Fast Support Vector Machines for Structural Kernels

Aliaksei Severyn and Alessandro Moschitti

Department of Computer Science and Engineering
University of Trento
Via Sommarive 5, 38123 POVO (TN) - Italy
{severyn,moschitti}@disi.unitn.it

Abstract. In this paper, we propose three important enhancements of the approximate cutting plane algorithm (CPA) to train Support Vector Machines with structural kernels: (i) we exploit a compact yet exact representation of cutting plane models using directed acyclic graphs to speed up both training and classification, (ii) we provide a parallel implementation, which makes the training scale almost linearly with the number of CPUs, and (iii) we propose an alternative sampling strategy to handle class-imbalanced problem and show that theoretical convergence bounds are preserved. The experimental evaluations on three diverse datasets demonstrate the soundness of our approach and the possibility to carry out fast learning and classification with structural kernels.

1 Introduction

Various kernels have been successfully applied to different Natural Language Processing (NLP) tasks, e.g. [13, 5, 17, 12, 6, 2]. However, previous work is limited to relatively small datasets. Indeed, the major drawback of kernel methods is the necessity to carry out learning in dual spaces, where training complexity typically is quadratic in the number of instances.

Recently, a number of efficient CPA-based algorithms have been proposed [10, 8]. Unfortunately, these algorithms scale well only when linear kernels are used. To address slow learning with non-linear kernels [12] propose to extract basis vectors to compactly represent cutting plane models, which speeds up both classification and learning. However, this requires to solve a non-trivial optimization problem when arbitrary kernel functions are used. Finding a set of basis vectors in high-dimensional spaces produced by arbitrary kernels, structural kernels in particular, is an open research area and is definitely worth further exploration.

Another approach of adapting CPA for non-linear kernels by reducing the number of kernel evaluations is studied in [23], where sampling is used to reduce the number of basis functions in the kernel expansion. [16] showed that the same algorithm can be successfully applied to SVM learning with structural kernels on very large data obtaining speedup up factors up to 10 over conventional SVMs.

In this paper, we provide three important improvements of the approximate CPA with sampling. The proposed techniques make SVMs with structural kernels a viable tool to tackle real-world tasks. In particular, we first present an idea

to use (Directed Acyclic Graphs) DAGs to compactly represent cutting plane models computed at each iteration of the CPA algorithm. This has the benefit of reducing the number of expensive kernel evaluations, since DAGs provide the means to avoid redundant computations over shared substructures. We present two algorithms that deliver impressive speedups for both training and testing. We also parallelize the code improving scalability even further. Finally, we provide an effective and sound method to handle class imbalanced datasets, which plays an important role to obtain the optimal balance between Precision and Recall.

2 Preliminaries: Cutting Plane Algorithm with Sampling

In this section, we illustrate a re-elaborated version of the cutting plane method (originally proposed in the context of structural SVMs) for binary classification. After briefly explaining it for linear SVMs, we point out the main source of inefficiency for the case when kernels are used. Next we present the idea of [23] to use sampling to alleviate high training costs for SVMs with non-linear kernels.

2.1 Cutting-plane algorithm (primal)

Consider a slight modification of SVM training problem, known as a 1-slack reformulation [10], to derive CPA for binary classification[1]:

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

where binary vector $\boldsymbol{c} = (c_1, \ldots, c_n) \in \{0,1\}^n$ forms a constraint that is a linear combinations of the constraints $y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i) \geq 1 - \xi_i$.

The CPA is presented in Alg. 1. It starts with an empty set of constraints S and computes the optimal solution to the unconstrained problem (1). Next, the algorithm forms a binary vector $\boldsymbol{c}$ to compute a cutting plane model defined by its offset $d^{(t)} = \frac{1}{n}\sum_{i=1}^{n} c_i$ and gradient $\boldsymbol{g}^{(t)} = \frac{1}{n}\sum_{i=1}^{n} c_i y_i x_i$ (lines 5-9). This produces a constraint $\boldsymbol{w}\cdot\boldsymbol{g}^{(t)} \geq d^{(t)} - \xi$ that is violated the most by the current solution $\boldsymbol{w}$, which is included in the set of active constraints S (line 10). This way, a series of successively tightening approximations to the original problem is constructed. The algorithm stops when no constraints are violated by more than ϵ, which is formalized by the criteria in line 12.

[1] Here we fix the bias term b at zero, as it could be easily incorporated in feature vectors as an additional constant.

Algorithm 1 Cutting Plane Algorithm (primal)

1: Input: $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, C, ϵ
2: $S \leftarrow \emptyset; t = 0$
3: **repeat**
4: $(\boldsymbol{w}, \xi) \leftarrow$ optimize (1) over the constraints in S
/ find a cutting plane */*
5: **for** $i = 1$ to n **do**
6: $c_i^{(t)} \leftarrow \begin{cases} 1 \textbf{ if } \big(y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i) \leq 1\big) \\ 0 \textbf{ otherwise} \end{cases}$
7: **end for**
8: $d^{(t)} \leftarrow \frac{1}{n} \sum_{i=1}^{n} c_i$
9: $\boldsymbol{g}^{(t)} \leftarrow \frac{1}{n} \sum_{i=1}^{n} c_i y_i \phi(\boldsymbol{x_i})$
/ add a constraint to the set of constraints */*
10: $S \leftarrow S \cup \{(d^{(t)}, \boldsymbol{g}^{(t)})\}$
11: $t \leftarrow t + 1$
12: **until** $\boldsymbol{w} \cdot \boldsymbol{g}^{(t)} \geq d^{(t)} - \xi + \epsilon$
13: **return** $\boldsymbol{w}, \xi$

2.2 Cutting-plane algorithm (dual)

To enable the use of kernels, we need to solve the Wolfe dual of the problem (1). Its solution $\boldsymbol{w}$ lies in the feature space defined by a kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_k) = \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_i)$. It can be easily verified (by deriving the the dual from (1)) that primal and dual variables are connected via:

$$\boldsymbol{w} = \sum_{j=1}^{t} \alpha_j \boldsymbol{g}^{(j)}, \tag{2}$$

where $\boldsymbol{g^{(j)}} = \frac{1}{n} \sum_{k=1}^{n} c_k^{(j)} y_k \phi(\boldsymbol{x}_k)$ denotes the gradient of the cutting plane model added at iteration j and t is the size of the set S.

As one can see, with the use of kernels the gradient $\boldsymbol{g}^{(j)}$ (that also defines the most violated constraint (MVC) added at iteration j) cannot be compactly represented as in the linear case by simply summing up n feature vectors since it now lies in the feature space spanned by $\phi(\cdot)$. We will address the problem of compact representation of the cutting plane models in the next section.

Computing an inner product between the weight vector $\boldsymbol{w}$ and an example $\boldsymbol{x}_i$ involves the sum of kernel evaluations for each example $\boldsymbol{x}_k$ in the constraint j for each constraint in S. In particular, using the expansion of $\boldsymbol{w}$ from (2), the inner product required to compute the MVC (steps 5-9 in the Alg. 1), renders as:

$$\boldsymbol{w} \cdot \phi(\boldsymbol{x}_i) = \sum_{j=1}^{t} \alpha_j \boldsymbol{g}^{(j)} \cdot \phi(\boldsymbol{x}_i) = \sum_{k=1}^{n} \Big(\sum_{j=1}^{t} \frac{1}{n} \alpha_j c_k^{(j)} y_k \Big) K(\boldsymbol{x}_i, \boldsymbol{x}_k), \tag{3}$$

The analysis of the inner product given by (3) reveals that the number of kernel evaluations at each iteration is $O(tn^2)$. Indeed, the number of non-zero elements in each $\boldsymbol{g}^{(j)}$ is proportional to the number of support vectors which grows linearly

with the training size n [19]. Summing over all constraints in the set S, the complexity of (3) is $O(tn)$. Since the inner product (3) needs to be computed for each training example (lines 5-7 in Alg. 1) we obtain $O(tn^2)$ scaling behavior at each iteration.

The obtained quadratic scaling in the number of examples makes cutting plane training for non-linear SVMs prohibitively expensive for even medium-sized datasets. To address this limitation [23] proposed to construct approximate cuts by sampling r examples from the training set. The idea is to replace the expensive computation of the MVC (lines 5-7, Alg. 1) over all training examples n by a sum over a smaller sample r, s.t. the number of examples in $\boldsymbol{g}^{(j)}$ is reduced from $O(n)$ to $O(r)$. In this case the double sum of kernel evaluations in (3) reduces from $\sum_{i,j=1}^{n} K(\boldsymbol{x}_i, \boldsymbol{x_j})$ to a more tractable $\sum_{i,j=1}^{r} K(\boldsymbol{x}_i, \boldsymbol{x_j})$.

3 Fast CPA for Structural Kernels

In this section we present an approach to significantly speed up the approximate CPA for structural kernels. We observe that for convolution structural kernels, e.g. tree kernels, the cutting plane model can be compactly represented as a Directed Acyclic Graph (DAG). This helps to speed up both the training and classification as the repeating kernel evaluations over shared substructures can be avoided. Most interestingly this approach can be parallelized during training thus making structural kernel learning practical on larger datasets.

3.1 Compacting cutting plane models using DAGs

In the previous section we have seen that computing an MVC at each iteration involves quadratic number of kernel evaluations. Using smaller samples to approximate the cutting plane helps to reduce the number of kernel evaluations.

Here we explore another avenue to reduce the number of kernel computations when convolution structural kernels are used. Indeed, when applied to structural data such as sequences, trees or graphs, we can take advantage of the fact that many examples share common sub-structures. Hence, we can use a compact representation of a cutting plane model to avoid redundant computations over repeating sub-structures. In particular, when dealing with tree-structured data, a collection of trees can be compactly represented as a DAG [1]. In the following we briefly introduce the idea behind using DAGs to compactly represent a tree forest and then show how it applies to speed up the learning algorithm.

3.2 DAG tree kernels

A DAG can efficiently represent a set of trees by including only the unique subtrees and accounting for the frequency of the repeated substructures. Fig. 1 shows three syntactic trees on the left and the resulting DAG on the right. As we can see, the subtree of the noun phrase `[NP [D a][N car]]` is repeated in two trees, thus the frequency of the corresponding node is updated to 2. Also

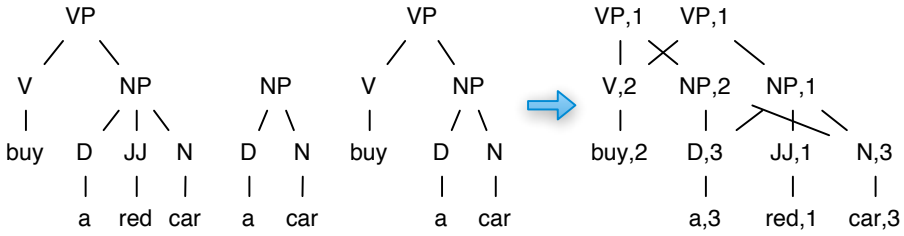


Fig. 1. Three syntactic trees and the resulting DAG.

smallesubtrees such as `[D a]` and `[D car]` are shared with a frequency of 3. The two subtrees rooted in `VP` are different and require different roots but they can still share some of their subparts, e.g. `[V buy]`.

Given a collection of trees, there are various methods to efficiently build a corresponding DAG and allow for fast access to its tree nodes, see for example [1]. In our approach, for each node in a tree, we generate a string representation of its subtree. This requires linear time in the number of tree nodes and can be done at the preprocessing step. These strings are unique identifiers of each respective node and serve as keys in the hash table, whose values are pointers to the corresponding nodes. To perform efficient search within a DAG, we maintain a simple and efficient nested structure of two associative arrays. The first is a hash table, which given a node retrieves the set of nodes associate with the same production rule. Each entry in the retrieved set contains a tuple of a pointer to the node and its current frequency. In this way we can efficiently enumerate all the candidate substructures to compute the tree kernel [4] between a DAG and a given tree.

Tree Kernels (TKs). Convolution TKs compute the number of common substructures between two trees T_1 and T_2 without explicitly considering the whole fragment space. For this purpose, let the set $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ be the substructure space and $\chi_i(n)$ be an indicator function, equal to 1 if the target t_i is rooted at node n and equal to 0 otherwise. A tree-kernel function over T_1 and T_2 is $TK(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2)$, N_{T_1} and N_{T_2} are the sets of the T_1's and T_2's nodes, respectively and $\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{T}|} \chi_i(n_1)\chi_i(n_2)$. The latter is equal to the number of common fragments rooted in the n_1 and n_2 nodes.

Theorem 1. *Let D be a DAG representing a tree forest F and $K_{dag}(D, T_2) = \sum_{n_1 \in N_D} \sum_{n_2 \in N_{T_2}} f(n_1)\Delta(n_1, n_2)$ then*

$$\sum_{T1 \in F} TK(T_1, T_2) = K_{dag}(D, T_2), \tag{4}$$

where $f(n_1)$ is the frequency associated with n_1 in the DAG.

Proof. Let $\mathcal{S}(F)$ the set of possible subtrees of F, i.e. the substructures whose leaves coincide with those of the original tree (in general $\mathcal{T} \neq \mathcal{S}$), then $\sum_{T_1 \in F} TK(T_1, T_2)$ $= \sum_{T_1 \in F} \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) = \sum_{T_1 \in F} \sum_{\substack{n_1 : t \in \mathcal{S}(T_1) \\ n_1 = r(t)}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2)$, where $r(t)$ is the root of the subtree t. The last expression is equal to $\sum_{\substack{n_1 : t \in \mathcal{S}(F) \\ n_1 = r(t)}}$ $\sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2)$. Let $\mathcal{S}'$ the unique subtrees of $\mathcal{S}$, we can rewrite the above equation as: $\sum_{\substack{n_1 : t \in \mathcal{S}'(F) \\ n_1 = r(t)}} f(n_1) \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) = \sum_{\substack{n_1 : t \in D \\ n_1 = r(t)}} \sum_{n_2 \in N_{T_2}} f(n_1)$ $\Delta(n_1, n_2) = \sum_{n_1 \in D} \sum_{n_2 \in N_{T_2}} f(n_1) \Delta(n_1, n_2)$. □

3.3 Fast Computation of the MVC on Structural Data

Having introduced the DAG tree kernel, we redefine the inner product (3) required to compute the MVC by compacting $\boldsymbol{g}^{(j)}$ into a single DAG model $\boldsymbol{dag}^{(j)}$:

$$\boldsymbol{w} \cdot \phi(\boldsymbol{x}_i) = \sum_{j=1}^{t} \alpha_j K_{dag}(\boldsymbol{dag}_{(j)}, \boldsymbol{x}_i) \tag{5}$$

Unlike (3), where each cutting plane $\boldsymbol{g}^{(j)}$ is an arithmetic sum of training examples, here we take advantage of the fact that a collection of trees can be efficiently put into an equivalent DAG. Please note that computing a kernel $K_{dag}(\cdot, \cdot)$ between an example and a DAG that represents a collection of trees yields an exact kernel value as shown in Th. 1. The benefit of such representation comes from the efficiency gains obtained by speeding up kernel evaluations over the sum of examples compacted into a single DAG.

Now we are ready to present the new cutting plane algorithm (see Alg. 2) adapted for the use of structural kernels. The weight vector $\boldsymbol{w}$ (line 6) is now expanded over the dual variables α_j obtained by solving Wolfe dual of (1) (line 5) and cutting plane models, each compactly represented by $\boldsymbol{dag}^{(j)}$. To compute the MVC we use a smaller set of examples uniformly sampled from the original training set. A binary vector $\boldsymbol{c}$ formed in (lines 8-12) defines the examples that are inserted into a DAG model (line 11). The obtained algorithm preserves all the theoretical benefits of the approximate CPA with sampling, while reducing the number of expensive kernel evaluations to compute the MVC.

To benefit even more from the compact representation offered by DAGs, we can put all cutting planes from the set S into a single DAG, such that the inner product (3) is reduced to a single kernel evaluation:

$$\boldsymbol{w} \cdot \phi(\boldsymbol{x}_i) = K_{dag}(\widehat{\boldsymbol{dag}}_{(t)}, \boldsymbol{x}_i) \tag{6}$$

where $\widehat{\boldsymbol{dag}}_{(t)}$ at iteration t is built by inserting nodes from $\boldsymbol{dag}_{(j)}$ together with the frequency counts multiplied by the value of the corresponding dual variable α_j. This ensures that a single K_{dag} evaluation over the full DAG model makes Eq. 6 equivalent to computing a weighted sum of K_{dag} using individual $\boldsymbol{dag}_{(j)}$ in Eq. 5. Even though $\widehat{\boldsymbol{dag}}_{(t)}$ has to be re-built at each iteration to

Algorithm 2 Cutting Plane Algorithm (dual) using DAG model representation

1: Input: $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, C, ϵ
2: $S \leftarrow \emptyset$; $\boldsymbol{dag} \leftarrow 0$; $t = 0$;
3: **repeat**
4: **Update the Gram matrix G with a new constraint**
5: $\boldsymbol{\alpha} \leftarrow \text{argmax}_{\boldsymbol{\alpha} \geq 0} \boldsymbol{h}^T \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^T G \boldsymbol{\alpha}$, s.t. $\boldsymbol{\alpha}^T \mathbf{1} \leq C$ where $h_i = d^{(i)}$ and $G = \boldsymbol{g}^{(i)} \cdot \boldsymbol{g}^{(j)}$
6: $\boldsymbol{w} = \sum_{j=1}^{|t|} \alpha_j \boldsymbol{dag}^{(j)}$
7: **Sample r examples from the training set**
/ find a cutting plane */*
8: **for** $i = 1$ to r **do**
9: $c_i^{(t)} \leftarrow \begin{cases} 1 & \textbf{if } \left(y_i \sum_{j=1}^{t} \alpha_j K_{dag}(\boldsymbol{dag}_j, \boldsymbol{x}_i) \leq 1\right) \\ 0 & \textbf{otherwise} \end{cases}$
10: **end for**
11: $\boldsymbol{dag}^{(t)} = \text{build\_dag}(\boldsymbol{c})$
12: $d^{(t)} = \frac{1}{r} \sum_{i=1}^{r} c_i$
13: */* add dag to the active set */*
14: $S \leftarrow S \cup \{(d^t, \boldsymbol{dag}^t)\}$
15: $t = t + 1$
16: **until** $d^{(t)} - \boldsymbol{w} \cdot \boldsymbol{dag}^{(t)} \leq \xi + \epsilon$
17: **return** $\boldsymbol{w}, \xi$

accommodate updated vector $\boldsymbol{\alpha}$, this imposes little computational overhead in practice. Another computational drawback of using full DAG model compared to the set of $\boldsymbol{dag}_{(j)}$ is that in the former case we need to compute the update of the Gram matrix column (line 4 in Alg.2) $G_{it} = \boldsymbol{g}^{(i)} \cdot \boldsymbol{g}^{(t)}$ for $1 \leq i \leq t$, while in the latter case it is obtained automatically from computing Eq. 5.

Even though the worst-case complexity for computing the MVC using both variants of using DAGs is still $O(r^2)$, it is highly unlikely to observe in practice, where input examples tend to share many common substructures. This speeds up both training and classification by avoiding redundant kernel computations.

3.4 Parallelization

The modular nature of the CPA suggests easy parallelization. In fact, in our experiments, we observed that at each iteration 95% of the total learning time is spent on computing the MVC (steps 8-12, Alg. 2). This involves computing Eq. 5 over the set of individual DAGs or Eq. 6 using full DAG model for r training examples in the sample. This observation suggests high parallelizability of the code: using p processors the complexity of this pre-dominant part can be brought down from $O(r^2)$ to $O(r^2/p)$.

4 Handling Class-Imbalanced data

In this section, we extend the theory of cutting-plane algorithm to tackle class-imbalance problem. Our approach is based on an alternative sampling strategy

that is effective for tuning up Precision and Recall on class-imbalanced data. We also provide a convergence proof of the proposed algorithm.

4.1 Cost-proportionate sampling

Conventional SVM problem formulation allows for natural incorporation of example dependent importance weights into the optimization problem. We can modify the objective function to include example dependent cost factors:

$$\begin{aligned} \underset{w,\xi_i \geq 0}{\text{minimize}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{n}\sum_{i}^{n} z_i \xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w} \cdot \boldsymbol{x_i}) \geq 1 - \xi_i,\ 1 \leq i \leq n \end{aligned} \tag{7}$$

where z_i is the importance weight of example i and $\frac{1}{n}\sum_i^n z_i\xi_i$ serves as an upper bound on the total cost-sensitive empirical risk. This problem formulation where there is an individual slack variable ξ_i for each example is typically referred to as "n-slack" formulation.

In the dual space, the example-dependent costs captured by cost factors z_i translate into the box constraints imposed on each dual variables: $0 \leq \alpha_i \leq z_i C,\ 1 \leq i \leq n$ such that the $z_i C$ sets an upper bound on the values of α_i. This feature to integrate importance weights z_i in the problem formulation is implemented in SVM-light software.

This natural modification of the quadratic problem, is, however, difficult to incorporate in the case of 1-slack formulation (1). Indeed, in the case of 1-slack formulation we have a single slack variable ξ that is shared among all the constraints. More importantly, moving to the dual space, the box constraints $0 \leq \alpha_i \leq C$ are no longer for each individual dual variable but for a sum: $\sum_i \alpha_i$. This makes the 1-slack problem formulation difficult to incorporate importance weights directly. Nevertheless, the idea of approximating the cutting plane model at each iteration via sampling suggests a straightforward solution.

Indeed, we can extend the original CPA to the case of cost-sensitive classification. A straight-forward way to do this is instead of using uniform sampling to build an approximation to the cutting plane model at each iteration (steps 8-12 in Alg. 2), we can draw examples according to their importance weights using the cost-proportionate rejection sampling technique (Alg. 3).

Algorithm 3 Cost-proportionate rejection sampling

1: Pick example $(\boldsymbol{x_i}, y_i, z_i)$ at random
2: Flip a coin with bias z_i/Z
3: **if** *heads* **then**
4: keep the example
5: **else**
6: discard it
7: **end if**

Here z_i is the importance weight of the i-th example and Z is an upper bound on any importance value in the dataset. This process is repeated until we sample the required number of examples r. This modification enables the control over the proportion of examples from different classes that will form a sample used to compute the MVC.

Unlike the conventional approaches for addressing the class-imbalance problem, that either under-sample the majority class or over-sample the minority class from the training data, the rejection sampling coupled with CPA does not completely discard examples from the training set. At each iteration it forms a sample according to the pre-assigned importance weights for each example, such that examples from both the majority and minority classes enter the sample in the desired proportion. This process is repeated until the algorithm converges. Thus, the learner has the chance to incorporate relevant information present in the data over a number of iterations before it converges. This way, the method preserves the global view on the dataset and no relevant information is lost during the iterative optimization process unlike in the "one-shot" sampling methods.

Another benefit of this approach is that by increasing the importance weight of the minority class, we give its examples more chance to end up in the MVC and hence, become support vectors. This way the imbalanced support-vector ratio is automatically tuned to include more examples from the minority class, which gives more control over the class-imbalance problem. Proving this property could be an interesting theoretical result.

4.2 Theoretical Analysis of the Algorithm

Cost proportionate rejection sampling allows for natural extension of the binary classification to importance weighted binary classification. It achieves this task by re-weighting the original distribution of examples D according to the importance weights of examples such that the training is effectively carried out under the new distribution $\hat{D}$.

In [24] it is shown that by transforming the original distribution D to a training set under $\hat{D}$, one can effectively train a **cost-insensitive** classifier on a dataset $\hat{D}$ such that it will minimize the expected risk under the original distribution D.

Theorem 2. (Translation Theorem; [24]) *Learning a classifier h to minimize the expected cost-sensitive risk under the original distribution D is equivalent to learning a decision function to minimize the expected **cost-insensitive** risk under the distribution $\hat{D}(x, y, z) \equiv \frac{z}{E_{(x,y,z)\sim D}[z]} D(x, y, z)$.*

The proof is a straight-forward application of the definitions and simply follows by establishing an equivalence relationship between the expected cost-sensitive risk $E_{(x,y,z)\sim D}[z\Delta(y, h(x))]$ under the original distribution D and the expected cost-insensitive risk $E_{(x,y,z)\sim \hat{D}}[\Delta(y, h(x))]$ under the transformed distribution $\hat{D}$. The theorem produces an important implication that by transforming the original distribution D to $\hat{D}$ according to example-dependent importance weights,

a classifier for the cost-sensitive problem over D can be obtained with a cost-insensitive learning algorithm over D. We can use this finding to show that the convergence proof for the original CPA with uniform sampling naturally applies to the proposed version of the algorithm that uses cost-proportionate rejection sampling:

Theorem 3. (Convergence) *Assume* $R = max_{1 \leq i \leq n} \|\phi(\boldsymbol{x}_i)\|$, *i.e.* R *is an upper bound on the norm of any* $\phi(\boldsymbol{x}_i)$, *and* $\Delta = max_{1 \leq i \leq n} \| \Delta(y, y_i)\|$, *the number of steps required by Alg. 2 using the sampling strategy of Alg. 3 is upper bounded by* $8C\Delta R^2/\epsilon^2$.

Proof. We first note that the cost-proportionate rejection sampling (Alg. 3), used to build the approximate cutting plane model, at each step re-weights the original distribution D according to the importance weights of the examples. This means that we are effectively training a cost insensitive classifier that draws examples to build the cutting plane model from the transformed distribution $\hat{D}$. By invoking the Translation Theorem (2), we establish that, to obtain a cost-sensitive classifier that minimizes the expected risk under the original distribution D, it is sufficient to learn a cost-insensitive classifier under the transformed distribution $\hat{D}$. The CPA that draws examples from D using rejection sampling is equivalent to the original CPA applying uniform sampling to the transformed distribution $\hat{D}$. Thus, we can reutilize the proof in [23] of the convergence bounds for the original CPA with uniform sampling over $\hat{D}$. This states that CPA with uniform sampling terminates after at most $8C\Delta R^2/\epsilon^2$ iterations. By applying such bound, we have proved the thesis of the theorem.

Remarks. The main idea to obtain convergence bounds in [23] is to set an upper bound on the value of the dual objective and if there exists a lower bound on the minimal improvement of the dual objective at each iteration, then the algorithm will terminate in a finite number of steps.

Indeed, using the relationship between primal and dual problems, we have that a feasible solution of the primal problem (1), such as, for example: $\boldsymbol{w} = \mathbf{0}$, $\xi = \Delta$, forms an upper bound $C\Delta$ on the dual objective of 1. Next, in [20] it is shown that the inclusion of ϵ-violated constraint at each iteration improves the dual objective by at least $\epsilon/8R^2$. Since the dual objective is upper bounded by $C\Delta$, the algorithm terminates after at most $8C\Delta R^2/\epsilon^2$ iterations.

The derivation of the bound on the minimal improvement of the dual objective obtained at each step only depends on the values of ϵ and R and does not rely on the assumption about distribution of the examples. Also note that each cutting plane model built via rejection sampling is a valid constraint for the optimization problem (1).

5 Experiments

In our experiments we pursue a three-fold goal: (i) study the effects of compacting the cutting plane model by using DAGs on both training and classification

runtimes; (ii) demonstrate the speedup factors one can obtain after straightforward parallelization offered by the CPA; and (iii) demonstrate the ability of the cost-proportionate sampling scheme to tune up Precision and Recall;

5.1 Experimental setup

We integrated CPA with uniform sampling as described in [23] within the framework of SVM-Light-TK [14, 9] to enable the use of structural kernels, e.g. tree kernels. For the DAG implementation we employ highly efficient Judy arrays[2]. For brevity, we refer to the CPA with uniform sampling as uSVM; uSVM where each cutting plane $\boldsymbol{g}^{(j)}$ is compacted into a $\boldsymbol{dag}_{(j)}$ as SDAG; uSVM with a single DAG that fits all active constraints in the set S as SDAG+; uSVM with rejection sampling as uSVM+j (Alg. 3), and SVM-light-TK as SVM. Parallel implementation relies on the OpenMP library.

To carry out learning, we used the subset tree (SST) kernel [4] since it has been indicated as the most accurate in similar tasks, e.g. [14]. As the stopping criteria of the algorithms, we fix the precision parameter ϵ at 0.001. The margin trade off parameter is fixed at 1.0. To measure the classification performance, we use Precision, Recall and F_1-score. We ran all the experiments on machines equipped with Intel® Xeon® 2.33GHz CPUs carrying 6Gb of RAM under Linux.

5.2 Data and models

To evaluate the efficiency of the compact model representation offered by SDAG and SDAG+ algorithms with respect to uSVM, we use Semantic Role Labeling (SRL) benchmark, using PropBank annotations [15] and automatic Charniak parse trees [3]. SRL dataset has already been used to extensively test uSVM for structural kernels and we follow the same setting as described in [16].

In the next set of experiments to study the ability of uSVM+j to tune up Precision and Recall we used two different natural language datasets: TREC 10 QA[3] (training: 5,483, test: 500) and Yahoo! Answers (YA)[4](train: up to 300k, test: 10k) to perform two similar tasks of QA classification. The task for the first dataset is to select the most appropriate type of the answer from a set of given possibilities. The goal of the experiments on these relatively small datasets is to demonstrate that rejection sampling is able to effectively handle class imbalance similar to SVM. For Yahoo! Answers dataset the classification task was set up as follows. Given pairs of questions and corresponding answers learn if in a given pair the answer is the 'best' answer for a question. The goal of this experiment is to have a large classification task (300k examples in our experiments) to demonstrate that class-imbalance problem can be handled effectively at a scale where SVM becomes too slow.

[2] `http://judy.sourceforge.net`
[3] `http://l2r.cs.uiuc.edu/cogcomp/Data/QA/QC/`
[4] retrieved through the Yahoo! Webscope program.

Table 1. Runtime comparison between uSVM, SDAG and SDAG+. Training on 100k subset of SRL across multiple values of the sample size (left) and classification on 10k subset when learning on SRL subsets of varying size (right). Time indicated in seconds; values in parenthesis for SDAG and SDAG+ are speedups w.r.t uSVM; #SVs- number of support vectors.

Training			
sample	uSVM	SDAG	SDAG+
1000	2196	312 (7.0)	283 (7.8)
2000	8282	1127 (7.3)	752 (11.0)
3000	18189	2509 (7.2)	1275 (14.3)
4000	31012	4306 (7.2)	1802 (17.2)
5000	50060	6591 (7.6)	2497 (20.0)

Classification				
Data	#SVs	uSVM	SDAG	SDAG+
10k	1686	11	9 (1.1)	1 (24.0)
25k	3392	41	25 (1.6)	1 (33.2)
50k	5876	82	40 (2.1)	3 (28.3)
75k	7489	112	55 (2.0)	5 (20.5)
100k	8674	131	59 (2.2)	7 (19.5)

5.3 Results and Analysis

Compact model representation using DAGs. The goal of this set of experiments is to study computational savings that come from using a compact representation of individual (SDAG) or the full set (SDAG+) of cutting plane models in S. As the baseline for the learning and classification runtime comparison we use plain uSVM algorithm. To carry out training we use 100k of SRL dataset. The number of iterations for the algorithms is fixed at 300 and the rest of the parameters are kept at default values. For evaluating speedups obtained during classification phase we carry out learning on SRL subsets of increasing size and then test trained models on 10k of data.

Runtime results for SDAG, SDAG+ and uSVM are reported in Table 1 (since SDAG and SDAG+ produce exact kernel evaluations, hence they train the same model as uSVM, accuracy is not of concern and is omitted). As one can see both SDAG and SDAG+ deliver significant speedups during the learning. SDAG+ is a clear winner here delivering speedups up to 20 when a large sample size is used. Regarding classification, SDAG+ is also the fastest, although as the number of support vectors of the learned model increases, the speedup factor decreases. This is due to the increased overhead of maintaining a single large DAG. As the number of elements in the DAG grows the inefficiencies from the implementation of the underlying data structure slow down the node lookup time. We plan to address this problem in the future.

Tuning up Precision and Recall. We first report experimental results on question classification corpus on six different categories in Table 2 (since the dataset is small, we only report the accuracy). For both uSVM and uSVM+j, we fixed the sample size to 100. For uSVM+j, we picked the value of j from {1, 2, 3, 4, 5, 10} and use the best results obtained on the validation set. For SVM, we carried out tuning of j parameter on a validation set. It is important to note that such parameter has slightly different meaning for uSVM+j and SVM. For the former, it controls the bias to reject negative examples during sampling (Alg. 3) to compute MVC, while for the latter it defines the factor by which training errors on positive examples outweigh errors on negative examples.

Table 2. Handling class-imbalance problem on TREC 10 (top) YA (bottom). Ratio - proportion of negative examples w.r.t. positive; P/R - precision (P) and recall (R). The bottom row in YA is the performance using bag-of-words features on 75k subset.

TREC 10

DATA	RATIO	uSVM		uSVM+J		SVM	
		F-1	P/R	F-1	P/R	F-1	P/R
ABBR	1:60	87.5	100.0/77.8	84.2	80.0/88.9	84.2	80.0/88.9
DESC	1:4	96.1	95.0/97.1	96.1	95.0/97.1	94.8	97.7/92.0
ENTY	1:3	72.3	91.8/59.6	79.1	79.6/78.7	80.4	82.2/78.7
HUM	1:3	88.1	98.1/80.0	90.3	94.9/86.2	87.5	88.9/86.2
LOC	1:3	81.4	96.6/70.4	87.0	87.5/86.4	82.6	86.5/79.0
NUM	1:5	86.0	98.9/76.1	91.2	96.1/86.7	89.9	98.9/82.3

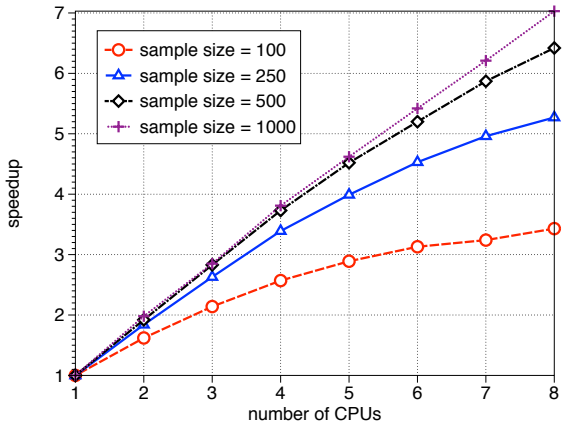
YAHOO ANSWERS

10K	1:1.5	37.4	33.5/42.2	39.1	29.6/57.7	37.9	24.2/87.7
50K	1:2.0	36.5	36.0/36.9	40.6	30.0/62.5	39.6	25.7/86.9
100K	1:2.4	33.4	36.2/31.1	40.2	30.2/59.9	40.3	26.6/83.5
150K	1:2.8	33.5	36.9/30.7	41.0	30.2/64.0	-	-
300K	1:3.4	23.8	40.1/16.9	41.4	30.7/63.8	-	-
BOW	1:2.0	34.2	33.2/35.3	38.1	27.5/61.7	36.3	22.5/93.5

Analyzing the results from Table 2 (top), we can see that uSVM algorithm that uses uniform sampling obtains high Precision, as it minimizes the training error dominated by examples from negative class. This results in lower values of the Recall. Its rather high F_1 for ABBR dataset shows that the model simply misclassifies the examples from the minority class saturating the Precision. On the other hand, uSVM+j is able to establish a much better balance between Precision and Recall resulting in high F_1 scores across the majority of categories. Also the performance of SVM with the optimal set of parameters suggests that our method has a better capacity to control the imbalance problem than SVM. This can be explained by the fact, as suggested in [22], that $z_i C$ imposes only an upper bound on dual variables α_i, which results in poorer flexibility to control the class-imbalance with the j parameter of SVM.

The results on Yahoo! Answers are displayed in Table 2 (bottom). For uSVM and uSVM+j, we fix the sample size at 500. Due to the constant time scaling behavior of uSVM [23], the training time for both uSVM and uSVM+j was slightly less than 10 hours across all subsets reported here. While being faster on small subsets of 5k, 10k and 25k, SVM begins to scale poorly on the subsets larger than 50k. Indeed, as studied in [23, 16], CPA with sampling begins to outperform SVM starting from datasets of moderate size (around 50k in our experiments). SVM did not finish the training within 5 days for 150k and 300k subsets, hence there are missing values. We set the value of j parameter for uSVM+j equal to the ratio of negative to positive examples. This natural setting of j parameter for uSVM+j is driven by the intuition to make the distribution of examples from different classes approximately balanced inside each sample, such that the classifier learns on a balanced data. As one can see, this gives much

Fig. 2. Speedups due to parallelizing SDAG/SDAG+ on 50k Yahoo! Answers dataset.



better trade-off between Precision and Recall compared to uSVM. Looking at the results of SVM, we conjecture that here j parameter, similar to the results in previous experiments, is not flexible enough to deliver the optimal P/R trade-off. Also note that training SVM on 100k subset requires almost 4 days, which makes uSVM+j a viable tool for advanced text classification on large datasets, where obtaining optimal balance between Precision and Recall is hindered by the class imbalance problem.

The bottom row of Table 2 reports the results using bag-of-words (BOW) feature representation on 75k subset. We note that SST kernel delivers an interesting 12% of relative improvement over BOW model on SVM. However, the main goal of this experiment was not to obtain the top classification performance on such noisy web data but rather to demonstrate that uSVM+j can efficiently deal with large imbalanced data.

Parallelization. To assess the effects of parallelization, we tested parallel versions of SDAG and SDAG+ on 50k subset of Yahoo! Answers dataset using up to 8 CPUs. The achieved speedups over the sequential algorithm are reported in Figure 2, where each curve corresponds to runtimes using different sample sizes: {100, 250, 500, 1000}. Increasing the sample size leads to the increase of the time spent to compute MVC, which makes the speedup achieved by parallelization for large sample sizes even more significant. Using the maximum number of 8 CPUs, we are able to achieve the speedup factor of about 7.0 (using sample size equal to 1000). Since classification can also be easily parallelized, we could experiment with larger sample sizes to obtain a more accurate model.

6 Related work

To improve the scaling properties of SVM-light, a number of efficient algorithms using CPA-based algorithms have been proposed. For example, SVM^{perf} [10] exhibits linear computational complexity in the number of examples when linear

kernels are used. While CPA-based approaches deliver state of the art performance w.r.t. accuracy and training time, they scale well only when linear kernels are used. The problem of efficient kernel learning for CPA has been studied in [11], where cutting plane models are compacted by extracting basis vectors. This, however, leads to a non-trivial optimization problem when arbitrary kernel functions are applied.

Regarding learning with structural kernels, compact representation of tree forests offered by DAGs was applied for speeding up training of the voted perceptron algorithm in [1]. Another interesting idea of hash kernels for structured data is proposed in [18], where hashing can generate explicit vector representation such that linear learning methods can be applied. However, it is likely that hashing all possible substructures generated by SST kernel, which is exponential in the tree length, will make the preprocessing step too expensive. Also, due to hash collisions, this method computes approximate kernel values and its implications on the accuracy need to be studied more extensively.

Concerning class-imbalance problem for SVM learning, the most widely adopted method is to introduce different cost factors in the objective function s.t. the training errors for positive and negative examples receive different penalties [21]. This approach is implemented as the j option in SVM-light [9] that has a super-linear scaling behavior, which prohibits its use on large datasets. Our approach to accomplish cost-sensitive classification shares the idea of reductions put forward in [24] together with the benefit of the conventional approach in SVMs [21] to incorporate importance weights directly into the optimization process.

7 Conclusions and Future Work

In this paper we have presented a set of techniques to make SVMs with structural kernels a more useful tool to apply in real-world tasks. First, we derive two learning algorithms SDAG and SDAG+ that compact cutting plane models using DAGs. This makes both learning and classification much faster when compared to the original CPA with sampling. Next, we present parallelized versions of both algorithms to deliver even faster runtimes. Finally, we propose an alternative sampling strategy to efficiently handle class-imbalanced data. The distinctive property of the proposed method is that it directly integrates the cost-proportionate sampling into the CPA optimization process, unlike the other sampling approaches based on the reductions idea of [24]. In other words, sampling is carried out iteratively, such that no information is discarded from training examples as in "one-shot" sampling methods.

Acknowledgements

This work has been partially supported by the EC project FP247758: Trustworthy Eternal Systems via Evolving Software, Data and Knowledge (EternalS).

References

1. Aiolli, F., Martino, G.D.S., Sperduti, A., Moschitti, A.: Efficient kernel-based learning for trees. In: CIDM. pp. 308–315 (2007)
2. Cancedda, N., Gaussier, E., Goutte, C., Renders, J.M.: Word sequence kernels. Journal of Machine Learning Research 3, 1059–1082 (2003)
3. Charniak, E.: A maximum-entropy-inspired parser. In: ANLP. pp. 132–139 (2000)
4. 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)
5. Cumby, C., Roth, D.: Kernel Methods for Relational Learning. In: Proceedings of ICML 2003 (2003)
6. Daumé III, H., Marcu, D.: A tree-position kernel for document compression. In: Proceedings of the DUC. Boston, MA (May 6 – 7 2004)
7. Fan, R., Chen, P., Lin, C.: Working set selection using the second order information for training svm. Journal of Machine Learning Research 6, 1889–1918 (2005)
8. Franc, V., Sonnenburg, S.: Optimized cutting plane algorithm for support vector machines. In: ICML. pp. 320–327 (2008)
9. Joachims, T.: Making large-scale SVM learning practical. In: Advances in Kernel Methods - Support Vector Learning, chap. 11, pp. 169–184. MIT Press, Cambridge, MA (1999)
10. Joachims, T.: Training linear SVMs in linear time. In: KDD (2006)
11. Joachims, T., Yu, C.N.J.: Sparse kernel svms via cutting-plane training. Machine Learning 76(2-3), 179–193 (2009), eCML
12. Kate, R.J., Mooney, R.J.: Using string-kernels for learning semantic parsers. In: ACL (July 2006)
13. Kudo, T., Matsumoto, Y.: Fast methods for kernel-based text analysis. In: Proceedings of ACL'03 (2003)
14. Moschitti, A.: Making tree kernels practical for natural language learning. In: EACL. The Association for Computer Linguistics (2006)
15. Palmer, M., Kingsbury, P., Gildea, D.: The proposition bank: An annotated corpus of semantic roles. Computational Linguistics 31(1), 71–106 (2005)
16. Severyn, A., Moschitti, A.: Large-scale support vector learning with structural kernels. In: ECML/PKDD (3). pp. 229–244 (2010)
17. Shen, L., Sarkar, A., Joshi, A.k.: Using LTAG Based Features in Parse Reranking. In: Proceedings of EMNLP'06 (2003)
18. Shi, Q., Petterson, J., Dror, G., Langford, J., Smola, A.J., Vishwanathan, S.V.N.: Hash kernels for structured data. JMLR 10, 2615–2637 (2009)
19. Steinwart, I.: Sparseness of support vector machines. Journal of Machine Learning Research 4, 1071–1105 (2003)
20. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. Journal of Machine Learning Research 6, 1453–1484 (2005)
21. Veropoulos, K., Campbell, C., Cristianini, N.: Controlling the sensitivity of support vector machines. In: Proceedings of the IJCAI. pp. 55–60 (1999)
22. Wu, G., Chang, E.: Class-boundary alignment for imbalanced dataset learning. ICML 2003 workshop on learning from imbalanced data sets II, Washington, DC pp. 49–56 (2003)
23. Yu, C.N.J., Joachims, T.: Training structural svms with kernels using sampled cuts. In: KDD. pp. 794–802 (2008)
24. Zadrozny, B., Langford, J., Abe, N.: Cost-sensitive learning by cost-proportionate example weighting. In: Proceedings of ICDM (2003)