convolution kernel on the parse-tree space is given in (Collins and Duffy, 2002). The aim was to design a novel syntactic parser by looking at the similarity between the testing parse-trees and the correct parse-trees available for training.

In this paper, we define a kernel in a semantic structure space to learn the classification function of predicate arguments. The main idea is to select portions of syntactic/semantic trees that include the target <predicate, argument> pair and to define a kernel function between these objects. If our similarity function is well defined the learning model will converge and provide an effective argument classification.

Experiments on PropBank data show not only that Support Vector Machines (SVMs) trained with the proposed semantic kernel converge but also that they have a higher accuracy than SVMs trained with a linear kernel on the standard features proposed in (Gildea and Jurasky, 2002). This provides a piece of evidence that convolution kernel can be used to learn semantic linguistic structures. Moreover, interesting research lines on the use of kernel for NLP are enabled, e.g. question classification in Question/Answering or automatic template designing in Information Extraction.

The remaining 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 approach based on the parse-tree kernel whereas in Section 4 we show our comparative results between SVMs using standard features and the proposed kernel. Finally, Section 5 summarizes the conclusions.

2 Automatic Predicate-Argument extraction

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

A direct 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. $F_{"give","IN"}$.

The above T^+ and T^- sets 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 (Pradhan 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[2], i.e. $argmax_{i \in S} \quad C_i$, where S is the target set of arguments.

[2]This is a basic method to pass from binary categorization

2.1 Standard feature space

The discovering 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 Jurasky, 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 a given parse tree.

- *Phrase Type*: This feature indicates the syntactic type of the phrase labeled as a predicate argument, e.g. NP for Arg_1 in Figure 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. $\mathsf{V} \uparrow \mathsf{VP} \downarrow \mathsf{NP}$ for Arg_1 in Figure 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 (see Figure 1).
- *Voice*: This feature distinguishes between active or passive voice for the predicate phrase, e.g. *active* for every argument (see Figure 1).
- *Head Word*: This feature contains the head word of the evaluated phrase. Case and morphological information are preserved, e.g. *lecture* for Arg_1 (see Figure 1).
- *Governing Category*: This feature applies to noun phrases only, and it indicates if the NP is dominated by a sentence phrase (typical for subject arguments with active voice predicates), or by a verb phrase (typical for object arguments), e.g. the NP associated with Arg_1 is dominated by a verbal phrase VP (see Figure 1).
- *Predicate Word*: In our implementation this feature consists of two components: (1) the word itself with the case and morphological information preserved, 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 (see Figure 1).

Table 1: Standard features extracted from parse-trees.

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, $\mathsf{V}\uparrow\mathsf{VP}\downarrow\mathsf{NP}$ is the path between the predicate *to give* and the argument 1, *a lecture*. If two pairs $<p_1, a_1>$ and $<p_2, a_2>$ have a *Path* that differs even for one character (e.g. a node in the parse-tree) the match will not be carried out, preventing the learning algorithm to generalize well on unseen data. In order to address also this problem, next section describes a novel kernel space for predicate argument classification.

into a multi-class categorization problem; several optimization have been proposed, e.g. (Goh et al., 2001).

3 A semantic kernel for argument classification

We consider the predicate argument structures annotated in PropBank as our semantic space. Many semantic structures may constitute the objects of our space. Some possibilities are: (a) the selection of the whole sentence parse-tree, in which the target predicate is contained or (b) the selection of the sub-tree that encloses the whole predicate annotation (i.e. all its arguments). However, both choices would cause an exponential explosion on the potential sub-parse-trees that have to be classified during the testing phase. In fact, during this phase we do not know which are the arguments associated with a predicate. Thus, we need to build all the possible structures, which contain groups of potential arguments for the target predicate. More in detail, assuming that S is the set of PropBank argument types, and m is the maximum number of entries that the target predicate can have, we have to evaluate $\binom{|S|}{m}$ argument combinations for each target predicate.

In order to define an efficient semantic space we select as objects only the minimal sub-structures that include one predicate with only one of its arguments. For example, Figure 2 illustrates the parse-tree of the sentence `"Paul delivers a lecture 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>*. In this formulation, only one of the above structures is associated with each predicate/argument pair, i.e. $F_{p,a}$ contain only one of the circled sub-trees.

We note that our approach has the following properties:

- The overall semantic feature space F contain substructures composed of syntactic information embodied by parse-tree dependencies and semantic information under the form of predicate/argument annotation.
- This solution is efficient as we have to classify at maximum $|\mathcal{A}|$ nodes for each predicate, i.e. the set of the parse-tree nodes of a testing sentence.
- 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}[3], asso-

[3] $F_{p,a}$ was defined as the set of features of our objects $<p, a>$. Since in our kernel we have only one element in $F_{p,a}$ with an abuse of notation we use it to indicate the objects themselves.

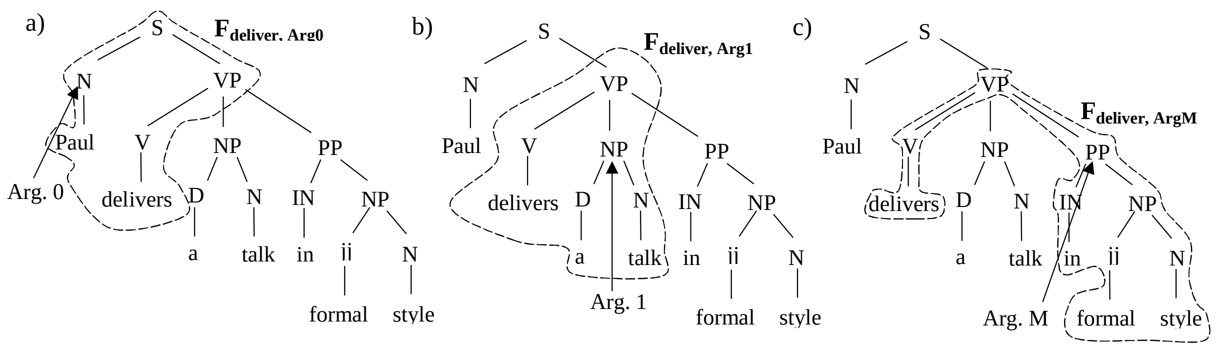

Figure 2: Semantic feature space for predicate argument classification.

ciated with two different arguments, 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.

Once defined our semantic space we need to design a kernel function to measure a similarity between two objects. These latter may still be seen as described by complex features but such a similarity is carried out avoiding the explicit feature computation. For this purpose we define a mapping $\phi : F \to F'$ such as:

$$\vec{x} = (x_1, ..., x_{|F|}) \to \phi(\vec{x}) = (\phi_1(\vec{x}), .., \phi_{|F'|}(\vec{x})),$$

where F' allows us to design an efficient semantic kernel $K(\vec{x}, \vec{z}) = <\phi(\vec{x}) \cdot \phi(\vec{z})>$.

3.1 The Semantic Kernel (SK)

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). Our feature set F' is the set of all possible substructures (enumerated from 1 to $|F'|$) of the semantic objects extracted from PropBank. For example, Figure 3 illustrates all valid fragments of the semantic structure $F_{deliver,Arg1}$ (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 3 since only a part of the production NP $\to$ D N is used in its generation. However, this constraint does not apply to the production VP $\to$ V NP PP along with the fragment [VP [V NP]] as the subtree [VP [PP [...]]] is not considered part of the semantic structure.

Even if the cardinality of F' will be very large the evaluation of the kernel function is polynomial in the number of parse-tree nodes.

More precisely, a semantic structure $\vec{x}$ is mapped in $\phi(\vec{x}) = (h_1(\vec{x}), h_2(\vec{x}), ...)$, where the feature function $h_i(\vec{x})$ simply counts the number of times that the i-th sub-structure of the training data appears in $\vec{x}$. Let

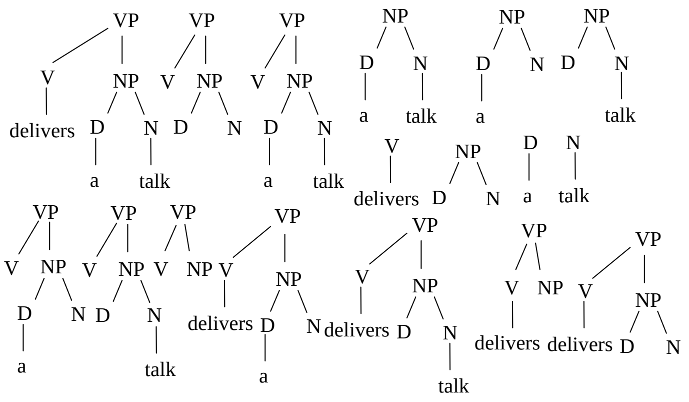

Figure 3: All 17 valid fragments of the semantic structure associated with Arg 1 (see Figure 2).

$I_i(n)$ be the indicator function: 1 if the sub-structure i is rooted at node n and 0 otherwise. It follows that $h(\vec{x}) = \sum_{n \in N} I_i(n)$, where N is the set of the $\vec{x}$'s nodes. Therefore, the kernel[4] function is:

$$K(\vec{x}, \vec{z}) = \vec{h}(\vec{x}) \cdot \vec{h}(\vec{z}) =$$

$$= \sum_i \Big(\sum_{n_x \in N_x} I_i(n_x) \Big) \Big(\sum_{n_z \in N_z} I_i(n_z) \Big) =$$

$$= \sum_{n_x \in N_x} \sum_{n_z \in N_z} \sum_i I_i(n_x) I_i(n_z) \qquad (1)$$

where N_x and N_z are the nodes in x and z, respectively. In (Collins and Duffy, 2002), it has been shown that Eq. 1 can be computed in $O(|N_x| \times |N_z|)$ by evaluating $\Delta(n_x, n_z) = \sum_i I_i(n_x) I_i(n_z)$ with the following recursive equations:

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

[4]Additionally, we carried out the normalization in the kernel space, thus the final kernel is $K'(\vec{x}, \vec{z}) = \frac{K(\vec{x},\vec{z})}{\sqrt{K(\vec{x},\vec{x}) \times K(\vec{z},\vec{z})}}$.

- if the production at n_x and n_z are the same, and n_x and n_z are pre-terminals then

$$\Delta(n_x, n_z) = 1; \quad (2)$$

- if the production at n_x and n_z are the same, and n_x and n_z are not pre-terminals then

$$\Delta(n_x, n_z) = \prod_{j=1}^{nc(n_x)} (1 + \Delta(ch(n_x, j), ch(n_z, j))), \quad (3)$$

  where $nc(n_x)$ is the number of 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 depend at all on the size of its structure. In fact two sentences such as:
(1) $[_{Arg_0}$ Paul $][_{predicate}$ delivers $]$ $[_{Arg_1}$ a lecture] and
(2) $[_{Arg_0}$ Paul $][_{predicate}$ delivers $][_{Arg_1}$ a plan on the detection of theorist groups active in the North Iraq]
have the same argument type with a very different size. To overcome this problem we can scale the relative importance of the tree fragments with their size. For this purpose a parameter λ is introduced in equations 2 and 3 obtaining:

$$\Delta(n_x, n_z) = \lambda \quad (4)$$

$$\Delta(n_x, n_z) = \lambda \prod_{j=1}^{nc(n_x)} (1 + \Delta(ch(n_x, j), ch(n_z, j))) \quad (5)$$

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 SK operates are different from the parse-tree kernel. For example, Figure 3 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 are not included in the parse-tree kernel representation of the sentence.

3.2 Comparison with Standard Features

We have synthesized the comparison between standard features and the SK representation in the following points. First, SK estimates a similarity between two semantic structures 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 argument types tend to appear in different structures.

On the contrary, if two structures differ only for a few nodes (especially terminal 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. This conservative property does not apply to the *Parse Tree Path* feature which is very sensible to small changes in the tree-structure, e.g. two predicates, expressed in different tenses, generate two different *Path* features.

Second, some information contained in the standard features is embedded in SK: *Phrase Type*, *Predicate Word* and *Head Word* explicitly appear as structure fragments. For example, in Figure 3 are shown fragments like [NP [DT] [N]] or [NP [DT a] [N talk]] which explicitly encode the *Phrase Type* feature NP for Arg 1 in Figure 2.b. The *Predicate Word* is represented by the fragment [V delivers] and the *Head Word* is present as [N talk].

Finally, *Governing Category*, *Position* and *Voice* cannot be expressed by SK. This suggests that a combination of the flat features (especially the named entity class (Surdeanu et al., 2003)) with SK could furthermore improve the predicate argument representation.

4 The Experiments

For the experiments, we used PropBank (www.cis.upenn.edu/∼ace) along with PennTreeBank[5] 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 (Gildea and Jurasky, 2002; Surdeanu et al., 2003; Hacioglu et al., 2003; Chen and Rambow, 2003; Gildea and Hockenmaier, 2003; Gildea and Palmer, 2002; Pradhan 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 from *Arg0* to *Arg9*, *ArgA* and *ArgM* even if only *Arg0* from *Arg4* and *ArgM* contain enough training/testing data to affect the global performance. In Table 2 some characteristics of the corpus used in our experiments are reported.

The classifier evaluations were carried out using the SVM-light software (Joachims, 1999) available at http://svmlight.joachims.org/ with the default linear kernel for the standard feature evaluations.

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

Table 2: Characteristics of the corpus used in the experiments.

	Number of Args		Number of unique
	train.	test-set	Std. features
Arg0	34,955	2,030	12,520
Arg1	44,369	2,714	14,442
Arg2	10,491	579	6,422
Arg3	2,028	106	1,591
Arg4	1,611	67	918
ArgM	30,464	1,930	7,647
Total	123,918	7,426	21,710

For processing our semantic structures, we implemented our own kernel and we used it inside SVM-light.

The classification performances were evaluated using the f_1 measure for single arguments as each of them has a different Precision and Recall and by using the accuracy for the final multi-class classifier as the global Precision = Recall = accuracy. The latter measure allows us to compare the results with previous literature works, e.g. (Gildea and Palmer, 2002; Surdeanu et al., 2003; Hacioglu et al., 2003; Chen and Rambow, 2003; Gildea and Hockenmaier, 2003).

To evaluate the effectiveness of our new kernel we divided the experiments in 3 steps:

- The evaluation of SVMs trained with standard features in a linear kernel, for comparison purposes.
- The estimation of the λ parameter (equations 4 and 5) for SK from the *validation-set* .
- The performance measurement of SVMs, using SK along with λ computed in the previous step.

Additionally, both Linear and SK kernels were evaluated using different percentages of training data to compare the gradients of their learning curves.

4.1 SVM performance on Linear and Semantic Kernel

The evaluation of SVMs using a linear kernel on the standard features did not raise particular problems. We used the default regularization parameter (i.e., $C = 1$ 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. Given the huge amount of training data, we used only 30% of training-set in these validation experiments. Once the parameters were derived, we learned 6 different classifiers (one for each role) and measured their performances on the *test-set*.

For SVM, using the Semantic Kernel, we derived that a *good* λ parameter for the *validation-set* is 0.4. In Figure 4 we report the curves, f_1 function of λ, for the 3 largest (in term of training examples) arguments on the *test-set*.

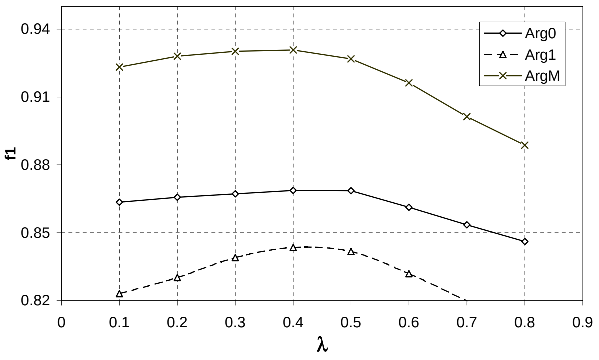

Figure 4: SVM f_1 for *Arg0*, *Arg1* and *ArgM* with respect to different λ values.

We note that the maximal value from the *validation-set* is also the maximal value from the *test-set* for every argument. This suggests that: (a) it is *easy* to detect an optimal parameter and (b) there is a common (to all arguments) λ-value which defines how much the size of two structures impacts on their similarity. Moreover, some experiments using λ greater than 1 have shown a remarkable decrease in performance, i.e. a correct λ seems to be essential to obtain a *good* generalization[6] of the *training-set*.

Table 3: f_1 of SVMs using linear and semantic kernel compared to literature models for argument classification.

Args	SVM	SVM	Prob.	C4.5	
	STD	SK	STD	STD	EXT
Arg0	87.79	88.35	-	-	-
Arg1	82.43	86.25	-	-	-
Arg2	54.10	68.52	-	-	-
Arg3	31.65	49.46	-	-	-
Arg4	62.81	66.66	-	-	-
ArgM	91.97	94.07	-	-	-
multi-class accuracy	84.07	86.78	82.8	78.76	83.74

Table 3 reports the performances of SVM trained with the standard features (STD column) and with the Semantic Kernel (SK column). In columns Prob. and C4.5 are reported the results for argument classification achieved in (Gildea and Palmer, 2002) and (Surdeanu et al., 2003). This latter used C4.5 model on standard feature set (STD sub-column) and on an extended feature set (EXT sub-column). We note that: (a) SVM performs better than the probabilistic approach and C4.5 learning model independently of the adopted features and (b) the Semantic Kernel considerably improves the standard feature set.

In order to investigate if SK generalizes better than the

[6]For example, $\lambda = 1$ would generate low kernel values between *small* and large *structures*. This is in contrast with the observation in Section 3.1, i.e. argument type is independent of its constituent size.

linear kernel, we measured the performances by selecting different percentages of training data. Figure 5 shows the curves for the three roles *Arg0*, *Arg1* and *ArgM*, respectively for linear and semantic kernel whereas Figure 6 shows the multi-class classifier $f1$ plots.

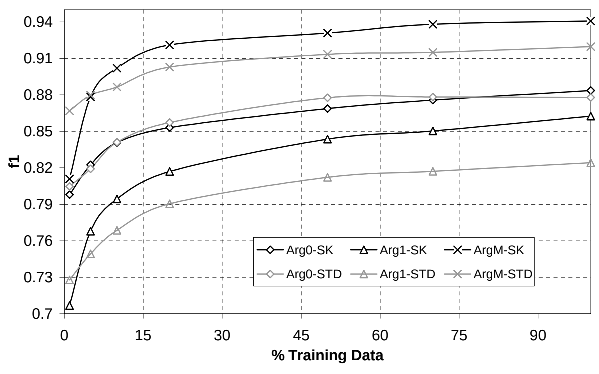

Figure 5: *Arg0*, *Arg1* and *ArgM* evaluations over SK and the linear kernel of standard features with respect to different percentages of training data.

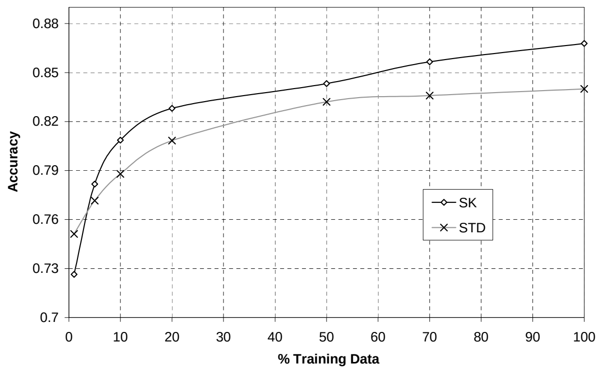

Figure 6: Accuracy of the multi-class classifier using standard features and SK with respect to different percentages of training data.

We note that not only SK produces higher accuracy but also the gradient of the learning curves is higher: for example, Figure 6 shows that with only 20% of training data, SVM using SK approaches the accuracy of SVM trained with all data on standard features.

Additionally, we carried out some preliminary experiments for argument identification (boundary detection), but the learning algorithm was not able to converge. In fact, for this task the non-inclusion property (discussed in Section 3) does not hold. A constituent a_i, which has incorrect boundaries, can include or be included in the correct argument a_c. Thus, the similarity $K(a_i, a_c)$ between a_i and a_c is quite high preventing the algorithm to learn the structures of correct arguments.

4.2 Discussion and Related Work

The material of the previous sections requires a discussion of the following points: firstly, in Section 3.2 we have noted that some standard features are explicitly coded in SK but *Governing Category*, *Position* and *Voice* features are not expressible as a single fragment of a semantic structure. For example, to derive the *Position* of an argument relatively to the target predicate is required a visit of the tree. No parse-tree information, i.e. node tags or edges, explicitly indicates this feature. A similar rationale applies to *Governing Category* and *Voice*, even if for the latter some tree fragments may code the *to be* feature. Since these three features have been proved important for role classification we argue that either (a) SK implicitly produces this kind of information or (b) SK is able to provide a different but equally effective information which allows it to perform better than the standard features. In this latter case, it would be interesting to study which features can be backported from SK to the linear kernel to obtain a fast and improved system (Cumby and Roth, 2003). As an example, the fragment [VP [V NP]] defines a sort of sub-categorization frame that may be used to cluster together syntactically similar *verbs*.

Secondly, it is worth noting that we compared SK against a linear kernel of standard features. A recent study, (Pradhan et al., 2003), has suggested that a polynomial kernel with $degree = 2$ performs better than the linear one. Using such a kernel, the authors obtained 88% in classification but we should take into account that they also used a larger set of flat features, i.e. *sub-categorization* information (e.g. VP $\rightarrow$ V NP PP for the tree in Figure 1), *Named Entity Classes* and a *Partial Path* feature.

Thirdly, this is one of the first massive use of convolution kernels for Natural Language Processing tasks, we trained SK and tested it on 123,918 and 7,426 arguments, respectively. For training each large argument (in term of instances) were required more than 1.5 billion of kernel iterations. This was a little time consuming (about a couple of days for each argument on a Intel Pentium 4, 1,70 GHz, 512 Mbytes Ram) as the SK computation complexity is quadratic in the number of semantic structure nodes[7]. This prevented us to carry out cross/fold validation. An important aspect is that a recent paper (Vishwanathan and Smola, 2002) assesses that the tree-kernel complexity can be reduced to linear one; this would make our approach largely applicable.

Finally, there is a considerable work in Natural Language Processing oriented kernel (Collins and Duffy, 2002; Lodhi et al., 2000; Gärtner, 2003; Cumby and Roth, 2003; Zelenko et al., 2003) about string, parse-

[7]More precisely, it is O($|F_{p,a}|^2$) where $F_{p,a}$ is the largest semantic structure of the training data.

tree, graph, and relational kernels but, to our knowledge, none of them was used to derive semantic information on the form of predicate argument structures. In particular, (Cristianini et al., 2001; Kandola et al., 2003) address the problem of semantic similarity between two terms by using, respectively, document sets as term context and the latent semantic indexing. Both techniques attempt to cluster together terms that express the same meaning. This is quite different in means and purpose of our approach that derives more specific semantic information expressed as argument/predicate relations.

5 Conclusions

In this paper, we have experimented an original kernel based on semantic structures from PropBank corpus. The results have shown that:

- the Semantic Kernel (SK) can be adopted to classify predicate arguments defined in PropBank;
- SVMs using SK performs better than SVMs trained with the linear kernel of standard features; and
- the higher gradient in the *accuracy*/*training percentage* plots shows that SK generalizes better than the linear kernel.

Finally, SK suggests that some features, contained in the fragments of semantic structures, should be backported in a flat feature space. Conversely, the good performance of the linear kernel suggests that standard features, e.g. *Head Word*, *Predicate Word* should be emphasized in the definition of a convolution kernel for argument classification. Moreover, other selections of predicate/argument substructures (able to capture different linguistic relations) as well as kernel combinations (e.g. flat features with SK) could furthermore improve semantic shallow parsing.

6 Acknowledgements

This research has been sponsored by the ARDA AQUAINT program. In addition, we would like to thank prof. Sanda Harabagiu to support us with interesting advices. Many thanks to the anonymous reviewers for their professional and committed suggestions.

References

John Chen and Owen Rambow. 2003. Use of deep linguistic features for the recognition and labeling of semantic arguments. *In Proceedings EMNLP03*.

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

Nello Cristianini, John Shawe-Taylor, and Huma Lodhi. 2001. Latent semantic kernels. *In Proceedings of ICML01*, pages 66–73, Williams College, US. Morgan Kaufmann Publishers, San Francisco, US.

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

Thomas Gärtner. 2003. A survey of kernels for structured data. *SIGKDD Explor. Newsl.*, 5(1):49–58.

Daniel Gildea and Julia Hockenmaier. 2003. Identifying semantic roles using combinatory categorial grammar. *In Proceedings of EMNLP03*.

Daniel Gildea and Daniel Jurasky. 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 ACL02, Philadelphia, PA*.

King-Shy Goh, Edward Chang, and Kwang-Ting Cheng. 2001. SVM binary classifier ensembles for image classification. *Proceedings of CIKM01*, pages 395–402.

Kadri Hacioglu, Sameer Pradhan, Wayne Ward, Jim Martin, and Dan Jurafsky. 2003. Shallow semantic parsing using Support Vector Machines. Technical report.

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

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

J. Kandola, N. Cristianini, and J. Shawe-Taylor. 2003. Learning semantic similarity. *In Advances in Neural Information Processing Systems, volume 15*.

Paul Kingsbury and Martha Palmer. 2002. From TreeBank to PropBank. *In Proceedings of LREC02, Las Palmas, Spain.*

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.

Sameer Pradhan, Kadri Hacioglu, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2003. Semantic role parsing: Adding semantic structure to unstructured text. *In Proceedings of ICDM03*.

Mihai Surdeanu, Sanda M. Harabagiu, John Williams, and John Aarseth. 2003. Using predicate-argument structures for information extraction. *In Proceedings of ACL03, Sapporo, Japan*.

S.V.N. Vishwanathan and A.J. Smola. 2002. Fast kernels on strings and trees. *In Proceedings of Neural Information Processing Systems*.

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