# Training Structural SVMs with Kernels Using Sampled Cuts

Chun-Nam John Yu
Department of Computer Science
Cornell University
Ithaca, NY 14853
cnyu@cs.cornell.edu

Thorsten Joachims
Department of Computer Science
Cornell University
Ithaca, NY 14853
tj@cs.cornell.edu

## ABSTRACT

Discriminative training for structured outputs has found increasing applications in areas such as natural language processing, bioinformatics, information retrieval, and computer vision. Focusing on large-margin methods, the most general (in terms of loss function and model structure) training algorithms known to date are based on cutting-plane approaches. While these algorithms are very efficient for linear models, their training complexity becomes quadratic in the number of examples when kernels are used. To overcome this bottleneck, we propose new training algorithms that use approximate cutting planes and random sampling to enable efficient training with kernels. We prove that these algorithms have improved time complexity while providing approximation guarantees. In empirical evaluations, our algorithms produced solutions with training and test error rates close to those of exact solvers. Even on binary classification problems where highly optimized conventional training methods exist (e.g. SVM-light), our methods are about an order of magnitude faster than conventional training methods on large datasets, while remaining competitive in speed on datasets of medium size.

## Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

## General Terms

Algorithms, Experimentation, Performance

## Keywords

Support Vector Machines, Kernels, Large-Scale Problems

## 1. INTRODUCTION

Large-margin methods for structured output prediction like Maximum-Margin Markov Networks [13] and Structural SVMs [15] have recently received substantial interest for challenging problems in natural language processing [14], bioinformatics [19], and information retrieval [20]. As training algorithms for these problems, cutting-plane approaches [15, 6] are among the most generally applicable methods that provide well-understood performance guarantees. First, cutting-plane methods can be use to train any type of structured linear prediction model for which inference and subgradient can be computed (or at least approximated) efficiently. This makes them applicable to problems ranging from HMM training and natural language parsing, to supervised clustering and learning ranking functions. Second, they allow optimizing directly to non-standard loss functions that do not necessarily have to decompose linearly (e.g. Average Precision, ROC-Area, $F_1$-score) [4, 20]. And third, their runtime provably scales linearly with the number of training examples for linear models. This makes cutting-plane methods not only attractive for training structured prediction models, but they are also orders of magnitude faster than conventional methods for training binary classifiers [5].

Unfortunately, the computational efficiency of cutting-plane methods becomes substantially worse for non-linear models that involve kernels. While it is possible to train kernel models, the computational complexity scales quadratically with the number of examples, not linearly as in the non-kernel case. In particular, each iteration of the algorithm requires a quadratic number of kernel evaluations. This makes it infeasible to train large-scale structural models that involve kernels, and it makes cutting-plane methods non-competitive for training kernelized binary classifiers compared to conventional decomposition methods like SVM-light.

In this paper we present new cutting-plane training methods for structural SVMs that can be used to train kernelized models efficiently. These methods are equally broadly applicable, requiring only the ability to compute subgradients efficiently, but exploit sparse approximations to each cut in order to limit the number of kernel computations. In particular, we present two new cutting-plane methods that exploit random sampling in computing a cut, so that the number of kernel evaluations depends only linearly on the number of examples in one algorithm, or is independent of the number of examples in the other algorithm. Instead, the number of kernel evaluations depends only on the quality of the solution that the user desires and that is sensible for the learning task. In addition to providing theoretical guarantees regarding runtime and quality of the solutions, we also provide empirical results in comparison to conventional decomposition methods and a subspace method that uses a Cholesky decomposition.

## 2. STRUCTURAL SVMS

Structual SVMs are a method for learning rules $h : \mathcal{X} \rightarrow \mathcal{Y}$ from some space $\mathcal{X}$ of complex and structured objects

*KDD'08,* August 24–27, 2008, Las Vegas, Nevada, USA.

$x \in \mathcal{X}$ to some space $\mathcal{Y}$ of complex and structured objects $y \in \mathcal{Y}$ (e.g. sentences $\mathcal{X}$ to parse trees $\mathcal{Y}$ in natural language parsing). Given a labeled training sample

$$S = ((x_1, y_1), \ldots, (x_N, y_N)),$$

structural SVMs learn a linear discriminant rule

$$h(x) = \text{argmax}_{y \in Y}\, \vec{w} \cdot \Psi(x, y)$$

by minimizing a regularized version of the empirical risk $R_S(h) = \sum_{i=1}^{N} \Delta(y_i, h(x_i))$ for a given non-negative loss function $\Delta$. In the case of margin-rescaling [15, 13] we consider in this paper, training a structural SVM amounts to solving the following quadratic program.

Optimization Problem 1. (Struct SVM Primal)

$$\begin{aligned} \min_{\vec{w}, \xi \geq 0} \quad & \frac{1}{2}\|\vec{w}\|^2 + \frac{C}{N}\sum_{i=1}^{N} \xi_i \\ s.t. \quad & \forall \hat{y}_1 \in \mathcal{Y} : \Delta(y_1, \hat{y}_1) - \vec{w} \cdot \delta\Psi_1(\hat{y}_1) \leq \xi_1 \\ & \vdots \\ & \forall \hat{y}_N \in \mathcal{Y} : \Delta(y_N, \hat{y}_N) - \vec{w} \cdot \delta\Psi_N(\hat{y}_N) \leq \xi_N \end{aligned}$$

We use the short-hand notation $\delta\Psi_i(\hat{y}) := \Psi(x_i, y_i) - \Psi(x_i, \hat{y})$. While this program is convex, it has an exponential or infinite number of constraints (i.e. proportional to $|Y|$) on most interesting problems, making naive approaches to its solution intractable. Fortunately, it can be shown that the cutting-plane Algorithm 1 can nevertheless solve OP1 to arbitrary precision $\epsilon$.

**Algorithm 1** 1-Slack Cutting Plane Algorithm

```
Input: $S = ((x_1, y_1), \ldots, (x_N, y_N)), C, \epsilon$
$J = \{\}, t = 0, \vec{w} = \vec{0}, \xi = 0$
$(c^{(t)}, \vec{g}^{(t)}) = $ Find_Cutting_Plane($\vec{w}$)
while $c^{(t)} + \vec{w} \cdot \vec{g}^{(t)} > \xi + \epsilon$ do
    $J = J \cup \{(c^{(t)}, \vec{g}^{(t)})\}$
    $t = t + 1$
    $(\vec{w}, \xi) = $ Solve_QP($J$)
    $(c^{(t)}, \vec{g}^{(t)}) = $ Find_Cutting_Plane($\vec{w}$)
end while
return $(\vec{w}, \xi)$
procedure Find_Cutting_Plane($\vec{w}$)
    for $i = 1$ to $N$ do
        $\hat{y}_i = \text{argmax}_{\hat{y} \in \mathcal{Y}}(\Delta(y_i, \hat{y}) + \vec{w} \cdot \Psi(x_i, \hat{y}))$
    end for
    return $(\frac{1}{N}\sum_{i=1}^{N} \Delta(y_i, \hat{y}_i), -\frac{1}{N}\sum_{i=1}^{N} \delta\Psi_i(\hat{y}_i))$
end procedure
procedure Solve_QP($J$)
    $(\vec{w}, \xi) = \begin{cases} \text{argmin}_{\vec{w}, \xi \geq 0} \frac{1}{2}\|\vec{w}\|^2 + C\xi \\ \text{s.t. } \forall (c, \vec{g}) \in J, c + \vec{w} \cdot \vec{g} \leq \xi \end{cases}$
    return $(\vec{w}, \xi)$
end procedure
```

This cutting plane algorithm is currently one of the fastest solution method for large margin structural learning problems. Its time complexity scales linearly with the number of examples $N$ [5, 6] when the learned discriminant function $\vec{w} \cdot \Psi(x, y)$ is linear. However, with the use of kernels, it becomes neccessary to work in the dual and Algorithm 1 now scales quadratic in the number of examples. To see this, let's look at this dual optimization problem.

Optimization Problem 2. (Cutting-Plane Dual)

$$\begin{aligned} \max_{\vec{\alpha} \in \mathbb{R}^t} \quad & -\frac{1}{2}\vec{\alpha}^T G \vec{\alpha} + \vec{h}^T \vec{\alpha} \\ s.t. \quad & \vec{\alpha} \geq \vec{0} \textit{ and } \vec{\alpha}^T \vec{1} \leq C \end{aligned}$$

*where* $G_{ij} = \vec{g}^{(i)} \cdot \vec{g}^{(j)}$ *and* $h_i = c^{(i)}$ *for* $i, j = 1$ *to* $t$.

The primal and dual solution are related via $\vec{w} = -\sum_{i=1}^{t} \alpha_i \vec{g}^{(i)}$. One of the major issue with the dual algorithm is the computation of the inner product $\vec{g}^{(i)} \cdot \vec{g}^{(j)}$ in the nonlinear case,

$$\begin{aligned} & \vec{g}^{(i)} \cdot \vec{g}^{(j)} \\ = & \frac{1}{N^2}\sum_{k=1}^{N}\sum_{l=1}^{N}[\Psi(x_k, y_k) - \Psi(x_k, \hat{y}_k^{(i)})] \cdot [\Psi(x_l, y_l) - \Psi(x_l, \hat{y}_l^{(j)})] \\ = & \frac{1}{N^2}\sum_{k=1}^{N}\sum_{l=1}^{N}[K(x_k, y_k, x_l, y_l) - K(x_k, y_k, x_l, \hat{y}_l^{(j)}) \\ & \qquad - K(x_k, \hat{y}_k^{(i)}, x_l, y_l) + K(x_k, \hat{y}_k^{(i)}, x_l, \hat{y}_l^{(j)})] \end{aligned} \tag{1}$$

which involves $O(N^2)$ kernel computations. This makes Algorithm 1 impractical even if $N$ is only moderately large. Removing this bottleneck is central to our approach.

## 3. RELATED WORKS

There has been many training methods proposed in the structural learning literature. The Maximum-Margin Markov Networks [13] use SMO[8] for training with linearly decomposable loss functions, while the more general framework of structural SVM [15] introduces the cutting plane method as a training procedure. Subgradient methods [9] have also been proposed as an efficient training method for structural learning. Recently a faster 1-slack version of the cutting plane algorithm [6] has been introduced to solve large margin structural learning problems. A generalization of the cutting plane method called the bundle method [12] has also been recently proposed for the minimization of different convex loss functions in structural learning. Most of these works consider only linear discriminant functions. Our work continues this line of research by extending the cutting plane method to structural learning with kernels.

Our work is also related to the use of stochastic optimization in structural learning. The work in [16] investigated the use of stochastic gradient in the training of Conditional Random Fields, while the work in [11] employed stochastic subgradient to train linear SVMs. In stochastic optimization methods, decreasing step sizes or more accurate estimates of the gradient is required as the optimization progresses. We aim to provide methods that automatically terminate when a solution with guaranteed precision is reached. We take a somewhat different approach by directly modifying the optimization method.

Besides structural learning there have also been extensive work on speeding up kernel methods based on kernel matrix approximation. The Nyström method has been proposed in [18] to approximate the kernel matrix used for Gaussian Process classification. Low-rank approximation has been exploited to speed up the training of kernel SVMs[2]. A greedy basis-pursuit-style algorithm is also proposed in [7] to build sparse kernel SVMs to speed up both training and classification.

## 4. THE CUT SUBSAMPLING ALGORITHMS

Our main idea is to speed up the expensive double sum kernel computations in Equation 1 with approximate cuts that involve fewer basis functions. Such approximate cuts could be constructed by various methods such as greedy approaches, but we take the simpler approach of sampling since it allows us to prove performance guarantees later. In the following we will present two different sampling strategies and analyze their complexity.

## 4.1 A Constant Time Algorithm

Our first algorithm has constant time scaling with respect to the training set size. Let us look at the new cutting plane oracle in Algorithm 2, modified from Algorithm 1. There are no other changes apart from the function FIND_CUTTING_PLANE(). The vector $\vec{S}$ contains $r$ indices sampled uniformly from 1 to $N$. Both the offset $c^{(t)}$ and the subgradient $\vec{g}^{(t)}$ are constructed from these $r$ examples instead of the full training set. In general, the approximate subgradient points in a different direction than the exact subgradient. If we regard the exact constraint as a statement of how we want the classifier to behave on the whole training set, we can regard the sampled cut as a statement on a bootstrap sample. Notice that the exit condition of the while loop on Line 4 of Algorithm 2 is now based on an estimate of the loss from a small sample instead of the whole training set.

**Algorithm 2** Constant Time Cut Subsampling Algorithm for Structural SVM

```
 1: Input: $S = ((x_1, y_1), \dots, (x_N, y_N)), C, \epsilon$
 2: $J = \{\}, t = 0, \vec{w} = \vec{0}, \xi = 0$
 3: $(c^{(t)}, \vec{g}^{(t)})$ = FIND_CUTTING_PLANE($\vec{w}$)
 4: while $c^{(t)} + \vec{w} \cdot \vec{g}^{(t)} > \xi + \epsilon$ do
 5:     $J = J \cup \{(c^{(t)}, \vec{g}^{(t)})\}$
 6:     $t = t + 1$
 7:     $(\vec{w}, \xi)$ = SOLVE_QP($J$)
 8:     $(c^{(t)}, \vec{g}^{(t)})$ = FIND_CUTTING_PLANE($\vec{w}$)
 9: end while
10: return $(\vec{w}, \xi)$
11: procedure FIND_CUTTING_PLANE($\vec{w}$)
12:     Sample $r$ examples uniformly for $\vec{S}$
13:     for $j = 1$ to $r$ do
14:         $\hat{y}_{S_j} = \text{argmax}_{\hat{y} \in \mathcal{Y}}(\Delta(y_{S_j}, \hat{y}) + \vec{w} \cdot \Psi(x_{S_j}, \hat{y}))$
15:     end for
16:     $(c, \vec{g}) = (\frac{1}{r}\sum_{j=1}^{r} \Delta(y_{S_j}, \hat{y}_{S_j}), -\frac{1}{r}\sum_{j=1}^{r} \delta\Psi_{S_j}(\hat{y}_{S_j}))$
17:     return $(c, \vec{g})$
18: end procedure
```

### 4.1.1 Complexity per Iteration

Since the optimization problem is solved in the dual, we focus on complexity analysis of the dual of Algorithm 2. We defer the analysis on the number of cutting planes required before convergence to the next section, and analyze the time and especially the number of kernel computations required in each iteration. The dual form of the argmax operation of line 14 in Algorithm 2 is:

$$\hat{y}_j = \text{argmax}_{\hat{y} \in \mathcal{Y}} \left\{ \Delta(y_{S_j}, \hat{y}) + \sum_{k=1}^{t-1} \alpha_k \vec{g}^{(k)} \cdot \Psi(x_{S_j}, \hat{y}) \right\} \quad (2)$$

Expanding the inner product in Equation 2,

$$\vec{g}^{(k)} \cdot \Psi(x_{S_j}, \hat{y})) = \frac{1}{r} \sum_{l=1}^{r} [K(x_{S_l^{(k)}}, y_{S_l^{(k)}}, x_{S_j}, \hat{y}) - K(x_{S_l^{(k)}}, \hat{y}_l^{(k)}, x_{S_j}, \hat{y})] \quad (3)$$

where $S_l^{(k)}$ and $\hat{y}_l^{(k)}$ denotes $S_l$ and $\hat{y}_l$ at the $k$th iteration. This involves $O(tr)$ kernel computations at iteration $t$ when we sum up from $k = 1$ to $t - 1$, provided the argmax computation over $\hat{y}$ involves only a small constant number of kernel computation overall for different $\hat{y}$. This is true for binary or multi-class classification, and also true for the case when the kernel function factorizes into components (e.g. MRF cliques). Since we need to compute this inner product for all the sampled examples $\hat{y}_j$ for $j = 1$ to $r$, the overall complexity of sampling a cut involves $O(tr^2)$ kernel computations.

For computing the Gram matrix $G$, we can update it incrementally from one iteration to the next. At iteration $t$, it involves expanding $G$ by computing $G_{it}$ for $1 \leq i \leq t$. Following from Equation 1 in the case of the exact algorithm, we can infer that the inner product of two sampled cuts $\vec{g}^{(i)} \cdot \vec{g}^{(j)}$ involves $O(r^2)$ kernel computations. It takes $O(tr^2)$ kernel computations overall since we need to do this for $1 \leq i \leq t$. We can see that the subsequent iterations are more expensive since the cost scales linearly with $t$. If it takes $T$ iterations for the algorithm to terminate, then the overall complexity would be $O(T^2 r^2)$ kernel computations. We omit the time spent on the quadratic program in this analysis since in practice kernel computations account for over 95% of training time.

## 4.2 A Linear Time Algorithm

The previous sampling approach never looks at the whole training set, making the complexity independent of the training set size $N$ in each iteration. Our second sampling algorithm trades off additional work in each iteration for the ability to sample in a more targeted way. Let us consider Algorithm 3, especially the changes to the cutting plane oracle. Like the exact algorithm, it computes the argmax and the loss over all examples. However, it only samples $r$ of the examples with non-zero loss to construct the cutting plane. This has the effect of focusing on those examples that are more important to determining the decision surface. Two cutting planes $(c^{(t)}, \vec{g}^{(t)})$ and $(c', \vec{g}')$ are returned, one for inclusion in the optimization problem while the other is used for the stopping criterion.

In the case of a linear feature space this sampling is not needed because the cutting plane can be represented compactly by just adding up the $N$ feature vectors returned by the argmax computation. But in the nonlinear kernel case, sampling helps because it reduces the number of basis functions used in the kernel expansion from $O(N)$ to $O(r)$. Since the argmax computation is performed on all $N$ examples, the algorithm has more information on the whole training set compared to the constant time algorithm, such as the average loss and the primal objective value. In particular we can use the exact cutting plane $(c', \vec{g}')$ as the stopping criterion of the algorithm.

### 4.2.1 Complexity per Iteration

Since we are computing the argmax over all $N$ examples, it is possible to save computation in return for increased memory usage. Suppose we have a structure $A_{ki}$ to store all the information required to compute $\vec{g}^{(k)} \cdot \Psi(x_i, \hat{y})$ for $1 \leq k \leq t$, $1 \leq i \leq N$, and for all $\hat{y} \in \mathcal{Y}$. This is a single inner product $\vec{g}^{(k)} \cdot \phi(x_i)$ for binary classification, and $m$ numbers for multi-class classification if there are $m$ classes, one for each class. For HMM with kernelized emissions, this involves storing the kernelized emission score at each position for each possible hidden state. In all of these cases it amounts to $O(N)$ storage requirement for each cut.

**Algorithm 3** Linear Time Cut Subsampling Algorithm for Structural SVM

```
1: Input: $S = ((x_1, y_1), \dots, (x_N, y_N)), C, \epsilon$
2: $J = \{\}, t = 0, \vec{w} = \vec{0}, \xi = 0$
3: $((c^{(t)}, \vec{g}^{(t)}), (c', \vec{g}')) = $ FIND_CUTTING_PLANE($\vec{w}$)
4: while $c' + \vec{w} \cdot \vec{g}' > \xi + \epsilon$ do
5:     $J = J \cup \{(c^{(t)}, \vec{g}^{(t)})\}$
6:     $t = t + 1$
7:     $(\vec{w}, \xi) = $ SOLVE_QP($J$)
8:     $((c^{(t)}, \vec{g}^{(t)}), (c', \vec{g}')) = $ FIND_CUTTING_PLANE($\vec{w}$)
9: end while
10: return $(\vec{w}, \xi)$
11: procedure FIND_CUTTING_PLANE($\vec{w}$)
12:     for $i = 1$ to $N$ do
13:         $\hat{y}_i = \mathrm{argmax}_{\hat{y} \in \mathcal{Y}}(\Delta(y_i, \hat{y}) + \vec{w} \cdot \Psi(x_i, \hat{y}))$
14:     end for
15:     $I = \{1 \le i \le N \mid \Delta(y_i, \hat{y}_i) > 0\}$
16:     $(c', \vec{g}') = (\frac{1}{N}\sum_{i=1}^N \Delta(y_i, \hat{y}_i), -\frac{1}{N}\sum_{i=1}^N \delta\Psi_i(\hat{y}_i))$
17:     repeat
18:         Sample $r$ examples uniformly from $I$ for $\vec{S}$
19:         $(c, \vec{g}) = (\frac{1}{N}\sum_{i=1}^N \Delta(y_i, \hat{y}_i), -\frac{|I|}{Nr}\sum_{j=1}^r \delta\Psi_{S_j}(\hat{y}_{S_j}))$
20:     until $c' + \vec{w} \cdot \vec{g}' \le \xi + \epsilon$ or $c + \vec{w} \cdot \vec{g} > \xi + \epsilon$
21:     return $((c, \vec{g}), (c', \vec{g}'))$
22: end procedure
```

The dual form of the argmax operation in line 16 is:

$$\hat{y}_i = \mathrm{argmax}_{\hat{y} \in \mathcal{Y}} \left\{ \Delta(y_i, \hat{y}) + \sum_{k=1}^{t-1} \alpha_k \vec{g}^{(k)} \cdot \Psi(x_i, \hat{y}) \right\} \quad (4)$$

With the saved kernel computations in $A_{ki}$ for $1 \le k < t$, the argmax computation requires no extra kernel computations since the term $\vec{g}^{(k)} \cdot \Psi(x_i, \hat{y})$ can be retrieved from $A_{ki}$.

Updating $A_{ti}$ for a new iteration $t$ involves computing

$$\vec{g}^{(t)} \cdot \Psi(x_i, \hat{y}) = \frac{1}{r}\sum_{l=1}^r [K(x_{S_l^{(t)}}, y_{S_l^{(t)}}, x_i, \hat{y}) - K(x_{S_l^{(t)}}, \hat{y}^{(t)}_{S_l^{(t)}}, x_i, \hat{y})].$$

This requires $O(r)$ kernel computations, assuming that computing and storing the information required for recontructing the above inner product for each $\hat{y}$ takes a constant number of kernel computations and storage. As this has to be done for all $N$ examples, the overall complexity is $O(Nr)$ kernel compuations for each update in each iteration.

The Gram matrix $G$ can be updated conveniently with the information stored in $A_{ki}$, since

$$\vec{g}^{(i)} \cdot \vec{g}^{(j)} = \frac{1}{r}\sum_{l=1}^r \vec{g}^{(i)} \cdot \Psi(x_{S_l^{(j)}}, y_{S_l^{(j)}}) - \frac{1}{r}\sum_{l=1}^r \vec{g}^{(i)} \cdot \Psi(x_{S_l^{(j)}}, \hat{y}^{(j)}_{S_l^{(j)}}).$$

This involves no new kernel computations since both $\vec{g}^{(i)} \cdot \Psi(x_{S_l^{(j)}}, y_{S_l^{(j)}})$ and $\vec{g}^{(i)} \cdot \Psi(x_{S_l^{(j)}}, \hat{y}^{(j)}_{S_l^{(j)}})$ can be reconstructed from $A_{iS_l^{(j)}}$.

Therefore if the algorithm terminates in $T$ iterations, the overall complexity is $O(TNr)$ kernel compuations, with $O(TN)$ storage required. Although storing each cut requires $O(N)$ storage, it is still feasible even for large datasets if the number of active cuts is small(e.g., less than 100). This is the basic assumption in this space-time tradeoff and is confirmed by our experiments in section 6.

# 5. ANALYSIS OF THE ALGORITHMS

In this section we analyze theoretically the termination and solution accuracies of the two algorithms. We first prove bounds on the number of iterations for the algorithms to terminate, and then use the results to prove error bounds on the solutions. We prove the results for the two algorithms under a general framework to show that these results could also apply to the design of other sampling schemes.

## 5.1 Termination

To prove termination for the above algorithms, we consider the following template of the cutting plane algorithm:

**Algorithm 4** Generic Cutting Plane Algorithm

```
1: $J = \{\}, t = 0, \vec{w}^{(0)} = \vec{0}, \xi = 0$
2: $((c^{(t)}, \vec{g}^{(t)}), (c', \vec{g}')) = $ FIND_CUTTING_PLANE($\vec{w}^{(t)}$)
3: while $c' + \vec{w}^{(t)} \cdot \vec{g}' > \xi + \epsilon$ do
4:     $J = J \cup \{(c^{(t)}, \vec{g}^{(t)})\}$
5:     $t = t + 1$
6:     $(\vec{w}^{(t)}, \xi) = $ SOLVE_QP($J$)
7:     $((c^{(t)}, \vec{g}^{(t)}), (c', \vec{g}')) = $ FIND_CUTTING_PLANE($\vec{w}^{(t)}$)
8: end while
9: return $(\vec{w}^{(t)}, \xi)$
```

Notice that what the above algorithm returns as solution depends crucially on the implementation of FIND_CUTTING_PLANE(). However the specific detail of the implementation does not affect the termination property of the above cutting plane algorithm, and we have the following theorem:

THEOREM 1. *Assume the following holds for the cuts $(c^{(t)}, \vec{g}^{(t)})$, $(c', \vec{g}')$ returned by the cutting plane oracle* FIND_CUTTING_PLANE*():*

*(i)* $0 \le c', c^{(t)} \le \bar{\Delta}$

*(ii)* $\|\vec{g}'\|, \|\vec{g}^{(t)}\| \le R$

*(iii) if* $c' + \vec{w}^{(t)} \cdot \vec{g}' > \xi + \epsilon$*, then* $c^{(t)} + \vec{w}^{(t)} \cdot \vec{g}^{(t)} > \xi + \epsilon$

*Then Algorithm 4 terminates after at most $8C\bar{\Delta}R^2/\epsilon^2$ calls to the cutting plane oracle* FIND_CUTTING_PLANE*().*

PROOF. Consider the optimization problem solved by SOLVE_QP() on line 6 of the generic cutting plane algorithm:

OPTIMIZATION PROBLEM 3.

$$\min_{\vec{w}, \xi \ge 0} \frac{1}{2}\|\vec{w}\|^2 + C\xi \qquad s.t.\ \forall(c, \vec{g}) \in J, c + \vec{w} \cdot \vec{g} \le \xi \quad (5)$$

Consider also the following optimization problem:

OPTIMIZATION PROBLEM 4.

$$\min_{\vec{w}, \xi \ge 0} \frac{1}{2}\|\vec{w}\|^2 + C\xi \qquad s.t.\ \forall(c, \vec{g}) \in \mathcal{C}, c + \vec{w} \cdot \vec{g} \le \xi \qquad where\ \mathcal{C} = \{(c, \vec{g}) \mid c \in \mathbb{R}, 0 \le c \le \bar{\Delta}, \vec{g} \in \mathcal{H}, \|\vec{g}\| \le R\} \quad (6)$$

$\mathcal{C}$ contains all possible bounded cutting planes where $c$ is bounded above by $\bar{\Delta}$ and $\vec{g}$ is bounded above in norm by $R$.

Since conditions (i) and (ii) hold for the cutting plane oracle, OP3 is always a relaxation of OP4. Therefore the value of the primal solution of OP3 is always smaller than the value of the primal solution of OP4, and hence the value of any feasible solution of OP4 upper bounds the value of any dual solution of OP3. As $\vec{w} = \vec{0}$, $\xi = \bar{\Delta}$ is a feasible solution to OP4, the value of the dual solution of OP3 is upper bounded by $C\bar{\Delta}$. By Proposition 17 of [15], the inclusion of each $\epsilon$-violated constraint increases the dual objective of OP3 by at least $\epsilon^2/8R^2$, where $R$ is the upper bound on the norm of any $\vec{g}$. As the dual objective is bounded from above by $C\bar{\Delta}$, at most $8C\bar{\Delta}R^2/\epsilon^2$ constraints could be added before the cutting plane algorithm terminates.

Condition (iii) ensures that whenever we are not terminating the while loop, an $\epsilon$-violated constraint $(c^{(t)}, \vec{g}^{(t)})$ will always be added to the working set. □

Corollary 1. *Let* $\bar{\Delta} = \max_{i,y} \Delta(y_i, y)$ *and* $R = \max_{i,y} \|\delta\Psi_i(y)\|$. *Algorithm 2 terminates after at most* $8C\bar{\Delta}R^2/\epsilon^2$ *calls to* Find_Cutting_Plane*()*.

Proof. First of all notice that Algorithm 2 fits into the generic template of Algorithm 4. The cut $(c^{(t)}, \vec{g}^{(t)})$ returned by Find_Cutting_Plane() in Algorithm 2 serves both as the cut to be included into the working cut set $J$ and also as the cut for the termination criterion $(c', \vec{g}')$ as in line 3 of Algorithm 4 above. Therefore condition (iii) of Theorem 1 holds trivially. Since $0 \leq c^{(t)} = \frac{1}{r}\sum_{j=1}^{r} \Delta(y_{S_j}, \hat{y}_{S_j}) \leq \bar{\Delta}$ and $\|\vec{g}^{(t)}\| = \|-\frac{1}{r}\sum_{j=1}^{r} \delta\Psi_{S_j}(\hat{y}_{S_j})\| \leq R$, both conditions (i) and (ii) hold. Invoking Theorem 1, we can conclude that at most $8C\bar{\Delta}R^2/\epsilon^2$ calls are made to Find_Cutting_Plane() before Algorithm 2 terminates. □

Corollary 2. *Let* $\bar{\Delta} = \max_{i,y} \Delta(y_i, y)$ *and* $R = \max_{i,y} \|\delta\Psi_i(y)\|$. *Algorithm 3 terminates after at most* $8C\bar{\Delta}R^2/\epsilon^2$ *calls to* Find_Cutting_Plane*()*.

Proof. The proof is very similar to the previous corollary. Algorithm 3 fits the generic template of Algorithm 4. First of all $c^{(t)} = c' = \frac{1}{N}\sum_{i=1}^{N} \Delta(y_i, \hat{y}_i) \leq \bar{\Delta}$, so condition (i) of Theorem 1 is satisfied. It is also easy to see that $\vec{g}^{(t)} = \frac{|I|}{Nr}\sum_{j=1}^{r} \delta\Psi_{S_j}(\hat{y}_{S_j})$ and $\vec{g}' = \frac{1}{N}\sum_{i=1}^{N} \delta\Psi_i(\hat{y}_i)$ are bounded in norm by $R$, so condition (ii) holds as well. It is also easy to see that the exit condition of the repeat loop on line 20 of Algorithm 3 makes condition (iii) hold. Therefore we can invoke Theorem 1 and conclude that at most $8C\bar{\Delta}R^2/\epsilon^2$ calls are made to Find_Cutting_Plane() before termination. Moreover, the repeat loop in Find_Cutting_Plane() always terminate in finte expected time. When the exact cutting plane is $\epsilon$-violated, we can always sample an $\epsilon$-violated approximate cut with probability bounded away from 0 (for example, by sampling the worst violating example $r$ times). □

## 5.2 Accuracy of Solution

After proving termination and bounding the number of cutting planes required, we turn our attention to the accuracy of the solutions. Specifically we will characterize the difference between the regularized risk of the exact solution and our approximate solutions. The main idea used in the proof is: if the error introduced by each approximate cut is small with high probability, then the difference between the exact and approximate solutions will also be small with high probability. Bounding the difference between the exact cut and the sampled cut can be done with Hoeffding's inequality.

Let us start the proofs by defining some notation. Let $f(\vec{w}) = \max_{1 \leq t \leq T}\left(c^{(t)} + \vec{w}\cdot\vec{g}^{(t)}\right)$ be an exact cutting plane model of the empirical risk, and let $\tilde{f}(\vec{w}) = \max_{1 \leq t \leq T}\left(\tilde{c}^{(t)} + \vec{w}\cdot\tilde{\vec{g}}^{(t)}\right)$ be an approximate cutting plane model, with $(\tilde{c}^{(t)}, \tilde{\vec{g}}^{(t)})$ being the approximate cutting planes. We have the following lemma:

Lemma 1. *Let a fixed* $\vec{v}$ *in the RKHS* $\mathcal{H}$ *be given. Suppose for some* $\gamma > 0$ *each of the cutting plane and its approximate counterpart satisfy*

$$\Pr\left((\tilde{c}^{(t)} + \vec{v}\cdot\tilde{\vec{g}}^{(t)}) - (c^{(t)} + \vec{v}\cdot\vec{g}^{(t)}) \geq \gamma\right) < p_\gamma,$$

*for* $t = 1 \ldots T$. *Then* $\tilde{f}(\vec{v}) < f(\vec{v}) + \gamma$ *with probability at least* $1 - Tp_\gamma$.

Proof. By union bound we know that $(\tilde{c}^{(t)} + \vec{v}\cdot\tilde{\vec{g}}^{(t)}) - (c^{(t)} + \vec{v}\cdot\vec{g}^{(t)}) < \gamma$ for $1 \leq t \leq T$ occurs with probability at least $1 - Tp_\gamma$. The following chain of implications holds:

$$\bigwedge_{t=1}^{T}\left((\tilde{c}^{(t)} + \vec{v}\cdot\tilde{\vec{g}}^{(t)}) - (c^{(t)} + \vec{v}\cdot\vec{g}^{(t)}) < \gamma\right)$$

$$\Rightarrow \max_{1 \leq t \leq T}\left((\tilde{c}^{(t)} + \vec{v}\cdot\tilde{\vec{g}}^{(t)}) - (c^{(t)} + \vec{v}\cdot\vec{g}^{(t)})\right) < \gamma$$

$$\Rightarrow \max_{1 \leq t \leq T}(\tilde{c}^{(t)} + \vec{v}\cdot\tilde{\vec{g}}^{(t)}) - \max_{1 \leq t \leq T}(c^{(t)} + \vec{v}\cdot\vec{g}^{(t)}) < \gamma$$

Hence $\tilde{f}(\vec{v}) < f(\vec{v}) + \gamma$ with probability at least $1 - Tp_\gamma$. □

The lemma shows that the approximate cutting plane model does not overestimate the loss by more than a certain amount with high probability. Notice that $T$ is a fixed number above. If $T$ is a bounded random variable such as the termination iteration, then we can replace $T$ by its upper bound $\bar{T}$ and the lemma still holds. From the termination bound in section 5.1 we have $\bar{T} = 8C\bar{\Delta}R^2/\epsilon^2$.

Now we are going to use this lemma to analyze the linear time algorithm Algorithm 3. In the linear time algorithm we denote the exact cutting plane $(c^{(t)}, \vec{g}^{(t)})$ with $(\frac{1}{N}\sum_{i=1}^{N}\Delta(y_i, \hat{y}_i), -\frac{1}{N}\sum_{i=1}^{N}\delta\Psi_i(\hat{y}_i))$, and the approximate cut $(\tilde{c}^{(t)}, \tilde{\vec{g}}^{(t)})$ with $(\frac{1}{N}\sum_{i=1}^{N}\Delta(y_i, \hat{y}_i), -\frac{1}{r}\sum_{j=1}^{r}\delta\Psi_{S_j}(\hat{y}_{S_j}))$. We can bound the difference between the exact cutting planes and the approximate cutting planes using Hoeffding's inequality in the following lemma:

Lemma 2. *Let a fixed* $\vec{v} \in \mathcal{H}$, $\|\vec{v}\| \leq \sqrt{2C\bar{\Delta}}$ *be given, and let the exact cutting planes* $(c^{(t)}, \vec{g}^{(t)})$ *and approximate cutting planes* $(\tilde{c}^{(t)}, \tilde{\vec{g}}^{(t)})$ *be defined as above. We have for each* $t = 1 \ldots T$,

$$\Pr\left((\tilde{c}^{(t)} + \vec{v}\cdot\tilde{\vec{g}}^{(t)}) - (c^{(t)} + \vec{v}\cdot\vec{g}^{(t)}) \geq \gamma\right) < \exp\left(\frac{-r\gamma^2}{4C\bar{\Delta}R^2\eta^2}\right)$$

*where* $\eta = |I|/N$, $I$ *being the index set at the* $t$*-th iteration.*

Proof. Define $Z_j = -\vec{v}\cdot\delta\Psi_{S_j}(\hat{y}_{S_j})$. Since $S_j$ are sampled uniformly from the index set $I$, $Z_j$'s are independent with $E(Z_j) = -\frac{1}{|I|}\sum_{i\in I}\vec{v}\cdot\delta\Psi_i(\hat{y}_i)$. Each $Z_j$ is also bounded between $[-\sqrt{2C\bar{\Delta}}R, \sqrt{2C\bar{\Delta}}R]$. Apply Hoeffding's inequality and after some arithmetic we obtain the result. □

Now we are ready to prove our main theorem relating the regularized risk of the optimal solution to our approximate solution. Let $\vec{v^*}$ be the optimal solution to OP1. We have the following theorem:

THEOREM 2. *Suppose Algorithm 3 terminates in $T$ iterations and return $\vec{w}^*$ as solution. Then with probability at least $1-\delta$,*

$$\frac{1}{2}\|\vec{w}^*\|^2 + CL(\vec{w}^*) \leq \frac{1}{2}\|\vec{v}^*\|^2 + CL(\vec{v}^*) + C\left(\epsilon + \sqrt{\frac{4C\bar{\Delta}R^2}{r}\log\frac{\bar{T}}{\delta}}\right),$$

*where $\bar{T} = 8C\bar{\Delta}R^2/\epsilon^2$, and $L(\vec{w})$ is the margin loss $\frac{1}{N}\sum_{i=1}^N \xi_i$ as in OP1.*

PROOF. With the exact cutting planes $(c^{(t)}, \vec{g}^{(t)})$ and approximate cutting planes $(\tilde{c}^{(t)}, \tilde{\vec{g}}^{(t)})$ as defined in Lemma 2, we apply Lemma 1. Put $\vec{v} = \vec{v}^*$, and $p_\gamma = \exp(-r\gamma^2/4C\bar{\Delta}R^2)$ (we omit $\eta$ since it is bounded above by 1), we obtain $\tilde{f}(\vec{v}^*) < f(\vec{v}^*) + \gamma$ with probability at least $1 - \bar{T}\exp(-r\gamma^2/4C\bar{\Delta}R^2)$. Inverting the statement and we have with probability at least $1-\delta$:

$$\tilde{f}(\vec{v}^*) < f(\vec{v}^*) + \sqrt{\frac{4C\bar{\Delta}R^2}{r}\log\frac{\bar{T}}{\delta}}$$

Since $\vec{w}^*$ is the optimal solution of $\min_{\vec{w}} \frac{1}{2}\|\vec{w}\|^2 + C\tilde{f}(\vec{w})$ at the $T$th iteration, we have the following:

$$\begin{aligned} \frac{1}{2}\|\vec{w}^*\|^2 + C\tilde{f}(\vec{w}^*) \leq\ & \frac{1}{2}\|\vec{v}^*\|^2 + C\tilde{f}(\vec{v}^*) \\ < \frac{1}{2}\|\vec{v}^*\|^2 + Cf(\vec{v}^*) + C&\sqrt{\frac{4C\bar{\Delta}R^2}{r}\log\frac{\bar{T}}{\delta}} \quad \text{(with prob. } 1-\delta) \\ \leq \frac{1}{2}\|\vec{v}^*\|^2 + CL(\vec{v}^*) + C&\sqrt{\frac{4C\bar{\Delta}R^2}{r}\log\frac{\bar{T}}{\delta}} \quad \text{(subgrad. property)} \end{aligned} \tag{7}$$

The last line makes use of the subgradient property that $f(\vec{w}) \leq L(\vec{w})$ for any exact cutting plane model $f$ of a convex loss function $L$. Since we are using the exact cutting plane as the condition for exiting the while loop, so we must have at termination:

$$c' + \vec{w}^* \cdot \vec{g}' \leq \xi + \epsilon$$

$$\frac{1}{N}\sum_{i=1}^N \Delta(y_i, \hat{y}_i) - \frac{1}{N}\vec{w}^* \cdot \sum_{i=1}^N \delta\Psi_i(\hat{y}_i) \leq \tilde{f}(\vec{w}^*) + \epsilon$$

$$\max_{(\hat{y}_1,\ldots,\hat{y}_N)\in\mathcal{Y}^N} \frac{1}{N}\sum_{i=1}^N [\Delta(y_i, \hat{y}_i) - \vec{w}^* \cdot \delta\Psi_i(\hat{y}_i)] \leq \tilde{f}(\vec{w}^*) + \epsilon$$

$$L(\vec{w}^*) \leq \tilde{f}(\vec{w}^*) + \epsilon$$

Therefore we have:

$$\begin{aligned} \frac{1}{2}\|\vec{w}^*\|^2 + CL(\vec{w}^*) \leq\ & \frac{1}{2}\|\vec{w}^*\|^2 + C(\tilde{f}(\vec{w}^*) + \epsilon) \\ \leq \frac{1}{2}\|\vec{v}^*\|^2 + CL(\vec{v}^*) + C&\left(\epsilon + \sqrt{\frac{4C\bar{\Delta}R^2}{r}\log\frac{\bar{T}}{\delta}}\right) \end{aligned}$$

with probability at least $1-\delta$. □

The theorem shows that as far as obtaining a finite precision solution to the regularized risk minimization problem is concerned, it is sufficient to use sampled cuts with sufficiently large sample size $r$ to match the desired accuracy $\epsilon$ of the solution. We will see in the experiment section that fairly small values of $r$ work well in practice.

We state a similar result for Algorithm 2. The proof is fairly similar with a few technical differences. We assign its proof to the appendix.

THEOREM 3. *Suppose Algorithm 2 terminates in $T$ iterations with $\vec{w}^*$ returned as solution. Then with probability at least $1-2\delta$,*

$$\frac{1}{2}\|\vec{w}^*\|^2 + CL(\vec{w}^*) \leq \frac{1}{2}\|\vec{v}^*\|^2 + CL(\vec{v}^*) + C\left(\epsilon + 2\sqrt{\frac{(\bar{\Delta} + 2\sqrt{2C\bar{\Delta}}R)^2}{2r}\log\frac{\bar{T}}{\delta}}\right)$$

*where $\bar{T} = 8C\bar{\Delta}R^2/\epsilon^2$.*

# 6. EXPERIMENTS

While theory gives us the worst case bounds that are reassuring, we now study the empirical behaviour of the algorithms.

## 6.1 Experiment Setup

We implemented Algorithm 2 and Algorithm 3 and evaluated them on the task of binary classification with kernels. We choose this task for evaluation because binary classification with kernels is a well-studied problem, and there are stable SVM solvers that are suitable for comparisons. Moreover, scaling up SVM with kernels to large datasets is an interesting research problem on its own [1].

In binary classification the loss function $\Delta$ is just the zero-one loss. The feature map $\Psi$ is defined by $\Psi(x,y) = y\phi(x)$, where $y \in \{1, -1\}$ and $\phi$ is the nonlinear feature map induced from a Mercer kernel (such as the commonly used polynomial kernels and Gaussian kernels).

We implemented the algorithms in C, using Mosek as the quadratic program solver and the SFMT implementation [10] of Mersenne Twister as the random number generator. The experiments were run on machines with Opteron 2.0Ghz CPUs with 2Gb of memory (with the exception of the control experiments with incomplete Cholesky factorization, which we ran on machines with 4Gb of memory).

For all the experiments below we fix the precision parameter $\epsilon$ at 0.001. We remove cuts that are inactive for 20 iterations. We found that the constant time algorithm has better performance if we use a more stringent stopping criterion. We terminate the algorithm only when for $p$ consecutive iterations, the sampled cut is not violated by more than $\epsilon$. In the experiments below we use $p = 4$. For each combination of parameters we ran the experiment for 3 runs using different random seeds, and report the average result in the plots and tables below. In section 6.4 we also investigate the stability of the algorithms by reporting the standard deviation of the results.

In the experiments below we test our algorithms on three different datasets: Checkers, Adult, and Covertype. Checkers is a synthetic dataset with 1 million training points, with classes alternating on a 4x4 checkerboard. We generated the data using the SimpleSVM toolbox [17], with noise level parameter sigma set to 0.02. The kernel width for the Gaussian kernel used for the Checkers dataset was determined by cross validation on a small subsample of 10000 examples. Adult is a medium-sized dataset with 32562 examples, with a sample of 22697 examples taken as training set. The Gaussian kernel width is taken from [8]. Covertype is a dataset with 522910 training points, the kernel width of the Gaussian kernel we use below is obtained from the study [1].

## 6.2 Scaling with Training Set Size

Our first set of experiments is about how the two algorithms scale with training set size. We perform the experiments on the two large datasets Checkers and Covertype. We pick $C$ to be 1 multiplied by the training set size, since

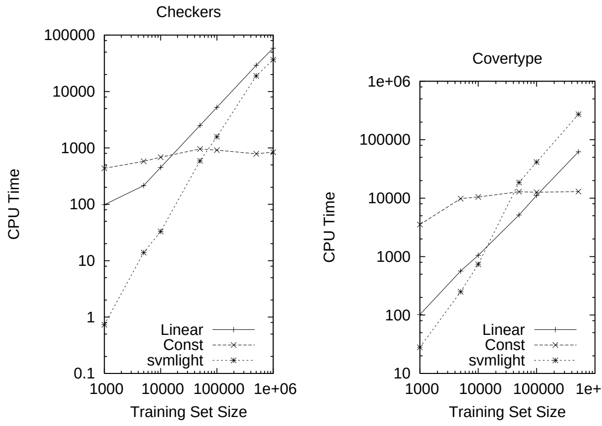

**Figure 1: CPU Time Against Training Set Size**

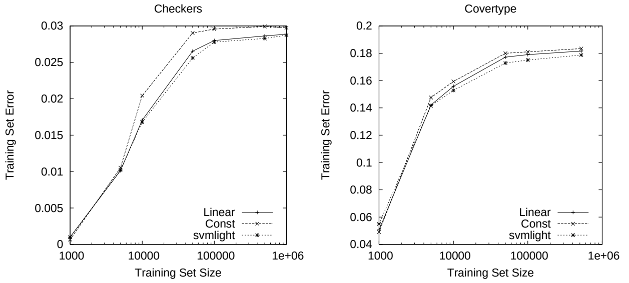

**Figure 2: Training Set Error Against Training Set Size**

that is the largest value of $C$ we could get SVM$^{light}$to train within 5 days. For the linear time algorithm we fix the sample size $r$ at 400, and for the constant time algorithm we use a sample size $r$ of 1000 to compensate for the less efficient sampling. We train SVM models on subsets of the full training sets of various sizes to evaluate scaling.

Figure 1 shows the CPU time required to train SVMs on training sets of different sizes on the Checkers and Covertype dataset. We can observe that the linear time algorithm scales roughly linearly in the log-log plot, while the constant time algorithm has a roughly flat curve in both plots. This confirms the scaling behaviour we expect from the complexity of each iteration. SVM$^{light}$ shows superlinear scaling on both of these datasets.

Figures 2 and 3 show the training and test set errors of the algorithms. In general SVM$^{light}$has the lowest training and test set errors, followed by the linear time algorithm and then the constant time algorithm. Both the training and test set errors lie within a very narrow band, and they are never more than 0.5 percentage point apart even in the worst case.

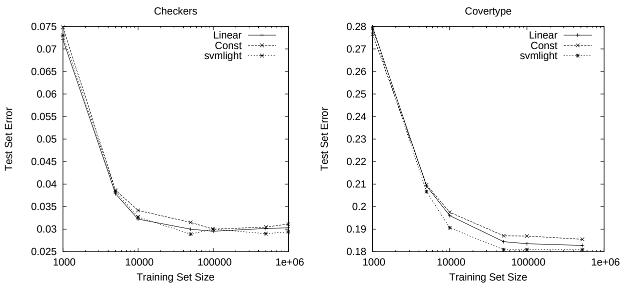

**Figure 3: Test Set Error Against Training Set Size**

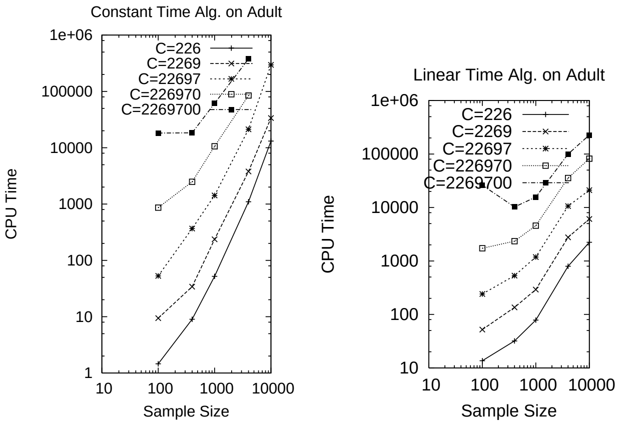

**Figure 4: CPU Time Against Sample Size**

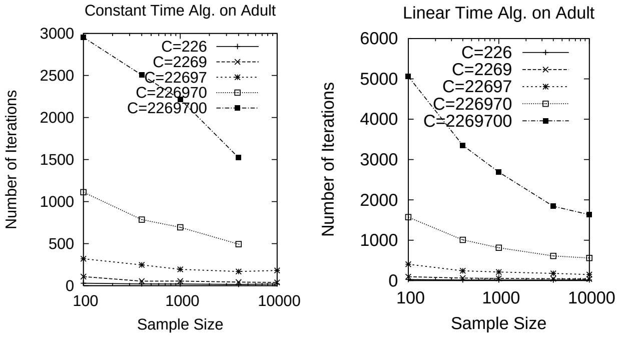

**Figure 5: Number of Iteration Against Sample Size**

## 6.3 Effect of Different Sample Sizes

The next set of experiments is about the effect of the sample size $r$ on training time and solution quality. We investigate the effect of sample size using the Adult dataset, since on this dataset it is easier to collect more data points for different sample sizes. We use sample sizes $r$ from {100, 400, 1000, 4000, 10000} and $C$ from {0.01, 0.1, 1, 10, 100} multiplied by the training set size 22697. The constant time algorithm did not finish the training within 5 days for the largest sample size 10000 and $C \in \{10, 100\}$, hence there are two missing data points in the figures.

In Figure 5 shows that the number of iterations required generally decreases with increasing sample size. However the decrease in the number of iterations to convergence does not result in overall savings in time due to the extra cost involved in each iteration with larger sample sizes. This can be observed from the CPU Time plots in Figure 4. In general, the linear time algorithm has better scaling behaviour with respect to sample size compared to the constant time algorithm. This is predicted by our complexity analysis. What is most interesting is the stability of training and test set errors with respect to changes to sample size, as shown in Figures 6 and 7. Except for very small sample sizes like 100 or small values of $C$ like 0.01 the sets of curves are essentially flat.

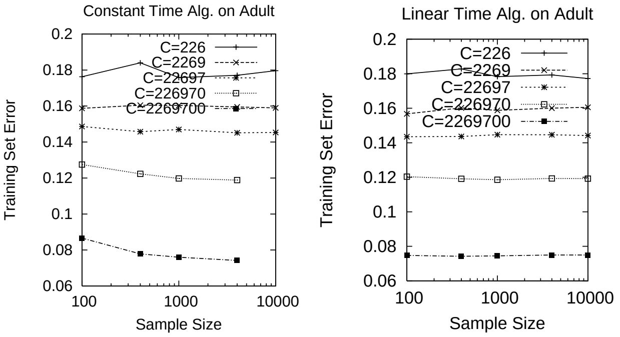

**Figure 6: Training Set Error Against Sample Size**

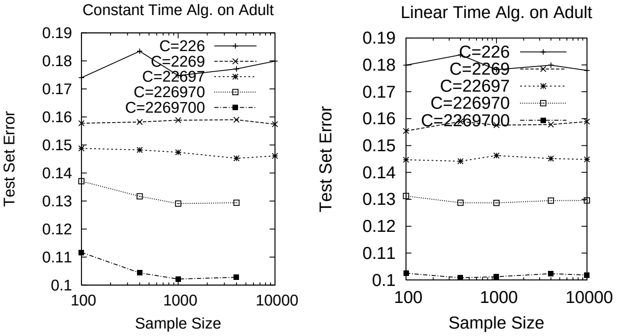

**Figure 7: Test Set Error Against Sample Size**

## 6.4 Quality of Solutions

Table 1 shows a comparison of the two algorithms against two conventional training methods, namely SVM$^{light}$ and a sampling-based method that uses Cholesky decomposition as described below. For each dataset we train different models using values of $C \in \{0.01, 0.1, 1, 10, 100\}$, multipled by the size of the training set. We used the results of SVM$^{light}$as a yardstick to compare against, and report the value of $C$ for which the test performance is optimal for SVM$^{light}$. For the larger datasets Checkers and Covertype, SVM$^{light}$terminated early due to slow progress for $C \geq 10$, so for those two datasets we use $C = 1$.

First of all, we notice from Table 1 that all the solutions have training and test set error rates very close to the solutions produced by SVM$^{light}$. For the constant time algorithm the error rates are usually within 0.3 to 0.5 above the SVM$^{light}$solutions, while the linear time algorithm has error rates usually within 0.2 above the SVM$^{light}$solutions. The error rates also have very small standard deviation, on the order of 0.1, which is the same as our tolerance parameter $\epsilon$. We also notice when using the same sample size $r$, the linear time algorithm provides more accurate solutions than the constant time algorithm due to its use of more focused sampling.

We also provide control experiments with Cholesky decomposition method, where we subsample a set of points from the training set, and then compute the projection of all the points in the training set onto the subspace spanned by these examples. Then we train a linear SVM using SVM$^{\mathrm{perf}}$[6] (with options `-t 2 -w 3 -b 0`) on the whole training set. Our implementation involves storing all the projected training vectors, and this consumes a lot of memory, especially for large datasets like Checkers and Covertype. We can only do 250 and 500 basis functions on those datasets respectively without running out of memory on a 4Gb machine, and on the Adult dataset we can only do up to 10000 basis functions. An alternative implementation with smaller storage requirement would involve recomputing the projected training vector when needed, but this would become prohibitively expensive.

We observe that the Cholesky decomposition is generally faster than all the other methods, but its accuracy is usually substantially below that of SVM$^{light}$ and our sampling algorithms. Moreover, unlike our algorithms, the accuracy of the Cholesky method depends crucially on the number of basis functions, which is difficult to pick in advance. The accuracies of our sampling algorithms are more stable with respect to the choice of sample size, where decreasing the sample size ususally results in more iterations to converge without much loss in accuracy of the solutions.

# 7. CONCLUSIONS

We presented two methods that make cutting-plane training of structural SVMs with kernels tractable through the use of random sampling in constructing a cut. The methods maintain the full generality of the cutting-plane approach, making it possible to kernelize any structural prediction problem where linear models are currently used. The theoretical analysis shows that these algorithms have linear or constant-time termination guarantees while providing bounds on the solution quality. Empirically, the algorithms can handle datasets with hundred-thousands of examples, and they are competitive or faster than conventional decomposition methods even on binary classification problems, where highly optimized special-purpose algorithms exist.

The current algorithms can be improved along several directions. The two sampling methods presented here are chosen for their simplicity and ease of analysis. Sampling efficiency can be improved by designing alternative sampling schemes, for example, by having different sampling rates for bound support vectors and non-bound support vectors following the popular shrinking heuristic used in training SVMs. On the other hand, one major bottleneck in the speed of the current algorithm is the large number of cuts required before convergence. Recently [3] proposes a stabilized cutting plane algorithm for linear SVMs with much improved convergence, and it will be interesting to extend their techniques to improve the speed of our sampled-cut algorithm for kernels.

# 8. ACKNOWLEDGMENTS

We would like to thank the reviewers for their careful reading and helpful comments for improving this paper. This work was supported in part by NSF Award IIS-0713483 and by a gift from Yahoo!.

# 9. REFERENCES

[1] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. In *NIPS*, 2002.

[2] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *JMLR*, 2:243–264, 2001.

[3] V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *ICML*, 2008.

[4] T. Joachims. A support vector method for multivariate performance measures. In *ICML*, 2005.

[5] T. Joachims. Training linear SVMs in linear time. In *KDD*, 2006.

| | Checkers | | | Adult | | | Covertype | | |
|---|---|---|---|---|---|---|---|---|---|
| | ($N$=1000000, $C$=1, $\sigma^2$=0.05) | | | ($N$=22697, $C$=100, $\sigma^2$=20) | | | ($N$=522910, $C$=1, $\sigma^2$=1.7) | | |
| Algorithm | Err(Train) | Err(Test) | CPU sec | Err(Train) | Err(Test) | CPU sec | Err(Train) | Err(Test) | CPU sec |
| Const ($r=400$) | 3.12(0.01) | 3.23(0.06) | 1411(40) | 7.90(0.05) | 10.45(0.08) | 17487(1175) | 18.49(0.11) | 18.61(0.13) | 3496(408) |
| Const ($r=1000$) | 2.96(0.01) | 3.14(0.13) | 1854(175) | 7.65(0.01) | 10.29(0.07) | 38617(604) | 18.34(0.09) | 18.55(0.02) | 12946(1400) |
| Linear ($r=400$) | 2.90(0.01) | 2.99(0.01) | 60101(1042) | 7.43(0.09) | 10.19(0.15) | 11136(640) | 18.16(0.03) | 18.28(0.02) | 62184(485) |
| Linear ($r=1000$) | 2.89(0.00) | 2.95(0.05) | 151759(3334) | 7.44(0.01) | 10.19(0.15) | 15191(612) | 17.96(0.03) | 18.22(0.02) | 155196(6649) |
| Cholesky(250) | 3.11 | 3.32 | 2447 | 15.10 | 15.09 | 604 | 21.73 | 21.85 | 1937 |
| Cholesky(500) | N/A | N/A | N/A | 14.60 | 14.50 | 537 | 20.29 | 20.35 | 3093 |
| Cholesky(5000) | N/A | N/A | N/A | 10.89 | 12.43 | 3386 | N/A | N/A | N/A |
| Cholesky(10000) | N/A | N/A | N/A | 9.12 | 11.41 | 8836 | N/A | N/A | N/A |
| SVM$^{light}$ | 2.87 | 2.94 | 36533 | 7.52 | 10.37 | 11630 | 17.87 | 18.09 | 273021 |

**Table 1: Runtime and training/test error of sampling algorithms compared to SVM$^{light}$ and Cholesky.**

[6] T. Joachims, T. Finley, and C.-N. Yu. Cutting plane training of structural SVMs. *MLJ*, To Appear.

[7] S. S. Keerthi, O. Chapelle, and D. DeCoste. Building support vector machines with reduced classifier complexity. *JMLR*, 7:1493–1515, 2006.

[8] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 12. MIT-Press, 1999.

[9] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich. (Online) subgradient methods for structured prediction. In *AISTATS*, 2007.

[10] M. Saito and M. Matsumoto. SIMD-oriented fast mersenne twister: a 128-bit pseudorandom number generator. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*, 2006.

[11] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *ICML*, 2007.

[12] A. J. Smola, S. V. N. Vishwanathan, and Q. Le. Bundle methods for machine learning. In *NIPS*, 2007.

[13] B. Taskar, C. Guestrin, and D. Koller. Maximum-Margin Markov networks. In *NIPS*, 2003.

[14] B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin Parsing. In *EMNLP*, 2004.

[15] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6:1453 – 1484, September 2005.

[16] S. V. N. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *ICML*, 2006.

[17] S. V. N. Vishwanathan, A. J. Smola, and M. N. Murty. SimpleSVM. In *ICML*, 2003.

[18] C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *NIPS*, 2001.

[19] C.-N. Yu, T. Joachims, R. Elber, and J. Pillardy. Support vector training of protein alignment models. In *RECOMB*, 2007.

[20] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR*, 2007.

# APPENDIX

Proof of Theorem 3. Define the exact cutting planes $(c^{(t)}, \vec{g}^{(t)})$ to be $(\frac{1}{N}\sum_{i=1}^{N}\Delta(y_i,\hat{y}_i), -\frac{1}{N}\sum_{i=1}^{N}\delta\Psi_i(\hat{y}_i))$, and the approximate cutting planes $(\tilde{c}^{(t)}, \tilde{\vec{g}}^{(t)})$ to be $\frac{1}{r}\sum_{j=1}^{r}\Delta(y_{S_j},\hat{y}_{S_j})$ and $-\frac{1}{r}\sum_{j=1}^{r}\delta\Psi_{S_j}(\hat{y}_{S_j})$. We can bound the difference between the approximate and exact cutting planes with Hoeffding's inequality:

Lemma 3. *Let a fixed $\vec{v} \in \mathcal{H}, \|\vec{v}\| \leq \sqrt{2C\bar{\Delta}}$ be given, and let the exact cutting planes $(c^{(t)}, \vec{g}^{(t)})$ and approximate cutting planes $(\tilde{c}^{(t)}, \tilde{\vec{g}}^{(t)})$ be defined as above. We have for $1 \leq t \leq T$,*

$$\Pr\left((\tilde{c}^{(t)}+\vec{v}\cdot\tilde{\vec{g}}^{(t)})-(c^{(t)}+\vec{v}\cdot\vec{g}^{(t)})\geq\gamma\right)<\exp\left(\frac{-2r\gamma^2}{(\bar{\Delta}+2\sqrt{2C\bar{\Delta}}R)^2}\right)$$

Proof. Define $Z_j = \Delta(y_{S_j},\hat{y}_{S_j}) - \vec{v}\cdot\delta\Psi_{S_j}(\hat{y}_{S_j})$. Since $S_j$ is sampled uniformly from 1 to $N$, $Z_j$'s are independent with $E(Z_j) = \frac{1}{N}(\Delta(y_i,\hat{y}_i) - \vec{v}\cdot\delta\Psi_i(\hat{y}_i))$. Each $Z_j$ is bounded in the interval $[-\sqrt{2C\bar{\Delta}}R, \bar{\Delta}+\sqrt{2C\bar{\Delta}}R]$. Apply Hoeffding's inequality and we obtain the result. □

Using Lemma 1 and Lemma 3, we can adopt a similar proof as in Theorem 2 up to Equation 7 and obtain

$$\frac{1}{2}\|\vec{w}^*\|^2+C\tilde{f}(\vec{w}^*)\leq\frac{1}{2}\|\vec{v}^*\|^2+CL(\vec{v}^*)+C\left(\epsilon+\sqrt{\frac{(\bar{\Delta}+2\sqrt{2C\bar{\Delta}}R)^2}{2r}\log\frac{\bar{T}}{\delta}}\right) \quad (8)$$

Unlike Algorithm 3, we terminate using an approximated objective instead of the true objective, so we do not have $L(\vec{w}^*) \leq \tilde{f}(\vec{w}^*) + \epsilon$. However we can show that with high probability $L(\vec{w}^*) \leq \tilde{f}(\vec{w}^*) + \epsilon + \gamma'$ for some $\gamma' > 0$. Analogous to Lemma 3, we can prove using Hoeffding's inequality the following bound:

$$\Pr\left((c^{(t)}+\vec{w}^{(t)}\cdot\vec{g}^{(t)})-(\tilde{c}^{(t)}+\vec{w}^{(t)}\cdot\tilde{\vec{g}}^{(t)})\geq\gamma\right)<\exp\left(\frac{-2r\gamma^2}{(\bar{\Delta}+2\sqrt{2C\bar{\Delta}}R)^2}\right)$$

where $\vec{w}^{(t)}$ is the weight vector at the beginning of iteration $t$. Notice $\vec{w}^{(t)}$ is fixed in the sense that it is chosen before sampling for $(\tilde{c}^{(t)}, \tilde{\vec{g}}^{(t)})$ occurs. Moreover, the value $c^{(t)} + \vec{w}^{(t)}\cdot\vec{g}^{(t)}$ equals $L(\vec{w}^{(t)})$ at successive iterates $\vec{w}^{(t)}$. Therefore by union bound we have $L(\vec{w}^{(t)}) - (\tilde{c}^{(t)} + \vec{w}^{(t)}\cdot\tilde{\vec{g}}^{(t)}) < \gamma'$ for $t = 1\ldots T$ with probability at least $1 - \bar{T}\exp(-2r\gamma'^2/(\bar{\Delta}+2\sqrt{2C\bar{\Delta}}R)^2)$. In particular at termination we have:

$$\xi + \epsilon \geq \tilde{c}^{(T)} + \vec{w}^*\cdot\tilde{\vec{g}}^{(T)}$$
$$\tilde{f}(\vec{w}^*) + \epsilon \geq L(\vec{w}^*) - \gamma'$$

with probability at least $1 - \bar{T}\exp(-2r\gamma'^2/(\bar{\Delta}+2\sqrt{2C\bar{\Delta}}R)^2)$. Inverting the statement we have with probability at least $1-\delta'$,

$$\tilde{f}(\vec{w}^*) + \epsilon \leq L(\vec{w}^*) + \sqrt{\frac{(\bar{\Delta}+2\sqrt{2C\bar{\Delta}}R)^2}{2r}\log\frac{\bar{T}}{\delta'}}$$

By putting $\delta' = \delta$, and combining with Equation 8, we obtain the theorem. □