Tree Kernel Engineering for Proposition Re-ranking

Alessandro Moschitti, Daniele Pighin, and Roberto Basili
{moschitti,basili}@info.uniroma2.it
daniele.pighin@gmail.com

Department of Computer Science
University of Rome "Tor Vergata", Italy

Abstract. Recent work on the design of automatic systems for semantic role labeling has shown that such task is complex from both modeling and implementation point of views. Tree kernels alleviate such complexity as kernel functions generate features automatically and require less software development for data pre-processing. In this paper, we study several tree kernel approaches for boundary detection, argument classification and, most notably, proposition re-ranking. The comparative experiments on Support Vector Machines with such kernels on the CoNLL 2005 dataset show that very simple tree manipulations trigger automatic feature engineering that highly improves accuracy and efficiency in every SRL phase.

1 Introduction

A lot of attention has been recently devoted to the design of systems for the automatic labeling of semantic roles (SRL) as defined in two important projects: FrameNet [1], inspired by Frame Semantics, and PropBank [2] based on Levin's verb classes. SRL is a complex task consisting in the recognition of predicate argument structures within natural language sentences.

Research on the design of automatic SRL systems has shown that (shallow or deep) syntactic information is necessary to achieve a good accuracy, e.g. [3,4]. A careful analysis of literature features encoding such information reveals that most of them are fragments of syntactic trees of training sentences. Thus, a natural way to represent them is the adoption of tree kernels as described in [5]. Tree kernels show important advantages: first, we can implement them very quickly as the feature extractor module only requires the writing of the procedure for subtree extraction. In contrast, traditional SRL systems are based on the extraction of more than thirty features [6], which require the writing of at least thirty different procedures. Second, combining tree kernels with a traditional attribute-value SRL system allows us to obtain a more accurate system. Usually the combination of two traditional systems (based on the same machine learning model) does not result in an improvement as their features are more or less equivalent as shown in [4]. Finally, the study of the effective structural features

can inspire the design of novel linear features, which can be used with a more efficient model (i. e. linear SVMs).

In this paper, we carry out tree kernel engineering [7,8] to increase the accuracy of the boundary detection, argument classification and proposition re-ranking steps. In the first two cases (Section 2.1), the engineering approach relates to marking the nodes of the encoding subtrees to generate substructures more strictly correlated with a particular argument, boundary or predicate. For the latter case (Section 2.2), i. e. proposition re-ranking, we try both marking large parts of the tree that dominates the whole predicate argument structure and utterly reworking the syntactic structure. Our extensive experimentation of the proposed tree kernels with Support Vector Machines on the CoNLL 2005 data set provides interesting insights on the design of performant SRL systems (Section 3).

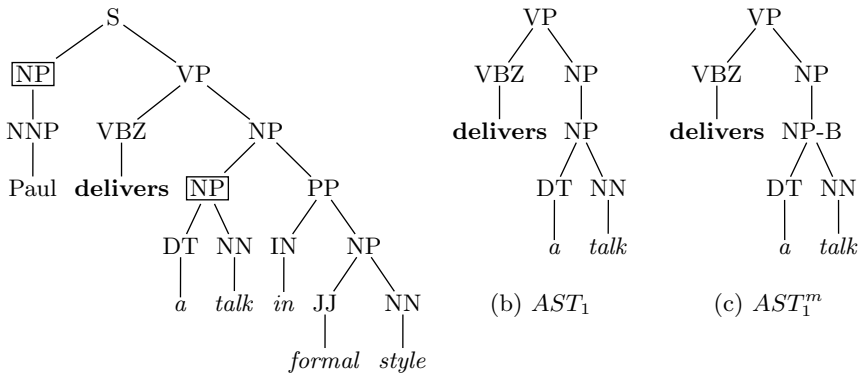
2 A Model for Semantic Role Labeling

The SRL approach that we adopt is based on the deep syntactic parse [9] of the sentence that we intend to semantically annotate. The standard algorithm concerns the classification of tree node pairs $\langle p, a\rangle$, where p is the node that exactly dominates the target predicate and a is the node dominating a potential argument. If $\langle p, a\rangle$ is selected as an argument, then the leaves of the tree rooted in a will be considered as the words constituting such argument. There are hundreds of pairs in a sentence, thus, if we use training corpora containing hundreds of thousands of sentences, we have to deal with millions of instances.

To limit such complexity, we can divide the problem in two subtasks: (a) boundary detection, in which a single classifier is trained on many instances to detect if a node is an argument or not, i. e. if the sequence of words dominated by the target node constitutes a correct boundary; and (b) argument classification, in which only the set of nodes corresponding to correct boundaries are considered. These can be used to train a multiclassifier that, for such nodes, selects the most appropriate labeling. For example, n classifiers can be combined with a One-vs-All approach, selecting for each argument node the role associated with the maximum among the n scores provided by the individual role classifiers.

The main advantage of this approach is the use of just one computationally expensive classifier, i. e. the one for boundary detection. Regarding the feature representation of $\langle p, a\rangle$, we can extract syntactic fragments from the sentence-parse tree proposed in [3], e.g. the *Phrase Type* or *Predicate Word*. An alternative to the manual fragment extraction is the use of Tree Kernels as suggested in [5]. Tree kernels are especially useful when the manual design of features is made complex by the use of a re-ranking module. This has been shown to be essential to obtain state-of-the-art performance [10].

The next sections describe our tree kernel approaches for the classification of boundaries and arguments and the re-ranking of complete predicative structures.



(a) A parse tree (b) AST_1 (c) AST_1^m

Fig. 1. Syntactic parse tree of the sentence *John delivers a talk in formal style* (a), AST_1 (b) and AST_1^m (c) for the argument A1 *a talk*.

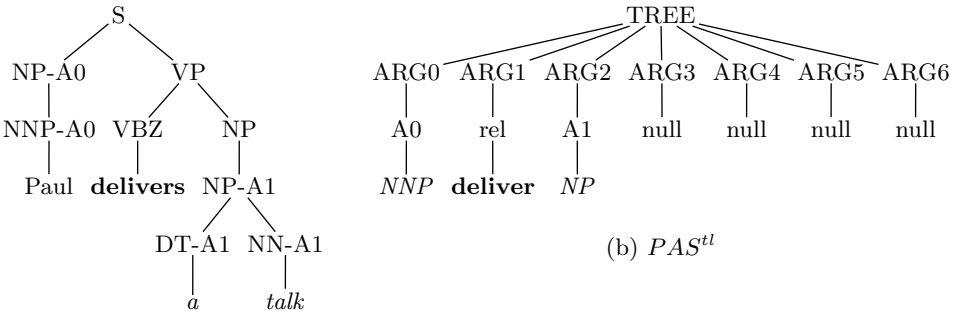
2.1 Kernels for Boundary Detection and Argument Classification

Once a basic kernel function is defined, we need to characterize the predicate-argument pair with a subtree. This allows the function to generate a large number of syntactic features related to such pair. The approach proposed in [5] selects the minimal subtree that includes a predicate with one of its arguments. For example, Figure 1(a) shows the parse tree of the sentence *Paul delivers a talk in formal style* whereas Frame (b) illustrates the AST_1 subtree that characterizes the predicate *to deliver* with its argument A1 *a talk*.

AST_1s are very effective for argument classification but not for boundary detection since two nodes that encode correct and incorrect boundaries may generate very similar AST_1s [5]. To solve this problem, we simply mark the argument node with the label B, denoting the boundary property. This new subtree is called a marked argument spanning tree (AST_1^m) and it is shown in Figure 1(c). A positive example for the AST_1^m classifier is a subtree in which the marked node exactly covers the boundaries of an argument, whereas the marking of any other node within the same subtree results in a negative example.

2.2 Tree Kernels for the Proposition Re-ranking Task

Our re-ranking mechanism is similar to that described in [11], where a Viterbi algorithm is used to evaluate the most likely labeling schemes for a given predicate and a re-ranking mechanism selects the best annotation. The re-ranker is a binary classifier trained with pairs $\langle s_i, s_j \rangle$ where s_i and s_j are taken from the set of the most m probable prepositions output by the Viterbi algorithm for the same target predicate. The classifier is meant to output a positive value if s_i is more



(a) AST_n^{cm}

(b) PAS^{tl}

Fig. 2. AST_n^{cm} and PAS^{tl} representations of the example proposition.

accurate that s_j and a negative value otherwise. Each candidate proposition s_i can be described by a structural feature t_i and by a vector of linear features v_i representing information that cannot be captured by t_i, e.g. the probability associated with the annotation output by the Viterbi algorithm. As a whole, each classifier example e_i is described by a tuple $\langle t_i^1, t_i^2, v_i^1, v_i^2 \rangle$, where $\langle t_i^1, v_i^1 \rangle$ and $\langle t_i^2, v_i^2 \rangle$ describe the first and second candidate annotations, respectively.

Using the above tuple, we can define the following kernels:

$$K_{tr}(e_1, e_2) = K_t(t_1^1, t_2^1) + K_t(t_1^2, t_2^2) - K_t(t_1^1, t_2^2) - K_t(t_1^2, t_2^1)$$
$$K_{pr}(e_1, e_2) = K_p(v_1^1, v_2^1) + K_p(v_1^2, v_2^2) - K_p(v_1^1, v_2^2) - K_p(v_1^2, v_2^1)$$

where K_t is a tree kernel function defined in [12] and K_p is a polynomial kernel applied to the feature vectors. The final kernel that we use for re-ranking is the following:

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

Among the many different structural features that we tested with our re-ranker, the most effective are the completely marked argument structure spanning tree (AST_n^{cm}) and the lemmatized type-only predicate argument structure (PAS^t).

An AST_n^{cm} (see Figure 2(a)) consists of the node spanning tree embracing the whole argument structure: each argument node's label is enriched with the role assigned to the node by the role multiclassifier, the labels of the descendants of each argument node being accordingly modified down to pre-terminal nodes. Marking the nodes' descendants is meant to force substructures to match only among homogeneous argument types. This representation is meant to provide rich syntactic and lexical information about the parse tree encoding the predicate structure.

A PAS^{tl} (see Figure 2(b)) is a completely different structure that represents the syntax of the predicate argument structure, i. e. the number, type and position of each argument, minimizing the amount of lexical and syntactic information derived from the parse tree. The syntactic links between the argument nodes are represented as a fake 1-level tree, which is shared by any PAS^{tl} and therefore does not influence the evaluation of similarity between pairs of structures. Such structure accommodates sequentially all the arguments of an annotation, each slot being attached a pre-terminal node standing for the node type and a terminal symbolizing the syntactic type of the argument node. In general, a proposition consists of m arguments, with $m < 7$. In this case, all the nodes ARGi, $i \leq m \leq 6$ are attached a dummy descendant marked *null*. The predicate is represented by means of a pre-terminal node labeled *rel* to which the lemmatization of the predicate word is attached as a leaf node.

Table 1. Correct (+), incorrect (-) and overall (tot) number of potential argument nodes from sections 2, 3 and 24 of the PropBank.

Section 2			Section 3			Section 24		
+	-	tot	+	-	tot	+	-	tot
12,741	185,178	197,919	7,023	139,823	146,846	8,234	130,489	138,723

Table 2. Performance improvement on the boundary detection and argument classification tasks using engineered tree kernels.

	Boundary detection	Argument classification
AST_1	75.24	82.07
AST_1^m	75.06	77.17

3 Experiments

In these experiments we evaluate the impact of our proposed kernels on the different phases of the SRL task. The resulting accuracy improvement confirms that the node marking approach enables the automatic engineering of effective SRL features.

The empirical evaluations were carried out within the setting of the CoNLL-2005 Shared Task [4] described in `www.lsi.upc.edu/~srlconll/` by means of SVM-light-TK available at `http://ai-nlp.info.uniroma2.it/moschitti/` which encodes fast tree kernel evaluation [12] in SVM-light [13]. We used a regularization parameter (option `-c`) equal to 1 and $\lambda = 0.4$ (see [5]).

3.1 Boundary Detection and Argument Classification Results

For the boundary detection experiments we used Section 02 for training and Section 24 for testing, whereas for argument classification also Section 03 was

used for training. Their characteristics in terms of potential argument nodes[1] are shown in Table 1.

Table 3. Number of distinct annotations output by the Viterbi algorithm and of pair comparisons (i. e. re-ranker input examples) in the PropBank sections used for the experiments.

	Section 12	Section 23	Section 24
Annotations	24,494	26,325	16,240
Comparisons	74,650	81,162	48,582

Table 4. Summary of the proposition re-ranking experiments with different training sets.

Training section	AST_n^{cm}	PAS^{tl}	PAS^{tl} + STD
12	-	78.27	77.61
24	76.47	78.15	77.77
12+24	-	78.44	-

The results obtained using the AST_1 and the AST_1^m based kernels are reported in Table 2 in rows 2 and 3, respectively. Columns 2 and 3 show their respective performance (in terms of F_1 measure) on the boundary detection and argument classification phases. We note that: (1) on boundary detection, AST_1^ms improve the F_1 over AST_1 by about 7 points, i. e. 82.07 vs. 75.24. This suggests that marking the argument node simplifies the generalization process; (2) using an engineered tree kernel also improves the argument classification task by about 2 points, i. e. 77.17 vs. 75.06. This confirms the outcome on boundary detection experiments and the fact that we need to distinguish the target node from the others.

3.2 Proposition Re-ranking Results

For our proposition re-ranking experiments, Section 23 was used for testing. On such test set, considering the first 5 alternatives output by the Viterbi algorithm, our model has a lower bound of the F_1 measure of 75.91 (corresponding to the selection of the first alternative, i. e. the most likely with respect to the probabilistic model) and an upper bound of 84.76 (corresponding to the informed selection of the best among the 5 alternatives, i. e. the theoretical output of a perfect re-ranker). The number of distinct annotations output by the Viterbi

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

algorithm for each section that we used is shown in Table 3, Row 2. In Row 3, the number of pair comparisons, i. e. the number of training/test examples for the classifier.

Table 4 summarizes the outcome of our experiments. First, we compared the accuracy of the AST_n^{cm} and PAS^{tl} classifiers trained on Section 24 (in Row 3, Columns 2 and 3) and discovered that the latter structure produces a noticeable F_1 improvement, i. e. 78.15 vs. 76.47. Second, we added the local (to each argument node) linear features commonly employed for the boundary detection and argument classification tasks, as in [10] to the PAS^{tl} kernel (Column 4). The comparison with the simple PAS^{tl} on 2 different training sets (Rows 2 and 3) shows that the introduction of the standard linear features produces a performance decrease on both sections 12 and 24 . Finally, we trained our best re-ranking kernel, i. e. the PAS^{tl}, with both sections 12 and 24 achieving an F_1 measure of 78.44 (Row 4).

These results suggest that: (1) the re-ranking task is very difficult from a ML point of view: in fact, adding or removing thousands of training examples has only a small impact on the classification accuracy; (2) the PAS^{tl} kernel is much more effective than the AST_n^{cm} one, which is always outperformed. This may be due to the fact that two AST_n^{cm}s always share a great number of substructures, since most alternative annotations tend to be very alike and the small differences among them only affect a small part of their enriched syntactic parse trees; (3) on the other hand, the little amount of local parsing information encoded in the PAS^{tl}s allows for a good generalization process; (4) the introduction of the standard, local linear features in our re-ranking model caused a performance loss of about 0.5 points on both Sections 12 and 24. This fact, which is in contrast with what has been shown in [10], might be the consequence of the small training sets that we employed. In fact, local linear features tend to be very sparse and their effectiveness should be evaluated against a larger data set.

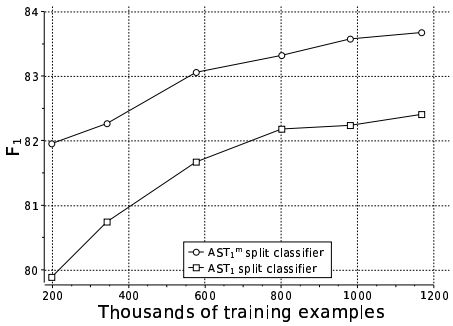


Fig. 3. Learning curve comparison for the boundary detection phase between the AST_1 and AST_1^m F_1 measures.

4 Conclusions

The design of automatic systems for the labeling of semantic roles requires the solution of complex problems. Among others, feature engineering is made difficult by the structural nature of the data, i. e. features should represent information contained in automatic parse trees. A system based on tree kernels alleviate such complexity as kernel functions can automatically generate effective features.

In this paper, we have improved tree kernels by studying different strategies, e.g. AST_1^ms highly improve accuracy in both the boundary detection (about 7%) and argument classification subtasks (about 2%). We have also engineered different structured features for the re-ranking module, which improves our system of about 2.5 percent points. This is quite a good results as it approaches the state-of-the-art using only a small fraction of all the available data. In the near future, we would like to use more such data along with other kernels described in [12].

Acknowledgments

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

References

1. Johnson, C.R., Fillmore, C.J.: The framenet tagset for frame-semantic and syntactic coding of predicate-argument structure. In proceedings of NAACL 2000, Seattle WA. (2000)
2. Kingsbury, P., Palmer, M.: From Treebank to PropBank. In proceedings of LREC'02, Las Palmas, Spain (2002)
3. Gildea, D., Jurasfky, D.: Automatic labeling of semantic roles. Computational Linguistic **28**(3) (2002) 496–530
4. Carreras, X., Màrquez, L.: Introduction to the CoNLL-2005 shared task: Semantic role labeling. In proceedings of CoNLL-2005, Ann Arbor, Michigan, (2005)
5. Moschitti, A.: A study on convolution kernels for shallow semantic parsing. In proceedings of ACL'04, Barcelona, Spain (2004)
6. Pradhan, S., Hacioglu, K., Krugler, V., Ward, W., Martin, J.H., Jurafsky, D.: Support vector learning for semantic argument classification. Machine Learning Journal (2005)
7. Moschitti, A., Coppola, B., Pighin, D., Basili, R.: Engineering of syntactic features for shallow semantic parsing. In proceedings of the ACL Workshop on Feature Engineering for Machine Learning in Natural Language Processing, Ann Arbor, Michigan, (2005)
8. Moschitti, A., Pighin, D., Basili, R.: Tree kernel engineering in semantic role labeling systems. In proceedings of the EACL Workshop on Learning Structured Information in Natural Language Applications, Trento, Italy, (2006)
9. Charniak, E.: A maximum-entropy-inspired parser. In: Proceedings of the 1st Meeting of NAACL. (2000)
10. Haghighi, A., Toutanova, K., Manning, C.: A joint model for semantic role labeling. In proceedings of CoNLL-2005, Ann Arbor, Michigan. (2005)
11. Moschitti, A., Pighin, D., Basili, R.: Semantic role labeling via tree kernel joint inference. In proceedings of CoNLL-X. (2006)
12. Moschitti, A.: Efficient convolution kernels for dependency and constituent syntactic trees. In proceedings of ECML 2006, Berlin, Germany. (2006)
13. Joachims, T.: Making large-scale SVM learning practical. In Schölkopf, B., Burges, C., Smola, A., eds.: Advances in Kernel Methods - Support Vector Learning. (1999)