Large-Scale Support Vector Learning with Structural Kernels

Aliaksei Severyn and Alessandro Moschitti

Department of Computer Science and Engineering
University of Trento
Via Sommarive 14, 38100 POVO (TN) - Italy
{severyn,moschitti}@disi.unitn.it

Abstract. In this paper, we present an extensive study of the cutting-plane algorithm (CPA) applied to structural kernels for advanced text classification on large datasets. In particular, we carry out a comprehensive experimentation on two interesting natural language tasks, e.g. predicate argument extraction and question answering. Our results show that (i) CPA applied to train a non-linear model with different tree kernels fully matches the accuracy of the conventional SVM algorithm while being ten times faster; (ii) by using smaller sampling sizes to approximate subgradients in CPA we can trade off accuracy for speed, yet the optimal parameters and kernels found remain optimal for the exact SVM. These results open numerous research perspectives, e.g. in natural language processing, as they show that complex structural kernels can be efficiently used in real-world applications. For example, for the first time, we could carry out extensive tests of several tree kernels on millions of training instances. As a direct benefit, we could experiment with a variant of the partial tree kernel, which we also propose in this paper.

Keywords: Structural Kernels; Support Vector Machines;Natural Language Processing;

1 Introduction

In many computer science areas such as Natural Language Processing (NLP), Bioinformatics, Data Mining and Information Retrieval, structural kernels have been widely used to capture rich syntactic information, e.g. [3, 11, 27, 28]. In particular, in NLP, tree kernel functions can capture a representation of syntactic parse trees, which is considerably more effective than the one provided by straightforward bag-of-words models. This often results in higher accuracy, e.g. [15, 17, 29, 10]. Unfortunately, the use of kernels forces us to solve SVM optimization problem in the dual space, which, in case of the use of very large datasets, makes SVM learning prohibitively expensive.

Recently, cutting plane approaches have been proposed to achieve a great speed-up in the learning of linear models on large datasets, but they still fail to provide the same training performance on non-linear learning tasks. Indeed, the number of kernel evaluations scales quadratically with the number of examples. To overcome this bottleneck, Yu and Joachims [26] developed two cutting plane algorithms (CPAs) that have linear or constant-time scaling behavior. However, their experiments were carried out using only Gaussian and polynomial kernels on unstructured data, e.g. topic categorization.

In this paper, we study models that combine the speed of CPAs with the efficiency of SVM-light-TK[1], which encodes state of the art structural kernels [16, 17] in SVM-light [6]. More specifically, by the means of extensive experimentation we examine the applicability of CPAs to well-known structural kernels, i.e. subtree (ST), subset tree (SST), and partial tree (PT) kernels, which provide a good sample of the efficient tree kernel technology currently available. To obtain more general results we considered two advanced and very different text categorization tasks for which syntactic information is essential: (i) Semantic Role Labeling (SRL) or predicate argument extraction [19, 14] whose associated dataset contains several millions of examples; and (ii) Question Classification (QC), e.g. [12, 24] from question answering domain.

Moreover, we conjectured that the best parameters for CPAs are invariant with respect to the sample size. In other words, we can use an extremely small number of examples to carry out a very fast SVM parameterization and kernel selection. The optimal set of parameters is also optimal for the conventional SVM-light.

Finally, we defined a novel kernel, unlexicalized PTK (uPTK), which is a variant of PTK. It excludes single nodes from the feature space so that structural features are better emphasized.

Our findings reveal that:

- As the theory suggests, CPAs can be successfully applied to the structural spaces. Indeed, we show that CPA is at least 10 times faster than the exact version while achieving the same classification accuracy. For example, to train a conventional SVM-light solver with tree kernels on 1 million examples requires more than seven days, while CPAs match the same accuracy in just a few hours.
- By decreasing the sampling size used in the approximation of the cutting planes, we can trade off the accuracy to a small degree for even faster training time. For example, learning a model with the accuracy that is only 1.0 percentage point apart from the exact model reduces the training time by a factor of 50.
- Using a sample size of only 100 instances for CPAs, which takes just a couple of minutes of learning on one million of examples, we can correctly estimate the best kernel, its hyper-parameters, and the trade-off parameter. The identified set of optimal parameters can be further used to train more computationally expensive and accurate models (i.e. CPAs with larger sample sizes or the original SVM-light).

[1] available at `http://disi.unitn.it/moschitti/Tree-Kernel.htm`

– Thanks to the gained speed-up we could efficiently compare different kernels on the large SRL datasets and establish an absolute rank, where uPTK proves to be more effective than PTK.

Our results open up new perspectives for the application of structural kernels in various NLP tasks. The experiments that could not be carried out until now on full subsets of very large corpora, e.g. training SVM-based SRL classifier such as in [21, 14] using either polynomial or tree kernels on the full dataset; training SVM re-rankers for syntactic parsing [3, 23] using tree kernels on the available data; and training SVM question classifiers using tree kernels and other features on large subsets of Yahoo! Answers dataset.

The aforementioned motivations encouraged us to make the new Tree Kernel toolkit used in this paper freely available to the community.

In the remainder of this paper, Section 2 reviews the related work, while Section 3 gives careful theoretical treatment to the workings of the cutting plane approach. Section 4 introduces well-known tree kernels along with the new uPTK. The experimental analysis (Section 5) describes our experiments on SRL and QC datasets and also discusses how approximate cutting plane algorithms could be used to drive parameter selection for the exact SVM solver. Finally, in Section 6, we draw conclusions and provide directions for the further research.

2 Related work

Since the introduction of SVMs, a number of fast algorithms that can efficiently train non-linear SVMs have been proposed - for example, decomposition method along with working set selection [6]. Decomposition methods work directly in the dual space and perform well on moderately large datasets but their performance degrades when the number of training examples reaches a level of millions of examples.

Recently, a number of efficient algorithms using cutting planes to train conventional SVM classifiers have been proposed. For example, SVM^{perf} [7] is based on a cutting plane algorithm and exhibits linear computational complexity in the number of examples when linear kernels are used. To improve the convergence rate of the underlying cutting plane algorithm, Franc and Sonnenburg [5] developed the optimized cutting plane algorithm (OCAS) that achieves speed-up factor of 29 over SVM^{perf}. Alternatively, another promising approach based on stochastic gradient descent and projection steps called Pegasos [22] has shown promising performance for linear kernels in binary classification tasks.

While the aforementioned algorithms deliver state of the art performance with respect to accuracy and training time, they scale well only when linear kernels are used. To overcome this bottleneck, an idea to use approximate cutting planes with random sampling was emloyed by Yu and Joachims [26]. At each iteration step an exact cutting plane is replaced by its approximation that is built from a small subset of examples sampled from the training dataset. The most recent attempt to address the poor scaling of non-liner models was done by Joachims [8], where a sparse set of basis vectors is extracted as a part of the cutting-plane optimization process. Not only the trained sparse model speeds up the classification time, but it also improves the training complexity. Even though the achieved scaling behavior is roughly linear, the method is limited to the use of only Gaussian kernels.

Following a different line of research, a number of methods that exploit low-rank approximation of a kernel matrix have been proposed in [4, 25]. However, the experiments were carried out only on fairly small datasets (thousands of examples).

The approach we employ in this paper is different in a sense that it is generally applicable to the learning of any non-linear discriminant function, structural kernels in particular, and can efficiently handle datasets with hundreds of thousands examples.

3 Cutting Plane Algorithm for Structural Kernels

In this section, we illustrate the the cutting plane approach. Previous work is focused on structural SVMs whereas the topic of this paper is binary classification, thus we present a re-elaborated version of the algorithm tailored for the binary classification task. This results in an easier discussion of the sampling approach to compute the approximate cutting planes.

3.1 Cutting-plane algorithm (dual)

The cutting-plane algorithm is based on a slight modification of SVM optimization problem, known as a 1-slack reformulation Joachims [7]:

$$\begin{aligned} \underset{\boldsymbol{w},\xi\geq 0}{\text{minimize}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\xi \\ \text{subject to} \quad & \forall \boldsymbol{c} \in \{0,1\}^n : \\ & \frac{1}{n}\boldsymbol{w}\cdot\sum_{i=1}^{n} c_i y_i \boldsymbol{x}_i \geq \frac{1}{n}\sum_{i=1}^{n} c_i - \xi, \end{aligned} \tag{1}$$

where each non-zero c_i of a vector $\boldsymbol{c} = (c_1, \ldots, c_n) \in \{0,1\}^n$ selects a corresponding constraint of the form: $y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i) \geq 1 - \xi_i$ from a standard optimization problem.

To derive the dual formulation, the Lagrangian of the primal problem (1) is computed as:

$$L_P = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\xi - \sum_{j=1}^{2^n} \alpha_j \Big(\frac{1}{n}\sum_{k=1}^{n} c_{kj}(y_k \boldsymbol{w}\cdot\boldsymbol{x}_k - 1) + \xi\Big), \tag{2}$$

where c_{kj} denotes the k^{th} component of a vector $\boldsymbol{c_j}$ that corresponds to the the j^{th} constraint, i.e. $\boldsymbol{c_j}$ defines the j^{th} constraint of (1), and $\alpha_j \geq 0$ are the Lagrange multipliers, one for each of the 2^n inequality constraints.

Using the fact that both gradients of L_P with respect to w and ξ vanish:

$$\begin{aligned} \frac{\delta L_P}{\delta \boldsymbol{w}} &= \boldsymbol{w} - \sum_{j=1}^{2^n} \alpha_j \Big(\frac{1}{n} \sum_{k=1}^{n} c_{kj} y_k \boldsymbol{x}_k \Big) = 0 \\ \frac{\delta L_P}{\delta \xi} &= \frac{1}{n} \sum_{j=1}^{2^n} \alpha_j - C = 0, \end{aligned} \tag{3}$$

and by substituting variables expressed from (3), we obtain the dual Lagrangian:

$$L_D = \sum_{i=1}^{2^n} \alpha_i d^{(i)} - \frac{1}{2} \sum_{i=1}^{2^n} \sum_{j=1}^{2^n} \alpha_i \alpha_j \boldsymbol{g}^{(i)} \cdot \boldsymbol{g}^{(j)} \tag{4}$$

where $d^{(i)} = \frac{1}{n} \sum_{k=1}^{n} c_{ki}$ and $\boldsymbol{g}^{(i)} = -\frac{1}{n} \sum_{k=1}^{n} c_{ki} y_k \boldsymbol{x}_k$ are respectively the bias and the subgradient that define a cutting plane model $(d^{(i)}, \boldsymbol{g}^{(i)})$ added to the set of constraints at each iteration (Algorithm 1, lines 12-13). Now we can state the dual variant of the optimization problem (1):

$$\begin{aligned} &\underset{\boldsymbol{\alpha} \geq 0}{\text{maximize}} \quad \boldsymbol{h}^T \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^T H \boldsymbol{\alpha} \\ &\text{subject to} \quad \boldsymbol{\alpha}^T \mathbf{1} \leq C, \end{aligned} \tag{5}$$

where $h_i = d^{(i)}$ and $H_{ij} = \boldsymbol{g}^{(i)} \cdot \boldsymbol{g}^{(j)}$. Using the first equation from (3), we get the connection between the primal and dual variables:

$$\boldsymbol{w} = \sum_{j=1}^{2^n} \alpha_j \Big(\frac{1}{n} \sum_{k=1}^{n} c_{kj} y_k \boldsymbol{x}_k \Big) = - \sum_{j=1}^{2^n} \alpha_j \boldsymbol{g}^{(j)}, \tag{6}$$

where $\boldsymbol{g}^{(j)}$ is a subgradient of a cutting plane model as defined in algorithm 1 (line 9). When working in the dual space, to find the most violated constraint (Algorithm 1, lines 9-11) we need to compute the following quantity for each training example:

$$\boldsymbol{w} \cdot \phi(\boldsymbol{x}_i) = - \sum_{j=1}^{|S|} \alpha_j \boldsymbol{g}^{(j)} = \sum_{k=1}^{n} \Big(\sum_{j=1}^{|S|} \frac{1}{n} \alpha_j c_{kj} y_k \Big) K(\boldsymbol{x}_i, \boldsymbol{x}_k), \tag{7}$$

where $K(\boldsymbol{x}_i, \boldsymbol{x}_k) = \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_i)$ is a kernel. Note that here we use the sum only over a small number of active constraints $|S|$ instead of 2^n. The cutting plane method maintains the maximum size of the working set $|S|$ constant which is normally much less that the total number of iterations T. This has another important implication, as the dual problem (5) has the number of dual variables independent of the number of training examples, which produces a solution that is very sparse. The analysis of the inner product given by (7) reveals that since it needs to be computed for each training example, it requires the time $O(n^2 + Tn)$.

Algorithm 1 Cutting Plane Algorithm (dual) with uniform sampling

```
1: Input: $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, $C$, $\epsilon$
2: $S \leftarrow \emptyset$; $t = 0$;
3: repeat
4: Update the Gram matrix $H$ with a new constraint
5: $\boldsymbol{\alpha} \leftarrow$ optimize (5)
6: $\xi = \frac{1}{C}(\boldsymbol{h}^T\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^T H\boldsymbol{\alpha})$
7: $\boldsymbol{w} = -\sum_{j=1}^{|S|} \alpha_j \boldsymbol{g}^{(j)}$
8: Sample $r$ examples from the training set
   /* find a cutting plane */
9: for $i = 1$ to $r$ do
10: $c_i \leftarrow \begin{cases} 1 \; y_i(\boldsymbol{w} \cdot \phi(\boldsymbol{x}_i)) \leq 1 \\ 0 \; otherwise \end{cases}$
11: end for
12: $d^{(t)} = \frac{1}{r}\sum_{i=1}^{r} c_i$
13: $\boldsymbol{g}^{(t)} = -\frac{1}{r}\sum_{i=1}^{r} c_i y_i \phi(\boldsymbol{x_i})$
   /* add a constraint to the active set */
14: $S \leftarrow S \cup \{(d^{(t)}, \boldsymbol{g}^{(t)})\}$
15: $t = t + 1$
16: until $d^{(t)} + \boldsymbol{w} \cdot \boldsymbol{g}^{(t)} \leq \xi + \epsilon$
17: return $w, \xi$
```

Similarly, as we add a cutting plane to S at each iteration t, a new column is added to the Gram matrix H (Algorithm 1, line 4) requiring the computation of

$$H_{it} = \boldsymbol{g}^{(i)} \cdot \boldsymbol{g}^{(t)} = \frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n} c_{ki}c_{lt}y_k y_l K(\boldsymbol{x}_k, \boldsymbol{x}_l) \tag{8}$$

which takes $O(Tn^2)$. Thus, the obtained $O(n^2)$ scaling behavior makes cutting plane training no better than conventional decomposition methods.

To address this limitation, we employ the approach of Yu and Joachims [26] to construct approximate cuts by sampling r examples from the training set. They suggest two strategies to sample examples, namely uniform and importance sampling (the pseudocode of the algorithm using sampling is presented in Algorithm 1). These two strategies derive constant-time and linear-time algorithms. The former uniformly samples r examples from the training set to approximate the cut. Thus, we approximate a subgradient with only r examples, which replaces the number of expensive kernel evaluations in (7) over n by a more tractable: $\sum_{i,j=1}^{r} K(\boldsymbol{x}_i, \boldsymbol{x_j})$ (lines 9-13 in Algorithm 1). The importance sampling acts in a more targeted way as it looks through the whole dataset to compute two cutting planes, one to be used in the optimization problem (Algorithm 1, line 5), and the other for termination criterion (Algorithm 1, line 16). The training complexity reduces from $O(n^2)$ to $O(T^2r^2)$, when the uniform sampling algorithm is used, to $O(Tnr)$ for the importance sampling.

4 Tree Kernels

The main idea underlying tree kernels is to compute the number of common substructures between two trees T_1 and T_2 without explicitly considering the whole fragment space. Let $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$ be the set of tree fragments and $\chi_i(n)$ an indicator function equal to 1 if the target f_i is rooted at node n and equal to 0 otherwise. A tree kernel function over T_1 and T_2 is defined as

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

where N_{T_1} and N_{T_2} are the sets of nodes in T_1 and T_2, respectively, and

$$\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} \chi_i(n_1)\chi_i(n_2).$$

The Δ function is equal to the number of common fragments rooted in nodes n_1 and n_2 and thus depends on the fragment type.

4.1 Fragment types

In [16], we pointed out that there are three main categories of fragments: the subtree (ST), the subset tree (SST) and the partial tree (PT) fragments corresponding to three different kernels. STs are fragments rooted in any node of a tree along with all its descendants. The SSTs are more general structures since, given the root node of an SST, not all its descendants (with respect to the referring tree) have to be included, i.e. the SST leaves can be non-terminal symbols. PT fragments are still more general since their nodes can contain a subset of the children of the original trees, i.e. partial sequences.

For example, Figure 1 illustrates the syntactic parse tree of the sentence `Autism is a disease` on the left along with some of the possible fragments on the right of the arrow. ST kernel generates complete structures like `[D a]` or `[NP [D a] [N disease]]`. SST kernel can generate more structures, e.g. `[NP [D] [N disease]]` whereas PT kernel can also separate children in the fragments, e.g. `[NP [N disease]]`, and generate the individual tree nodes as features, e.g. `Autism` or `VBZ`.

[28] provided a version of SST kernel, which also generates leaves, i.e. words, as features, hereafter, SST-bow. However, such lexical features, when the data is very sparse, tend to cause overfitting. Thus, we give the definition of a variant of PTK, namely, the unlexicalized partial tree kernel (uPTK), which does not include lexicals and individual nodes in the feature space. This will promote the importance of structural information.

4.2 Unlexicalized Partial Tree Kernel (uPTK)

The algorithm for the uPTK computation straightforwardly follows from the definition of the Δ function of PTK provided in [16]. Given two nodes n_1 and n_2 in the corresponding two trees T_1 and T_2, Δ is evaluated as follows:

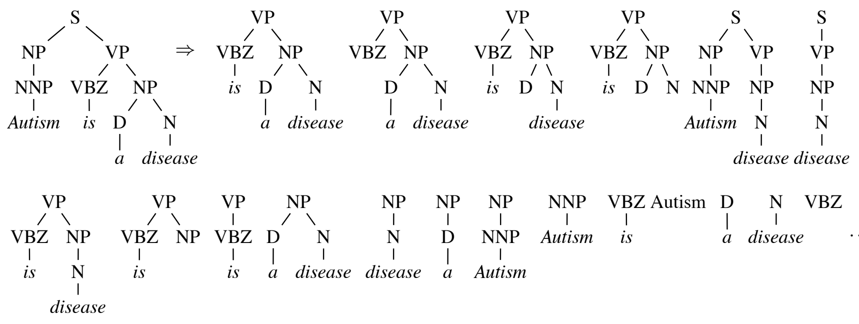

Fig. 1. A tree for the sentence “Autism is a disease” (top left) with some of its partial tree fragments (PTFs).

1. if the node labels of n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
2. else $\Delta(n_1, n_2) = \mu\Big(\lambda^2 + \sum_{\boldsymbol{I}_1, \boldsymbol{I}_2, l(\boldsymbol{I}_1)=l(\boldsymbol{I}_2)} \lambda^{d(\boldsymbol{I}_1)+d(\boldsymbol{I}_2)} \prod_{j=1}^{l(\boldsymbol{I}_1)} \Delta(c_{n_1}(\boldsymbol{I}_{1j}), c_{n_2}(\boldsymbol{I}_{2j}))\Big)$,

where: (a) $\boldsymbol{I}_1 = \langle h_1, h_2, h_3, ..\rangle$ and $\boldsymbol{I}_2 = \langle k_1, k_2, k_3, ..\rangle$ are index sequences associated with the ordered child sequences c_{n_1} of n_1 and c_{n_2} of n_2, respectively; (b) $\boldsymbol{I}_{1j}$ and $\boldsymbol{I}_{2j}$ point to the j-th child in the corresponding sequence; (c) $l(\cdot)$ returns the sequence length, i.e. the number of children; (d) $d(\boldsymbol{I}_1) = \boldsymbol{I}_{1l(\boldsymbol{I}_1)} - \boldsymbol{I}_{11} + 1$ and $d(\boldsymbol{I}_2) + 1 = \boldsymbol{I}_{2l(\boldsymbol{I}_2)} - \boldsymbol{I}_{21} + 1$; and (e) μ and λ 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 λ^2 from the equation in the step 2. An efficient algorithm for the computation of PTK is given in [16]. This evaluates Δ by summing the contribution of tree structures coming from different types of sequences, e.g. those composed by p children such as:

$$\Delta(n_1, n_2) = \mu\big(\lambda^2 + \textstyle\sum_{p=1}^{lm} \Delta_p(c_{n_1}, c_{n_2})\big), \tag{9}$$

where Δ_p evaluates the number of common subtrees rooted in subsequences of exactly p children (of n_1 and n_2) and $lm = min\{l(c_{n1}), l(c_{n2})\}$. It is easy to verify that we can use the recursive computation of Δ_p proposed in [16] by simply removing λ^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 [26] with SVM-light-TK [16]. For brevity, in this section we will refer to the algorithm that uses uniform sampling as uSVM, importance sampling (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 ϵ 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 λ 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 [13], PropBank annotation [19] 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 [14], 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`.

The second corpus, is a Question Classification (QC) dataset, whose testset comes from the TREC 10 - Question Answering evaluation campaign whereas

[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.

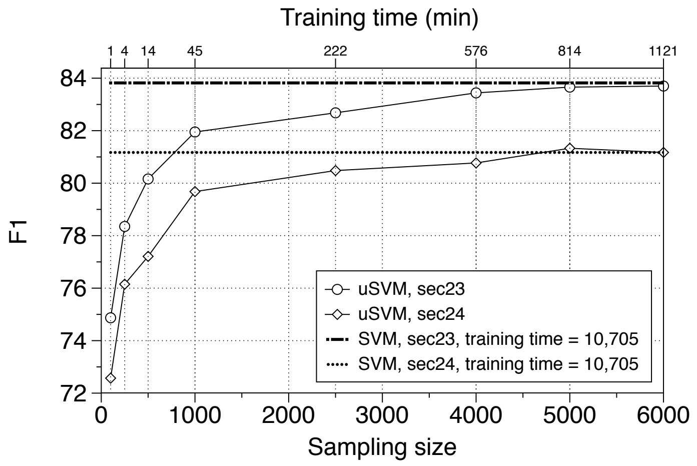

Fig. 2. F_1-score as a function of the sampling size on SRL dataset (1 million examples). Horizontal axis at top is the training time of the uSVM algorithm.

the training set[3] was developed in [12]. The task consists in selecting the most appropriate type of the answer from a set of given possibilities. The *coarse grained* question taxonomy [28, 12] consists of six non overlapping classes: Abbreviations (ABBR), Descriptions (DESC, e.g. definitions or explanations), Entity (ENTY, e.g. animal, body or color), Human (HUM, e.g. group or individual), Location (LOC, e.g. cities or countries) and Numeric (NUM, e.g. amounts or dates). For each question, we used the full parse tree as its representation (similarly to [28, 16, 18]). This is automatically extracted by means of the Stanford parser[4] [9]. We actually have only 5,483 questions in our training set, due to parsing issues with a few of them. The testset is constituted by 500 questions and the size of the categories varies from one thousands to few hundreds.

5.3 Accuracy and Efficiency vs. Sampling Size

In these experiments, we test the trade-off between speed and accuracy of CPAs. We use uSVM, since it is faster than iSVM, and compare it against SVM on the SRL task by training on 1 million examples and testing on the two usual testing sections, i.e. *sec23* and *sec24*. We used the SST kernel since it has been indicated as the most accurate in similar tasks, e.g. [16]. Figure 2 plots the F_1-score for different values of the sampling size. The dashed horizontal lines denote the accuracy achieved by the exact SVM. The training time for the uSVM algorithm is plotted along the top horizontal axis.

We note that:

– changing the sample size allows us to tune the trade-off between accuracy and speed. Indeed, as we increase the sampling size, the F_1-score grows until

[3] `http://l2r.cs.uiuc.edu/cogcomp/Data/QA/QC/`

[4] `http://nlp.stanford.edu/software/lex-parser.shtml`

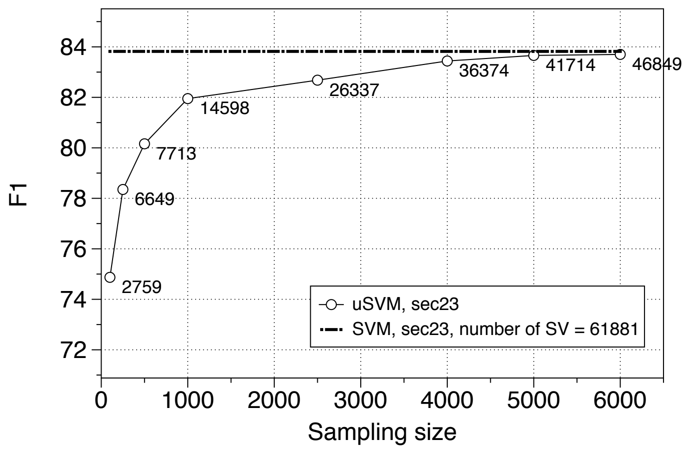

Fig. 3. Number of support vectors (displayed beside each data point) as a function of the sampling size for the 1-slack algorithm.

it reaches the accuracy of the exact SVM (at 5,000). In this case, the uSVM produces the same accuracy of SVM while being 10 times faster.

– with a sample size of 2,500 the accuracy of the uSVM is only 1.0 percentage point apart fromt the exact model whereas the training time savings are of a factor over 50. This corresponds to a training time smaller than 4 hours for uSVM vs. 7.5 days for SVM.
– finally, we note that our reported F_1-score for boundary classification is state-of-the-art only if tree kernels are used, e.g. [20].

5.4 Producing a Sparse Model

Another interesting dimension to compare sampling algorithms with the exact SVM solver would be to evaluate the sparseness of the produced solutions. As we have already mentioned in Section 3.1, uSVM and iSVM employ the 1-slack formulation that produces very sparse models [7]. This becomes especially important when very large datasets are used, as the improved sparsity can greatly reduce the classification time. Figure 3 is different from Figure 2, as it displays the classification results for sec23 subset of SRL dataset along with the number of support vectors (plotted beside a data point) learned by each model.

Indeed, the number of support vectors grows, as we increase the sampling size in an attempt to match the accuracy of the exact SVM. The model learned by the exact SVM contains 61,881 support vectors, while uSVM model produces 41,714 support vectors. We would like to have more experimental data to make the comparison more sound but the slow training of the exact SVM makes this endeavor almost infeasible (just the training of the exact SVM on 1 million of examples takes over 7.5 days!)

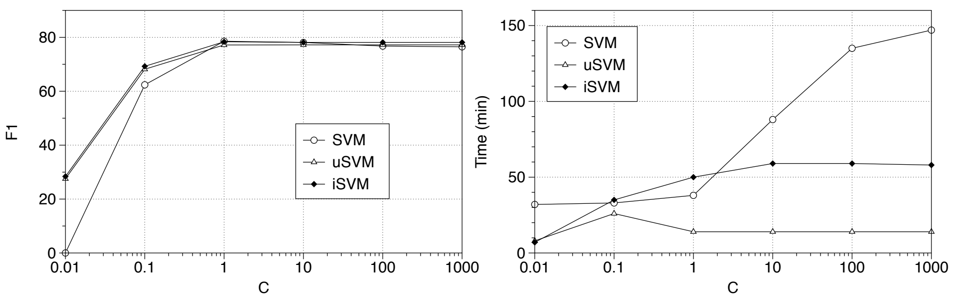

Fig. 4. F_1-score (left) and training time (right) as a function of margin parameter C on the 100,000 subset of SRL dataset (C is on a logarithmic scale).

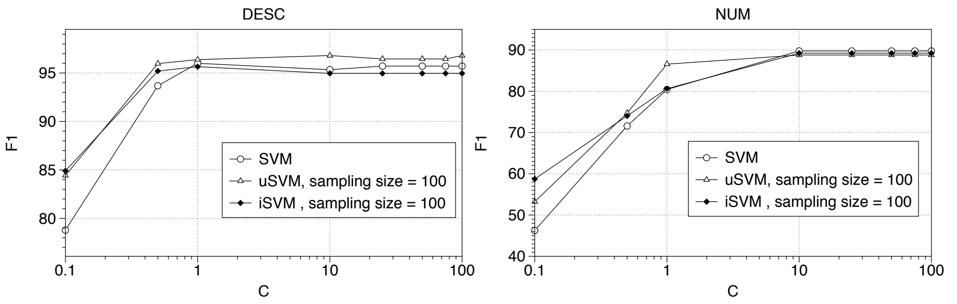

Fig. 5. F_1-score as a function of margin parameter C on the DESC (left) and NUM (right) categories of Question Classification dataset (C is on a logarithmic scale).

5.5 Fast Parameterization

On very large datasets, finding the best set of parameters for an exact solver, e.g. SVM-light, using structural kernels becomes prohibitively expensive, thus greatly limiting the possibility to find the best performing model. Hence, we test the idea of fast parameter selection for the exact SVM using CPAs and small samples.

We chose the trade-off parameter, C, as our target parameter and we select a subset of 100,000 examples from SRL data to train uSVM, iSVM, and SVM with $C \in \{0.01, 0.1, 1, 10, 100, 1000\}$. We could not use 1 million dataset since SVM prevents to carry out experiments within a tractable time frame. The left plot of Figure 4 shows that F1 of the three models has the same behavior according to C. Thus, we can select the optimum value according to the fast method (e.g. with a sample size 1000) to estimate the best value of C of SVM.

Moreover, it is interesting to observe the plot on the right of Figure 4. This shows the training time on the previous subset with respect to C values. It reveals that the use of sampling algorithms, particularly uSVM, provides substantial speed-up in the training time, especially when large values for C are used. Not surprisingly, uSVM is a preferable choice, since it has a constant-time scaling

Sample size	**ST**			**SST**			**SST-bow**			**PT**			**uPT**		
	F_1		time	F_1		time	F_1		time	F_1		time	F_1		time
	sec23	*sec24*		*sec23*	*sec24*		*sec23*	*sec24*		*sec23*	*sec24*		*sec23*	*sec24*	
100	6.8	6.5	3.9	**74.9**	**72.6**	**1.0**	74.3	73.0	2.2	71.9	70.7	3.7	73.3	71.4	3.7
250	14.2	13.0	14.4	**78.4**	**76.2**	**4.1**	78.5	76.3	6.1	74.6	73.2	13.7	76.5	74.6	14.2
500	20.3	18.7	46.6	**80.2**	**77.2**	**14.0**	79.5	77.3	16.9	76.3	74.4	45.0	78.3	76.3	47.6
1000	23.5	21.2	143.7	**82.0**	**79.7**	**45.3**	81.3	79.0	55.9	78.2	76.2	158	79.6	76.9	158.8
SVM	12.6	10.94	213.8	**80.78**	**78.56**	**37.5**	80.38	78.13	42.2	74.39	73.47	89.1	77.54	75.87	100.4

Table 1. F_1 measured on two testsets, *sec23* and *sec24*, for five different kernels: ST, SST, SST-bow, PT, and uPT kernels, trained on 1 million instances. The best results are shown in bold. The training time is given in minutes. The bottom row is the performance of SVM trained on only **100k** examples.

behavior. These results show that the proposed algorithms can provide fast and reliable parameter search for the best model.

To generalize the findings above we carried out the same experiment on the second dataset. We ran tests for a range of values on all six categories of the QC dataset. Since each category has only 5500 examples, here, however, the main concern is not the training time, but the ability of uSVM and iSVM to match the behavior of the exact SVM with respect to the parameter C. For brevity, we provide the results only for DESC and NUM categories. Figures 5 shows that both sampling algorithms match well the behavior of the exact SVM for DESC and NUM categories of QC dataset.

5.6 Kernel Testing and Selection

The previous section has shown that sampling algorithms can be used for fast selection of the parameter C. Here we investigate if the same approach can be applied for the efficient selection of the best kernel. The aim is to check if a very fast uSVM algorithm, i.e. using a small sample size, can identify the best performing kernel. Thus, we ran uSVM algorithm on 1 million subset of SRL dataset by varying the sample size and using five different structural kernels: ST, SST, SST-bow, PT and uPT kernels. The results reported in Table 1 show that:

– uSVM has a consistent behavior across different sample sizes for all kernel functions. This suggests that a small sample, e.g. 100 training examples, can be used to select the most suitable kernel for a given task in a matter of a couple of minutes.
– the bottom row of the Table 1 reports the results of SVM trained on a smaller subset of 100k examples (only one experiment with SVM on a subset of 1 million examples takes over 7 days), which demonstrates that the results obtained with uSVM are consistent with the exact solver.
– as already happened in similar tasks, SST kernel appears to provide the best performance with respect to the training time and accuracy whereas the ST

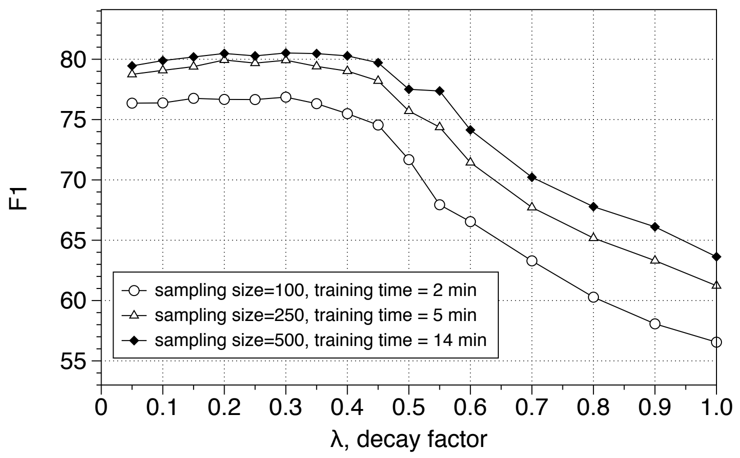

Fig. 6. F_1-score as a function of λ decay factor (length of the child sequences) for SST kernel. Training carried out by uSVM on SRL (1million examples) for samples sizes 100, 250 and 500.

kernel, as expected, cannot generate enough features to characterize correct or incorrect boundary. Indeed, its F_1 is very low even when large amount of data is used.

– surprisingly the SST-bow kernel which simply adds bow to SST is less accurate. This suggests that adding words can reduce the generalization ability of the tree kernels, while the syntactic information is very important like in SRL. This aspect is confirmed by the F_1 of uPT, which is higher than PT. Indeed, the former does not generate *pure* lexical features, i.e. words, whereas the latter does.
– finally, since we can quickly pick the most appropriate kernel, we can also perform a fast tuning of kernel hyper-parameters. For this task, we experiment with λ, which is one of the most important factors defining tree kernel performance. Figure 6 displays the behavior of F_1 with respect to the range of λ values. The plot shows that F_1 varies considerably so several values should be tested. We carried out these experiment in a few minutes but using SVM we would have required several weeks.

6 Conclusions

In this paper, we have presented the cutting plane technique for training classification SVMs and pointed out why it works with structural kernels. We have presented the dual formulation of CPA for binary classification task along with two sampling algorithms - uSVM and iSVM. The experiments on Semantic Role Labeling and Question Classification show very promising results with respect to accuracy and efficiency. Our major achievement is a speed-up factor of over 10 compared to the exact SVM, while obtaining the same precise solution. In addition, the proposed method gives the flexibility to train very fast models by trading off accuracy for the training time.

We have also demonstrated that the proposed approach works well for the fast parameter estimation task, so that the best model could be found efficiently. Although, more experiments are required to compare sparseness of the obtained solutions, we still get a hint that sampling algorithms, following 1-slack problem formulation, find a sparser solution.

In the short-term future, we plan to extend our work in fast parameter and kernel selection to design the best model.

References

1. Carreras, X., Màrquez, L.: Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In: Proceedings of the 9th Conference on Natural Language Learning, CoNLL-2005. Ann Arbor, MI USA (2005)
2. Charniak, E.: A maximum-entropy-inspired parser. In: ANLP. pp. 132–139 (2000)
3. Collins, M., Duffy, N.: New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In: ACL. pp. 263–270 (2002)
4. Fine, S., Scheinberg, K.: Efficient svm training using low-rank kernel representations. Journal of Machine Learning Research 2, 243–264 (2001)
5. Franc, V., Sonnenburg, S.: Optimized cutting plane algorithm for support vector machines. In: Cohen, W.W., McCallum, A., Roweis, S.T. (eds.) ICML. ACM International Conference Proceeding Series, vol. 307, pp. 320–327. ACM (2008)
6. Joachims, T.: Making large-scale SVM learning practical. In: Schölkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods - Support Vector Learning, chap. 11, pp. 169–184. MIT Press, Cambridge, MA (1999)
7. Joachims, T.: Training linear SVMs in linear time. In: ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD). pp. 217–226 (2006)
8. Joachims, T., Yu, C.N.J.: Sparse kernel svms via cutting-plane training. Machine Learning 76(2-3), 179–193 (2009), european Conference on Machine Learning (ECML) Special Issue
9. Klein, D., Manning, C.D.: Fast exact inference with a factored model for natural language parsing. In: Becker, S., Thrun, S., Obermayer, K. (eds.) NIPS. pp. 3–10. MIT Press (2002)
10. Kudo, T., Matsumoto, Y.: Fast methods for kernel-based text analysis. In: Proceedings of ACL'03 (2003)
11. Leslie, C., Eskin, E., Cohen, A., Weston, J., Noble, W.S.: Mismatch string kernels for discriminative protein classification. Bioinformatics 20(4), 467–76 (2004)
12. Li, X., Roth, D.: Learning question classifiers: the role of semantic information. Natural Language Engineering 12(3), 229–249 (2006)
13. Marcus, M., Santorini, B., Marcinkiewicz, M.: Building a large annotated corpus of English: the Penn Treebank. Computational Linguistics 19(2), 313–330 (1993)
14. Moschitti, A., Pighin, D., Basili, R.: Tree kernels for semantic role labeling. Computational Linguistics 34(2), 193–224 (2008)
15. Moschitti, A., Zanzotto, F.: Fast and effective kernels for relational learning from texts. In: Ghahramani, Z. (ed.) Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007) (2007)
16. Moschitti, A.: Making tree kernels practical for natural language learning. In: EACL. The Association for Computer Linguistics (2006)

17. Moschitti, A.: Kernel methods, syntax and semantics for relational text categorization. In: Proceeding of CIKM '08. NY, USA (2008)
18. Moschitti, A., Quarteroni, S., Basili, R., Manandhar, S.: Exploiting syntactic and shallow semantic kernels for question/answer classification. In: Proceedings of ACL'07 (2007)
19. Palmer, M., Kingsbury, P., Gildea, D.: The proposition bank: An annotated corpus of semantic roles. Computational Linguistics 31(1), 71–106 (2005)
20. Pighin, D., Moschitti, A.: Efficient linearization of tree kernel functions. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009). pp. 30–38. Association for Computational Linguistics, Boulder, Colorado (June 2009), `http://www.aclweb.org/anthology/W09-1106`
21. Pradhan, S., Hacioglu, K., Krugler, V., Ward, W., Martin, J.H., Jurafsky, D.: Support vector learning for semantic argument classification. Mach. Learn. 60(1-3), 11–39 (2005)
22. Shalev-Shwartz, S., Singer, Y., Srebro, N.: Pegasos: Primal estimated sub-gradient solver for svm. In: Ghahramani, Z. (ed.) ICML. ACM International Conference Proceeding Series, vol. 227, pp. 807–814. ACM (2007)
23. Shen, L., Joshi, A.K.: An svm-based voting algorithm with application to parse reranking. In: Daelemans, W., Osborne, M. (eds.) Proceedings of CoNLL HLT-NAACL 2003. pp. 9–16 (2003), `http://www.aclweb.org/anthology/W03-0402.pdf`
24. Surdeanu, M., Ciaramita, M., Zaragoza, H.: Learning to rank answers on large online QA collections. In: Proceedings of ACL-08: HLT. Columbus, Ohio (2008), `http://www.aclweb.org/anthology/P/P08/P08-1082`
25. Williams, C., Seeger, M.: Using the nystrm method to speed up kernel machines. In: Advances in Neural Information Processing Systems 13. pp. 682–688. MIT Press (2001)
26. Yu, C.N.J., Joachims, T.: Training structural svms with kernels using sampled cuts. In: ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD). pp. 794–802 (2008)
27. Zaki, M.J.: Efficiently mining frequent trees in a forest. In: KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 71–80. ACM Press, New York, NY, USA (2002)
28. Zhang, D., Lee, W.S.: Question classification using support vector machines. In: SIGIR. pp. 26–32. ACM (2003)
29. Zhang, M., Zhang, J., Su, J.: Exploring Syntactic Features for Relation Extraction using a Convolution tree kernel. In: Proceedings of NAACL. pp. 288–295. New York City, USA (2006), `http://www.aclweb.org/anthology/N/N06/N06-1037`