Cutting Plane Training for Linear Support Vector Machines

Nicholas A. Arnosti and Jugal K. Kalita

Abstract—Support Vector Machines (SVMs) have been shown to achieve high performance on classification tasks across many domains, and a great deal of work has been dedicated to developing computationally efficient training algorithms for linear SVMs. One approach [1] approximately minimizes risk through use of cutting planes, and is improved by [2], [3]. We build upon this work, presenting a modification to the algorithm developed by [2]. We demonstrate empirically that our changes can reduce cutting-plane training time by up to 40%, and discuss how changes in data sets and parameter settings affect the effectiveness of our method.

Index Terms—Linear support vector machine, Cutting plane SVM.

1 Introduction

ALTHOUGH SVMs have been shown to perform well on many classification tasks, their training requires solving an optimization problem whose complexity grows with the number of training examples. Some tasks require a large number of examples to achieve optimal performance, and traditional SVM solvers are often prohibitively slow in these cases. For this reason, it is important to develop training algorithms whose asymptotic time complexity scales more reasonably.

Given a set of examples of the form $(\mathbf{x}_i, y_i)$, where each $\mathbf{x}_i$ is a vector of feature values taken from the space $\mathcal{X}$, and y_i is a class label taken from the set $\mathcal{Y}$, classifier learning algorithms aim to find a function $g : \mathcal{X} \to \mathcal{Y}$ such that for unseen examples $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}, g(\mathbf{x}) = y$. Throughout this paper, n is the number of training examples, p the number of features for each example and $\mathcal{X} = \mathbb{R}^p$, and $k = |\mathcal{Y}|$, the number of distinct classes.

For binary classification, $\mathcal{Y} = \{-1, +1\}$, and the SVM finds a function of the form $g(\mathbf{x}_i) = sgn(\mathbf{w}^T\mathbf{x}_i + b)$ for fixed $\mathbf{w} \in \mathbb{R}^p, b \in \mathbb{R}$ [4]. For linearly separable data sets, the goal is to find the choice of $\mathbf{w}$ which maximizes the margin between the two classes. This is equivalent to minimizing $\mathbf{w}^T\mathbf{w}$ subject to $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \ \forall i$.

Most data sets are not linearly separable, so it is necessary to introduce error terms ("slack variables") ξ_i for each training example. The objective function to be minimized has two terms: one penalizing for training errors and the other penalizing for a narrow margin between classes (to avoid over-fitting). Using an L1-loss function, this is captured by the following optimization problem [5]:

$$\begin{aligned} &\text{minimize:} && \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + \tfrac{C}{n}\textstyle\sum_{i=1}^{n}\xi_i \\ &\text{subject to:} && y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \ , \xi_i \geq 0 \, \forall i, \end{aligned} \tag{1}$$

where $C \in \mathbb{R}_+$ is known as the regularization constant and determines the relative importance of the two objectives. For simplicity, this paper will not include the bias term b, as it can be "absorbed" into the vector $\mathbf{w}$ by adding a feature of weight 1 to each vector $\mathbf{x}_i$.

For multiclass classification, the learned model consists of one vector $\mathbf{w}_y$ for each class label, and the decision function given input $\mathbf{x}$ is $\hat{y} = \text{argmax}_{y \in \mathcal{Y}}(\mathbf{w}_y^T\mathbf{x})$. Commonly, the weight vectors $\mathbf{w}_y$ are optimized individually by breaking the task into a series of binary problems. Examples of such approaches include *one-against-all*. *one-against-one* [6], and *error-correcting output coding* [7]. Other approaches, such as those presented by [8] and [9], optimize the vectors $\mathbf{w}_y$ simultaneously rather than independently.

In all of the above formulations, the number of slack variables scales linearly with n, the number of training examples. Minimizing the objective function requires optimizing each of these variables, which generally takes time at least quadratic in n. As a result, traditional SVM formulations are unsuitable for large data sets. Recent work [1], [2], [10], [11], [12] has focused on finding close-to-optimal and relatively efficient solutions to the primal problem. This paper extends the cutting plane approach presented in [2] in an effort to further reduce SVM training time. Section 2 describes previous cutting plane training algorithms. Section 3 introduces our modified cutting plane algorithm, and Section 4 presents an empirical analysis of the effect of our changes. We close with a summary and discussion of future work.

2 Previous Work

Joachims [1] presents a structural formulation of the primal (1), which reduces the number of slack variables

- *Nicholas Arnosti graduated from Williams College, MA, in June 2011. He is now pursuing a Ph.D. in Operations Research at Stanford University. (e-mail: narnosti@stanford.edu).*
- *Jugal K. Kalita is with the Department of Computer Science, University of Colorado, Colorado Springs, CO 80918 USA. (e-mail: jkalita@uccs.edu).*

from n to one, and allows him to quickly find good approximations to the optimal objective value. Teo et al. [10] generalize Joachims' [1] cutting plane approach, and Franc and Sonnenburg [2] improve upon this work. Throughout this paper, we will call Joachims' algorithm the Cutting Plane Algorithm (CPA), and refer to the work presented by [2] as the Optimized Cutting Plane Algorithm (OCA). Both algorithms aim to solve the following reformulation of the SVM primal (1):

$$\text{minimize } F(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + CR(\mathbf{w}), \tag{2}$$
$$\text{where } R(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\max\{0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\}.$$

In [1], it is proven that that (1) and (2) are mathematically equivalent, meaning that a vector $\mathbf{w}^*$ is a solution to one if and only if it is a solution to the other.

The risk function $R(\mathbf{w})$ consists of a sum of n non-differentiable terms, making it difficult to find the point $\mathbf{w}^*$ which minimizes $F(\mathbf{w})$. Instead, CPA and OCA create a series of approximations $R_t(\mathbf{w})$ (where t represents the number of iterations) of the actual risk $R(\mathbf{w})$. At each iteration, these algorithms find the point $\mathbf{w}_t$ which minimizes the simplified objective function F_t:

$$\mathbf{w}_t = \text{argmin}_{\mathbf{w}}\{F_t(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + CR_t(\mathbf{w})\}. \tag{3}$$

The approximation $R_t(\mathbf{w})$ is constructed using linear approximations of the risk R ("cutting planes"). Because R is a convex function, for any point $\mathbf{w}'$ and any sub-gradient $\mathbf{a}'$ of R at this point, the linear approximation $R(\mathbf{w}') + \langle\mathbf{a}', \mathbf{w} - \mathbf{w}'\rangle$ provides a lower bound for $R(\mathbf{w})$. For notational simplicity, we define

$$b' = R(\mathbf{w}') - \langle\mathbf{a}', \mathbf{w}'\rangle \tag{4}$$

so that $R(\mathbf{w}) \geq \langle\mathbf{a}', \mathbf{w}\rangle + b'$. Given a collection of functions $P_i(\mathbf{w}) = \langle\mathbf{a}_i, \mathbf{w}\rangle + b_i$, $R_t(\mathbf{w})$ is their pointwise maximum subject to a non-negativity constraint:

$$R_t(\mathbf{w}) = \max(0, \max_{1\leq i\leq t}\{P_i(\mathbf{w})\}). \tag{5}$$

It follows that for all $\mathbf{w}$, $F_t(\mathbf{w}) \leq F(\mathbf{w})$, so by definition of $\mathbf{w}_t$ and $\mathbf{w}^*$ we have that $F_t(\mathbf{w}_t) \leq F_t(\mathbf{w}^*) \leq F(\mathbf{w}^*) \leq F(\mathbf{w}_t)$. It is natural to terminate the algorithm when $F_t(\mathbf{w}_t)$ is within some relative tolerance ϵ of $F(\mathbf{w}_t)$, as this ensures that $F(\mathbf{w}_t)$ is within that tolerance of the true optimum $F(\mathbf{w}^*)$. A concise description of CPA is presented in Algorithm 1.

Although there are many subgradients of R at a given point $\mathbf{w}'$, for all cutting-plane algorithms discussed in this paper, the following subgradient is used:

$$\mathbf{a}' = -\frac{1}{n}\sum_{i=1}^{n}\pi_i y_i \mathbf{x}_i, \text{ where} \tag{6}$$
$$\pi_i = \begin{cases} 1 & \text{if } y_i\langle\mathbf{w}', \mathbf{x}_i\rangle \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

Joachims [1] proves that the number of iterations required to reach ϵ-precision depends only on ϵ and the regularization constant C (he bounds the growth at a rate of $\mathcal{O}(C/\epsilon)$) and not on the number of training examples n. From this, he concludes that CPA runs in $\mathcal{O}(np)$ time. Further details and analysis of CPA can be found in [13].

Franc and Sonnenburg [2] note several ways in which CPA is sub-optimal. Most notably, while the under-estimate $F_t(\mathbf{w}_t)$ increases monotonically with respect to t, the over-estimate $F(\mathbf{w}_t)$ fluctuates significantly, which can lead to slow convergence. To solve this problem, a new value $\mathbf{w}_t^b$ is defined:

$$\mathbf{w}_t^b = \text{argmin}_{\mu\geq 0}F(\mu\mathbf{w}_t + (1-\mu)\mathbf{w}_{t-1}^b). \tag{7}$$

OCA uses $F(\mathbf{w}_t^b)$, rather than $F(\mathbf{w}_t)$ to bound $F(\mathbf{w}^*)$ from above. The process of solving equation (7) is referred to as the line search. For notational convenience, we define $f(\mu) = F(\mu\mathbf{w}_t + (1-\mu)\mathbf{w}_{t-1}^b)$. Note that $f(0) = F(\mathbf{w}_{t-1}^b)$, so choosing the value of μ which minimizes $f(\mu)$ ensures that $F(\mathbf{w}_t^b) \leq F(\mathbf{w}_{t-1}^b)$. Thus, the sequence $\{F(\mathbf{w}_t^b)\}_{t\geq 1}$ decreases monotonically.

Note that CPA can be considered a special case of OCA which, rather than conducting the line search, simply sets $\mu = 1$. While this means that each iteration of CPA is faster than the corresponding OCA iteration, OCA outperforms CPA because the line search "pays" for itself by reducing the number of iterations required to converge. The net effect of OCA modifications often reduces training time by an order of magnitude or more [2]. OCA is summarized in Algorithm 2. The value λ used in the algorithm is an internal OCA parameter, which the authors suggest setting to 0.1.

There are many other descent methods which have been used to solve the convex optimization problem of minimizing $F(\mathbf{w})$. Examples that have been developed for linear SVMs include stochastic subgradient descent such as Pegasos [12] and SGD [14], dual coordinate descent implemented in LIBLINEAR [11], and more generalized cutting plane techniques [10]. Tests presented by [2], [3] indicate that OCA compares favorably to many of these approaches. It is noted in [3] that OCA outperforms Pegasos and the algorithms presented by [10], while converging at rates comparable to those of SGD. The authors report that OCA is competitive with LIBLINEAR for many data sets and parameter settings, although for large values of C, LIBLINEAR is superior. Our work seeks to modify and improve OCA.

3 Modifying the OCA Line Search

Franc and Sonnenburg [2] present a method which solves equation (7) exactly, and establish that it runs in $\mathcal{O}(np + n\log n)$ time. In this section we propose instead to use an inexact line search, and offer one such search algorithm. Our modifications prove beneficial for two reasons. The first is that our inexact method is computationally more efficient than the exact approach. The second is that in some cases, our algorithm converges in significantly fewer iterations than the search algorithm

Algorithm 1 Cutting Plane Algorithm (CPA)

$t = 0, R_0(\mathbf{w}) = 0$
repeat
  Compute $\mathbf{w}_t$, the solution of problem (3)
  Compute $\mathbf{a}_{t+1}$ and b_{t+1} using equations (6) and (4) at the point $\mathbf{w}_t$.
  Define $R_{t+1}(\mathbf{w}) = \max\{R_t(\mathbf{w}), \langle \mathbf{a}_{t+1}, \mathbf{w} \rangle + b_{t+1}\}$
  t = t+1
until $1 - \frac{F_t(\mathbf{w}_t)}{F(\mathbf{w}_t)} < \epsilon$

Algorithm 2 Optimized Cutting Plane Algorithm (OCA)

$t = 1, R_0(\mathbf{w}) = 0, \mathbf{w}_0^b = \mathbf{0}$
repeat
  Compute $\mathbf{w}_t$, the solution of problem (3)
  Compute $\mathbf{w}_t^b$ according to equation (7)
  Compute $\mathbf{a}_{t+1}$ and b_{t+1} using equations (6) and (4) at the point $\mathbf{w}_t^c = \mathbf{w}_t^b(1-\lambda) + \mathbf{w}_t\lambda$.
  Define $R_{t+1}(\mathbf{w}) = \max\{R_t(\mathbf{w}), \langle \mathbf{a}_{t+1}, \mathbf{w} \rangle + b_{t+1}\}$
  t = t+1
until $1 - \frac{F_t(\mathbf{w}_t)}{F(\mathbf{w}_t^b)} < \epsilon$

Fig. 1: The basic cutting plane approach to training SVMs, and the improvement upon which our work is focused.

presented by [3]. These two advantages are further discussed in sections 4.1 and 4.2, respectively.

Franc and Sonnenburg [2] note that $f(\mu)$ is of the form

$$f(\mu) = \frac{1}{2}A\mu^2 + B_0\mu + C_0 + \sum_{i=1}^{n} \max\{0, B_i\mu + C_i\}$$

for constants $A > 0, B_i, C_i, i = 0 \ldots n$. Hence, the subdifferential $\partial f(\mu)$ is a sum of an increasing linear function and up to n step functions which have threshold values at the points $k_i = -C_i/B_i$ (or are identically zero if $B_i = 0$). Because $\partial f(\mu)$ increases monotonically, it is possible to solve for the value μ^* satisfying $0 \in \partial f(\mu^*)$ by computing and sorting the threshold values k_i. Algorithm 3 precisely describes this process. The multi-class implementation is similar, and discussed in detail by [3]. In this paper, we refer to the implementation of OCA that uses Algorithm 3 to determine $\mathbf{w}_t^b$ as OCA-E.

Rather than solving equation (7) analytically, we present an algorithm that evaluates $f(\mu)$ at several values of μ and uses this information to make an intelligent choice for $\mathbf{w}_t^b$. This task is facilitated by the observation in [2], [3] that f is a convex function with no extraneous local minima. Our algorithm uses three values, `low` $<$ `mid` $<$ `high` to define a search window for the optimal value of μ. As long as $f($`high`$) < f($`mid`$)$, the convexity of f implies that the optimal value of μ is greater than `mid`, so the search window is shifted "to the right" (i.e. to larger values of μ). If $f($`low`$) < f($`mid`$)$, then the optimal value of μ is less than `mid`, so the window is shifted towards smaller values of μ.

Once $f($`low`$) > f($`mid`$)$ and $f($`high`$) > f($`mid`$)$, it is certain that the optimal value of μ lies between `low` and `high`. At this point, the algorithm selects `mid` as its value of μ, rather than searching more carefully within the interval (`low`, `high`). We made this choice upon observing that additional search rarely changed the selected value of μ in a meaningful way, but often carried a significant computational cost. Algorithm 4 provides a precise description of our "three point" line search. In this paper, we use OCA-T to refer to an implementation of OCA that uses Algorithm 4 to determine $\mathbf{w}_t^b$.

One crucial component of OCA-T is the step size δ. If it is large, the search is coarse, leading to poorer choices of μ and slower convergence (measured by number of iterations). A small step size, meanwhile, may require $f(\mu)$ to be evaluated at many points if the initial search window is far from the optimal choice. This causes each iteration to take more time. To balance these competing factors, OCA-T uses a variable step size. Initially, δ is small. Each time the search window shifts, δ increases, to ensure that starting far from the optimal value does not dramatically slow the algorithm.

Our implementation somewhat arbitrarily initializes δ to 0.02 and doubles it each time the search window shifts. Although the theoretical properties of OCA-T are not sensitive to these choices, one might expect that they would impact the algorithm's empirical performance. While carefully tuning δ and its rate of growth might have improved the numbers reported in the following section, we avoided doing this for two reasons.

The first is that parameters values that optimize performance on our baseline data sets are unlikely to be the same as those that would optimize performance on new data sets. Thus, results from trials using carefully tuned values for δ would likely be overly optimistic. The fact that OCA-T outperforms OCA-E without tuning δ means that the results in the following section are likely to generalize to untested domains.

The second reason for not experimenting further with the parameter δ is that a cursory investigation suggested that the performance of OCA-T is not highly sensitive to how δ is initialized and at what rate it is increased. When collecting data for this report, we ran limited tests using versions of OCA-T with different initial δ values (up to about 0.1) and grown rates (between 1 and 3). With the exception of growth rates very close to one, all implementations performed comparably.

4 Experimental Results

Tests were run on five data sets, all available from the LIBSVM data sets website (www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/). Relevant information about these data sets is displayed in Figure 3. The first four are the primary data sets used

Algorithm 3 Exact Line Search Algorithm

Compute $A, B_i, C_i, 0 \leq i \leq n$
if max $\partial f(0) > 0$ **then**
  return 0
else
  Sort the set $K = \{k_i = -C_i/B_i \mid k_i > 0\}$
  Define $s_1 < s_2 < \ldots < s_{|K|}$ so that $\{s_i\}_{i=1}^{|K|} = K$
  for $i = 1 : |K|$ **do**
    if $0 \in \partial f(s_i)$ **then**
      return s_i
    else if $\partial f(s_{i-1}) < 0 < \partial f(s_i)$ **then**
      return the value in the interval (s_{i-1}, s_i) where $\partial f(\mu) = 0$.
    end if
  end for
end if

Algorithm 4 Three Point Line Search Algorithm

$\delta = 0.02$, low $= \mu' - \delta$, mid $= \mu'$, high $= \mu' + \delta$
while f(high) $< f$(mid) **do** /* shift right */
  $\delta = 2\delta$
  low $=$ mid
  mid $=$ high
  high $=$ high $+ \delta$
end while
while f(low) $< f$(mid) **do** /* shift left */
  $\delta = 2\delta$
  high $=$ mid
  mid $=$ low
  low $=$ low $- \delta$
end while
$\mu' =$ mid /* save for next iteration */
return mid

Fig. 2: Two approaches for the OCA line search (7). We propose Algorithm 4 as an alternative to Algorithm 3.

for comparison. The fifth, POKER, is highly linearly inseparable but serves as an illustrative example in Section 4.2. All tests used a 2.66 GHz Intel Core with 32 GB of memory running Ubuntu 10.

Figures 4 and 5 show comparative performance of OCA-T and OCA-E across the four primary data sets for a variety of parameter values. In most cases, OCA-T requires more iterations before convergence, but the difference is rarely more than 5%. As expected, OCA-T reduces the amount of time spent on the line search (by over 75% on all data sets except NEWS20). As a result, on COVTYPE, MNIST, and RCV1, OCA-T trains more quickly than OCA-E. On NEWS20, OCA-T does not significantly reduce time spent on the line search, and OCA-T and OCA-E perform comparably. The relative performances of OCA-T and OCA-E seem to be independent of ϵ, but vary with C. For parameter settings which minimized test set error, OCA-T reduced training time of COVTYPE, MNIST, RCV1, and NEWS20 by 40.5%, 12.2%, 9.7%, and -1.6%, respectively.

Because OCA-E and OCA-T have identical termination criteria, their classification performance is equivalent. For all data sets and parameter settings, error rates on the test set for the two algorithms were within 0.0025 (and frequently quite closer). Figure 6 shows percent error on test sets for each data set as C and ϵ vary.

Name	Train/Test Size	Features	#Class
COVTYPE	581,012/115,826	54	7
MNIST	60,000/10,000	780	10
RCV1	518,571/15,564	47,236	53
NEWS20	15,935/3,993	62,061	20
POKER	25,010/1,000,000	10	10

Fig. 3: The data sets used for this report, selected from the LIBSVM site. Each has large numbers of examples and/or features, and COVTYPE, MNIST, RCV1, and NEWS20 appear frequently in the literature.

Accuracy was highly dependent on the choice of C, but $C = 1$ gave consistently good performance. Interestingly, classification accuracy did not appear to depend significantly on ϵ.

4.1 Data Set Dimensionality

Clearly, OCA-T is most helpful when a relatively large percentage of OCA-E training time is dedicated to the line search. But when is this the case? The exact line search computes each of the points k_i where the subdifferential $\partial f(\mu)$ is an interval, then sorts these values and considers them in sorted order. While time spent computing k_i depends on the number of features p, the process is dominated by the sorting step [2], which takes time depending only on n. Accordingly, for data sets with a large number of training examples, the time spent on the line search should not be greatly impacted by the number of features p. Other portions of the algorithm, however, take time which scales linearly with p. Therefore, we expect the fraction of training time spent on the line search to increase as the number of features decreases.

Empirical results confirm this expectation: on RCV1, which has over 47,000 features, the exact line search consumed 12% of OCA-E training time. On MNIST, which incorporates 780 features, 22% of the total time was devoted to the line search. COVTYPE, meanwhile, uses only 54 features, and the line search consumes 61% of the total time. To investigate this further we created reduced training sets from RCV1 and MNIST, each containing all of the original examples but eliminating many of the original features. The results of running OCA-E and OCA-T (using $C = 1, \epsilon = 0.01$) on these training sets are shown in Figure 7. These tests confirm that as the number of features decreases, the line search becomes an increasingly significant portion of the training process.

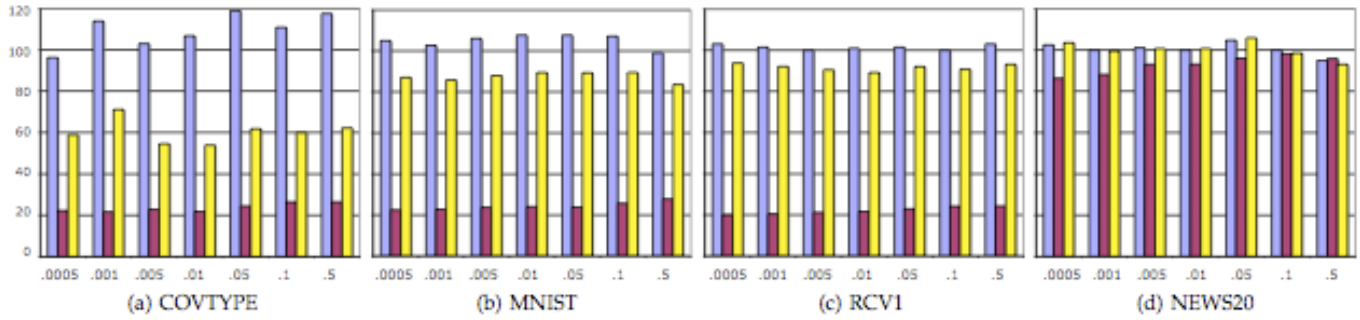

Fig. 4: Results when varying the tolerance ϵ (x-axis) while keeping the regularization constant $C = 1$. The y-axis values represent OCA-T performance as a percentage of corresponding OCA-E results. On COVTYPE, MNIST, and RCV1, OCA-T reduces time spent on the line search by nearly 80%. On these three data sets, OCA-T reduced training time across all values of ϵ. Generally, performance did not vary with ϵ.

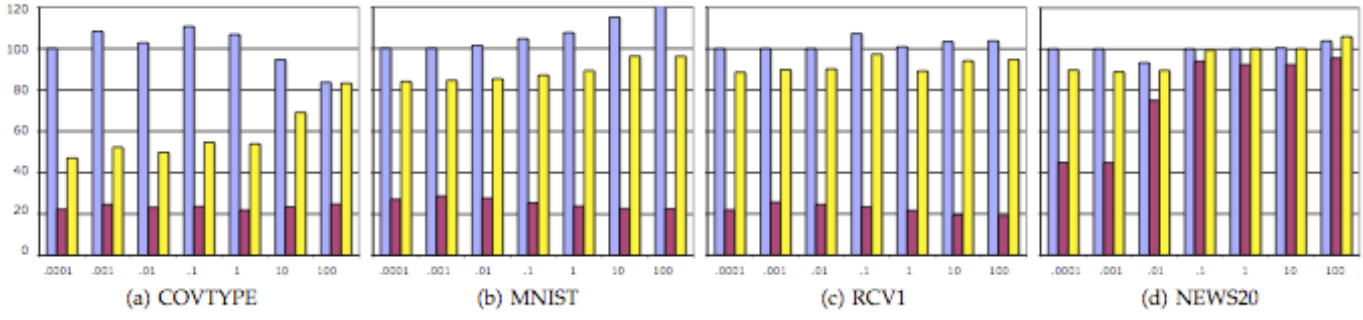

Fig. 5: Results when varying the regularization constant C (x-axis) while keeping the tolerance $\epsilon = 0.01$. The y-axis values represent OCA-T performance as a percentage of corresponding OCA-E results. For COVTYPE, MNIST, and RCV1, OCA-T outperforms OCA-E for all choices of C, while on NEWS20, OCA-E outperformed OCA-T for $C = 100$. Generally, as C increased, the advantage provided by OCA-T decreased.

	COVTYPE	MNIST	RCV1	NEWS20
0.5000	35.70	7.20	7.86	14.70
0.1000	35.75	7.23	7.83	14.78
0.0500	35.74	7.23	7.83	14.80
0.0100	35.73	7.22	7.82	14.73
0.0050	35.73	7.08	7.95	14.70
0.0010	35.99	7.13	7.68	14.75
0.0005	36.16	6.97	7.86	15.28

(a) Percent error as ϵ varies, $C = 1$

	COVTYPE	MNIST	RCV1	NEWS20
10^{-4}	37.72	11.72	26.20	94.99
10^{-3}	35.83	8.53	18.01	94.99
10^{-2}	35.77	7.22	10.79	20.19
10^{-1}	35.80	7.10	7.90	17.34
10^{0}	35.73	7.22	7.82	14.73
10^{1}	35.72	7.61	9.84	15.75
10^{2}	35.74	7.78	11.91	16.25

(b) Percent error as C varies, $\epsilon = 0.01$

Fig. 6: Percent error on an unseen test set for varying parameter values. Figure 6a shows that error is static as ϵ varies, suggesting that equation (2) need not be well-approximated to learn a good model. Figure 6b indicates that accuracy is highly dependent on the choice of C, with optimal performance obtained at or near the default value $C = 1$ in all cases.

4.2 The Downside of Greed

When seeking an approximate solution to the primal optimization problem (2), both CPA and OCA use a greedy approach: each iteration attempts to reduce the objective value by as much as possible. Since the ultimate goal is to find an objective value within a tolerance ϵ of the optimal value, this is a reasonable approach. It should be noted, however, that the choice of $\mathbf{w}_t^b$ has a notable impact on values of $\mathbf{w}_i^b$ for $i > t$. A slightly "worse" choice of $\mathbf{w}_t^b$ during the current iteration may result in better options in the future, and lead to faster convergence.

This begs the question of whether it is possible to identify values of μ which will be particularly good or bad for future convergence. In general, this is difficult, but there is one value of μ which has a clear drawback: if $\mu = 0$, $\mathbf{w}_t^b = \mathbf{w}_{t-1}^b$. If this occurs for many iterations in a row, the algorithm has spent a significant amount of time without improving its best so-far solution.

This concern is illustrated in dramatic fashion on the POKER data set. When run on POKER using the default value $C = 1$, OCA-E determined that $\mu = 0$ was optimal for each of the first 1451 iterations, leading to very slow convergence. OCA-T, by contrast, is designed so that it

chooses small but nonzero values of μ. While the first several iterations of OCA-T do not reduce $F(\mathbf{w}_t^b)$ by quite as much as the first iterations of OCA-E do, the final result is that OCA-T converges to within 1% of optimal after 258 iterations and 22 seconds, compared to 1454 iterations and 219 seconds for OCA-E.

While POKER is not well-suited to classification using linear SVMs, we argue that the observations above still illustrate the superiority of OCA-T over OCA-E. One reason for this is that regardless of the separability of the data set in question, one would hope to achieve the best result possible in relatively few iterations, which OCA-E clearly does not. More importantly, similar behavior was observed when working with other data sets. For example, when trained on the COVTYPE data set with parameters $C = 100, \epsilon = .01$, OCA-E required 744 iterations to converge, while OCA-T needed only 622. Upon investigation we found that for 490 of the 744 OCA-E iterations (or 66%), the value $\mu = 0$ was selected. Generally speaking, the observations in this section indicate that the shortcomings of the greedy approaches used by all cutting plane algorithms described in this paper may be significant. An immediate consequence of this fact is that that even when OCA-E spends relatively little time on the line search, approximation techniques such as OCA-T may offer significant benefits.

5 Conclusion and Future Work

In this paper, we have observed that the exact line search used to compute the best-so-far solution $\mathbf{w}_t^b$ at each iteration of the OCA algorithm can be computationally costly. Our modifications reduce time spent solving the reduced optimization problem by approximately 77% on COVTYPE, MNIST, and RCV1. This modification reduces training time using optimal parameter settings by approximately 41%, 12%, and 10%, respectively. On the very sparse data set NEWS20, OCA-T performs comparably to the original OCA-E. In general, the advantages of OCA-T are most significant on data sets with fewer features, since these are the situations when the line search dominates the OCA-E training process. This suggests that OCA-T would compare even more favorably to OCA-E after performing feature selection on the unfiltered data sets used in this report.

Data Set	Features	Search %	Speedup
MNIST	780	22.3	1.12
	473	30.4	1.27
	337	44.3	1.42
	139	64.3	1.83
RCV1	47,236	11.7	1.11
	4,373	36.3	1.42
	676	49.3	1.56
	186	62.7	1.94

Fig. 7: For both MNIST and RCV1, as the number of features decreases, the percentage of OCA-E training time spent on the exact line search increases. Correspondingly, the speedup offered by OCA-T increases.

Even on high-dimensional data sets, however, OCA-T can offer advantages. While performance on the 47,236-feature RCV1 is one such example, even more noteworthy are the cases where OCA-T converges in fewer iterations than OCA-E. The somewhat surprising result that an inexact search can produce faster convergence than an exact one is explained primarily by the fact that OCA-T avoids selecting $\mu = 0$. This finding suggests that the greedy approach of reducing the objective function by as much as possible at each iteration can yield far-from-optimal performance. An open question for future investigation is *which* greedy choices are problematic and whether there are better search methods than the approaches that have so far been employed.

Acknowledgement

This research was conducted with partial support from NSF grant ARRA 0851783. Nick Arnosti was a research intern at UCCS, funded by this grant, during the summer of 2010.

References

[1] T. Joachims, "Training linear SVMs in linear time," in *ACM SIGKDD*, 2006, pp. 217–226.
[2] V. Franc and S. Sonnenburg, "Optimized cutting plane algorithm for support vector machines," in *ICML*, 2008, pp. 320–327.
[3] ——, "Optimized cutting plane algorithm for large-scale risk minimization," *J. of Mach. Learn. Res.*, vol. 10, pp. 2157–2192, 2009.
[4] E. Mayoraz and E. Alpaydin, "Support vector machines for multi-class classification," in *LNCS, Engineering Applications of Bio-Inspired Artificial Neural Networks*. Springer, 1999, pp. 833–842.
[5] C. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.
[6] C. Hsu and C. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.
[7] T. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *J. of AI Res.*, vol. 2, no. 263, p. 286, 1995.
[8] J. Weston and C. Watkins, "Support vector machines for multi-class pattern recognition," in *7th European Symp. on Artificial Neural Networks*, vol. 4, no. 6, 1999, pp. 219–224.
[9] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *J. of Mach. Learn. Res.*, vol. 2, pp. 265–292, 2002.
[10] C. H. Teo, A. Smola, S. V. Vishwanathan, and Q. V. Le, "A scalable modular convex solver for regularized risk minimization," in *ACM SIGKDD*, 2007, pp. 727–736.
[11] S. Keerthi, S. Sundararajan, K. Chang, C. Hsieh, and C. Lin, "A sequential dual method for large scale multi-class linear SVMs," in *ACM SIGKDD*, 2008, pp. 408–416.
[12] S. Shalev-Shwartz and N. Srebro, "Svm optimization: inverse dependence on training set size," in *ICML*, 2008, pp. 928–935.
[13] T. Joachims, T. Finley, and C. Yu, "Cutting-plane training of structural SVMs," *Machine Learning*, vol. 77, no. 1, pp. 27–59, 2009.
[14] A. Bordes, L. Bottou, and P. Gallinari, "Sgd-qn: Careful quasi-newton stochastic gradient descent," *J. Mach. Learn. Res.*, vol. 10, pp. 1737–1754, December 2009.
[15] C. Lin, C. Chang, and C. Hsu, "A practical guide to support vector classification," National Taiwan U., www.csie.ntu.edu.tw/cjlin/papers/guide/guide.pdf, 2003.
[16] H. Lei and V. Govindaraju, "Half-against-half multi-class support vector machines," in *LNCS, Multiple Classifier Systems*. Springer, 2005, pp. 156–164.