Cutting-Plane Training of Structural SVMs

Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu

Abstract Discriminative training approaches like structural SVMs have shown much promise for building highly complex and accurate models in areas like natural language processing, protein structure prediction, and information retrieval. However, current training algorithms are computationally expensive or intractable on large datasets. To overcome this bottleneck, this paper explores how cutting-plane methods can provide fast training not only for classification SVMs, but also for structural SVMs. In particular, we show that in an equivalent "1-slack" reformulation of the linear SVM training problem, our cutting-plane method has time complexity linear in the number of training examples, linear in the desired precision, and linear also in all other parameters. Furthermore, we present an extensive empirical evaluation of the method applied to binary classification, multi-class classification, HMM sequence tagging, and CFG parsing. The experiments show that the cutting-plane algorithm is broadly applicable and fast in practice. On large datasets, it is typically several orders of magnitude faster than conventional training methods derived from decomposition methods like SVM^{light}, or conventional cutting-plane methods. Implementations of our methods are available online.

Key words: Structural SVMs, Support Vector Machines, Structured Output Prediction, Training Algorithms

Thorsten Joachims
Dept. of Computer Science, Cornell University, Ithaca, NY, USA, e-mail: tj@cs.cornell.edu

Thomas Finley
Dept. of Computer Science, Cornell University, Ithaca, NY, USA, e-mail: tomf@cs.cornell.edu

Chun-Nam John Yu
Dept. of Computer Science, Cornell University, Ithaca, NY, USA, e-mail: cnyu@cs.cornell.edu

1 Introduction

Consider the problem of learning a function with complex outputs, where the prediction is not a single univariate response (e.g., 0/1 for classification or a real number for regression), but a structured multivariate object. Examples of such structured prediction problems are the prediction of parse trees in natural language processing, the prediction of total orderings in web search, or the prediction of sequence alignments in protein threading.

Recent years have provided intriguing advances in extending methods like Logistic Regression, Perceptrons, and Support Vector Machines (SVMs) to global training of such structured prediction models (e.g., Lafferty et al, 2001; Collins, 2004; Collins and Duffy, 2002; Taskar et al, 2003; Tsochantaridis et al, 2004). In contrast to conventional generative training, these methods are discriminative (e.g., conditional likelihood, empirical risk minimization). Akin to moving from Naive Bayes to an SVM for classification, this provides greater modeling flexibility through avoidance of independence assumptions, and it was shown to provide substantially improved prediction accuracy in many domains (e.g., Lafferty et al, 2001; Taskar et al, 2003; Tsochantaridis et al, 2004; Taskar et al, 2004; Yu et al, 2007). By eliminating the need for modeling statistical dependencies between features, discriminative training enables us to freely use more complex and possibly interdependent features, which provides the potential to learn models with improved fidelity. However, training these rich models with a sufficiently large training set is often beyond the reach of current discriminative training algorithms.

We focus on the problem of training structural SVMs in this paper. Formally, this can be thought of as solving a convex quadratic program (QP) with a large (typically exponential or infinite) number of constraints. Existing algorithm fall into two groups. The first group of algorithms relies on an elegant polynomial-size reformulation of the training problem (Taskar et al, 2003; Anguelov et al, 2005), which is possible for the special case of margin-rescaling (Tsochantaridis et al, 2005) with linearly decomposable loss. These smaller QPs can then be solved, for example, with general-purpose optimization methods (Anguelov et al, 2005) or decomposition methods similar to SMO (Taskar et al, 2003; Platt, 1999). Unfortunately, decomposition methods are known to scale super-linearly with the number of examples (Platt, 1999; Joachims, 1999), and so do general-purpose optimizers, since they do not exploit the special structure of this optimization problem. But most significantly, the algorithms in the first group are limited to applications where the polynomial-size reformulation exists. Similar restrictions also apply to the extragradient method (Taskar et al, 2005), which applies only to problems where subgradients of the QP can be computed via a convex real relaxation, as well as exponentiated gradient methods (Bartlett et al, 2004; Globerson et al, 2007), which require the ability to compute “marginals” (e.g. via the sum-product algorithm). The second group of algorithms works directly with the original, exponentially-sized QP. This is feasible, since a polynomially-sized subset of the constraints from the original QP is already sufficient for a solution of arbitrary accuracy (Joachims, 2003; Tsochantaridis et al, 2005). Such algorithms either take stochastic subgradient steps (Collins, 2002; Ratliff et al, 2007; Shalev-Shwartz et al, 2007), or build a cutting-plane model which is easy to solve directly (Tsochantaridis et al, 2004). The algorithm in (Tsochantaridis et al, 2005) shows how such a cutting-plane can be constructed efficiently. Compared to the subgradient methods, the cutting-plane approach does not take a single gradient step, but always takes an optimal step in the current cutting-plane model. It requires only the existence of an efficient separation oracle, which makes it applicable to many problems for which no polynomially-sized reformulation is known. In practice, however, the cutting-plane method of Tsochantaridis et al (2005) is known to scale super-linearly with the number of training examples. In particular, since the size of the cutting-plane model typically grows linearly with the dataset size (see Tsochantaridis et al, 2005, and Section 5.5), QPs of increasing size need to be solved to compute the optimal steps, which leads to the super-linear runtime.

In this paper, we explore an extension of the cutting-plane method presented in (Joachims, 2006) for training linear structural SVMs, both in the margin-rescaling and in the slack-rescaling formulation (Tsochantaridis et al, 2005). In contrast to the cutting-plane method presented in (Tsochantaridis et al, 2005), we show that the size of the cutting-plane models and the number of iterations are independent of the number of training examples n. Instead, their size and the number of iterations can be upper bounded by $O(\frac{C}{\varepsilon})$, where C is the regularization constant and ε is the desired precision of the solution (see Optimization Problems OP2 and OP3 in Section 2). Since each iteration of the new algorithm takes $O(n)$ time and memory, it also scales $O(n)$ overall with the number of training examples both in terms of computation time and memory. Empirically, the size of the cutting-plane models and the QPs that need to be solved in each iteration is typically very small (less than a few hundred variables) even for problems with millions of features and hundreds of thousands of examples.

A key conceptual difference of the new algorithm compared to the algorithm of Tsochantaridis et al (2005) and most other SVM training methods is that not only individual data points are considered as potential Support Vectors (SVs), but also linear combinations of those. This increased flexibility allows for solutions with far fewer non-zero dual variables, and it leads to the small cutting-plane models discussed above.

The new algorithm is applicable to all structural SVM problems where the separation oracle can be computed efficiently, which makes it just as widely applicable as the most general training algorithms known to date. Even further, following the original publication in (Joachims, 2006), Teo et al (2007) have already shown that the algorithm can also be extended to Conditional Random Field training. We provide a theoretical analysis of the algorithm's correctness, convergence rate, and scaling behavior for structured prediction. Furthermore, we present empirical results for several structured prediction problems (i.e., multi-class classification, part-of-speech tagging, and natural language parsing), and compare against conventional algorithms also for the special case of binary classification. On all problems, the new algorithm is substantially faster than conventional decomposition methods and cutting-plane methods, often by several orders of magnitude for large datasets.

2 Structural Support Vector Machines

Structured output prediction describes the problem of learning a function

$$h : \mathscr{X} \longrightarrow \mathscr{Y}$$

where $\mathscr{X}$ is the space of inputs, and $\mathscr{Y}$ is the space of (multivariate and structured) outputs. In the case of natural language parsing, for example, $\mathscr{X}$ is the space of sentences, and $\mathscr{Y}$ is the space of trees over a given set of non-terminal grammar symbols. To learn h, we assume that a training sample of input-output pairs

$$S = ((x_1,y_1),\ldots,(x_n,y_n)) \in (\mathscr{X} \times \mathscr{Y})^n$$

is available and drawn i.i.d.[1] from a distribution $P(X,Y)$. For a given hypothesis space $\mathscr{H}$, the goal is to find a function $h \in \mathscr{H}$ that has low prediction error, or, more generally, low risk

$$R_P^{\Delta}(h) = \int_{\mathscr{X}\times\mathscr{Y}} \Delta(y,h(x))\, dP(x,y)\,.$$

$\Delta(y,\bar{y})$ is a loss function that quantifies the loss associated with predicting $\bar{y}$ when y is the correct output value. We assume that $\Delta(y,y) = 0$ and $\Delta(y,\bar{y}) \geq 0$ for $y \neq \bar{y}$. We follow the Empirical Risk Minimization Principle (Vapnik, 1998) to infer a function h from the training sample S. The learner evaluates the quality of a function $h \in \mathscr{H}$ using its empirical risk $R_S^{\Delta}(h)$ on the training sample S.

$$R_S^{\Delta}(h) = \frac{1}{n}\sum_{i=1}^{n} \Delta(y_i,h(x_i))$$

Support Vector Machines select an $h \in \mathscr{H}$ that minimizes a regularized empirical risk on S. For conventional binary classification where $\mathscr{Y} = \{-1,+1\}$, SVM training is typically formulated as the following convex quadratic optimization problem [2] (Cortes and Vapnik, 1995; Vapnik, 1998).

Optimization Problem 1 (Classification SVM (primal))

$$\min_{\mathbf{w},\xi_i\geq 0} \ \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$
$$s.t.\quad \forall i \in \{1,\ldots,n\} : y_i(\mathbf{w}^T x_i) \geq 1 - \xi_i$$

It was shown that SVM training can be generalized to structured outputs (Altun et al, 2003; Taskar et al, 2003; Tsochantaridis et al, 2004), leading to an optimiza-

[1] Note, however, that all formal results in this paper also hold for non i.i.d. data, since our algorithms do not rely on the order or distribution of the examples.

[2] For simplicity, we consider the case of hyperplanes passing through the origin. By adding a constant feature, an offset can easily be simulated.

tion problem that is similar to multi-class SVMs (Crammer and Singer, 2001) and extending the Perceptron approach described in (Collins, 2002). The idea is to learn a discriminant function $f : \mathscr{X} \times \mathscr{Y} \rightarrow \Re$ over input/output pairs from which one derives a prediction by maximizing f over all $y \in \mathscr{Y}$ for a given input x.

$$h_{\mathbf{w}}(x) = \operatorname*{argmax}_{y \in \mathscr{Y}} f_{\mathbf{w}}(x,y)$$

We assume that $f_{\mathbf{w}}(x,y)$ takes the form of a linear function

$$f_{\mathbf{w}}(x,y) = \mathbf{w}^T \Psi(x,y)$$

where $\mathbf{w} \in \Re^N$ is a parameter vector and $\Psi(x,y)$ is a feature vector relating input x and output y. Intuitively, one can think of $f_{\mathbf{w}}(x,y)$ as a compatibility function that measures how well the output y matches the given input x. The flexibility in designing Ψ allows employing structural SVMs for problems as diverse as natural language parsing (Taskar et al, 2004), protein sequence alignment (Yu et al, 2007), supervised clustering (Finley and Joachims, 2005), learning ranking functions that optimize IR performance measures (Yue et al, 2007), and segmenting images (Anguelov et al, 2005).

For training the weights $\mathbf{w}$ of the linear discriminant function, the standard SVM optimization problem can be generalized in several ways (Altun et al, 2003; Joachims, 2003; Taskar et al, 2003; Tsochantaridis et al, 2004, 2005). This paper uses the formulations given in (Tsochantaridis et al, 2005), which subsume all other approaches. We refer to these as the "*n*-slack" formulations, since they assign a different slack variable to each of the *n* training examples. Tsochantaridis et al (2005) identify two different ways of using a hinge loss to convex upper bound the loss, namely "margin-rescaling" and "slack-rescaling". In margin-rescaling, the position of the hinge is adapted while the slope is fixed,

$$\Delta_{MR}(y, h_{\mathbf{w}}(x)) = \max_{\bar{y} \in \mathscr{Y}} \{\Delta(y,\bar{y}) - \mathbf{w}^T \Psi(x,y) + \mathbf{w}^T \Psi(x,\bar{y})\} \geq \Delta(y, h_{\mathbf{w}}(x)) \qquad (1)$$

while in slack-rescaling, the slope is adjusted while the position of the hinge is fixed.

$$\Delta_{SR}(y, h_{\mathbf{w}}(x)) = \max_{\bar{y} \in \mathscr{Y}} \{\Delta(y,\bar{y})(1 - \mathbf{w}^T \Psi(x,y) + \mathbf{w}^T \Psi(x,\bar{y}))\} \geq \Delta(y, h_{\mathbf{w}}(x)) \qquad (2)$$

This leads to the following two training problems, where each slack variable ξ_i is equal to the respective $\Delta_{MR}(y_i, h_{\mathbf{w}}(x_i))$ or $\Delta_{SR}(y_i, h_{\mathbf{w}}(x_i))$ for training example (x_i, y_i).

Optimization Problem 2 (*n*-SLACK STRUCTURAL SVM WITH MARGIN-RESCALING (PRIMAL))

$$\min_{\mathbf{w},\boldsymbol{\xi}\geq\mathbf{0}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}+\frac{C}{n}\sum_{i=1}^{n}\xi_i$$
$$s.t. \quad \forall\bar{y}_1\in\mathscr{Y}:\mathbf{w}^T[\Psi(x_1,y_1)-\Psi(x_1,\bar{y}_1)]\geq\Delta(y_1,\bar{y}_1)-\xi_1$$
$$\vdots$$
$$s.t. \quad \forall\bar{y}_n\in\mathscr{Y}:\mathbf{w}^T[\Psi(x_n,y_n)-\Psi(x_n,\bar{y}_n)]\geq\Delta(y_n,\bar{y}_n)-\xi_n$$

Optimization Problem 3 (*n*-SLACK STRUCTURAL SVM WITH SLACK-RESCALING (PRIMAL))

$$\min_{\mathbf{w},\boldsymbol{\xi}\geq\mathbf{0}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}+\frac{C}{n}\sum_{i=1}^{n}\xi_i$$
$$s.t. \quad \forall\bar{y}_1\in\mathscr{Y}:\mathbf{w}^T[\Psi(x_1,y_1)-\Psi(x_1,\bar{y}_1)]\geq 1-\frac{\xi_1}{\Delta(y_1,\bar{y}_1)}$$
$$\vdots$$
$$s.t. \quad \forall\bar{y}_n\in\mathscr{Y}:\mathbf{w}^T[\Psi(x_n,y_n)-\Psi(x_n,\bar{y}_n)]\geq 1-\frac{\xi_n}{\Delta(y_n,\bar{y}_n)}$$

The objective is the conventional regularized risk used in SVMs. The constraints state that for each training example (x_i,y_i) the score $\mathbf{w}^T\Psi(x_i,y_i)$ of the correct structure y_i must be greater than the score $\mathbf{w}^T\Psi(x_i,\bar{y})$ of all incorrect structures $\bar{y}$ by a required margin. This margin is 1 in slack-rescaling, and equal to the loss $\Delta(y_i,\bar{y})$ in margin rescaling. If the margin is violated, the slack variable ξ_i of the example becomes non-zero. Note that ξ_i is shared among constraints from the same example. It is easy to verify that for both margin-rescaling and for slack-rescaling, $\sum\xi_i$ is an upper bound on the empirical risk $R_S^\Delta(h)$ on the training sample S (see Tsochantaridis et al, 2005).

It is not immediately obvious that Optimization Problems OP2 and OP3 can be solved efficiently, since they have $O(n|\mathscr{Y}|)$ constraints. $|\mathscr{Y}|$ is typically extremely large (e.g., all possible alignments of two amino-acid sequence) or even infinite (e.g., real-valued outputs). For the special case of margin-rescaling with linearly decomposable loss functions Δ, Taskar et al. (Taskar et al, 2003) have shown that the problem can be reformulated as a quadratic program with only a polynomial number of constraints and variables.

A more general algorithm that applies to both margin-rescaling and slack-rescaling under a large variety of loss functions was given in (Tsochantaridis et al, 2004, 2005). The algorithm relies on the theoretical result that for any desired precision ε, a greedily constructed cutting-plane model of OP2 and OP3 requires only $O(\frac{n}{\varepsilon^2})$ many constraints (Joachims, 2003; Tsochantaridis et al, 2005). This greedy algorithm for the case of margin-rescaling is Algorithm 1, for slack-rescaling it leads to Algorithm 2. The algorithms iteratively construct a working set $\mathscr{W}=\mathscr{W}_1\cup\ldots\cup\mathscr{W}_n$

Algorithm 1 for training Structural SVMs (with margin-rescaling) via the *n*-Slack Formulation (OP2).

```
1:  Input: $S = ((x_1,y_1),\dots,(x_n,y_n))$, $C$, $\varepsilon$
2:  $\mathscr{W}_i \leftarrow \emptyset$, $\xi_i \leftarrow 0$ for all $i = 1,\dots,n$
3:  repeat
4:      for i=1,...,n do
5:          $\hat{y} \leftarrow \mathrm{argmax}_{\hat{y}\in\mathscr{Y}}\{\Delta(y_i,\hat{y}) - \mathbf{w}^T[\Psi(x_i,y_i) - \Psi(x_i,\hat{y})]\}$
6:          if $\Delta(y_i,\hat{y}) - \mathbf{w}^T[\Psi(x_i,y_i) - \Psi(x_i,\hat{y})] > \xi_i + \varepsilon$ then
7:              $\mathscr{W}_i \leftarrow \mathscr{W}_i \cup \{\hat{y}\}$
8:              $(\mathbf{w},\boldsymbol{\xi}) \leftarrow \mathrm{argmin}_{\mathbf{w},\boldsymbol{\xi}\geq 0}\ \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{n}\sum_{i=1}^{n}\xi_i$
                    s.t. $\forall\bar{y}_1 \in \mathscr{W}_1 : \mathbf{w}^T[\Psi(x_1,y_1) - \Psi(x_1,\bar{y}_1)] \geq \Delta(y_1,\bar{y}_1) - \xi_1$
                         $\vdots$
                         $\forall\bar{y}_n \in \mathscr{W}_n : \mathbf{w}^T[\Psi(x_n,y_n) - \Psi(x_n,\bar{y}_n)] \geq \Delta(y_n,\bar{y}_n) - \xi_n$
9:          end if
10:     end for
11: until no $\mathscr{W}_i$ has changed during iteration
12: return($\mathbf{w}$,$\xi$)
```

Algorithm 2 for training Structural SVMs (with slack-rescaling) via the *n*-Slack Formulation (OP3).

```
1:  Input: $S = ((x_1,y_1),\dots,(x_n,y_n))$, $C$, $\varepsilon$
2:  $\mathscr{W}_i \leftarrow \emptyset$, $\xi_i \leftarrow 0$ for all $i = 1,\dots,n$
3:  repeat
4:      for i=1,...,n do
5:          $\hat{y} \leftarrow \mathrm{argmax}_{\hat{y}\in\mathscr{Y}}\{\Delta(y_i,\hat{y})(1 - \mathbf{w}^T[\Psi(x_i,y_i) - \Psi(x_i,\hat{y})])\}$
6:          if $\Delta(y_i,\hat{y})(1 - \mathbf{w}^T[\Psi(x_i,y_i) - \Psi(x_i,\hat{y})]) > \xi_i + \varepsilon$ then
7:              $\mathscr{W}_i \leftarrow \mathscr{W}_i \cup \{\hat{y}\}$
8:              $(\mathbf{w},\boldsymbol{\xi}) \leftarrow \mathrm{argmin}_{\mathbf{w},\boldsymbol{\xi}\geq 0}\ \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{n}\sum_{i=1}^{n}\xi_i$
                    s.t. $\forall\bar{y}_1 \in \mathscr{W}_1 : \mathbf{w}^T\Delta(y_1,\bar{y}_1)[\Psi(x_1,y_1) - \Psi(x_1,\bar{y}_1)] \geq \Delta(y_1,\bar{y}_1) - \xi_1$
                         $\vdots$
                         $\forall\bar{y}_n \in \mathscr{W}_n : \mathbf{w}^T\Delta(y_n,\bar{y}_n)[\Psi(x_n,y_n) - \Psi(x_n,\bar{y}_n)] \geq \Delta(y_n,\bar{y}_n) - \xi_n$
9:          end if
10:     end for
11: until no $\mathscr{W}_i$ has changed during iteration
12: return($\mathbf{w}$,$\xi$)
```

of constraints, starting with an empty working set $\mathscr{W} = \emptyset$. The algorithms iterate through the training examples and find the constraint that is violated most by the current solution $\mathbf{w}, \boldsymbol{\xi}$ (Line 5). If this constraint is violated by more than the desired precision ε (Line 6), the constraint is added to the working set (Line 7) and the QP is solved over the extended $\mathscr{W}$ (Line 8). The algorithms terminate when no constraint is added in the previous iteration, meaning that all constraints in OP2 or OP3 are fulfilled up to a precision of ε. The algorithm is provably efficient whenever the most violated constraint can be found efficiently. The argmax in Line 5 has an efficient solution for a wide variety of choices for Ψ, $\mathscr{Y}$, and Δ (see e.g., Tsochantaridis et al, 2005; Joachims, 2005; Yu et al, 2007; Yue et al, 2007), and often the algorithm for making predictions (see Eq. 1) can be adapted to compute this solution.

Related to Algorithm 1 is the method proposed in (Anguelov et al, 2005), which applies to the special case where the argmax in Line 5 can be computed as a linear program. This allows them not to explicitly maintain a working set, but implicitly represent it by folding n linear programs into the quadratic program OP2. To this special case also applies the method of Taskar et al (2005), which casts the training of max-margin structured predictors as a convex-concave saddle-point problem. It provides improved scalability compared to an explicit reduction to a polynomially-sized QP, but involves the use of a special min-cost quadratic flow solver in the projection steps of the extragradient method.

Exponentiated gradient methods, originally proposed for online learning of linear predictors (Kivinen and Warmuth, 1997), have also been applied to the training of structured predictors (Globerson et al, 2007; Bartlett et al, 2004). They solve the optimization problem in the dual, and treat conditional random field and structural SVM within the same framework using Bregman divergences. Stochastic gradient methods Vishwanathan et al (2006) have been applied to the training of conditional random field on large scale problems, and exhibit faster rate of convergence than BFGS methods. Recently, subgradient methods and their stochastic variants (Ratliff et al, 2007) have also been proposed to solve the optimization problem in max-margin structured prediction. While not yet explored for structured prediction, the PEGASOS algorithm (Shalev-Shwartz et al, 2007) has shown promising performance for binary classification SVMs. Related to such online methods is also the MIRA algorithm (Crammer and Singer, 2003), which has been used for training structured predictors (e.g. McDonald et al, 2005). However, to deal with the exponential size of $\mathscr{Y}$, heuristics have to be used (e.g. only using a k-best subset of $\mathscr{Y}$), leading to only approximate solutions of Optimization Problem OP2.

3 Training Algorithm

While polynomial runtime was established for most algorithms discussed above, training general structural SVMs on large-scale problems is still a challenging problem. In the following, we present an equivalent reformulation of the training problems for both margin-rescaling and slack-rescaling, leading to a cutting-plane training algorithm that has not only provably linear runtime in the number of training examples, but is also several orders of magnitude faster than conventional cutting-plane methods (Tsochantaridis et al, 2005) on large-scale problems. Nevertheless, the new algorithm is equally general as Algorithms 1 and 2.

3.1 1-Slack Formulation

The first step towards the new algorithm is a reformulation of the optimization problems for training. The key idea is to replace the n cutting-plane models of the hinge loss – one for each training example – with a single cutting plane model for the sum of the hinge-losses. Since there is only a single slack variable in the new formulations, we refer to them as “1-slack” formulations.

Optimization Problem 4 (1-SLACK STRUCTURAL SVM WITH MARGIN-RESCALING (PRIMAL))

$$\min_{\mathbf{w},\xi\geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w}+C\xi$$

$$s.t. \quad \forall(\bar{y}_1,...,\bar{y}_n)\in\mathscr{Y}^n: \frac{1}{n}\mathbf{w}^T\sum_{i=1}^{n}[\Psi(x_i,y_i)-\Psi(x_i,\bar{y}_i)] \geq \frac{1}{n}\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i)-\xi$$

Optimization Problem 5 (1-SLACK STRUCTURAL SVM WITH SLACK-RESCALING (PRIMAL))

$$\min_{\mathbf{w},\xi\geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w}+C\xi$$

$$s.t. \quad \forall(\bar{y}_1,...,\bar{y}_n)\in\mathscr{Y}^n: \frac{1}{n}\mathbf{w}^T\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i)[\Psi(x_i,y_i)-\Psi(x_i,\bar{y}_i)] \geq \frac{1}{n}\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i)-\xi$$

While OP4 has $|\mathscr{Y}|^n$ constraints, one for each possible combination of labels $(\bar{y}_1,...,\bar{y}_n)\in\mathscr{Y}^n$, it has only one slack variable ξ that is shared across all constraints. Each constraint corresponds to a tangent to $R_S^{\Delta_{MR}}(h)$ and $R_S^{\Delta_{SR}}(h)$ respectively, and the set of constraints forms an equivalent model of the risk function. Specifically, the following theorems show that $\xi^* = R_S^{\Delta_{MR}}(h_{w^*})$ at the solution (w^*,ξ^*) of OP4, and $\xi^* = R_S^{\Delta_{SR}}(h_{w^*})$ at the solution (w^*,ξ^*) of OP5, since the n-slack and the 1-slack formulations are equivalent in the following sense.

Theorem 1. (EQUIVALENCE OF OP2 AND OP4)
Any solution $\mathbf{w}^*$ of OP4 is also a solution of OP2 (and vice versa), with $\xi^* = \frac{1}{n}\sum_{i=1}^{n}\xi_i^*$.

Proof. Generalizing the proof in (Joachims, 2006), we will show that for every $\mathbf{w}$ the smallest feasible ξ and $\sum_i^n \xi_i$ are equal.

For a given $\mathbf{w}$, each ξ_i in OP2 can be optimized individually, and the smallest feasible ξ_i given $\mathbf{w}$ is achieved for

$$\xi_i = \max_{\bar{y}_i\in\mathscr{Y}}\{\Delta(y_i,\bar{y}_i)-\mathbf{w}^T[\Psi(x_i,y_i)-\Psi(x_i,\bar{y}_i)]\}.$$

For OP4, the smallest feasible ξ for a given $\mathbf{w}$ is

Algorithm 3 for training Structural SVMs (with margin-rescaling) via the 1-Slack Formulation (OP4).

```
 1: Input: $S = ((x_1,y_1),\dots,(x_n,y_n))$, $C$, $\varepsilon$
 2: $\mathscr{W} \leftarrow \emptyset$
 3: repeat
 4:     $(\mathbf{w},\xi) \leftarrow \operatorname{argmin}_{\mathbf{w},\xi\geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi$
            s.t. $\forall(\bar{y}_1,\dots,\bar{y}_n)\in\mathscr{W}: \frac{1}{n}\mathbf{w}^T\sum_{i=1}^{n}[\Psi(x_i,y_i)-\Psi(x_i,\bar{y}_i)] \geq \frac{1}{n}\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i)-\xi$
 5:     for i=1,...,n do
 6:         $\hat{y}_i \leftarrow \operatorname{argmax}_{\hat{y}\in\mathscr{Y}}\{\Delta(y_i,\hat{y}) + \mathbf{w}^T\Psi(x_i,\hat{y})\}$
 7:     end for
 8:     $\mathscr{W} \leftarrow \mathscr{W} \cup \{(\hat{y}_1,\dots,\hat{y}_n)\}$
 9: until $\frac{1}{n}\sum_{i=1}^{n}\Delta(y_i,\hat{y}_i) - \frac{1}{n}\mathbf{w}^T\sum_{i=1}^{n}[\Psi(x_i,y_i)-\Psi(x_i,\hat{y}_i)] \leq \xi+\varepsilon$
10: return($\mathbf{w}$,$\xi$)
```

$$\xi = \max_{(\bar{y}_1,\dots,\bar{y}_n)\in\mathscr{Y}^n}\left\{\frac{1}{n}\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i) - \mathbf{w}^T\frac{1}{n}\sum_{i=1}^{n}[\Psi(x_i,y_i)-\Psi(x_i,\bar{y}_i)]\right\}.$$

Since the function can be decomposed linearly in $\bar{y}_i$, for any given $\mathbf{w}$, each $\bar{y}_i$ can be optimized independently.

$$\xi = \sum_{i=1}^{n}\max_{\bar{y}_i\in\mathscr{Y}}\left\{\frac{1}{n}\Delta(y_i,\bar{y}_i) - \frac{1}{n}\mathbf{w}^T[\Psi(x_i,y_i)-\Psi(x_i,\bar{y}_i)]\right\} = \frac{1}{n}\sum_{i=1}^{n}\xi_i$$

Therefore, the objective functions of both optimization problems are equal for any $\mathbf{w}$ given the corresponding smallest feasible ξ and ξ_i. Consequently this is also true for $\mathbf{w}^*$ and its corresponding smallest feasible slacks ξ^* and ξ_i^*. □

Theorem 2. (EQUIVALENCE OF OP3 AND OP5)
Any solution $\mathbf{w}^*$ of OP5 is also a solution of OP3 (and vice versa), with $\xi^* = \frac{1}{n}\sum_{i=1}^{n}\xi_i^*$.

Proof. *Analogous to Theorem 1.* □

3.2 *Cutting-Plane Algorithm*

What could we possibly have gained by moving from the *n*-slack to the 1-slack formulation, exponentially increasing the number of constraints in the process? We will show in the following that the dual of the 1-slack formulation has a solution that is extremely sparse, with the number of non-zero dual variables independent of the number of training examples. To find this solution, we propose Algorithms 3 and 4, which are generalizations of the algorithm in (Joachims, 2006) to structural SVMs.

Similar to the cutting-plane algorithms for the *n*-slack formulations, Algorithms 3 and 4 iteratively construct a working set $\mathscr{W}$ of constraints. In each iteration, the al-

Algorithm 4 for training Structural SVMs (with slack-rescaling) via the 1-Slack Formulation (OP5).

```
1:  Input: $S = ((x_1, y_1), \dots, (x_n, y_n))$, $C$, $\varepsilon$
2:  $\mathcal{W} \leftarrow \emptyset$
3:  repeat
4:      $(\mathbf{w}, \xi) \leftarrow \mathrm{argmin}_{\mathbf{w}, \xi \geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi$
            s.t. $\forall(\bar{y}_1, \dots, \bar{y}_n) \in \mathcal{W}: \frac{1}{n}\mathbf{w}^T \sum_{i=1}^{n} \Delta(y_i, \bar{y}_i)[\Psi(x_i, y_i) - \Psi(x_i, \bar{y}_i)] \geq \frac{1}{n}\sum_{i=1}^{n}\Delta(y_i, \bar{y}_i) - \xi$
5:      for i=1,...,n do
6:          $\hat{y}_i \leftarrow \mathrm{argmax}_{\hat{y} \in \mathcal{Y}}\{\Delta(y_i, \hat{y})(1 - \mathbf{w}^T[\Psi(x_i, y_i) - \Psi(x_i, \hat{y})])\}$
7:      end for
8:      $\mathcal{W} \leftarrow \mathcal{W} \cup \{(\hat{y}_1, \dots, \hat{y}_n)\}$
9:  until $\frac{1}{n}\sum_{i=1}^{n}\Delta(y_i, \hat{y}_i) - \frac{1}{n}\mathbf{w}^T\sum_{i=1}^{n}\Delta(y_i, \hat{y}_i)[\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \leq \xi + \varepsilon$
10: return($\mathbf{w}$,$\xi$)
```

gorithms compute the solution over the current $\mathcal{W}$ (Line 4), find the most violated constraint (Lines 5-7), and add it to the working set. The algorithm stops once no constraint can be found that is violated by more than the desired precision ε (Line 9). Unlike in the n-slack algorithms, only a single constraint is added in each iteration. The following theorems characterize the quality of the solutions returned by Algorithms 3 and 4.

Theorem 3. (Correctness of Algorithm 3)
For any training sample $S = ((x_1, y_1), \dots, (x_n, y_n))$ and any $\varepsilon > 0$, if $(\mathbf{w}^*, \xi^*)$ is the optimal solution of OP4, then Algorithm 3 returns a point $(\mathbf{w}, \xi)$ that has a better objective value than $(\mathbf{w}^*, \xi^*)$, and for which $(\mathbf{w}, \xi + \varepsilon)$ is feasible in OP4.

Proof. We first verify that Lines 5-7 in Algorithm 3 compute the vector $(\hat{y}_1, \dots, \hat{y}_n) \in \mathcal{Y}^n$ that maximizes

$$\xi' = \max_{(\hat{y}_1, \dots, \hat{y}_n) \in \mathcal{Y}^n} \left\{ \frac{1}{n}\sum_{i=1}^{n}\Delta(y_i, \hat{y}_i) - \frac{1}{n}\mathbf{w}^T\sum_{i=1}^{n}[\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \right\}.$$

ξ' is the minimum value needed to fulfill all constraints in OP4 for the current $\mathbf{w}$. The maximization problem is linear in the $\hat{y}_i$, so one can maximize over each $\hat{y}_i$ independently.

$$\xi' = \frac{1}{n}\sum_{i=1}^{n}\max_{\hat{y} \in \mathcal{Y}}\{\Delta(y_i, \hat{y}) - \mathbf{w}^T[\Psi(x_i, y_i) - \Psi(x_i, \hat{y})]\} \quad (3)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\mathbf{w}^T\Psi(x_i, y_i) + \frac{1}{n}\sum_{i=1}^{n}\max_{\hat{y} \in \mathcal{Y}}\{\Delta(y_i, \hat{y}) + \mathbf{w}^T\Psi(x_i, \hat{y})\} \quad (4)$$

Since the first sum in Equation (4) is constant, the second term directly corresponds to the assignment in Line 6. As checked in Line 9, the algorithm terminates only if ξ' does not exceed the ξ from the solution over $\mathcal{W}$ by more than ε as desired.

Since the $(\mathbf{w},\xi)$ *returned by Algorithm 3 is the solution on a subset of the constraints from OP4, it holds that* $\frac{1}{2}\mathbf{w}^{*T}\mathbf{w}^{*}+C\xi^{*}\geq\frac{1}{2}\mathbf{w}^{T}\mathbf{w}+C\xi$. □

Theorem 4. (CORRECTNESS OF ALGORITHM 4)
For any training sample $S=((x_1,y_1),\ldots,(x_n,y_n))$ *and any* $\varepsilon>0$, *if* $(\mathbf{w}^{*},\xi^{*})$ *is the optimal solution of OP5, then Algorithm 4 returns a point* $(\mathbf{w},\xi)$ *that has a better objective value than* $(\mathbf{w}^{*},\xi^{*})$, *and for which* $(\mathbf{w},\xi+\varepsilon)$ *is feasible in OP5.*

Proof. Analogous to the proof of Theorem 3. □

Using a stopping criterion based on the accuracy of the empirical risk ξ^{*} is very intuitive and practically meaningful, unlike the stopping criteria typically used in decomposition methods. Intuitively, ε can be used to indicate how close one wants to be to the empirical risk of the best parameter vector. In most machine learning applications, tolerating a training error that is suboptimal by 0.1% is very acceptable. This intuition makes selecting the stopping criterion much easier than in other training methods, where it is usually defined based on the accuracy of the Kuhn-Tucker Conditions of the dual (see e.g., Joachims, 1999). Nevertheless, it is easy to see that ε also bounds the duality gap of the solution by $C\varepsilon$. Solving the optimization problems to an arbitrary but fixed precision of ε is essential in our analysis below, making sure that computation time is not wasted on computing a solution that is more accurate than necessary.

We next analyze the time complexity of Algorithms 3 and 4. It is easy to see that each iteration of the algorithm takes n calls to the separation oracle, and that for the linear kernel the remaining work in each iteration scales linearly with n as well. We show next that the number of iterations until convergence is bounded, and that this upper bound is independent of n.

The argument requires the Wolfe-dual programs, which are straightforward to derive (see Appendix). For a more compact notation, we denote vectors of labels as $\bar{\mathbf{y}}=(\bar{y}_1,\ldots,\bar{y}_n)\in\mathcal{Y}^n$. For such vectors of labels, we then define $\Delta(\bar{\mathbf{y}})$ and the inner product $H_{MR}(\bar{\mathbf{y}},\bar{\mathbf{y}}')$ as follows. Note that y_i and y_j denote correct training labels, while $\bar{y}_i$ and $\bar{y}'_j$ denote arbitrary labels:

$$\Delta(\bar{\mathbf{y}})=\frac{1}{n}\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i) \tag{5}$$

$$\begin{aligned}H_{MR}(\bar{\mathbf{y}},\bar{\mathbf{y}}')=\tfrac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\Big[&\Psi(x_i,y_i)^T\Psi(x_j,y_j)-\Psi(x_i,y_i)^T\Psi(x_j,\bar{y}'_j)\\&-\Psi(x_i,\bar{y}_i)^T\Psi(x_j,y_j)+\Psi(x_i,\bar{y}_i)^T\Psi(x_j,\bar{y}'_j)\Big]\end{aligned} \tag{6}$$

The inner products $\Psi(x,y)^T\Psi(x',y')$ are computed either explicitly or via a Kernel $K(x,y,x',y')=\Psi(x,y)^T\Psi(x',y')$. Note that it is typically more efficient to compute

$$H_{MR}(\bar{\mathbf{y}},\bar{\mathbf{y}}')=\frac{1}{n^2}\left[\sum_{i=1}^{n}(\Psi(x_i,y_i)-\Psi(x_i,\bar{y}_i))\right]^T\left[\sum_{j=1}^{n}(\Psi(x_j,y_j)-\Psi(x_j,\bar{y}'_j))\right] \tag{7}$$

if no kernel is used. The dual of the 1-slack formulation for margin-rescaling is:

Optimization Problem 6 (1-SLACK STRUCTURAL SVM WITH MARGIN-RESCALING (DUAL))

$$\max_{\boldsymbol{\alpha}\geq 0} \; D(\boldsymbol{\alpha}) = \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \Delta(\bar{\mathbf{y}})\alpha_{\bar{\mathbf{y}}} - \frac{1}{2}\sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n}\sum_{\bar{\mathbf{y}}'\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}}\alpha_{\bar{\mathbf{y}}'} H_{MR}(\bar{\mathbf{y}},\bar{\mathbf{y}}')$$
$$s.t. \quad \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}} = C$$

For the case of slack-rescaling, the respective $H(\bar{\mathbf{y}},\bar{\mathbf{y}}')$ is defined as follows, and we again give an analogous factorization that is more efficient to compute if no kernel is used:

$$H_{SR}(\bar{\mathbf{y}},\bar{\mathbf{y}}') = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta(y_i,\bar{y}_i)\Delta(y_j,\bar{y}'_j)\Big[\Psi(x_i,y_i)^T\Psi(x_j,y_j) - \Psi(x_i,y_i)^T\Psi(x_j,\bar{y}'_j) - \Psi(x_i,\bar{y}_i)^T\Psi(x_j,y_j) + \Psi(x_i,\bar{y}_i)^T\Psi(x_j,\bar{y}'_j)\Big] \tag{8}$$

$$= \frac{1}{n^2}\left[\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i)(\Psi(x_i,y_i) - \Psi(x_i,\bar{y}_i))\right]^T \left[\sum_{j=1}^{n}\Delta(y_j,\bar{y}'_j)(\Psi(x_j,y_j) - \Psi(x_j,\bar{y}'_j))\right] \tag{9}$$

The dual of the 1-slack formulation is:

Optimization Problem 7 (1-SLACK STRUCTURAL SVM WITH SLACK-RESCALING (DUAL))

$$\max_{\boldsymbol{\alpha}\geq 0} \; D(\boldsymbol{\alpha}) = \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \Delta(\bar{\mathbf{y}})\alpha_{\bar{\mathbf{y}}} - \frac{1}{2}\sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n}\sum_{\bar{\mathbf{y}}'\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}}\alpha_{\bar{\mathbf{y}}'} H_{SR}(\bar{\mathbf{y}},\bar{\mathbf{y}}')$$
$$s.t. \quad \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}} = C$$

Using the respective dual solution $\boldsymbol{\alpha}^*$, one can compute inner products with the weight vector $\mathbf{w}^*$ solving the primal via

$$\mathbf{w}^{*T}\Psi(x,y) = \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \alpha^*_{\bar{\mathbf{y}}}\left(\frac{1}{n}\sum_{j=1}^{n}\left[\Psi(x,y)^T\Psi(x_j,y_j) - \Psi(x,y)^T\Psi(x_j,\bar{y}_j)\right]\right)$$
$$= \left[\frac{1}{n}\sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n}\alpha^*_{\bar{\mathbf{y}}}\sum_{j=1}^{n}\left[\Psi(x_j,y_j) - \Psi(x_j,\bar{y}_j)\right]\right]^T \Psi(x,y)$$

for margin-rescaling and via

$$\mathbf{w}^{*T}\Psi(x,y) = \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \alpha^*_{\bar{\mathbf{y}}}\left(\frac{1}{n}\sum_{j=1}^{n}\Delta(y_j,\bar{y}_j)\left[\Psi(x,y)^T\Psi(x_j,y_j) - \Psi(x,y)^T\Psi(x_j,\bar{y}_j)\right]\right)$$
$$= \left[\frac{1}{n}\sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n}\alpha^*_{\bar{\mathbf{y}}}\sum_{j=1}^{n}\Delta(y_j,\bar{y}_j)\left[\Psi(x_j,y_j) - \Psi(x_j,\bar{y}_j)\right]\right]^T \Psi(x,y)$$

for slack-rescaling. We will show in the following that only a small (i.e., polynomial) number of the $\alpha_{\bar{\mathbf{y}}}$ is non-zero at the solution. In analogy to classification SVMs, we will refer to those $\bar{\mathbf{y}}$ with non-zero $\alpha_{\bar{\mathbf{y}}}$ as *Support Vectors*. However, note that Support Vectors in the 1-slack formulation are linear combinations of multiple examples. We can now state the theorem giving an upper bound on the number of iterations of the 1-slack algorithms. The proof extends the one in (Joachims, 2006) to general structural SVMs, and is based on the technique introduced in (Joachims, 2003) and generalized in (Tsochantaridis et al, 2005). The final step of the proof uses an improvement developed in (Teo et al, 2007).

Theorem 5. (1-SLACK MARGIN-RESCALING SVM ITERATION COMPLEXITY) *For any $0 < C$, $0 < \varepsilon \le 4R^2C$ and any training sample $S = ((x_1,y_1),\dots,(x_n,y_n))$, Algorithms 3 terminates after at most*

$$\left\lceil \log_2\left(\frac{\Delta}{4R^2C}\right)\right\rceil + \left\lceil \frac{16R^2C}{\varepsilon}\right\rceil \tag{10}$$

iterations. $R^2 = \max_{i,\bar{y}} ||\Psi(x_i,y_i) - \Psi(x_i,\bar{y})||^2$, $\Delta = \max_{i,\bar{y}} \Delta(y_i,\bar{y})$, and $\lceil .. \rceil$ is the integer ceiling function.

Proof. *We will show that adding each new constraint to $\mathscr{W}$ increases the objective value at the solution of the quadratic program in Line 4 by at least some constant positive value. Since the objective value of the solution of OP6 is upper bounded by $C\Delta$ (since $\mathbf{w} = 0$ and $\xi = \Delta$ is a feasible point in the primal), the algorithm can only perform a constant number of iterations before termination. The amount by which the solution increases by adding one constraint that is violated by more then ε (i.e., the criteria in Line 9 of Algorithm 3 and Algorithm 4) to $\mathscr{W}$ can be lower bounded as follows.*

Let $\hat{\mathbf{y}}$ be the newly added constraint and let $\boldsymbol{\alpha}$ be the solution of the dual before the addition. To lower bound the progress made by the algorithm in each iteration, consider the increase in the dual that can be achieved with a line search

$$\max_{0\le\beta\le C}\{D(\boldsymbol{\alpha}+\beta\boldsymbol{\eta})\} - D(\boldsymbol{\alpha}). \tag{11}$$

The direction $\boldsymbol{\eta}$ is constructed by setting $\eta_{\hat{\mathbf{y}}} = 1$ and $\eta_{\bar{\mathbf{y}}} = -\frac{1}{C}\alpha_{\bar{\mathbf{y}}}$ for all other $\bar{\mathbf{y}}$. Note that the constraints on β and the construction of $\boldsymbol{\eta}$ ensure that $\boldsymbol{\alpha}+\beta\boldsymbol{\eta}$ never leaves the feasible region of the dual.

To apply Lemma 2 (see Appendix) for computing the progress made by a line search, we need a lower bound for $\nabla D(\boldsymbol{\alpha})^T\boldsymbol{\eta}$ and an upper bound for $\boldsymbol{\eta}^T H \boldsymbol{\eta}$. Starting with the lower bound for $\nabla D(\boldsymbol{\alpha})^T\boldsymbol{\eta}$, note that

$$\frac{\partial D(\boldsymbol{\alpha})}{\partial \alpha_{\bar{\mathbf{y}}}} = \Delta(\bar{\mathbf{y}}) - \sum_{\bar{\mathbf{y}}'\in\mathscr{W}} \alpha_{\bar{\mathbf{y}}'} H_{MR}(\bar{\mathbf{y}},\bar{\mathbf{y}}') = \xi \tag{12}$$

for all $\bar{\mathbf{y}}$ with non-zero $\alpha_{\bar{\mathbf{y}}}$ at the solution over the previous working set $\mathscr{W}$. For the newly added constraint $\hat{\mathbf{y}}$ and some $\gamma > 0$,

$$\frac{\partial D(\boldsymbol{\alpha})}{\partial \alpha_{\hat{\mathbf{y}}}} = \Delta(\hat{\mathbf{y}}) - \sum_{\bar{\mathbf{y}}' \in \mathscr{W}} \alpha_{\bar{\mathbf{y}}'} H_{MR}(\hat{\mathbf{y}}, \bar{\mathbf{y}}') = \xi + \gamma \geq \xi + \varepsilon \tag{13}$$

by construction due to Line 9 of Algorithms 3. It follows that

$$\begin{aligned} \nabla D(\boldsymbol{\alpha})^T \boldsymbol{\eta} &= \xi + \gamma - \sum_{\bar{\mathbf{y}} \in \mathscr{W}} \frac{\alpha_{\bar{\mathbf{y}}}}{C} \xi && (14) \\ &= \xi \left(1 - \frac{1}{C} \sum_{\bar{\mathbf{y}} \in \mathscr{W}} \alpha_{\bar{\mathbf{y}}} \right) + \gamma && (15) \\ &= \gamma. && (16) \end{aligned}$$

The following gives an upper bound for $\boldsymbol{\eta}^T H \boldsymbol{\eta}$*, where* $H_{\bar{\mathbf{y}}\bar{\mathbf{y}}'} = H_{MR}(\bar{\mathbf{y}}, \bar{\mathbf{y}}')$ *for* $\bar{\mathbf{y}}, \bar{\mathbf{y}}' \in \mathscr{W} \cup \{\hat{\mathbf{y}}\}$.

$$\begin{aligned} \boldsymbol{\eta}^T H \boldsymbol{\eta} &= H_{MR}(\hat{\mathbf{y}}, \hat{\mathbf{y}}) - \frac{2}{C} \sum_{\bar{\mathbf{y}} \in \mathscr{W}} \alpha_{\bar{\mathbf{y}}} H_{MR}(\bar{\mathbf{y}}, \hat{\mathbf{y}}) + \frac{1}{C^2} \sum_{\bar{\mathbf{y}} \in \mathscr{W}} \sum_{\bar{\mathbf{y}}' \in \mathscr{W}} \alpha_{\bar{\mathbf{y}}} \alpha_{\bar{\mathbf{y}}'} H_{MR}(\bar{\mathbf{y}}, \bar{\mathbf{y}}') && (17) \\ &\leq R^2 + \frac{2}{C} C R^2 + \frac{1}{C^2} C^2 R^2 && (18) \\ &= 4R^2 && (19) \end{aligned}$$

The bound uses that $-R^2 \leq H_{MR}(\bar{\mathbf{y}}, \hat{\mathbf{y}}) \leq R^2$.

Plugging everything into the bound of Lemma 2 shows that the increase of the objective is at least

$$\max_{0 \leq \beta \leq C} \{ D(\boldsymbol{\alpha} + \beta \boldsymbol{\eta}) \} - D(\boldsymbol{\alpha}) \geq \min \left\{ \frac{C\gamma}{2}, \frac{\gamma^2}{8R^2} \right\} \tag{20}$$

Note that the first case applies whenever $\gamma \geq 4R^2 C$*, and that the second case applies otherwise.*

The final step of the proof is to use this constant increase of the objective value in each iteration to bound the maximum number of iterations. First, note that $\alpha_{\bar{\mathbf{y}}} = 0$ *for all incorrect vectors of labels* $\bar{\mathbf{y}}$ *and* $\alpha_{\bar{\mathbf{y}}^*} = C$ *for the correct vector of labels* $\bar{\mathbf{y}}^* = (y_1, \ldots, y_n)$ *is a feasible starting point* $\boldsymbol{\alpha}_0$ *with a dual objective of* 0*. This means the initial optimality gap* $\delta(0) = D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}_0)$ *is at most* $C\Delta$*, where* $\boldsymbol{\alpha}^*$ *is the optimal dual solution. An optimality gap of* $\delta(i) = D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}_i)$ *ensures that there exists a constraint that is violated by at least* $\gamma \geq \frac{\delta(i)}{C}$*. This means that the first case of (20) applies while* $\delta(i) \geq 4R^2C^2$*, leading to a decrease in the optimality gap of at least*

$$\delta(i+1) \leq \delta(i) - \frac{1}{2}\delta(i) \tag{21}$$

in each iteration. Starting from the worst possible optimality gap of $\delta(0) = C\Delta$*, the algorithm needs at most*

$$i_1 \leq \left\lceil \log_2 \left(\frac{\Delta}{4R^2C} \right) \right\rceil \tag{22}$$

iterations until it has reached an optimality gap of $\delta(i_1) \leq 4R^2C^2$*, where the second case of (20) becomes valid. As proposed in (Teo et al, 2007), the recurrence equation*

$$\delta(i+1) \leq \delta(i) - \frac{1}{8R^2C^2}\delta(i)^2 \tag{23}$$

for the second case of (20) can be upper bounded by solving the differential equation $\frac{\partial \delta(i)}{\partial i} = -\frac{1}{8R^2C^2}\delta(i)^2$ *with boundary condition* $\delta(0) = 4R^2C^2$*. The solution is* $\delta(i) \leq \frac{8R^2C^2}{i+2}$*, showing that the algorithms does not need more than*

$$i_2 \leq \frac{8R^2C^2}{C\varepsilon} \tag{24}$$

iterations until it reaches an optimality gap of $C\varepsilon$ *when starting at a gap of* $4R^2C^2$*, where* ε *is the desired target precision given to the algorithm. Once the optimality gap reaches* $C\varepsilon$*, it is no longer guaranteed that an* ε*-violated constraint exists. However, such constraints may still exist and so the algorithm does not yet terminate. But since each such constraint leads to an increase in the dual objective of at least* $\frac{\varepsilon^2}{8R^2}$*, only*

$$i_3 \leq \left\lceil \frac{8R^2C}{\varepsilon} \right\rceil \tag{25}$$

can be added before the optimality gap becomes negative. The overall bound results from adding i_1*,* i_2*, and* i_3*.* □

Note that the proof of the theorem requires only a line search in each step, while Algorithm 4 actually computes the full QP solution. This suggests the following. On the one hand, the actual number of iterations in Algorithm 4 might be substantially smaller in practice than what is predicted by the bound. On the other hand, it suggests a variant of Algorithm 4, where the QP solver is replaced by a simple line search. This may be beneficial in structured prediction problems where the separation oracle in Line 6 is particularly cheap to compute.

Theorem 6. (1-SLACK SLACK-RESCALING SVM ITERATION COMPLEXITY) *For any* $0 < C$, $0 < \varepsilon \leq 4\Delta^2R^2C$ *and any training sample* $S = ((x_1, y_1), \ldots, (x_n, y_n))$*, Algorithms 4 terminate after at most*

$$\left\lceil \log_2 \left(\frac{1}{4R^2\Delta C} \right) \right\rceil + \left\lceil \frac{16R^2\Delta^2C}{\varepsilon} \right\rceil \tag{26}$$

iterations. $R^2 = \max_{i,\bar{y}} ||\Psi(x_i,y_i) - \Psi(x_i,\bar{y})||^2$*,* $\Delta = \max_{i,\bar{y}} \Delta(y_i,\bar{y})$*, and* $\lceil .. \rceil$ *is the integer ceiling function.*

Proof. The proof for the case of slack-rescaling is analogous. The only difference is that $-\Delta^2R^2 \leq H_{SR}(\bar{y},\bar{y}') \leq \Delta^2R^2$. □

The $O(\frac{1}{\varepsilon})$ convergence rate in the bound is tight, as the following example shows. Consider a multi-class classification problem with infinitely many classes $\mathscr{Y} = \{1,...,\infty\}$ and a feature space $\mathscr{X} = \Re$ that contains only one feature. This problem can be encoded using a feature map $\Psi(x,y)$ which takes value x in position y and 0 everywhere else. For a training set with a single training example $(x,y) = ((1),1)$ and using the zero/one-loss, the 1-slack quadratic program for both margin-rescaling and slack-rescaling is

$$\begin{aligned} \min_{\mathbf{w},\xi\geq 0} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi \\ s.t. \quad & \mathbf{w}^T[\Psi(x,1)-\Psi(x,2)] \geq 1-\xi \\ & \mathbf{w}^T[\Psi(x,1)-\Psi(x,3)] \geq 1-\xi \\ & \mathbf{w}^T[\Psi(x,1)-\Psi(x,4)] \geq 1-\xi \\ & \vdots \end{aligned} \tag{27}$$

Let's assume without loss of generaltity that Algorithm 3 (or equivalently Algorithm 4) introduces the first constraint in the first iteration. For $C \geq \frac{1}{2}$ the solution over this working set is $\mathbf{w}^T = (\frac{1}{2}, -\frac{1}{2}, 0, 0, \ldots)$ and $\xi = 0$. All other constraints are now violated by $\frac{1}{2}$ and one of them is selected at random to be added to the working set in the next iteration. It is easy to verify that after adding k constraints, the solution over the working set is $\mathbf{w}^T = (\frac{k}{k+1}, -\frac{1}{k+1}, \ldots, -\frac{1}{k+1}, 0, 0, \ldots)$ for $C \geq \frac{1}{2}$, and all constraints outside the working set are violated by $\varepsilon = \frac{1}{k+1}$. It therefore takes $O(\frac{1}{\varepsilon})$ iterations to reach a desired precision of ε.

The $O(C)$ scaling with C is tight as well, at least for small values of C. Considering the same examples and $C \leq \frac{1}{2}$, the solution over the working set after adding k constraints is $\mathbf{w}^T = (C, -\frac{C}{k}, \ldots, -\frac{C}{k}, 0, 0, \ldots)$. This means that after k constraints, all constraints outside the working set are violated by $\varepsilon = \frac{C}{k}$. Consequently, the bounds in (10) and (26) accurately reflect the worst-case scaling with C up to the log-term for $C \leq \frac{1}{2}$.

The following theorem summarizes our characterization of the time complexity of the 1-slack algorithms. In real applications, however, we will see that Algorithms 3 scales much better than what is predicted by these worst-case bounds both w.r.t. C and ε. Note that a "support vector" (i.e. point with non-zero dual variable) no longer corresponds to a single data point in the 1-slack dual, but is typically a linear combination of data points.

Corollary 1. (Time Complexity of Algorithms 3 and 4 for Linear Kernel)

For any n training examples $S = ((x_1,y_1),\ldots,(x_n,y_n))$ with $\max_{i,\bar{y}}||\Psi(x_i,y_i)-\Psi(x_i,\bar{y})||^2 \leq R^2 < \infty$ and $\max_{i,\bar{y}}\Delta(y_i,\bar{y}) \leq \Delta < \infty$ for all

n, the 1-slack cutting plane Algorithms 3 and 4 with constant ε and C using the linear kernel

- *require at most $O(n)$ calls to the separation oracle,*
- *require at most $O(n)$ computation time outside the separation oracle,*
- *find a solution where the number of support vectors (i.e. the number of non-zero dual variables in the cutting-plane model) does not depend on n,*

for any fixed value of $C > 0$ and $\varepsilon > 0$.

Proof. Theorems 5 and 6 shows that the algorithms terminate after a constant number of iterations that does not depend on n. Since only one constraint is introduced in each iteration, the number of support vectors is bounded by the number of iterations. In each iteration, the algorithm performs exactly n calls to the separation oracle, which proves the first statement. Similarly, the QP that is solved in each iteration is of constant size and therefore requires only constant time. It is easily verified that the remaining operations in each iterations can be done in time $O(n)$ using Eqs. (7) and (9). □

We further discuss the time complexity for the case of kernels in the following section. Note that the linear-time algorithm proposed in (Joachims, 2006) for training binary classification SVMs is a special case of the 1-slack methods developed here. In particular, for binary classification $\mathscr{X} = \Re^N$ and $\mathscr{Y} = \{-1,+1\}$, and plugging

$$\Psi(x,y) = \frac{1}{2}yx \text{ and } \Delta(y,y') = \begin{cases} 0 \text{ if } y = y' \\ 1 \; otherwise \end{cases} \tag{28}$$

into either n-slack formulation OP2 or OP3 produces the standard SVM optimization problem OP1. The 1-slack formulations and algorithms are then equivalent to those in (Joachims, 2006). However, the $O(\frac{1}{\varepsilon})$ bound on the maximum number of iterations derived here is tighter than the $O(\frac{1}{\varepsilon^2})$ bound in (Joachims, 2006). Using a similar argument, it can also be shown that the ordinal regression method in (Joachims, 2006) is a special case of the 1-slack algorithm.

3.3 Kernels and Low-Rank Approximations

For problems where a (non-linear) kernel is used, the computation time in each iteration is $O(n^2)$ instead of $O(n)$, since Eqs. (7) and (9) not longer apply. However, the 1-slack algorithm can easily exploit rank-k approximations, which we will show reduces the computation time outside of the separation oracle from $O(n^2)$ to $O(nk+k^3)$. Let $(x'_1,y'_1),...,(x'_k,y'_k)$ be a set of basis functions so that the subspace spanned by $\Psi(x'_1,y'_1),...,\Psi(x'_k,y'_k)$ (approximately) contains the solution $\mathbf{w}^*$ of OP4 and OP5 respectively. Algorithms for finding such approximations have been suggested in (Keerthi et al, 2006; Fukumizu et al, 2004; Smola and Schölkopf, 2000) for classifications SVMs, and at least some of them can be extended to structural SVMs as well. In the simplest case, the set of k basis functions can be chosen randomly from the set of training examples.

For a kernel $K(.)$ and the resulting Gram matrix K with $K_{ij} = \Psi(x'_i,y'_i)^T \Psi(x'_j,y'_j) = K(x'_i,y'_i,x'_j,y'_j)$, we can compute the inverse L^{-1} of the Cholesky Decomposition L of K in time $O(k^3)$. Assuming that $\mathbf{w}^*$ actually lies in the subspace, we can equivalently rewrite the 1-slack optimization problems as

Optimization Problem 8 (1-SLACK STRUCTURAL SVM WITH MARGIN-RESCALING AND k BASIS FUNCTIONS (PRIMAL))

$$\min_{\boldsymbol{\beta},\xi\geq 0} \frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{\beta} + C\,\xi$$

$$s.t. \forall(\bar{y}_1,\ldots,\bar{y}_n) \in \mathscr{Y}^n: \ \frac{1}{n}\boldsymbol{\beta}^T L^{-1}\sum_{i=1}^{n}\begin{pmatrix} K(x_i,y_i,x'_1,y'_1) - K(x_i,\bar{y}_i,x'_1,y'_1) \\ \vdots \\ K(x_i,y_i,x'_k,y'_k) - K(x_i,\bar{y}_i,x'_k,y'_k) \end{pmatrix} \geq \frac{1}{n}\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i) - \xi$$

Optimization Problem 9 (1-SLACK STRUCTURAL SVM WITH SLACK-RESCALING AND k BASIS FUNCTIONS (PRIMAL))

$$\min_{\boldsymbol{\beta},\xi\geq 0} \frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{\beta} + C\,\xi$$

$$s.t. \forall(\bar{y}_1,\ldots,\bar{y}_n) \in \mathscr{Y}^n: \ \frac{1}{n}\boldsymbol{\beta}^T L^{-1}\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i)\begin{pmatrix} K(x_i,y_i,x'_1,y'_1) - K(x_i,\bar{y}_i,x'_1,y'_1) \\ \vdots \\ K(x_i,y_i,x'_k,y'_k) - K(x_i,\bar{y}_i,x'_k,y'_k) \end{pmatrix} \geq \frac{1}{n}\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i) - \xi$$

Intuitively, the values of the kernel $K(.)$ with each of the k basis functions form a new feature vector $\Psi'(x,y)^T = (K(x,y,x'_1,y'_1),\ldots,K(x,y,x'_k,y'_k))^T$ describing each example (x,y). After multiplication with L^{-1}, OP8 and OP9 become identical to a problem with linear kernel and k features, and it is straightforward to see that Algorithms 3 and 4 apply to this new representation.

Corollary 2. (TIME COMPLEXITY OF ALGORITHMS 3 AND 4 FOR NON-LINEAR KERNEL)

For any n training examples $S = ((x_1,y_1),\ldots,(x_n,y_n))$ with $\max_{i,\bar{y}} \|\Psi(x_i,y_i) - \Psi(x_i,\bar{y})\|^2 \leq R^2 < \infty$ and $\max_{i,\bar{y}} \Delta(y_i,\bar{y}) \leq \Delta < \infty$ for all n, the 1-slack cutting plane Algorithms 3 and 4 using a non-linear kernel

- *require at most $O(n)$ calls to the separation oracle,*
- *require at most $O(n^2)$ computation time outside the separation oracle,*
- *require at most $O(nk+k^3)$ computation time outside the separation oracle, if a set of $k \leq n$ basis functions is used,*
- *find a solution where the number of support vectors does not depend on n,*

for any fixed value of $C > 0$ and $\varepsilon > 0$.

Proof. The proof is analogous to that of Corollary 1. For the low-rank approximation, note that it is more efficient to once compute $\mathbf{w}^T = \boldsymbol{\beta}^T L^{-1}$ *before entering the loop in Line 5, than to compute* $L^{-1}\Psi'(x,y)$ *for each example.* k^3 *is the cost of the Cholesky Decomposition, but this needs to be computed only once.* □

4 Implementation

We implemented both the *n*-slack algorithms and the 1-slack algorithms in software package called *SVM*^{struct}, which we make publicly available for download at `http://svmlight.joachims.org`. *SVM*^{struct} uses *SVM*^{light} as the optimizer for solving the QP sub-problems. Users may adapt *SVM*^{struct} to their own structural learning tasks by implementing API functions corresponding to task-specific Ψ, Δ, separation oracle, and inference. User API functions are in C. A popular extension is *SVM*^{python}, which allows users to write API functions in Python instead, and eliminates much of the drudge work of C including model serialization/deserialization and memory management. This extension is available for download at `http://www.cs.cornell.edu/~tomf/svmpython2/`.

An efficient implementation of the algorithms required a variety of design decisions, which are summarized in the following. These design decisions have a substantial influence on the practical efficiency of the algorithms.

Restarting the QP Sub-Problem Solver from the Previous Solution. Instead of solving each QP subproblem from scratch, we restart the optimizer from the dual solution of the previous working set as the starting point. This applies to both the *n*-slack and the 1-slack algorithms.

Batch Updates for the *n*-Slack Algorithm. Algorithm 1 recomputes the solution of the QP sub-problem after each update to the working set. While this allows the algorithm to potentially find better constraints to be added in each step, it requires a lot of time in the QP solver. We found that it is more efficient to wait with recomputing the solution of the QP sub-problem until 100 constraints have been added.

Managing the Accuracy of the QP Sub-Problem Solver. In the initial iterations, a relatively low precision solution of the QP sub-problems is sufficient for identifying the next violated constraint to add to the working set. We therefore adjust the precision of the QP sub-problem optimizer throughout the optimization process for all algorithms.

Removing Inactive Constraints from the Working Set. For both the *n*-slack and the 1-slack algorithm, constraints that were added to the working set in early iterations often become inactive later in the optimization process. These constraints can be removed without affecting the theoretical convergence guarantees of the algorithm, leading to smaller QP's being solved in each iteration. At the end of each iteration, we therefore remove constraints from the working set that have not been active in the last 50 QP sub-problems.

Caching $\Psi(x_i,y_i) - \Psi(x_i,\hat{y}_i)$ in the 1-Slack Algorithm. If the separation oracle returns a label $\hat{y}_i$ for an example x_i, the constraint added in the n-slack algorithm ensures that this label will never again produce an ε-violated constraint in a subsequent iteration. This is different, however, in the 1-slack algorithm, where the same label can be involved in an ε-violated constraint over and over again. We therefore cache the f most recently used $\Psi(x_i,y_i) - \Psi(x_i,\hat{y}_i)$ for each training example x_i (typically $f = 10$ in the following experiments). Let's denote the cache for example x_i with $\mathscr{C}_i$. Instead of asking the separation oracle in every iteration, the algorithm first tries to construct a sufficiently violated constraint from the caches via

```
for i=1,...,n do
    $\hat{y}_i \leftarrow \max_{\hat{y} \in \mathscr{C}_i}\{\Delta(y_i,\hat{y}) + \mathbf{w}^T\Psi(x_i,\hat{y})\}$
end for
```

or the analogous variant for the case of slack-rescaling. Only if this fails will the algorithm ask the separation oracle and update the cache. The goal of this caching strategy is to decrease the number of calls to the separation oracle. Note that in many applications, the separation oracle is very expensive (e.g., CFG parsing).

Parallelization. While currently not implemented, the loop in Lines 5-7 of the 1-slack algorithms can easily be parallelized. In principle, one could make use of up to n parallel threads, each computing the separation oracle for a subset of the training sample. For applications like CFG parsing, where more then 99% of the overall runtime is spent on the separation oracle (see Section 5), parallizing this loop will lead to a substantial speed-up that should be almost linear in the number of threads.

Solving the Dual of the QP Sub-Problems in the 1-Slack Algorithm. As indicated by Theorems 5 and 6, the working sets in the 1-slack algorithm stay small independent of the size of the training set. In practice, typically less then 100 constraints are active at the solutions and we never encountered a single instance where the working set grew beyond 1000 constraints. This makes it advantageous to store and solve the QP sub-problems in the dual instead of in the primal, since the dual is not affected by the dimensionality of $\Psi(x,y)$. The algorithm explicitly stores the Hessian H of the dual and adds or deletes a row/column whenever a constraint is added or removed from the working set. Note that this is not feasible for the n-slack algorithm, since the working set size is typically orders of magnitude larger (often $> 100,000$ constraints).

5 Experiments

For the experiments in this paper we will consider the following four applications, namely binary classification, multi-class classification, sequence tagging with linear chain HMMs, and CFG grammar learning. They cover the whole spectrum of possible applications, from multi-class classification involving a simple $\mathscr{Y}$ of low cardinality and with a very inexpensive separation oracle, to CFG parsing with large and complex structural objects and an expensive separation oracle. The particular setup for the different applications is as follows.

Binary Classification. For binary classification $\mathscr{X} = \Re^N$ and $\mathscr{Y} = \{-1, +1\}$. Using

$$\Psi(x,y) = \frac{1}{2}yx \text{ and } \Delta(y,\bar{y}) = 100[y \neq \bar{y}] = \begin{cases} 0 & \text{if } y = \bar{y} \\ 100 & otherwise \end{cases} \tag{29}$$

in the 1-slack formulation, OP4 results in the algorithm presented in (Joachims, 2006) and implemented in the SVM-perf software[3]. In the n-slack formulation, one immediately recovers Vapnik et al.'s original classification SVM formulation of OP1 (Cortes and Vapnik, 1995; Vapnik, 1998) (up to the more convenient percentage-scale rescaling of the loss function and the absence of the bias term), which we solve using SVM^{light}.

Multi-Class Classification. This is another simple instance of a structual SVM, where $X = \Re^N$ and $\mathscr{Y} = \{1, ..., k\}$. Using $\Delta(y,\bar{y}) = 100[y \neq \bar{y}]$ and

$$\Psi_{multi}(x,y) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ x \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{30}$$

where the feature vector x is stacked into position y, the resulting n-slack problem becomes identical to the multi-class SVM of Crammer and Singer (2001). Our SVM-multiclass (V2.13) implementation[3] is also built via the SVM^{struct} API. The argmax for the separation oracle and the prediction are computed by explicit enumeration.

We use the Covertype dataset of Blackard, Jock & Dean as our benchmark for the multi-class SVM. It is a 7-class problem with $n = 522,911$ examples and 54 features. This means that the dimensionality of $\Psi(x,y)$ is $N = 378$.

Sequence Tagging with Linear Chain HMMs. In sequence tagging (e.g., Part-of-Speech Tagging) each input $x = (x_1, ..., x_l)$ is a sequence of feature vectors (one for each word), and $y = (y_1, ..., y_l)$ is a sequence of labels $y_i \in \{1, ..., k\}$ of matching length. Isomorphic to a linear chain HMM, we model dependencies between each y_i and x_i, as well as dependencies between y_i and y_{i-1}. Using the definition of $\Psi_{multi}(x,y)$ from above, this leads to a joint feature vector of

[3] Available at `svmlight.joachims.org`

$$\Psi_{HMM}((x_1,...,x_l),(y_1,...,y_l)) = \sum_{i=1}^{l} \begin{pmatrix} \Psi_{multi}(x_i,y_i) \\ [y_i = 1][y_{i-1} = 1] \\ [y_i = 1][y_{i-1} = 2] \\ \vdots \\ [y_i = k][y_{i-1} = k] \end{pmatrix}. \tag{31}$$

We use the number of misclassified tags $\Delta((y_1,...,y_l),(y'_1,...,y'_l)) = \sum_{i=1}^{l}[y_i \neq y'_i]$ as the loss function. The argmax for prediction and the separation oracle are both computed via the Viterbi algorithm. Note that the separation oracle is equivalent to the prediction argmax after adding 1 to the node potentials of all incorrect labels. Our SVM-HMM (V3.10) implementation based on SVM^{struct} is also available online[3].

We evaluate on the Part-of-Speech tagging dataset from the Penn Treebank corpus (Marcus et al, 1993). After splitting the dataset into training and test set, it has $n = 35,531$ training examples (i.e., sentences), leading to a total of $854,022$ tags over $k = 43$ labels. The feature vectors x_i describing each word consist of binary features, each indicating the presence of a particular prefix or suffix in the current word, the previous word, and the following word. All prefixes and suffixes observed in the training data are used as features. In addition, there are features encoding the length of the word. The total number of features is approximately $430,000$, leading to a $\Psi_{HMM}(x,y)$ of dimensionality $N = 18,573,781$.

Parsing with Context Free Grammars. We use natural language parsing as an example application where the cost of computing the separation oracle is comparatively high. Here, each input $x = (x_1,...,x_l)$ is a sequence of feature vectors (one for each word), and y is a tree with x as its leaves. Admissible trees are those that can be constructed from a given set of grammar rules — in our case, all grammar rules observed in the training data. As the loss function, we use $\Delta(y,\bar{y}) = 100[y \neq \bar{y}]$, and $\Psi_{CFG}(x,y)$ has one feature per grammar rule that counts how often this rule was applied in y. The argmax for prediction can be computed efficiently using a CKY parser. We use the CKY parser implementation[4] of Johnson (1998). For the separation oracle the same CKY parser is used after extending it to also return the second best solution. Again, our SVM-CFG (V3.01) implementation based on SVM^{struct} is available online[3].

For the following experiments, we use all sentences with at most 15 words from the Penn Treebank corpus (Marcus et al, 1993). Restricting the dataset to short sentences is not due to a limitation of SVM^{struct}, but due to the CKY implementation we are using. It becomes very slow for long sentences. Faster parsers that use pruning could easily handle longer sentences as well. After splitting the data into training and test set, we have $n = 9,780$ training examples (i.e., sentences) and $\Psi_{CFG}(x,y)$ has a dimensionality of $N = 154,655$.

[4] Available at `http://www.cog.brown.edu/~mj/Software.htm`

			CPU-Time		# Sep. Oracle		# Support Vec.	
	n	N	1-slack	n-slack	1-slack	n-slack	1-slack	n-slack
`MultiC`	522,911	378	1.05	1180.56	4,183,288	10,981,131	98	334,524
`HMM`	35,531	18,573,781	0.90	177.00	1,314,647	4,476,906	139	83,126
`CFG`	9,780	154,655	2.90	8.52	224,940	479,220	70	12,890

Table 1 Training CPU-time (in hours), number of calls to the separation oracle, and number of support vectors for both the 1-Slack (with caching) and the n-Slack Algorithm. n is the number of training examples and N is the number of features in $\Psi(x,y)$.

5.1 Experiment Setup

Unless noted otherwise, the following parameters are used in the experiments reported below. Both the 1-slack algorithms (*SVM*^{struct} options "`-w 3`" and "`-w 4`" with caching) and the n-slack algorithms (option "`-w 0`") use $\varepsilon = 0.1$ as the stopping criterion (option "`-e 0.1`"). Given the scaling of the loss for multi-class classification and CFG parsing, this corresponds to a precision of approximately 0.1% of the empirical risk for the 1-slack algorithm, and it is slightly higher for the HMM problem. For the n-slack problem it is harder to interpret the meaning of this ε, but we will see in Section 5.7 that it gives solutions of comparable precision. As the value of C, we use the setting that achieves the best prediction performance on the test set when using the full training set ($C = 10{,}000{,}000$ for multi-class classification, $C = 5{,}000$ for HMM sequence tagging, and $C = 20{,}000$ for CFG parsing) (option "`-c`"). As the cache size we use $f = 10$ (option "`-f 10`"). For multi-class classification, margin-rescaling and slack-rescaling are equivalent. For the others two problems we use margin-rescaling (option "`-o 2`"). Whenever possible, runtime comparisons are done on the full training set. All experiments are run on 3.6 MHz Intel Xeon processors with 4GB of main memory under Linux.

5.2 How Fast is the 1-Slack Algorithm Compared to the n*-Slack Algorithm?*

We first examine absolute runtimes of the 1-slack algorithm, and then analyze and explain various aspects of its scaling behavior in the following. Table 1 shows the CPU-time that both the 1-slack and the n-slack algorithm take on the multi-class, sequence tagging, and parsing benchmark problems. For all problems, the 1-slack algorithm is substantially faster, for multi-class and HMM by several orders of magnitude.

The speed-up is largest for the multi-class problem, which has the least expensive separation oracle. Not counting constraints constructed from the cache, less than 1% of the time is spend on the separation oracle for the multi-class problem, while it is 15% for the HMM and 98% for CFG parsing. Therefore, it is interesting to also compare the number of calls to the separation oracle. In all cases, Table 1 shows that

				CPU-Time		# Support Vec.	
	n	N	s	1-slack	SVM^{light}	1-slack	SVM^{light}
`Reuters CCAT`	804,414	47,236	0.16%	58.0	20,075.5	8	230388
`Reuters C11`	804,414	47,236	0.16%	71.3	5,187.4	6	60748
`ArXiv Astro-ph`	62,369	99,757	0.08%	4.4	80.1	9	11318
`Covertype 1`	522,911	54	22.22%	53.4	25,514.3	27	279092
`KDD04 Physics`	150,000	78	38.42%	9.2	1,040.2	13	99123

Table 2 Training CPU time (in seconds) for five binary classification problems comparing the 1-slack algorithm (without caching) with SVM^{light}. n is the number of training examples, N is the number of features, and s is the fraction of non-zero elements of the feature vectors. The SVM^{light} results are quoted from (Joachims, 2006), the 1-slack results are re-run with the latest version of SVM-struct using the same experiment setup as in (Joachims, 2006).

the 1-slack algorithm requires by a factor between 2 and 4 fewer calls, accounting for much of the time saved on the CFG problem.

The most striking difference between the two algorithms lies in the number of support vectors they produce (i.e., the number of dual variables that are non-zero). For the n-slack algorithm, the number of support vectors lie in the tens or hundreds of thousands, while all solutions produced by the 1-slack algorithm have only about 100 support vectors. This means that the working sets that need to be solved in each iteration are orders of magnitude smaller in the 1-slack algorithm, accounting for only 26% of the overall runtime in the multi-class experiment compared to more than 99% for the n-slack algorithm. We will further analyze this in the following.

5.3 *How Fast is the 1-Slack Algorithm Compared to Conventional SVM Training Algorithms?*

Since most work on training algorithms for SVMs was done for binary classification, we compare the 1-slack algorithms against algorithms for the special case of binary classification. While there are training algorithms for linear SVMs that scale linearly with n (e.g., Lagrangian SVM (Mangasarian and Musicant, 2001) (using the $\sum \xi_i^2$ loss), Proximal SVM (Fung and Mangasarian, 2001) (using an L_2 regression loss), and Interior Point Methods (Ferris and Munson, 2003)), they use the Sherman-Morrison-Woodbury formula (or similar matrix factorizations) for inverting the Hessian of the dual. This requires operating on $N \times N$ matrices, which makes them applicable only for problems with small N. The L2-SVM-MFN method (Keerthi and DeCoste, 2005) avoids explicitly representing $N \times N$ matrices using conjugate gradient techniques. While the worst-case cost is still $O(sn\min(n,N))$ per iteration for feature vectors with sparsity s, they observe that their method empirically scales much better. The discussion in (Joachims, 2006) concludes that runtime is comparable to the 1-slack algorithm implemented in SVM-perf. The 1-slack algorithm scales linearly in both n and the sparsity s of the feature vectors, even if the total number N of features is large (Joachims, 2006). Note that it is unclear whether any of the conventional algorithms can be extended to structural SVM training.

The most widely used algorithms for training binary SVMs are decomposition methods like SVM^{light} (Joachims, 1999), SMO (Platt, 1999), and others (Chang and Lin, 2001; Collobert and Bengio, 2001). Taskar et al (2003) extended the SMO algorithm to structured prediction problems based on their polynomial-size reformulation of the n-slack optimization problem OP2 for the special case of decomposable models and decomposable loss functions. In the case of binary classification, their SMO algorithm reduces to a variant of the traditional SMO algorithm, which can be seen as a special case of the SVM^{light} algorithm. We therefore use SVM^{light} as a representative of the class of decomposition methods. Table 2 compares the runtime of the 1-slack algorithm to SVM^{light} on five benchmark problems with varying numbers of features, sparsity, and numbers of training examples. The benchmarks include two text classification problems from the Reuters RCV1 collection[5] (Lewis et al, 2004), a problem of classifying ArXiv abstracts, a binary classifier for class 1 of the `Covertype` dataset[6] of Blackard, Jock & Dean, and the `KDD04 Physics` task from the KDD-Cup 2004 (Caruana et al, 2004). In all cases, the 1-slack algorithm is faster than SVM^{light}, which is highly optimized to binary classification. On large datasets, the difference spans several orders of magnitude.

After the 1-slack algorithm was originally introduced, new stochastic subgradient descent methods were proposed that are competitive in runtime for classification SVMs, especially the PEGASOS algorithm (Shalev-Shwartz et al, 2007). While currently only explored for classification, it should be possible to extend PEGASOS also to structured prediction problems. Unlike exponentiated gradient methods (Bartlett et al, 2004; Globerson et al, 2007), PEGASOS does not require the computation of marginals, which makes it equally easy to apply as cutting-plane methods. However, unlike for our cutting-plane methods where the theory provides a practically effective stopping criterion, it is less clear when to stop primal stochastic subgradient methods. Since they do not maintain a dual program, the duality gap cannot be used to characterize the quality of the solution at termination. Furthermore, there is a questions of how to incorporate caching into stochastic subgradient methods while still maintaining fast convergence. As shown in the following, caching is essential for problems where the separation oracle (or, equivalently, the computation of subgradients) is expensive (e.g. CFG parsing).

5.4 How does Training Time Scale with the Number of Training Examples?

A key question is the scalability of the algorithm for large datasets. While Corollary 1 shows that an upper bound on the training time scales linearly with the number

[5] http://jmlr.csail.mit.edu/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm

[6] http://www.ics.uci.edu/∼mlearn/MLRepository.html

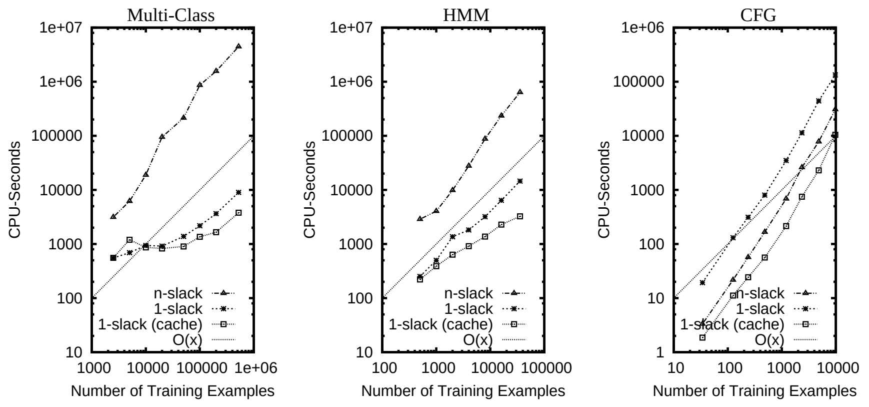


Fig. 1 Training times for multi-class classification (left) HMM part-of-speech tagging (middle) and CFG parsing (right) as a function of *n* for the *n*-slack algorithm, the 1-slack algorithm, and the 1-slack algorithm with caching.

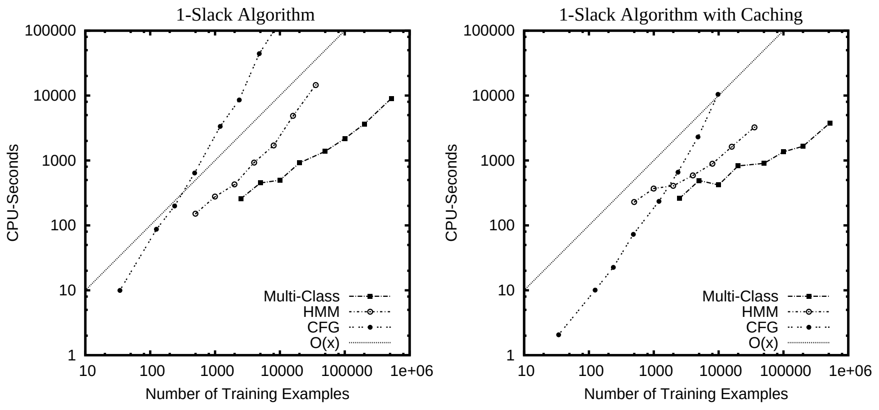


Fig. 2 Training times as a function of *n* using the optimal value of *C* at each training set size for the the 1-slack algorithm (left) and the 1-slack algorithm with caching (right).

of training examples, the actual behavior underneath this bound could potentially be different. Figure 1 shows how training time relates to the number of training examples for the three structural prediction problems. For the multi-class and the HMM problem, training time does indeed scale at most linearly as predicted by Corollary 1, both with and without using the cache. However, the cache helps for larger datasets, and there is a large advantage from using the cache over the whole range for CFG parsing. This is to be expected, given the high cost of the separation oracle in the case of parsing.

As shown in Figure 2, the scaling behavior of the 1-slack algorithm remains largely unchanged even if the regularization parameter *C* is not held constant, but is set to the value that gives optimal prediction performance on the test set for each training set size. The scaling with *C* is analyzed in more detail in Section 5.9.

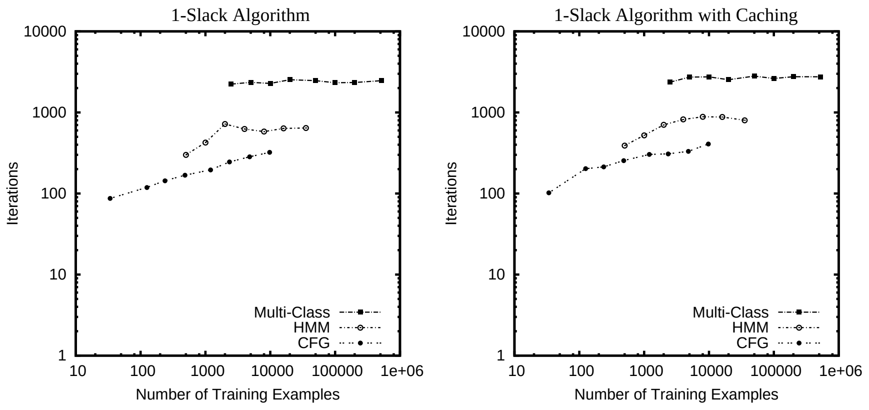


Fig. 3 Number of iterations as a function of n for the the 1-slack algorithm (left) and the 1-slack algorithm with caching (right).

The n-slack algorithm scales super-linearly for all problems, but so does the 1-slack algorithm for CFG parsing. This can be explained as follows. Since the grammar is constructed from all rules observed in the training data, the number of grammar rules grows with the number of training examples. Even from the second-largest to the largest training set, the number of rules in the grammar still grows by almost 70% (3550 rules vs. 5182 rules). This has two effects. First, the separation oracle becomes slower, since its time scales with the number of rules in the grammar. In particular, the time the CFG parser takes to compute a single argmax increases more then six-fold from the smallest to the largest training set. Second, additional rules (in particular unary rules) introduce additional features and allow the construction of larger and larger "wrong" trees $\bar{y}$, which means that $R^2 = \max_{i,\bar{y}} ||\Psi(x_i,y_i) - \Psi(x_i,\bar{y})||^2$ is not constant but grows. Indeed, Figure 3 shows that — consistent with Theorem 5 — the number of iterations of the 1-slack algorithm is roughly constant for multi-class classification and the HMM[7], while it grows slowly for CFG parsing.

Finally, note that in Figure 3 the difference in the number of iterations of the algorithm without caching (left) and with caching (right) is small. Despite the fact that the constraint from the cache is typically not the overall most violated constraint, but only a sufficiently violated constraint, both versions of the algorithm appear to make similar progress in each iteration.

[7] Note that the HMM always considers all possible rules in the regular language, so that there is no growth in the number of rules once all symbols are added.

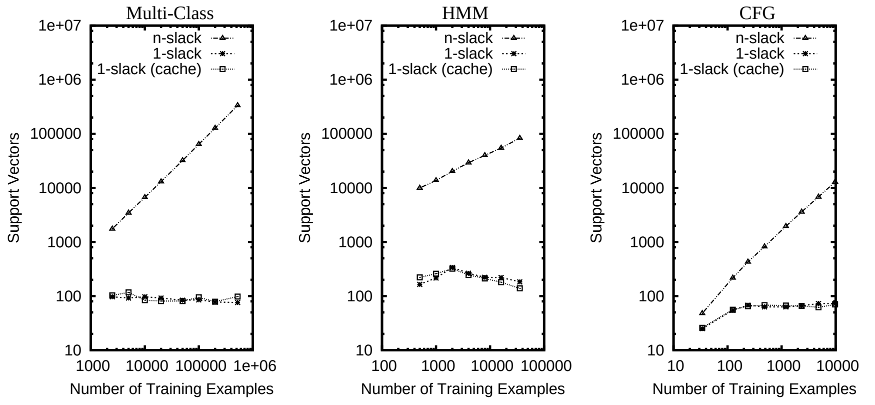


Fig. 4 Number of support vectors for multi-class classification (left) HMM part-of-speech tagging (middle) and CFG parsing (right) as a function of *n* for the *n*-slack algorithm, the 1-slack algorithm, and the 1-slack algorithm with caching.

5.5 *What is the Size of the Working Set?*

As already noted above, the size of the working set and its scaling has a substantial influence on the overall efficiency of the algorithm. In particular, large (and growing) working sets will make it expensive to solve the quadratic programs. While the number of iterations is an upper bound on the working set size for the 1-slack algorithm, the number of support vectors shown in Figure 4 gives a much better idea of its size, since we are removing inactive constraints from the working set. For the 1-slack algorithm, Figure 4 shows that the number of support vectors does not systematically grow with n for any of the problems, making it easy to solve the working set QPs even for large datasets. This is very much in contrast to the n-slack algorithm, where the growing number of support vectors makes each iteration increasingly costly, and is starting to push the limits of what can be kept in main memory.

5.6 *How often is the Separation Oracle Called?*

Next to solving the working set QPs in each iteration, computing the separation oracle is the other major expense in each iteration. We now investigate how the number of calls to the separation oracle scales with n, and how this is influenced by caching. Figure 5 shows that for all algorithms the number of calls scales linearly with n for the multi-class problem and the HMM. It is slightly super-linear for CFG parsing due to the increasing number of interations as discussed above. For all problems and training set sizes, the 1-slack algorithm with caching requires the fewest calls.

The size of the cache has a surprisingly little influence on the reduction of calls to the separation oracle. Figure 6 shows that a cache of size $f = 5$ already provides all

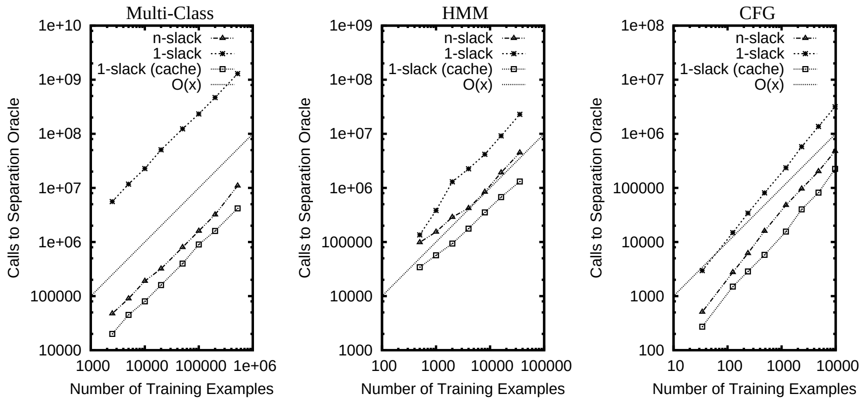


Fig. 5 Number of calls to the separation oracle for multi-class classification (left) HMM part-of-speech tagging (middle) and CFG parsing (right) as a function of *n* for the *n*-slack algorithm, the 1-slack algorithm, and the 1-slack algorithm with caching.

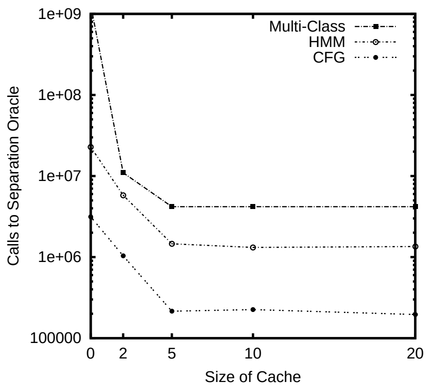
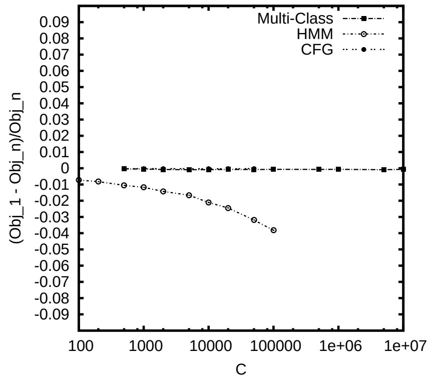


Fig. 6 Number of calls to the separation oracle as a function of cache size for the the 1-slack algorithm.

of the benefits, and that larger cache sizes do not further reduce the number of calls. However, we conjecture that this might be an artifact of our simple least-recently-used caching strategy, and that improved caching methods that selectively call the separation oracle for only a well-chosen subset of the examples will provide further benefits.

5.7 Are the Solutions Different?

Since the stopping criteria are different in the 1-slack and the *n*-slack algorithm, it remains to verify that they do indeed compute a solution of comparable effectiveness. The plot in Figure 7 shows the dual objective value of the 1-slack solution



Task	Measure	1-slack	*n*-slack
`MultiC`	Accuracy	72.33	72.35
`HMM`	Token Accuracy	96.71	96.69
`CFG`	Bracket F_1	70.22	70.09

Fig. 7 Relative difference in dual objective value of the solutions found by the 1-slack algorithm and by the *n*-slack algorithm as a function of *C* at the maximum training set size (left), and test-set prediction performance for the optimal value of *C* (right).

relative to the *n*-slack solution. A value below zero indicates that the *n*-slack solution has a better dual objective value, while a positive value shows by which fraction the 1-slack objective is higher than the *n*-slack objective. For all values of *C* the solutions are very close for the multi-class problem and for CFG parsing, and so are their prediction performances on the test set (see table in Figure 7). This is not surprising, since for both the *n*-slack and the 1-slack formulation the respective ε bound the duality gap by $C\varepsilon$.

For the HMM, however, this $C\varepsilon$ is a substantial fraction of the objective value at the solution, especially for large values of *C*. Since the training data is almost linearly separable for the HMM, $C\varepsilon$ becomes a substantial part of the slack contribution to the objective value. Furthermore, note the different scaling of the HMM loss (i.e., number of misclassified tags in the sentence), which is roughly 5 time smaller than the loss function on the other problems (i.e., 0 to 100 scale). So, an $\varepsilon = 0.1$ on the HMM problem is comparable to an $\varepsilon = 0.5$ on the other problems. Nevertheless, the per-token test accuracy of 96.71% for the 1-slack solution is even slightly better than the 96.69% accuracy of the *n*-slack solution.

5.8 *How does the 1-Slack Algorithm Scale with ε?*

While the scaling with *n* is the most important criterion from a practical perspective, it is also interesting to look at the scaling with ε. Theorem 5 shows that the number of iterations (and therefore the number of calls to the separation oracle) scales $O(\frac{1}{\varepsilon})$ in the worst cast. Figure 8, however, shows that the scaling is much better in practice. In particular, the number of calls to the separation oracle is largely independent of ε and remains constant when caching is used. It seems like the additional iterations can be done almost entirely from the cache.

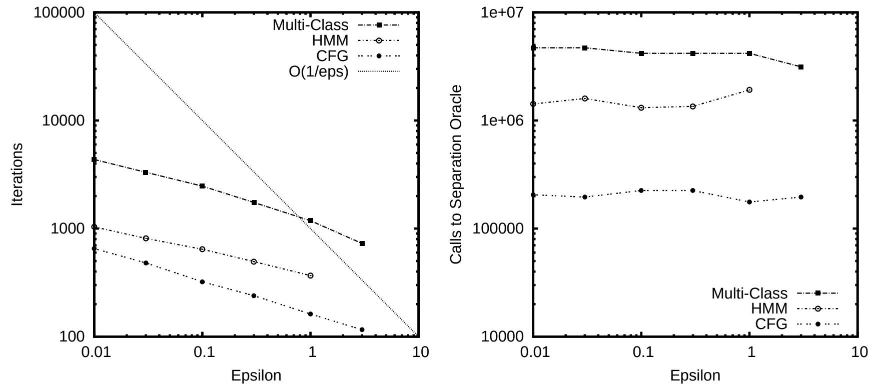


Fig. 8 Number of iterations for the 1-slack algorithm (left) and number of calls to the separation oracle for the 1-slack algorithm with caching (right) as a function of ε at the maximum training set size.

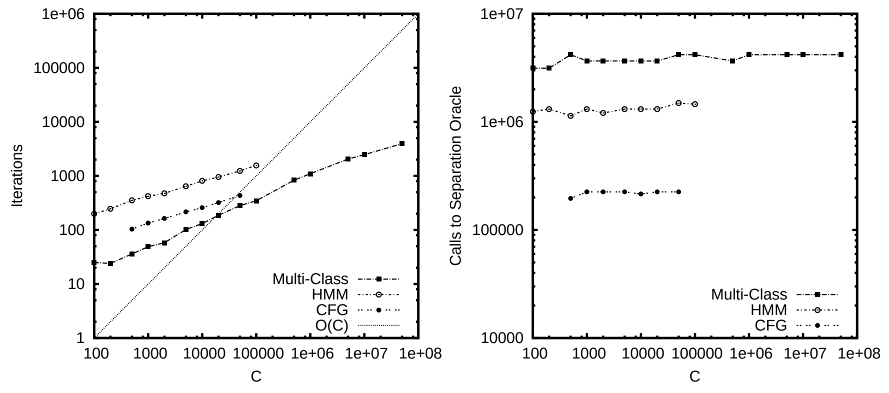


Fig. 9 Number of iterations for the 1-slack algorithm (left) and number of calls to the separation oracle for the 1-slack algorithm with caching (right) as a function of C at the maximum training set size.

5.9 How does the 1-Slack Algorithm Scale with C?

With increasing training set size n, the optimal value of C will typically change (some theoretical results suggest an increase on the order of $\sqrt{n}$). In practice, finding the optimal value of C typically requires training for a large range of C values as part of a cross-validation experiment. It is therefore interesting to know how the algorithm scales with C. While Theorem 5 bounds the number of iterations with $O(C)$, Figure 9 shows that the actual scaling is again much better. The number of iterations increases much more slowly on all problems. Furthermore, as already observed for ε above, the additional iterations are almost entirely based on the cache, so that C has hardly any influence on the number of calls to the separation oracle.

6 Conclusions

We presented a cutting-plane algorithm for training structural SVMs. Unlike existing cutting-plane methods for this problems, the number of constraints that are generated does not depend on the number of training examples n, but only on C and the desired precision ε. Empirically, the new algorithm is substantially faster than existing methods, in particular decomposition methods like SMO and SVM^{light}, and it includes the training algorithm of Joachims (2006) for linear binary classification SVMs as a special case. An implementation of the algorithm is available at `svmlight.joachims.org` with instances for multi-class classification, HMM sequence tagging, CFG parsing, and binary classification.

Acknowledgements We thank Evan Herbst for implementing a prototype of the HMM instance of SVM^{struct}, which was used in some of our preliminary experiments. This work was supported in part through the grant NSF IIS-0713483 from the National Science Foundation and through a gift from Yahoo!.

References

Altun Y, Tsochantaridis I, Hofmann T (2003) Hidden Markov support vector machines. In: International Conference on Machine Learning (ICML), pp 3–10

Anguelov D, Taskar B, Chatalbashev V, Koller D, Gupta D, Heitz G, Ng AY (2005) Discriminative learning of Markov random fields for segmentation of 3D scan data. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, pp 169–176

Bartlett P, Collins M, Taskar B, McAllester D (2004) Exponentiated algorithms for large-margin structured classification. In: Advances in Neural Information Processing Systems (NIPS), pp 305–312

Caruana R, Joachims T, Backstrom L (2004) KDDCup 2004: Results and analysis. ACM SIGKDD Newsletter 6(2):95–108

Chang CC, Lin CJ (2001) LIBSVM: a library for support vector machines. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm

Collins M (2002) Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In: Empirical Methods in Natural Language Processing (EMNLP), pp 1–8

Collins M (2004) Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In: New Developments in Parsing Technology, Kluwer, (paper accompanied invited talk at IWPT 2001)

Collins M, Duffy N (2002) New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In: Annual Meeting of the Association for Computational Linguistics (ACL), pp 263–270

Collobert R, Bengio S (2001) SVMTorch: Support vector machines for large-scale regression problems. Journal of Machine Learning Research (JMLR) 1:143–160

Cortes C, Vapnik VN (1995) Support–vector networks. Machine Learning 20:273–297

Crammer K, Singer Y (2001) On the algorithmic implementation of multiclass kernel-based vector machines. Journal of Machine Learning Research (JMLR) 2:265–292

Crammer K, Singer Y (2003) Ultraconservative online algorithms for multiclass problems. Journal of Machine Learning Research (JMLR) 3:951–991

Ferris M, Munson T (2003) Interior-point methods for massive support vector machines. SIAM Journal of Optimization 13(3):783–804

Finley T, Joachims T (2005) Supervised clustering with support vector machines. In: International Conference on Machine Learning (ICML), pp 217–224

Fukumizu K, Bach F, Jordan M (2004) Dimensionality reduction for supervised learning with reproducing kernel Hilbert spaces. Journal of Machine Learning Research (JMLR) 5:73–99

Fung G, Mangasarian O (2001) Proximal support vector classifiers. In: ACM Conference on Knowledge Discovery and Data Mining (KDD), pp 77–86

Globerson A, Koo TY, Carreras X, Collins M (2007) Exponentiated gradient algorithm for log-linear structured prediction. In: International Conference on Machine Learning (ICML), pp 305–312

Joachims T (1999) Making large-scale SVM learning practical. In: Schölkopf B, Burges C, Smola A (eds) Advances in Kernel Methods - Support Vector Learning, MIT Press, Cambridge, MA, chap 11, pp 169–184

Joachims T (2003) Learning to align sequences: A maximum-margin approach, online manuscript

Joachims T (2005) A support vector method for multivariate performance measures. In: International Conference on Machine Learning (ICML), pp 377–384

Joachims T (2006) Training linear SVMs in linear time. In: ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD), pp 217–226

Johnson M (1998) PCFG models of linguistic tree representations. Computational Linguistics 24(4):613–632

Keerthi S, DeCoste D (2005) A modified finite Newton method for fast solution of large scale linear SVMs. Journal of Machine Learning Research (JMLR) 6:341–361

Keerthi S, Chapelle O, DeCoste D (2006) Building support vector machines with reduced classifier complexity. Journal of Machine Learning Research (JMLR) 7:1493–1515

Kivinen J, Warmuth MK (1997) Exponentiated gradient versus gradient descent for linear predictors. Information and Computation 132(1):1–63

Lafferty J, McCallum A, Pereira F (2001) Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: International Conference on Machine Learning (ICML)

Lewis D, Yang Y, Rose T, Li F (2004) Rcv1: A new benchmark collection for text categorization research. Journal of Machine Learning Research (JMLR) 5:361–397

Mangasarian O, Musicant D (2001) Lagrangian support vector machines. Journal of Machine Learning Research (JMLR) 1:161–177

Marcus M, Santorini B, Marcinkiewicz MA (1993) Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics 19(2):313–330

McDonald R, Crammer K, Pereira F (2005) Online large-margin training of dependency parsers. In: Annual Meeting of the Association for Computational Linguistics (ACL), pp 91–98

Platt J (1999) Fast training of support vector machines using sequential minimal optimization. In: Schölkopf B, Burges C, Smola A (eds) Advances in Kernel Methods - Support Vector Learning, MIT-Press, chap 12

Ratliff ND, Bagnell JA, Zinkevich MA (2007) (Online) subgradient methods for structured prediction. In: Conference on Artificial Intelligence and Statistics (AISTATS)

Shalev-Shwartz S, Singer Y, Srebro N (2007) PEGASOS: Primal Estimated sub-GrAdient SOlver for SVM. In: International Conference on Machine Learning (ICML), ACM, pp 807–814

Smola A, Schölkopf B (2000) Sparse greedy matrix approximation for machine learning. In: International Conference on Machine Learning, pp 911–918

Taskar B, Guestrin C, Koller D (2003) Maximum-margin Markov networks. In: Advances in Neural Information Processing Systems (NIPS)

Taskar B, Klein D, Collins M, Koller D, Manning C (2004) Max-margin parsing. In: Empirical Methods in Natural Language Processing (EMNLP)

Taskar B, Lacoste-Julien S, Jordan MI (2005) Structured prediction via the extragradient method. In: Advances in Neural Information Processing Systems (NIPS)

Teo CH, Smola A, Vishwanathan SV, Le QV (2007) A scalable modular convex solver for regularized risk minimization. In: ACM Conference on Knowledge Discovery and Data Mining (KDD), pp 727–736

Tsochantaridis I, Hofmann T, Joachims T, Altun Y (2004) Support vector machine learning for interdependent and structured output spaces. In: International Conference on Machine Learning (ICML), pp 104–112

Tsochantaridis I, Joachims T, Hofmann T, Altun Y (2005) Large margin methods for structured and interdependent output variables. Journal of Machine Learning Research (JMLR) 6:1453–1484

Vapnik V (1998) Statistical Learning Theory. Wiley, Chichester, GB

Vishwanathan SVN, Schraudolph NN, Schmidt MW, Murphy KP (2006) Accelerated training of conditional random fields with stochastic gradient methods. In: International Conference on Machine Learning (ICML), pp 969–976

Yu CN, Joachims T, Elber R, Pillardy J (2007) Support vector training of protein alignment models. In: Proceeding of the International Conference on Research in Computational Molecular Biology (RECOMB), pp 253–267

Yue Y, Finley T, Radlinski F, Joachims T (2007) A support vector method for optimizing average precision. In: ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp 271–278

Appendix

Lemma 1.

$$\max_{\boldsymbol{\alpha}\geq 0} \; D(\boldsymbol{\alpha}) = \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \Delta(\bar{\mathbf{y}})\alpha_{\bar{\mathbf{y}}} - \frac{1}{2}\sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n}\sum_{\bar{\mathbf{y}}'\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}}\alpha_{\bar{\mathbf{y}}'} H_{MR}(\bar{\mathbf{y}},\bar{\mathbf{y}}') \qquad s.t. \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}} = C$$

and the Wolfe-Dual of the 1*-slack optimization problem OP5 for slack-rescaling is*

$$\max_{\boldsymbol{\alpha}\geq 0} \; D(\boldsymbol{\alpha}) = \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \Delta(\bar{\mathbf{y}})\alpha_{\bar{\mathbf{y}}} - \frac{1}{2}\sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n}\sum_{\bar{\mathbf{y}}'\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}}\alpha_{\bar{\mathbf{y}}'} H_{SR}(\bar{\mathbf{y}},\bar{\mathbf{y}}') \qquad s.t. \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}} = C$$

Proof. The Lagrangian of OP4 is

$$L(\mathbf{w},\xi,\boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi + \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}} \left[\frac{1}{n}\sum_{i=1}^{n} \Delta(y_i,\bar{y}_i) - \xi - \frac{1}{n}\mathbf{w}^T \sum_{i=1}^{n} [\Psi(x_i,y_i) - \Psi(x_i,\bar{y}_i)] \right].$$

Differentiating with respect to $\mathbf{w}$ and setting the derivative to zero gives

$$\mathbf{w} = \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}} \left(\frac{1}{n}\sum_{i=1}^{n} [\Psi(x_i,y_i) - \Psi(x_i,\bar{y}_i)] \right).$$

Similarly, differentiating with respect to ξ and setting the derivative to zero gives

$$\sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}} = C.$$

Plugging $\mathbf{w}$ into the Lagrangian with constraints on $\boldsymbol{\alpha}$ we obtain the dual problem:

$$\begin{aligned}
\max \; & -\frac{1}{2}\sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n}\sum_{\bar{\mathbf{y}}'\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}}\alpha_{\bar{\mathbf{y}}'} \frac{1}{n^2} \left[\sum_{i=1}^{n} [\Psi(x_i,y_i) - \Psi(x_i,\bar{y}_i)] \right]^T \left[\sum_{j=1}^{n} [\Psi(x_j,y_j) - \Psi(x_j,\bar{y}'_j)] \right] \\
& + \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}} \left(\frac{1}{n}\sum_{i=1}^{n} \Delta(y_i,\bar{y}_i) \right) \\
s.t. & \sum_{\bar{\mathbf{y}}\in\mathscr{Y}^n} \alpha_{\bar{\mathbf{y}}} = C \text{ and } \forall \bar{\mathbf{y}} \in \mathscr{Y}^n : \alpha_{\bar{\mathbf{y}}} \geq 0
\end{aligned}$$

The derivation of the dual of OP5 is analogous. □

Lemma 2. *For any unconstrained quadratic program*

$$\max_{\boldsymbol{\alpha}\in\Re^n}\{\Theta(\boldsymbol{\alpha})\}<\infty,\ \ \Theta(\boldsymbol{\alpha})=\boldsymbol{h}^T\boldsymbol{\alpha}-\frac{1}{2}\boldsymbol{\alpha}^T H\boldsymbol{\alpha} \tag{32}$$

with positive semi-definite H, and derivative $\partial\Theta(\boldsymbol{\alpha})=\boldsymbol{h}-H\boldsymbol{\alpha}$, a line search starting at $\boldsymbol{\alpha}$ along an ascent direction $\boldsymbol{\eta}$ with maximum step-size $C>0$ improves the objective by at least

$$\max_{0\le\beta\le C}\{\Theta(\boldsymbol{\alpha}+\beta\boldsymbol{\eta})\}-\Theta(\boldsymbol{\alpha})\ge\frac{1}{2}\min\left\{C,\frac{\nabla\Theta(\boldsymbol{\alpha})^T\boldsymbol{\eta}}{\boldsymbol{\eta}^T H\boldsymbol{\eta}}\right\}\nabla\Theta(\boldsymbol{\alpha})^T\boldsymbol{\eta}. \tag{33}$$

Proof. *For any β and $\boldsymbol{\eta}$, it is easy to verify that*

$$\Theta(\boldsymbol{\alpha}+\beta\boldsymbol{\eta})-\Theta(\boldsymbol{\alpha})=\beta\left[\nabla\Theta(\boldsymbol{\alpha})^T\boldsymbol{\eta}-\frac{1}{2}\beta\boldsymbol{\eta}^T H\boldsymbol{\eta}\right]. \tag{34}$$

Maximizing this expression with respect to an unconstrained β by setting the derivative to zero, the solution β^* is

$$\beta^*=\frac{\nabla\Theta(\boldsymbol{\alpha})^T\boldsymbol{\eta}}{\boldsymbol{\eta}^T H\boldsymbol{\eta}}. \tag{35}$$

Note that $\boldsymbol{\eta}^T H\boldsymbol{\eta}$ is non-negative, since H is positive semi-definite. Furthermore, $\boldsymbol{\eta}^T H\boldsymbol{\eta}\neq 0$, since otherwise $\boldsymbol{\eta}$ being an ascent direction would contradict that $\max_{\boldsymbol{\alpha}\in\Re^n}\{\Theta(\boldsymbol{\alpha})\}<\infty$. Plugging β^* into (34) shows that

$$\max_{\beta\in\Re}\{\Theta(\boldsymbol{\alpha}+\beta\boldsymbol{\eta})\}-\Theta(\boldsymbol{\alpha})=\frac{1}{2}\frac{(\nabla\Theta(\boldsymbol{\alpha})^T\boldsymbol{\eta})^2}{\boldsymbol{\eta}^T H\boldsymbol{\eta}}. \tag{36}$$

It remains to check whether the unconstrained solution β^* fulfills the constraints $0\le\beta^*\le C$. Since $\boldsymbol{\eta}$ is an ascent direction, β^* is always non-negative. But one needs to consider the case that $\beta^*>C$, which happens when $\nabla\Theta(\boldsymbol{\alpha})^T\boldsymbol{\eta}>C\boldsymbol{\eta}^T H\boldsymbol{\eta}$. In that case, the constrained optimum is at $\beta=C$ due to convexity. Plugging C into (34) shows that

$$\max_{\beta\in\Re}\{\Theta(\boldsymbol{\alpha}+\beta\boldsymbol{\eta})\}-\Theta(\boldsymbol{\alpha})=C\nabla\Theta(\boldsymbol{\alpha})^T\boldsymbol{\eta}-\frac{1}{2}C^2\boldsymbol{\eta}^T H\boldsymbol{\eta} \tag{37}$$

$$\ge\frac{1}{2}C\nabla\Theta(\boldsymbol{\alpha})^T\boldsymbol{\eta}. \tag{38}$$

The inequality follow from $C\le\frac{\nabla\Theta(\boldsymbol{\alpha})^T\boldsymbol{\eta}}{\boldsymbol{\eta}^T H\boldsymbol{\eta}}$. □