4.2 Unlexicalized Partial Tree Kernel (uPTK)

The algorithm for the uPTK computation straightforwardly follows from the definition of the $\Delta$ function of PTK provided in [17]. Given two nodes $n_1$ and $n_2$ in the corresponding two trees $T_1$ and $T_2$, $\Delta$ is evaluated as follows:

1. if the node labels of $n_1$ and $n_2$ are different then $\Delta(n_1,n_2) = 0$;

2. else $\Delta(n_1,n_2) = \mu \left( \lambda^2 + \sum_{I_1,I_2,l(I_1)=l(I_2)} \lambda^{d(I_1)+d(I_2)} \prod_{j=1}^{l(I_1)} \Delta(c_{n_1}(I_{1j}),c_{n_2}(I_{2j})) \right)$

where: (a) $I_1 = \langle h_1, h_2, h_3, .. \rangle$ and $I_2 = \langle k_1, k_2, k_3, .. \rangle$ are index sequences associated with the ordered child sequences $c_{n_1}$ of $n_1$ and $c_{n_2}$ of $n_2$, respectively; (b) $I_{1j}$ and $I_{2j}$ point to the $j$-th child in the corresponding sequence; (c) $l(\cdot)$ returns the sequence length, i.e. the number of children; (d) $d(I_1) = I_{1l(I_1)} - I_{11} + 1$ and $d(I_2) + 1 = I_{2l(I_2)} - I_{21} + 1$; and (e) $\mu$ and $\lambda$ are two decay factors for the size of the tree and for the length of the child subsequences with respect to the original sequence, i.e. we account for gaps.

The uPTK, can be obtained by removing $\lambda^2$ from the equation in the step 2. An efficient algorithm for the computation of PTK is given in [17]. This evaluates $\Delta$ by summing the contribution of tree structures coming from different types of sequences, e.g. those composed by $p$ children such as:

$$\Delta(n_1,n_2) = \mu \left( \lambda^2 + \sum_{p=1}^{lm} \Delta_p(c_{n_1}, c_{n_2}) \right),$$  \hspace{1cm} (9)

where $\Delta_p$ evaluates the number of common subtrees rooted in subsequences of exactly $p$ children (of $n_1$ and $n_2$) and $lm = min\{l(c_{n_1}), l(c_{n_2})\}$. It is easy to verify that we can use the recursive computation of $\Delta_p$ proposed in [17] by simply removing $\lambda^2$ from Eq. 9.

5 Experiments

In these experiments, we study the impact of the cutting plane algorithms (CPAs), reviewed in Section 3, on learning complex text classification tasks in
structural feature spaces. For this purpose, we compare the accuracy and the learning time of CPAs, according to different sample size against the conventional SVMs.

In the second set of experiments, we investigate the possibility of using fast parameter and kernel selection with CPA for conventional SVM. For this purpose, we carried out experiments with different classifiers on two different domains.

5.1 Experimental Setup

We integrated two approximate cutting plane algorithms using sampling [28] with SVM-light-TK [17]. For brevity, in this section we will refer to the algorithm that uses uniform sampling as uSVM, importance sampling as iSVM, and SVM-light-TK as SVM. While the implementation of sampling algorithms uses MOSEK to optimize quadratic problem, SVM is based on SVM-light 5.0 solver. As the stopping criteria of the algorithms, we fix the precision parameter $\epsilon$ at 0.001.

We experimented with five different kernels: the ST, SST, SST-bow, PT, uPT kernels described in Section 4, which are also normalized in the related kernel space. All the experiments that do not involve parameter tuning use the default trade-off parameter (i.e. 1 for normalized kernels) and the default $\lambda$ fixed at 0.4.

As a measure of classification accuracy we use the harmonic average of the Precision and Recall, i.e. $F_1$-score. All the experiments were run on machines equipped with Intel® Xeon® 2.33GHz CPUs carrying 6Gb of RAM under Linux 2.6.18 kernel.

5.2 Data

We used two different natural language datasets corresponding to two different tasks: Semantic Role Labeling (SRL) and Question Answering.

The first consists of the Penn Treebank texts [14], PropBank annotation [20] and automatic Charniak parse trees [2] as provided by the CoNLL 2005 evaluation campaign [1]. In particular, we tackle the task of identification of the argument boundaries (i.e. the exact sequence of words compounding an argument). This corresponds to the classification of parse tree nodes in correct or not correct boundaries\(^2\). For this purpose, we train a binary Boundary Classifier (BC) using the AST subtree defined in [15], i.e. the minimal subtree, extracted from the sentence parse tree, including the predicate and the target argument nodes. To test the learned model, we extract two sections, namely sec23 and sec24, that contain 234,416 and 149,140 examples respectively. The models are trained on two subsets of 100,000 and 1,000,000 examples. The proportion of positive examples in the whole corpus is roughly 5%. The dataset along with the exact structural representation is available at [http://danielepighin.net/cms/research/MixedFeaturesForSRL](http://danielepighin.net/cms/research/MixedFeaturesForSRL).

\(^2\) In the automatic trees some boundary may not correspond to any node. In this case, we choose the lower node dominating all the argument words.