Training Linear SVMs in Linear Time

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

ABSTRACT

Linear Support Vector Machines (SVMs) have become one of the most prominent machine learning techniques for high-dimensional sparse data commonly encountered in applications like text classification, word-sense disambiguation, and drug design. These applications involve a large number of examples n as well as a large number of features N, while each example has only $s << N$ non-zero features. This paper presents a Cutting-Plane Algorithm for training linear SVMs that provably has training time $O(sn)$ for classification problems and $O(sn\log(n))$ for ordinal regression problems. The algorithm is based on an alternative, but equivalent formulation of the SVM optimization problem. Empirically, the Cutting-Plane Algorithm is several orders of magnitude faster than decomposition methods like SVM-Light for large datasets.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance, Experimentation

Keywords

Support Vector Machines (SVM), Training Algorithms, Ordinal Regression, Large-Scale Problems, ROC-Area

1. INTRODUCTION

Many applications of machine learning deal with problems where both the number of features N as well as the number of examples n is large (in the millions). Examples of such problems can be found in text classification, word-sense disambiguation, and drug design. While problems of such size seem daunting at first glance, the examples mentioned above have extremely sparse feature vectors, which gives hope that these problems can be handled efficiently.

Linear Support Vector Machines (SVMs) are among the most prominent machine learning techniques for such high-dimensional and sparse data. On text classification problems, for example, linear SVMs provide state-of-the-art prediction accuracy [10, 5, 17]. While conventional training methods for linear SVMs, in particular decomposition methods like SVM-Light [11], SMO [19], LIBSVM [2], and SVM-Torch [3] handle problems with a large number of features N quite efficiently, their super-linear scaling behavior with n [11, 19, 9] makes their use inefficient or even intractable on large datasets. On the other hand, while there are training methods that scale linear in n (e.g. [18, 7, 6, 15]), such methods empirically (or at least in the worst case) scale quadratically with the number of features N.

Even more difficult is the current situation for training linear Ordinal Regression SVMs (OR-SVMs) [8]. In Herbrich et al.'s formulation, an ordinal regression SVM over n examples is solved by translating it into a classification SVM with $O(n^2)$ examples, which obviously makes scaling with n even worse than for straightforward classification SVMs. Nevertheless, OR-SVM are very interesting even beyond actual ordinal regression problems like those in information retrieval. When applied to problems with only two ranks, OR-SVMs are know to directly optimize the ROC-Area of the classification rule [20, 14]. This is a desirable criterion to optimize in many applications.

In this paper, we propose the first general training algorithm for linear SVMs that provably scales $O(s\,n)$ for classification and $O(s\,n\log(n))$ for ordinal regression, where s is the average number of non-zero features. Obviously, this scaling is very attractive for high-dimensional and sparse data. The algorithm is based on an alternative, yet equivalent formulation of the SVM training problem. Compared to existing methods, the algorithm has several advantages. First, it is very simple and easy to implement. Second, it is several orders of magnitude faster than existing decomposition methods on large classification problems. On a text classification problem with 800,000 examples and 47,000 features, the new algorithm is roughly 100 times faster than SVM-Light. Third, the algorithm has a meaningful stopping criterion that directly relates to training error. This avoids wasting time on solving the optimization problem to a higher precision than necessary. And, fourth, the algorithm can handle ordinal regression problems with hundred-thousands of examples with ease, while existing methods become intractable with only a few thousand examples.

2. STRUCTURAL SVMS

We first introduce the formulation of the SVM optimization problem that provides the basis of our algorithm, both for classification and for ordinal regression SVMs. Both for-

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.

mulations are derived from Structural SVMs [24, 14] previously used for predicting structured outputs and optimizing to multivariate performance measures. For each alternative formulation, we will show that it is equivalent to the respective conventional SVM optimization problem.

2.1 Classification

For a given training set $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)$ with $\mathbf{x}_i \in \Re^N$ and $y_i \in \{-1, +1\}$, training a binary classification SVM means solving the following optimization problem. For simplicity of the following theoretical results, we focus on classification rules $h_{\mathbf{w}}(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x}+b)$ with $b = 0$. A non-zero b can easily be modeled by adding an additional feature of constant value to each $\mathbf{x}$ (see e.g. [18]).

OP 1. (Classification SVM (primal))

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

Note that we adopted the formulation of [22, 21] where $\sum \xi_i$ is divided by n to better capture how C scales with the training set size. Most training algorithms solve either OP1 or its dual (see [21] for the dual).

The algorithm we explore in the following considers a different optimization problem, which was proposed for training SVMs to predict structured outputs [24] and to optimize multivariate performance measures like F_1-Score or the Precision/Recall Break-Even Point [14]. The following is a specialization of this formulation for the case of error rate, and we will refer to it as the "structural" formulation.

OP 2. (Structural Classification SVM (primal))

$$\min_{\mathbf{w},\xi\geq 0} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\,\xi$$
$$s.t. \quad \forall \mathbf{c} \in \{0,1\}^n: \; \frac{1}{n}\mathbf{w}^T\sum_{i=1}^{n} c_i y_i \mathbf{x}_i \geq \frac{1}{n}\sum_{i=1}^{n} c_i - \xi$$

While OP2 has 2^n constraints, one for each possible vector $\mathbf{c} = (c_1, ..., c_n) \in \{0, 1\}^n$, it has only one slack variable ξ that is shared across all constraints. Each constraint in this structural formulation corresponds to the sum of a subset of constraints from OP1, and the c_i select the subset. $\frac{1}{n}\sum_{i=1}^{n} c_i$ can be seen as the maximum fraction of training errors possible over each subset, and ξ is an upper bound on the fraction of training errors made by $h_{\mathbf{w}}$. Interestingly, OP1 and OP2 are equivalent in the following sense.

Theorem 1. *Any solution $\mathbf{w}^*$ of OP2 is also a solution of OP1 (and vice versa), with $\xi^* = \frac{1}{n}\sum_{i=1}^{n}\xi_i^*$.*

Proof. *Adapting the proof from [14], we will show that both optimization problems have the same objective value and an equivalent set of constraints. In particular, for every $\mathbf{w}$ the smallest feasible ξ and $\sum_i \xi_i$ are related as $\xi = \frac{1}{n}\sum_i \xi_i$.*

For a given $\mathbf{w}$, the ξ_i in OP1 can be optimized individually, and the optimum is achieved for $\xi_i = max\{0, 1-y_i(\mathbf{w}^T\mathbf{x}_i)\}$. For OP2, the optimal ξ for a given $\mathbf{w}$ is

$$\xi = \max_{\mathbf{c}\in\{0,1\}^n}\left\{\frac{1}{n}\sum_{i=1}^{n} c_i - \frac{1}{n}\sum_{i=1}^{n} c_i y_i(\mathbf{w}^T\mathbf{x}_i)\right\}$$

Since the function is linear in c_i, each c_i can be optimized independently.

$$\begin{aligned}\min C\xi &= \sum_{i=1}^{n}\max_{c_i\in\{0,1\}}\left\{\frac{1}{n}c_i - \frac{1}{n}c_i y_i(\mathbf{w}^T\mathbf{x}_i)\right\} \\ &= \sum_{i=1}^{n}\max\left\{0, \frac{1}{n} - \frac{1}{n}y_i(\mathbf{w}^T\mathbf{x}_i)\right\} = \min \frac{C}{n}\sum_{i=1}^{n}\xi_i\end{aligned}$$

Therefore, the objective functions of both optimization problems are equal for any $\mathbf{w}$ given the optimal ξ and ξ_i, and consequently so are their optima. $\square$

The theorem shows that it is possible to solve OP2 instead of OP1 to find the same soft-margin hyperplane. While OP2 does not appear particularly attractive at first glance, we will show in Section 3 that its Wolfe Dual has desirable sparseness properties. Before we show that a similar formulation also exists for Ordinal Regression SVMs [8], we first state the Wolfe dual of OP2, since it will be referred to later. Denote with $\mathbf{x_c}$ the sum $\frac{1}{n}\sum_{i=1}^{n} c_i y_i \mathbf{x}_i$ and with $||\mathbf{c}||_1$ the L1-norm of $\mathbf{c}$ (i.e. the number ones in $\mathbf{c}$ for binary $\mathbf{c}$).

OP 3. (Structural Classification SVM (dual))

$$\max_{\alpha\geq 0} \quad \sum_{\mathbf{c}\in\{0,1\}^n}\frac{||\mathbf{c}||_1}{n}\alpha_{\mathbf{c}} - \frac{1}{2}\sum_{\mathbf{c}\in\{0,1\}^n}\sum_{\mathbf{c}'\in\{0,1\}^n}\alpha_{\mathbf{c}}\alpha_{\mathbf{c}'}\mathbf{x}_{\mathbf{c}}^T\mathbf{x}_{\mathbf{c}'}$$
$$s.t. \quad \sum_{\mathbf{c}\in\{0,1\}^n}\alpha_{\mathbf{c}} \leq C$$

2.2 Ordinal Regression

In ordinal regression, the label y_i of an example $(\mathbf{x}_i, y_i)$ indicates a rank instead of a nominal class. Without loss of generality, let $y_i \in \{1, ..., R\}$ so that the values $1, ..., R$ are related on an ordinal scale. In the formulation of Herbrich et al. [8], the goal is to learn a function $h(\mathbf{x})$ so that for any pair of examples $(\mathbf{x}_i, y_i)$ and $(\mathbf{x}_j, y_j)$ it holds that

$$h(\mathbf{x}_i) > h(\mathbf{x}_j) \Longleftrightarrow y_i > y_j.$$

Given a training set $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)$ with $\mathbf{x}_i \in \Re^N$ and $y_i \in \{1, ..., R\}$, Herbrich et al. formulate the following ordinal regression SVM (OR-SVM). Denote with $\mathcal{P}$ the set of pairs (i, j) for which example i has a higher rank than example j, i.e. $\mathcal{P} = \{(i, j) : y_i > y_j\}$, and let $m = |\mathcal{P}|$.

OP 4. (Ordinal Regression SVM (primal))

$$\min_{\mathbf{w},\xi_{ij}\geq 0} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{m}\sum_{(i,j)\in\mathcal{P}}\xi_{ij}$$
$$s.t. \quad \forall (i,j) \in \mathcal{P}: \; (\mathbf{w}^T\mathbf{x}_i) \geq (\mathbf{w}^T\mathbf{x}_j) + 1 - \xi_{ij}$$

Intuitively, this formulation finds a large-margin linear function $h(\mathbf{x})$ that minimizes the number of pairs of training examples that are swapped w.r.t. their desired order. Like for classification SVMs, OP4 is a convex quadratic program. Ordinal regression problems have applications in learning retrieval functions for search engines [12]. Furthermore, if the labels y take only two values, OP4 optimizes the ROC-Area of the classification rule [20, 14].

In general, OP4 has $m \in O(n^2)$ constraints and slack variables. While this problem can be brought into the same form as OP1 by rewriting the constraints as $\mathbf{w}^T(\mathbf{x}_i - \mathbf{x}_j) \geq 1 - \xi_{ij}$, even relatively small training sets with only a few thousand examples are already intractable for conventional training methods. So far, researchers have tried to cut down on the number of constraints with various heuristics [20] which, however, cannot guarantee that the computed solution is optimal.

We will now derive a similar structural formulation of the ordinal regression SVM as we have already done for the binary classification SVM.

OP 5. (Structural Ord. Regr. SVM (primal))

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

$$s.t. \quad \forall(i,j)\in\mathcal{P}\ \forall c_{ij}\in\{0,1\}: \frac{1}{m}\mathbf{w}^T\sum_{i=1}^{m} c_{ij}(\mathbf{x}_i - \mathbf{x}_j) \geq \frac{1}{m}\sum_{i=1}^{m} c_{ij} - \xi$$

Like for classification, the structural formulation has $O(2^n)$ constraints, but only a single slack variable ξ. Analogous to the classification SVM, the following theorem establishes that both formulations of the OR-SVM have equivalent solutions.

Theorem 2. *Any solution $\mathbf{w}^*$ of OP5 is also a solution of OP4 (and vice versa), with $\xi^* = \frac{1}{m}\sum_{(i,j)\in\mathcal{P}} \xi^*_{ij}$.*

Proof. *As mentioned above, the constraints in OP4 can be rewritten as $y_{ij}(\mathbf{w}^T(\mathbf{x}_i - \mathbf{x}_j)) \geq 1 - \xi_{ij}$ with all y_{ij} set to 1. Theorem 1 applies immediately after substituting $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$, since OP4 has the same form as OP1, and OP5 has the same form as OP2.* $\square$

3. CUTTING-PLANE ALGORITHM

While the alternative formulations of the classification and the ordinal regression SVM from above have an exponential number of constraints, they are very convenient in several ways.

First, there is a single slack variable ξ that measures training loss, and there is a direct correspondence between ξ and the (in)feasibility of the set of constraints. In particular, if we have a point $(\mathbf{w},\xi)$ which fulfills all constraints up to precision ϵ, then $(\mathbf{w},\xi+\epsilon)$ is feasible. So, the approximation accuracy ϵ of an approximate solution to OP2 or OP5 is directly related to training loss, which provides an intuitive precision criterion.

Second, OP2 and OP5 are special cases of Structural SVMs [24, 14]. As we will detail below, for this type of formulation it is possible to prove bounds on the sparsity of an ϵ-approximate solution of the Wolfe dual. In particular, we will show that the sparsity is independent of the training set size n, and that simple Cutting-Plane Algorithms [16] find an ϵ-approximate solution in a constant number of iterations for both OP2 or OP5.

3.1 Classification

Algorithm 1 is our adaptation of the Cutting-Plane Algorithm for the Classification SVM optimization problem OP2. It is an adaptation of the Structural SVM training algorithm [24, 14]. The algorithm iteratively constructs a sufficient subset $\mathcal{W}$ of the set of constraints in OP2. The algorithm starts with an empty set of constraints $\mathcal{W}$. In each iteration, it first computes the optimum over the current working set $\mathcal{W}$ (i.e. $\mathbf{w} = 0$ and $\xi = 0$ in the first iteration) in Line 4. In Lines 5-7 it then finds the most violated constraint in OP2 and adds it to the working set $\mathcal{W}$ in Line 8.

Algorithm 1 for training Classification SVMs via OP2.

1: Input: $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$, C, ϵ
2: $\mathcal{W} \leftarrow \emptyset$
3: **repeat**
4: $\quad(\mathbf{w},\xi) \leftarrow \mathrm{argmin}_{\mathbf{w},\xi\geq 0}\ \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi$
$\qquad\qquad$ s.t. $\forall\mathbf{c}\in\mathcal{W}$: $\frac{1}{n}\mathbf{w}^T\sum_{i=1}^{n} c_i y_i \mathbf{x}_i \geq \frac{1}{n}\sum_{i=1}^{n} c_i - \xi$
5: $\quad$**for** i=1,...,n **do**
6: $\quad\quad c_i \leftarrow \begin{cases} 1 & y_i(\mathbf{w}^T\mathbf{x}_i) < 1 \\ 0 & \textit{otherwise} \end{cases}$
7: $\quad$**end for**
8: $\quad\mathcal{W} \leftarrow \mathcal{W} \cup \{\mathbf{c}\}$
9: **until** $\frac{1}{n}\sum_{i=1}^{n} c_i - \frac{1}{n}\sum_{i=1}^{n} c_i y_i(\mathbf{w}^T\mathbf{x}_i)\} \leq \xi + \epsilon$
10: return($\mathbf{w}$,ξ)

Note that this assignment to $(c_1, ..., c_n) = \mathbf{c}$ corresponds to the constraint in OP2 that requires the largest ξ to make it feasible given the current $\mathbf{w}$, i.e.

$$\mathbf{c} = \underset{\mathbf{c}\in\{0,1\}^n}{\mathrm{argmax}} \left\{ \frac{1}{n}\sum_{i=1}^{n} c_i - \frac{1}{n}\sum_{i=1}^{n} c_i y_i(\mathbf{w}^T\mathbf{x}_i) \right\}.$$

The algorithm then continues in Line 4 by optimizing over the new working set, unless the most violated constraint is not violated by more than the desired precision ϵ.

In the following, we will analyze the correctness and the time complexity of the algorithm. We will show that the algorithm always terminates after a polynomial number of iterations that does not depend on the size n of the training set. Regarding its correctness, the following theorem characterizes the accuracy of the solution computed by Algorithm 1.

Theorem 3. (Correctness of Algorithm 1) *For any training sample $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$ and any $\epsilon > 0$, if $(\mathbf{w}^*, \xi^*)$ is the optimal solution of OP2, then Algorithm 1 returns a point $(\mathbf{w},\xi)$ that has a better objective value than $(\mathbf{w}^*, \xi^*)$, and for which $(\mathbf{w}, \xi+\epsilon)$ is feasible in OP2.*

Proof. *We first verify that Lines 5-7 compute the vector $\mathbf{c} \in \{0,1\}^n$ that maximizes*

$$\xi' = \max_{\mathbf{c}\in\{0,1\}^n} \left\{ \frac{1}{n}\sum_{i=1}^{n} c_i - \frac{1}{n}\sum_{i=1}^{n} c_i y_i(\mathbf{w}^T\mathbf{x}_i) \right\}.$$

ξ' is the minimum value needed to fulfill all constraints in OP2 for the current $\mathbf{w}$. Since the function is linear in c_i, each c_i can be maximized independently.

$$\xi' = \frac{1}{n}\sum_{i=1}^{n} \max_{c_i\in\{0,1\}} \{c_i - c_i y_i(\mathbf{w}^T\mathbf{x}_i)\} = \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i)\}$$

This 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 1 is the solution on a subset of the constraints from OP2, it holds that $\frac{1}{2}\mathbf{w}^{*T}\mathbf{w}^* + C\xi^* \geq \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi$. $\square$

Using a stopping criterion based on the accuracy of the training loss 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 error rate of the best hyperplane. 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 decomposition methods, where it is usually defined based on the accuracy of the Kuhn-Tucker Conditions of the dual (see e.g. [11]). Solving OP2 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 Algorithm 1. It is easy to see that each iteration of the algorithm takes polynomial time, and that time scales linearly with n and s. We then show that the number of iterations until convergence is bounded, and that this upper bound is independent of n.

LEMMA 1. *Each iteration of Algorithm 1 takes time $O(sn)$ for a constant working set size $|\mathcal{W}|$.*

PROOF. *Each dot-product in Lines 6 and 9 takes time $O(s)$ when using sparse vector algebra, and n dot-products are computed in each line. Instead of solving the primal quadratic program, one can instead solve the dual OP3 in Line 4. Setting up the dual over $\mathcal{W}$ in Line 4 is dominated by computing the $O(|\mathcal{W}|^2)$ elements of the Hessian, which can be done in $O(|\mathcal{W}|^2 sn)$ after first computing $\frac{1}{n}\sum_{i=1}^{n} c_i y_i \mathbf{x}_i$ for each constraint in $\mathcal{W}$. Note that $N \leq sn$. The time for solving the dual is then independent of n and s. This leads to an overall time complexity of $O(sn)$ per iteration.* $\square$

LEMMA 2. *For any $\epsilon > 0$, $C > 0$, and any training sample $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$, Algorithms 1 and 2 terminate after at most*

$$\max\left\{\frac{2}{\epsilon}, \frac{8CR^2}{\epsilon^2}\right\} \qquad (1)$$

iterations. $R = \max_i ||\mathbf{x}_i||$ for Algorithm 1 and for Algorithm 2 it is $R = 2\max_i ||\mathbf{x}_i||$.

PROOF. *Following the proof scheme in [24, 13], we will show that adding each new constraint to $\mathcal{W}$ 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 OP2 is upper bounded by C (since $\mathbf{w} = 0$ and $\xi = 1$ 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 Lines 9 and 26 respectively) to $\mathcal{W}$ is characterized by Proposition 17 in [24]. A lower bound on the increase is*

$$\min\left\{\frac{C\epsilon}{2}, \frac{\epsilon^2}{8Q^2}\right\}$$

where Q is an upper bound on the L_2-norm of the coefficient vectors in the constraints. For OP2

$$Q = \max_{\mathbf{c}\in\{0,1\}^n} \left|\left|\frac{1}{n}\sum_{i=1}^{n} c_i y_i \mathbf{x}_i\right|\right| \leq \frac{1}{n}\sum_{i=1}^{n} c_i \max_i ||\mathbf{x}_i|| \leq R$$

in the case of Algorithm 1 and for OP5

$$Q = \max_{\mathbf{c}\in\{0,1\}^m} \left|\left|\frac{1}{m}\sum_{(i,j)\in\mathcal{P}} c_{ij}(\mathbf{x}_i - \mathbf{x}_j)\right|\right| \leq \frac{1}{m}\sum_{(i,j)\in\mathcal{P}} c_{ij} 2\max_i ||\mathbf{x}_i|| \leq R$$

in the case of Algorithm 2. Due to this constant increase of the objective value in each iteration, either algorithm can add at most $\max\left\{\frac{2}{\epsilon}, \frac{8CR^2}{\epsilon^2}\right\}$ constraints before the objective value exceeds C, which is an upper bound on the objective value at the solution of OP2 and OP5. $\square$

Note that the formulation of OP2 with the scaled $\frac{C}{n}$ instead of C in the objective is essential for this lemma. We will empirically evaluate the adequacy of this scaling in Section 4. Putting everything together leads to the following bound on the time complexity of Algorithm 1.

THEOREM 4. (TIME COMPLEXITY OF ALGORITHM 1) *For any distribution $P(X, Y)$ that generates feature vectors of bounded L_2-norm $||\mathbf{x}||$ and any fixed value of $C > 0$ and $\epsilon > 0$, Algorithm 1 has time complexity $O(sn)$ for any training sample of size n and sparsity s.*

PROOF. *Lemma 2 bounds the number of iterations (and therefore the maximum working set size $|\mathcal{W}|$) to a constant that is independent of n and s. Each iteration has time complexity $O(sn)$ as established by Lemma 1.* $\square$

To our knowledge, Algorithm 1 has the best scaling behavior of all known training algorithms for linear SVMs. Decomposition methods like SVM-Light [11], SMO [19], LIBSVM [2], and SVMTorch [3] handle sparse problems with a large number of features N quite efficiently. However, their super-linear scaling behavior with n [11, 19, 9] makes them inefficient or even intractable on large datasets. We will compare our algorithm against SVM-Light as a representative decomposition methods.

Other methods sacrifice the statistical robustness [22] of the $\sum \xi_i$ loss in the objective for the numerically more convenient $\sum \xi_i^2$ loss. With additional restrictions on how the data is normalized, Core Vector Machines [23] are shown to scale linear in n. However, the restrictions make the method inapplicable to many datasets. Generally applicable are Lagrangian SVM [18] (using the $\sum \xi_i^2$ loss), Proximal SVM [7] (using an L_2 regression loss), and Interior Point Methods [6]. While these method scale linearly with n, they use the Sherman-Morrison-Woodbury formula 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. As a representative of this group of methods, we will compare against the Lagrangian SVM in Section 4.

The recent L2-SVM-MFN method [15] avoids explicitly representing $N \times N$ matrices using conjugent gradient techniques. While the worst-case cost is still $O(sn\min(n,N))$ per iteration, they observe that their method empirically scales better. We will compare against this method as well.

3.2 Ordinal Regression

Algorithm 2 solves the ordinal regression SVM in the form of OP5 and has a structure that is very similar to Algorithm 1. It is a generalization of the algorithm for optimizing ROC-Area in [14] and similar to the algorithm independently developed in [4]. The key difference to Algorithm 1 lies in computing the most violated constraint of OP5

$$\mathbf{c}' = \underset{\mathbf{c}\in\{0,1\}^m}{\operatorname{argmax}} \left\{\frac{1}{m}\sum_{(i,j)\in\mathcal{P}} c_{ij} - \frac{1}{m}\sum_{(i,j)\in\mathcal{P}} c_{ij}\mathbf{w}^T(\mathbf{x}_i - \mathbf{x}_j)\right\}$$

without enumerating all $m \in O(n^2)$ constraints from OP4. To avoid $O(n^2)$ cost, Algorithm 2 makes use of a condensed

Algorithm 2 for training Ord. Regr. SVMs via OP5.

1: Input: $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$, C, ϵ
2: $\mathcal{W} \leftarrow \emptyset$
3: **repeat**
4: $\quad (\mathbf{w}, \xi) \leftarrow \operatorname{argmin}_{\mathbf{w},\xi \geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi$
$\qquad\qquad$ s.t. $\forall(\mathbf{c}^+,\mathbf{c}^-)\in\mathcal{W}$: $\frac{1}{m}\mathbf{w}^T\sum_{i=1}^{n}(c_i^+ - c_i^-)\mathbf{x}_i \geq \frac{1}{2m}\sum_{i=1}^{n}(c_i^+ + c_i^-) - \xi$
5: $\quad$ sort S by decreasing $\mathbf{w}^T\mathbf{x}_i$
6: $\quad \mathbf{c}^+ \leftarrow 0; \mathbf{c}^- \leftarrow 0$
7: $\quad n_r \leftarrow$ number of examples with $y_i = r$
8: $\quad$ **for** $r = 2, ..., R$ **do**
9: $\quad\quad i \leftarrow 1; j \leftarrow 1; a \leftarrow 0; b \leftarrow 0$
10: $\quad\quad$ **while** $i \leq n$ **do**
11: $\quad\quad\quad$ **if** $y_i = r$ **then**
12: $\quad\quad\quad\quad$ **while** $(j \leq n) \wedge (\mathbf{w}^T\mathbf{x}_i - \mathbf{w}^T\mathbf{x}_j < 1)$ **do**
13: $\quad\quad\quad\quad\quad$ **if** $y_j < r$ **then**
14: $\quad\quad\quad\quad\quad\quad b++;\ c_j^- \leftarrow c_j^- + (n_r - a + 1)$
15: $\quad\quad\quad\quad\quad$ **end if**
16: $\quad\quad\quad\quad\quad j++$
17: $\quad\quad\quad\quad$ **end while**
18: $\quad\quad\quad\quad a++;\ c_i^+ \leftarrow c_i^+ + b$
19: $\quad\quad\quad$ **end if**
20: $\quad\quad\quad i++$
21: $\quad\quad$ **end while**
22: $\quad$ **end for**
23: $\quad \mathcal{W} \leftarrow \mathcal{W} \cup \{(\mathbf{c}^+, \mathbf{c}^-)\}$
24: **until** $\frac{1}{2m}\sum_{i=1}^{n}(c_i^+ + c_i^-) - \frac{1}{m}\sum_{i=1}^{n}(c_i^+ - c_i^-)(w^T\mathbf{x}_i) \leq \xi + \epsilon$
25: return($\mathbf{w}$,ξ)

representation of the constraints as follows. While the left-hand side of the linear constraints in OP4 contains a sum over m vectors of differences $(\mathbf{x}_i - \mathbf{x}_j)$, most individual vectors $\mathbf{x}_i$ are added and subtracted multiple time. With proper coefficients c_i^+ and c_j^-, each constraint can be rewritten as a sum of n vectors

$$\frac{1}{m}\mathbf{w}^T\sum_{i=1}^{n}(c_i^+ - c_i^-)\mathbf{x}_i \geq \frac{1}{2m}\sum_{i=1}^{n}(c_i^+ + c_i^-) - \xi$$

where c_i^+ is the number of times $\mathbf{x}_i$ occurs with positive sign (i.e. $c_{ij} = 1$) and c_i^- is the number of times $\mathbf{x}_i$ occurs with negative sign (i.e. $c_{ji} = 1$). If $\mathbf{c}^+$ and $\mathbf{c}^-$ are known, each constraint can be evaluated in time $O(sn)$ instead of $O(sm)$. Furthermore, the right hand side of each constraint can be computed from $\mathbf{c}^+$ and $\mathbf{c}^-$ in time $O(n)$ instead of $O(m)$, since $\frac{1}{m}\sum_{i=1}^{m} c_{ij} = \frac{1}{2m}\sum_{i=1}^{n}(c_i^+ + c_i^-)$. The following theorem shows that Algorithm 2 computes the coefficient vectors $\mathbf{c}^+$ and $\mathbf{c}^-$ of the most violated constraint, and therefore converges to the optimal solution in the same sense as Algorithm 1.

Theorem 5. (Correctness of Algorithm 2) *For any training sample $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$ and any $\epsilon > 0$, if $(\mathbf{w}^*, \xi^*)$ is the optimal solution of OP5, then Algorithm 2 returns $(\mathbf{w}, \xi)$ that have a better objective value than $(\mathbf{w}^*, \xi^*)$, and for which $(\mathbf{w}, \xi + \epsilon)$ is feasible in OP5.*

Proof. *Analogous to the proof of Theorem 3,*

$$\mathbf{c}' = \operatorname*{argmax}_{\mathbf{c}\in\{0,1\}^m}\left\{\frac{1}{m}\sum_{(i,j)\in\mathcal{P}} c_{ij} - \frac{1}{m}\sum_{(i,j)\in\mathcal{P}} c_{ij}\mathbf{w}^T(\mathbf{x}_i - \mathbf{x}_j)\right\}$$

is reached for

$$c_{ij} \leftarrow \begin{cases} 1 & (\mathbf{w}^T\mathbf{x}_i) - (\mathbf{w}^T\mathbf{x}_j) < 1 \\ 0 & \textit{otherwise} \end{cases}.$$

This means that the number of times $\mathbf{x}_i$ enters with positive and negative sign is

$$c_i^+ = |\{j : (y_i > y_j) \wedge ((\mathbf{w}^T\mathbf{x}_i) - (\mathbf{w}^T\mathbf{x}_j) < 1)\}|,$$
$$c_i^- = |\{j : (y_j > y_i) \wedge ((\mathbf{w}^T\mathbf{x}_j) - (\mathbf{w}^T\mathbf{x}_i) < 1)\}|.$$

To compute these quantities efficiently, Algorithm 2 first sorts the training examples by decreasing value of $\mathbf{w}^T\mathbf{x}_i$. Then, for each rank r in turn, it updates the values of $\mathbf{c}^+$ and $\mathbf{c}^+$ for all constraints $(i, j) \in \mathcal{P}$ in OP4 with $y_i = r$. By going through the examples in order of $\mathbf{w}^T\mathbf{x}_i$, the algorithm can keep track of

$$a = |l : (y_l = r) \wedge (\mathbf{w}^T\mathbf{x}_l > \mathbf{w}^T\mathbf{x}_i)|$$
$$b = |l : (y_l < r) \wedge (\mathbf{w}^T\mathbf{x}_l > \mathbf{w}^T\mathbf{x}_i - 1)|$$

via incremental updates. Whenever it encounters an example with $y_i = r$, there are exactly b constraints $(i, j) \in \mathcal{P}$ in OP4 with $y_j < r$ and $((\mathbf{w}^T\mathbf{x}_i) - (\mathbf{w}^T\mathbf{x}_j) < 1)$. Similarly, whenever it encounters an example with $y_j < r$, there are exactly $(n_r - a)$ constraints $(i, j) \in \mathcal{P}$ in OP4 with $y_i = r$ and $((\mathbf{w}^T\mathbf{x}_i) - (\mathbf{w}^T\mathbf{x}_j) < 1)$. By exhaustively going through all r, $y_i = r$, and $y_j < r$ and adding the respective quantities to c_i^+ and c_j^-, the algorithm implicitly considers all constraints in OP4.

Like Algorithm 1, the iteration terminate only if no constraint in OP5 is violated by more than ϵ, and

$$\frac{1}{2}\mathbf{w}^{*T}\mathbf{w}^* + C\xi^* \geq \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi$$

since $\mathcal{W}$ is a subset of the constraints in OP4. □

The following lemma characterizes the time Algorithm 2 takes in each iteration as a function of n and s.

Lemma 3. *Each iteration of Algorithm 2 requires time $O(sn + n\log(n) + Rn)$ for a constant working set size $|\mathcal{W}|$.*

Proof. *The proof is analogous to that of Lemma 1. The greatest expense per iteration in terms of n is the sort in Line 5 and the computation of n inner products $\mathbf{w}^T\mathbf{x}_i$. Lines 8-24 take $R - 1$ passes through the training set. Due to the condensed representation, setting up the quadratic program in Line 4 can again be done in time $O(|\mathcal{W}|^2 sn)$ analogous to Lemma 1.* □

Lemma 2 already established an upper bound on the number of iterations of Algorithm 2. Analogous to Theorem 4, the following characterizes its the scaling behavior.

Theorem 6. (Time Complexity of Algorithm 2) *For any distribution $P(X, Y)$ that generates feature vectors of bounded L_2-norm $||\mathbf{x}||$ and any fixed value of $C > 0$ and $\epsilon > 0$, Algorithm 2 has time complexity $O(sn\log(n))$ for any training sample of size n and sparsity s.*

Note that conventional methods for training ordinal regression SVMs based on OP4 have much worse scaling behavior. They scale roughly $O(sn^3)$ even under the (optimistic) assumption that a problem with m constraints can be solved in $O(m)$ time. Only small training sets with hundreds or at best a few thousand examples are tractable.

Table 1: Training time in CPU-seconds.

				Classification		Ordinal Regression	
	n	N	s	SVM-Perf	SVM-Light	SVM-Perf	SVM-Light
`Reuters CCAT`	804,414	47,236	0.16%	149.7	20,075.5	304.1	NA
`Reuters C11`	804,414	47,236	0.16%	178.9	5,187.4	499.1	NA
`Arxiv astro-ph`	62,369	99,757	0.08%	16.9	80.1	26.1	NA
`Covertype 1`	522,911	54	22.22%	171.7	25,514.3	1,109.1	NA
`KDD04 Physics`	150,000	78	38.42%	31.9	1,040.2	132.5	NA

Heuristic approaches for pushing the limits by removing constraints offer no performance guarantees [20]. We will see in the following experiments that Algorithm 2 can handle problems with hundred-thousands of examples with ease.

4. EXPERIMENTS

While Theorems 4 and 6 characterize the asymptotic scaling of Algorithms 1 and 2, the behavior for small sample sizes may be different. We will empirically analyze the scaling behavior in the following experiments, as well as its sensitivity to C and ϵ. Furthermore, we compare the algorithms against existing methods, in particular the decomposition method SVM-Light.

We implemented Algorithms 1 and 2 using SVM-Light as the basic quadratic programming software that is called in Line 4 of each algorithm. However, other quadratic programming tools would work just as well, since $|\mathcal{W}|$ remained small in all our experiments. We will refer to our implementation of Algorithms 1 and 2 as SVM-Perf in the following. SVM-Perf is available at `http://svmlight.joachims.org`.

We use 5 datasets in our experiments, selected to cover a wide range of properties.

1. First, we consider the binary text classification task `CCAT` from the Reuters RCV1 collection[1] [17]. There are 804,414 examples split into 23,149 training and 781,265 test examples, and there are 47,236 features with sparsity 0.16%. This task has an almost balanced class ratio.

2. Second, we include the task `C11` from the RCV1 collection, since it has an unbalanced class ratio. Existing decomposition methods like SVM-Light are know to run faster for unbalanced tasks.

3. The third problem is classifying abstracts of scientific papers from the Physics ArXiv by whether they are in the `Astro-physics` section. We picked this task since it has a large number of features (99,757) with high sparsity (0.08%). There are 62,369 examples split into 29,882 training examples and 32,487 test examples.

4. The fourth problem is class 1 in the `Covertype` dataset[2] of Blackard, Jock & Dean, which is comparably low-dimensional with 54 features and a sparsity of 22.22%. There are 581,012 examples which we split into 522,911 training examples and 58101 test examples.

5. Finally, we added the `KDD04 Physics` task from the KDD-Cup 2004 [1], with 78 features (sparsity 38.42%) and 150,000 examples, which are split into 50,000 training examples and 100,000 test examples.

We use the Precision/Recall Break-Even Point (PRBEP) (see e.g. [10]) as the measure of performance for the text-classification tasks, and Accuracy for the other problems.

The following parameters are used in our experiments, unless noted otherwise. Both SVM-Light and SVM-Perf use $\epsilon = 0.001$ (note that their interpretation of ϵ is different, though). As the value of C, we use the setting that achieves the best performance on the test set when using the full training set ($C = 10,000$ for `Reuters CCAT`, $C = 50,000$ for `Reuters C11`, $C = 20,000$ for `Arxiv astro-ph`, $C = 1,000,000$ for `Covertype 1`, and $C = 20,000$ for `KDD04 Physics`). Whenever possible, runtime comparisons are done on the full set of examples, joining training and test data together to get larger datasets. Experiments that compare prediction performance report results for the standard test/training split. All experiments are run on 3.6 Mhz Intel Xeon processors with 2GB main memory under Linux.

4.1 How Fast are the Algorithms Compared to Existing Methods?

Table 1 compares the CPU-time of SVM-Perf and SVM-Light on the full data for the 5 tasks described above. For the classification SVM, SVM-Perf is substantially faster than SVM-Light on all problems, achieving a speedup of several orders of magnitude on most problems. We will analyze these results in detail in the following sections.

We also applied the ordinal regression SVM to these datasets, treating the binary classification problems as ordinal problems with two classes. An alternative view on this setup is that the OR-SVM learns a classification rule that optimizes ROC Area [20, 14]. The runtimes are somewhat slower than for classification, but still very tractable. We tried to train SVM-Light in its ordinal regression mode on these problems as well. However, training with SVM-Light is intractable with more than $\approx 4,000$ examples.

A method that was recently proposed for training linear SVMs is the L2-SVM-MFN algorithm [15]. While they do not provide an implementation of their method, they report training times for the two publicly available dataset `Adult` and `Web` in the version produced by John Platt. On the `Adult` data with 32,562 examples and 120 features, they report a training time of 1.6 CPU-seconds for the value of $C = 0.0625 \cdot 32{,}562$ achieving optimal cross-validation error, which is comparable to 3.1 CPU-seconds needed by SVM-Perf for $C = 0.05 \cdot 32{,}562$ as recommended by Platt. Similarly, for the `Web` data with 49,749 examples and 300 features, L2-SVM-MFN is reported to take 5.0 CPU-seconds ($C = 1 \cdot 49{,}749$) while SVM-Perf takes 7.6 CPU-seconds for the same value of C. While both methods seem to perform comparably for these rather small training sets, it is unclear

[1] http://jmlr.csail.mit.edu/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm
[2] http://www.ics.uci.edu/~mlearn/MLRepository.html

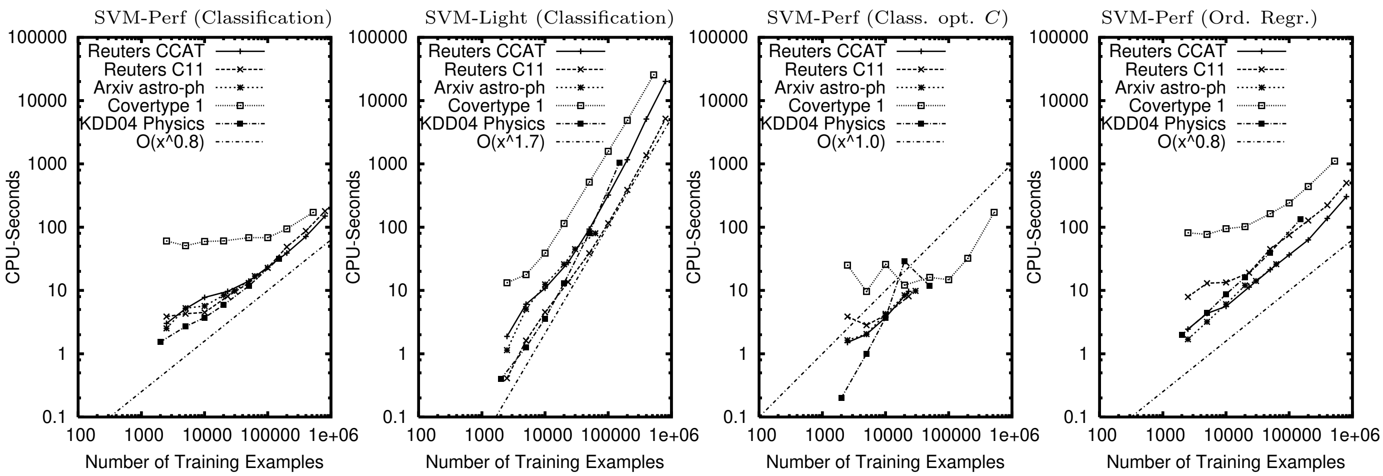

Figure 1: Training time of SVM-Perf (left) and SVM-Light (left-middle) for classification as a function of n for the value of C that gives best test set performance for the maximum training set size. The middle-right plot shows training time of SVM-Perf for the value of C with optimum test set performance for the respective training set size. The right-most plot is the CPU-time of SVM-Perf for ordinal regression.

how L2-SVM-MFN scales. In the worst case, the authors conclude that each iteration may scale $O(sn\min\{n,N\})$, although practical scaling is likely to be substantially better. Finally, note that L2-SVM-MFN uses squared slack variables $\sum \xi_i^2$ to measure training loss instead of linear slacks $\sum \xi_i$ like in SVM-Light and SVM-Perf.

The Lagrangian SVM (LSVM) [18] is another method particularly suited for training linear SVMs. Like the L2-SVM-MFN, the LSVM uses squared slack variables $\sum \xi_i^2$ to measure training loss. The LSVM can be very fast if the number of features N is small, scaling roughly as $O(nN^2)$. We applied the implementation of Mangasarian and Musicant[3] to the `Adult` and the `Web` data using the values of C from above. With 31.4 CPU-seconds, the training time of the LSVM is still comparable on `Adult`. For the higher-dimensional `Web` task, the LSVM runs into convergence problems. Applying the LSVM to tasks with thousands of features is not tractable, since the algorithm requires storing and inverting an $N \times N$ matrix.

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

Figure 1 shows log-log plots of how CPU-time increases with the size of the training set. The left-most plot shows the scaling of SVM-Perf for classification, while the left-middle plot shows the scaling of SVM-Light. Lines in a log-log plot correspond to polynomial growth $O(n^d)$, where d corresponds to the slope of the line. The middle plot shows that SVM-Light scales roughly $O(n^{1.7})$, which is consistent with previous observations [11]. SVM-Perf has much better scaling, which is (to some surprise) better than linear with roughly $O(n^{0.8})$ over much of the range.

Figure 2 gives insight into the reason for this scaling behavior. The graph shows the number of iterations of SVM-Perf (and therefore the maximum number of constraints in the working set) in relation to the training set size n. It turns out that the number of iterations is not only upper bounded independent of n as shown in Lemma 2, but that

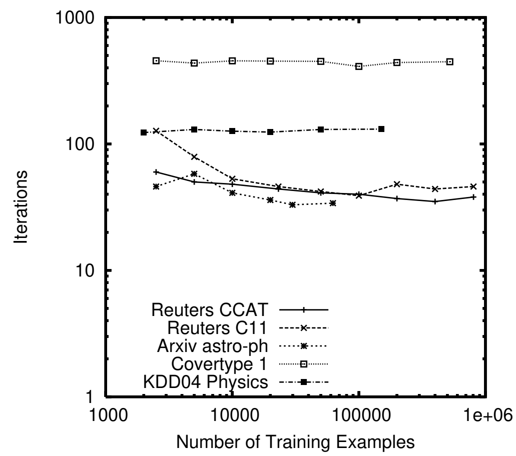

Figure 2: Number of iterations of SVM-Perf for classification as a function of sample size n.

it does not grow with n even in the non-asymptotic region. In fact, for some of the problems the number of iterations decreases with n, which explains the sub-linear scaling in CPU-time. Another explanation lies in the high "fixed cost" that is independent of n, which is mostly the cost for solving a quadratic program in each iteration.

Since Lemma 2 identifies that the number of iterations depends on the value of C, scaling for the optimal value of C might be different if the optimal C increases with training set size. To analyze this, the middle-right plot of Figure 1 shows training time for the optimal value of C. While the curves look more noisy, the scaling still seems to be roughly linear.

Finally, the right-most plot in Figure 1 shows training time of SVM-Perf for ordinal regression. The scaling is slightly steeper than for classification as expected. The number of iterations is virtually identical to the case of classification shown in Figure 2. Note that training time of SVM-Light would scale roughly $O(n^{3.4})$ on this problem.

[3] http://www.cs.wisc.edu/dmi/lsvm/

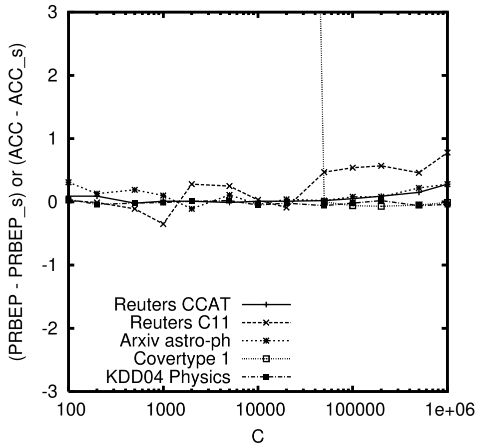

Figure 3: Difference in prediction performance between SVM-Perf and SVM-Light for classification as a function of C.

4.3 Is the Prediction Performance of SVM-Perf Different from SVM-Light?

One potential worry is that the speedup of SVM-Perf over SVM-Light somehow comes at the expense of prediction accuracy, especially due to the choice of $\epsilon = 0.001$. However, this is not the case. Figure 3 shows the difference in test set accuracy / PRBEP between the classifiers produced by SVM-Light and SVM-Perf. For better readability, the difference is shown in terms of percentage points. A positive value indicates that SVM-Perf has higher prediction performance, a negative value indicates that SVM-Light performs better. For almost all values of C both methods perform almost identically. In particular, there is not indication that the rules learned by SVM-Perf are less accurate. One case where there is a large difference is the Covertype 1 task for small values of C, since SVM-Light stops before fully converging.

4.4 How Small does ϵ need to be?

The previous section showed that $\epsilon = 0.001$ is sufficient to get prediction accuracy comparable to SVM-Light. But maybe a lower precision would suffice and reduce training time? Figure 4 shows the difference (in percentage points) in prediction accuracy / PRBEP compared to the performance SVM-Perf reaches for $\epsilon = 0.001$. Values above (below) 0 indicate that the accuracy of SVM-Perf for that ϵ is better (worse) than the accuracy at $\epsilon = 0.001$. The graph shows that for all $\epsilon \leq 0.01$ the prediction performance is within half a percentage point. For larger values of ϵ the resulting rules are starting to have more variable and less reliable performance. So, overall, $\epsilon = 0.001$ seems accurate enough with some "safety margin". However, one might elect to use larger ϵ at the expense of prediction accuracy, if training time was substantially faster. We will evaluate this next.

4.5 How does Training Time Scale with ϵ?

Lemma 2 indicates that the number of iterations, and therefore the training time, should decrease as ϵ increases. Figure 5 shows number of iterations as a function of ϵ. Interestingly, the empirical scaling of roughly $O(\frac{1}{\epsilon^{0.3}})$ is much better than $O(\frac{1}{\epsilon^2})$ in the bound from Lemma 2. For training time, as shown in Figure 6, the scaling is $O(\frac{1}{\epsilon^{0.4}})$.

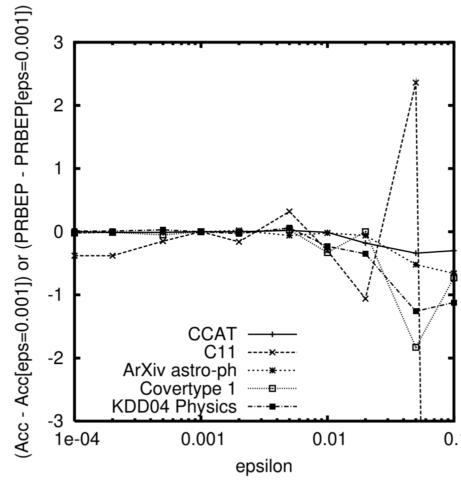

Figure 4: Difference in Accuracy or PRBEP of SVM-Perf compared to its performance at $\epsilon = 0.001$ as a function of ϵ.

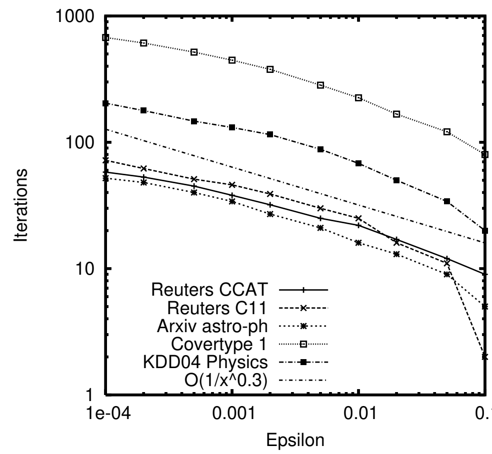

Figure 5: Number of iterations of SVM-Perf for classification as a function of ϵ.

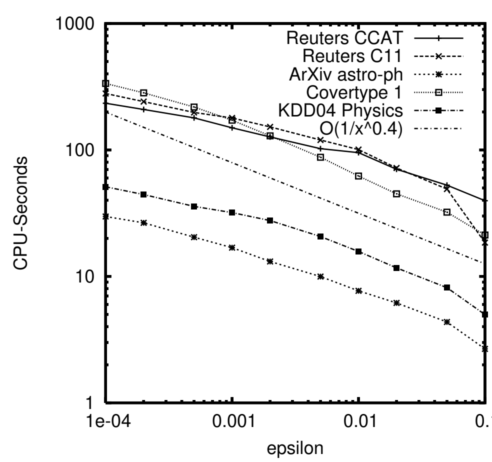

Figure 6: CPU-time of SVM-Perf for classification as a function of ϵ.

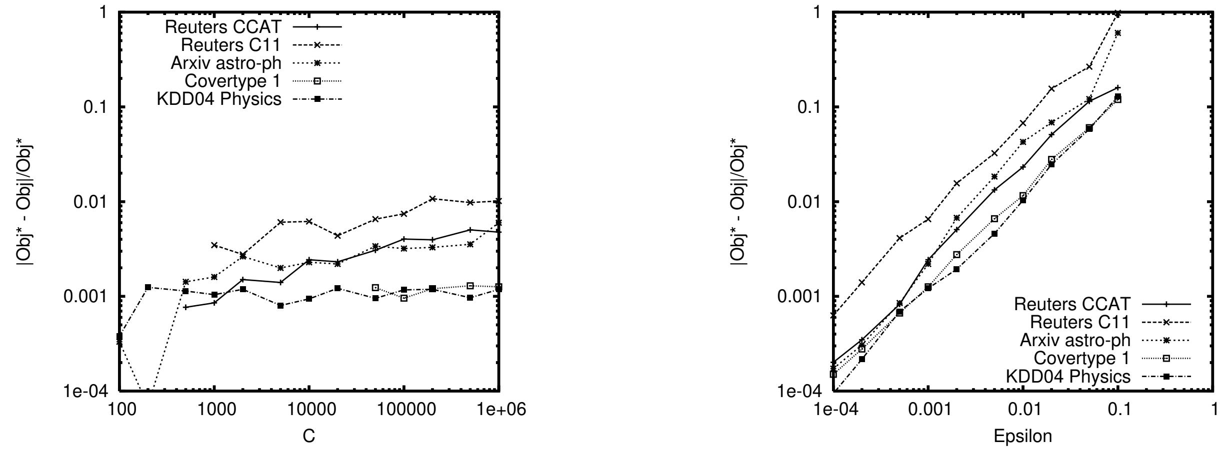

Figure 7: Relative difference between the objective value of the SVM-Perf solution for classification and the (approximately) true solution as a function of C (left) (with $\epsilon = 0.001$) and as a function of ϵ (right) (with C is set to maximize test set prediction performance).

Could the runtime of SVM-Light be improved by increasing the value of ϵ as well? While SVM-Light converges faster for larger values of ϵ, the difference is much smaller. Even when increasing ϵ to 0.5, the speedup is less than a factor of 2 on all five problems.

4.6 Is the Solution Computed by SVM-Perf Close to Optimal?

While we have already established that training with $\epsilon = 0.001$ gives rules of comparable prediction accuracy, it is also interesting to look at how close the objective value of the relaxed solution is to the true optimum for different values of ϵ. As Theorems 3 and 5 show, the objective value is lower than the true objective, but by how much? Figure 7 shows the relative difference

$$\frac{Obj(\text{SVM-Light}) - Obj(\text{SVM-Perf})}{Obj(\text{SVM-Light})}$$

between the solution of SVM-Perf and a high-precision solution computed by SVM-Light. The left-hand plot of Figure 7 indicates that for $\epsilon = 0.001$ the relative error is roughly between 0.1% and 1% over all values of C. The missing points correspond to values where SVM-Light failed to converge. The right-hand plot shows how the relative error decreases with ϵ.

4.7 How does Training Time Scale with C?

Finally, let's examine how the number of iterations of SVM-Perf scales with the value of C. The upper bound of Lemma 2 suggest a linear scaling, however, Figure 8 shows that the actual scaling is much better with $O(\sqrt{C})$ for classification (and similarly for ordinal regression). Figure 9 shows the resulting training times (left) and compares them against those of SVM-Light (right). Except for excessively large values of C, the training time of SVM-Perf scales sublinearly with C. Note that the optimal values of C lie between $10,000$ and $50,000$ for all tasks except `Covertype 1`. For all values of C, SVM-Perf is faster than SVM-Light.

5. ACKNOWLEDGMENTS

This research was supported under NSF Award IIS-0412894 and through a gift from Google.

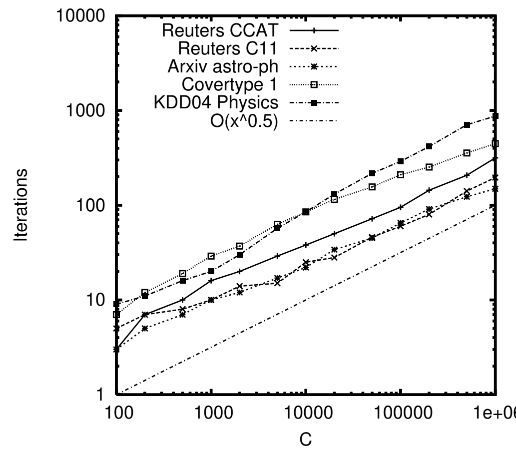

Figure 8: Number of iterations of SVM-Perf as a function of C.

6. CONCLUSIONS

We presented a simple Cutting-Plane Algorithm for training linear SVMs that is shown to converge in time $O(sn)$ for classification and $O(sn\log(n))$ for ordinal regression. It is based on an alternative formulation of the SVM optimization problem that exhibits a different form of sparsity compared to the conventional formulation. The algorithm is empirically very fast and has an intuitively meaningful stopping criterion.

The algorithm opens several areas for research. Since it takes only a small number of sequential iterations through the data, it is promising for parallel implementations using out-of-core memory. Also, the algorithm can in principle be applied to SVMs with Kernels. While a straightforward implementation is slower by a factor of n, matrix approximation techniques and the use of sampling might overcome this problem.

7. REFERENCES

[1] R. Caruana, T. Joachims, and L. Backstrom. Kddcup 2004: Results and analysis. *ACM SIGKDD Newsletter*, 6(2):95–108, 2004.

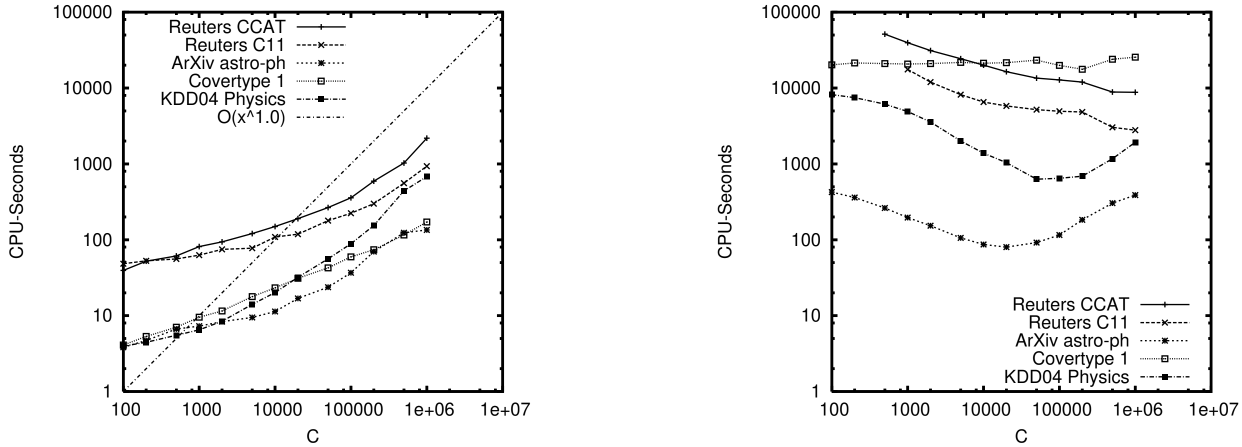

Figure 9: CPU-time of SVM-Perf (left) and SVM-Light (right) as a function of C.

[2] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

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

[4] J. Dez, J. del Coz, and A. Bahamonde. A support vector method for ranking minimizing the number of swapped pairs. Technical report, Artificial Intelligence Centre, Universidad de Oviedo at Gijn, 2006.

[5] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of ACM-CIKM98*, November 1998.

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

[7] G. Fung and O. Mangasarian. Proximal support vector classifiers. In *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*, 2001.

[8] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, pages 115–132. MIT Press, Cambridge, MA, 2000.

[9] D. Hush and C. Scovel. Polynomial-time decomposition algorithms for support vector machines. *Machine Learning*, 51:51–71, 2003.

[10] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning*, pages 137 – 142, Berlin, 1998. Springer.

[11] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

[12] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2002.

[13] T. Joachims. Learning to align sequences: A maximum-margin approach. online manuscript, August 2003.

[14] T. Joachims. A support vector method for multivariate performance measures. In *International Conference on Machine Learning (ICML)*, 2005.

[15] S. Keerthi and D. DeCoste. A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research (JMLR)*, 6:341–361, 2005.

[16] J. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial Applied Mathematics*, 8:703–712, 1960.

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

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

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

[20] A. Rakotomamonjy. Svms and area under roc curve. Technical report, PSI-INSA de Rouen, 2004.

[21] B. Schoelkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, Cambridge, MA, 2002.

[22] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

[23] I. Tsang, J. Kwok, and P.-M. Cheung. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research (JMLR)*, 6:363–392, 2005.

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