Satisfying Real-world Goals with Dataset Constraints

Andrew Cotter[1], Michael Friedlander[2], Gabriel Goh[2], and Maya Gupta[1]

[1]Google Research
[2]Department of Mathematics, UC Davis

June 27, 2016

Abstract

The goal of minimizing misclassification error on a training set is often just one of several real-world goals that might be defined on different datasets. For example, one may require a classifier to also make positive predictions at some specified rate for some subpopulation (fairness), or to achieve a specified empirical recall. Other real-world goals include reducing churn with respect to a previously deployed model, or stabilizing online training. In this paper we propose handling multiple goals on multiple datasets by training with dataset constraints, using the ramp penalty to accurately quantify costs, and present an efficient algorithm to approximately optimize the resulting non-convex constrained optimization problem. Experiments on both benchmark and real-world industry datasets demonstrate the effectiveness of our approach.

1 Real-world goals

We consider a broad set of design goals and issues important for making classifiers work well in real-world applications, ranging from the standard precision and recall, to some new proposals. The key theme is that these goals can be expressed in terms of the positive and negative classification rates on multiple datasets, as we show in Table 1.

Coverage: One may wish to control how often a classifier predicts the positive (or negative) class. For example, one may want to ensure that only 10% of customers are selected to receive a printed catalog due to budget constraints, or perhaps to compensate for a biased training set. In practice, constraining the "coverage rate" (the expected proportion of positive predictions) is often easier than measuring e.g. accuracy or precision because coverage can be computed on unlabeled data—labeling data can be expensive, but acquiring a large number of unlabeled examples is often very easy.

Coverage was also considered by Mann and McCallum [2007], who proposed what they call "label regularization", in which one adds a regularizer that penalizes the relative entropy between the mean classifier score for each class and the desired distribution, with an additional correction to avoid degeneracies.

Churn: Work does not stop once a machine learning model has been adopted. There will be new training data, improved features, and potentially new model structures. Hence, in practice, one will deploy a *series* of models, each improving slightly upon the last. In this setting, determining whether each candidate should be deployed is surprisingly challenging: if we evaluate on the *same* held-out testing set every time a new candidate is proposed, and deploy it if it outperforms its predecessor, then every compare-and-deploy decision will increase the statistical dependence between the deployed model and the testing dataset, causing the model sequence to fit the originally-independent testing data. This problem is magnified if, as is typical, the candidate models tend to disagree only on a relatively small number of examples near the true decision boundary. For example, with a fixed test set of 10 000 random examples, only 100 may be near to the decision boundary, so the risk of the model sequence fitting these 100 examples is heightened.

A simple and safe solution is to draw a *fresh* testing sample every time one wishes to compare two models in the sequence, only considering examples on which the two models disagree. Because labeling data is expensive, one would like these freshly sampled testing datasets to be as small as possible. It is here that the problem of "churn" arises. Imagine that model A, our deployed model, is 70% accurate, and that model B, our candidate, is 75% accurate. In the best case, only 5% of test samples would be labeled differently, and all differences would be "wins" for classifier B. Then only a dozen or so examples would need to be labeled in order to establish that B is the statistically significantly better classifier with 95% confidence. In the worst case, model A would be correct and model B is incorrect 25% of the time, model B correct and model A incorrect 30% of the time, and both models correct the remaining 45% of the time. Then 55% of testing examples will be labeled differently, and closer to 1000 examples would need to be labeled to determine that model B is better.

We define the "churn rate" as the expected proportion of examples on which the prediction of the model being considered (model B above) differs from that of a baseline model (model A). During training, we propose constraining the empirical churn rate with respect to a given baseline model on a large unlabeled dataset.

Stability: A special case of minimizing churn is to ensure stability of an online classifier as it evolves, by constraining it to not deviate too far from a trusted classifier on a large held-out unlabeled dataset.

Fairness: A practitioner may be required to guarantee *fairness* of a learned classifier, in the sense that it makes positive predictions for members of different subgroups at certain rates. For example, one might require that housing loans be given equally to people of different genders. As noted by Zafar et al. [2015], fairness is sometimes specified by a proportion, such as the 80% rule in US law that certain decisions must be in favor of group B individuals at least 80% as often as in favor of group A individuals [e.g. Biddle, 2005, Vuolo and Levy, 2013]. Zafar et al. [2015] propose learning fair classifiers by imposing linear constraints on the covariance between the predicted labels and the values of certain features. In our framework, rate constraints such as the 80% rule can be imposed directly.

Recall and Precision: Requirements of real-world classifiers are often expressed in terms of precision and recall, especially when examples are highly imbalanced between positives and negatives. In our framework, we can handle this problem via Neyman-Pearson classification [e.g. Scott and Nowak, 2005, Davenport et al., 2010], in which one seeks to minimize the false negative rate subject to a constraint on the false positive rate. Indeed, our ramp-loss formulation is equivalent to that of Gasso et al. [2011] in this setting.

Egregious Examples: For certain classification applications, examples may be discovered that are particularly embarrassing if classified incorrectly. One standard approach to handling such examples is to increase their weights during training, but this is difficult to get right, because too large a weight may distort the classifier too much in the surrounding feature space, whereas too small a weight may not fix the problem. Worse, over time the dataset will often be augmented with new training examples and new features, causing the ideal weights to drift. We propose instead simply adding a constraint ensuring that some proportion of a set of such egregious examples is correctly classified. Such constraints should be used with extreme care: if too many are added then the problem may become infeasible.

2 Optimization problem

A key aspect of many of the goals of Section 1 is that they are defined on different datasets. For example, we might seek to maximize the accuracy of our classifier on a set of labeled examples drawn in some biased manner, require that its recall be at least 90% on 50 small datasets sampled in an unbiased manner from 50 different countries, desire low churn relative to a deployed classifier on a large unbiased unlabeled dataset, and require that 100 given egregious examples be classified correctly.

Another characteristic common to the metrics of Section 1 is that they can be expressed in terms of the positive and negative classification rates on various datasets, where for simplicity, we treat all datasets as unlabeled, as described in Table 1. We'll restrict our attention to the problem of learning a *linear* classification function $f(x) = \langle w, x \rangle - b$ parameterized by a weight vector $w \in \mathbb{R}^d$ and bias $b \in \mathbb{R}$, for which these rates are:

Table 1: Dataset notation.

Notation	Dataset
D	Any dataset
D^+, D^-	Sets of examples labeled positive/negative, respectively
$D^{++}, D^{+-}, D^{-+}, D^{--}$	Sets of examples with ground-truth positive/negative labels, and for which a baseline classifier makes positive/negative predictions
D^A, D^B	Sets of examples belonging to category A and B, respectively

Table 2: The quantities discussed in Section 1, expressed in the notation used in Problem 2, with the dependence on w and b dropped for notational simplicity, and using the dataset notation of Table 1.

Metric	Expression
Coverage rate	$s_p\left(D\right)$
#TP, #TN, #FP, #FN	$\left\lvert D^+\right\rvert s_p\left(D^+\right)$, $\left\lvert D^-\right\rvert s_n\left(D^-\right)$, $\left\lvert D^-\right\rvert s_p\left(D^-\right)$, $\left\lvert D^+\right\rvert s_n\left(D^+\right)$
#Errors	#FP + #FN
Error rate	$\#\text{Errors}/\left(\left\lvert D^+\right\rvert + \left\lvert D^-\right\rvert\right)$
Recall	$\#\text{TP}/\left(\#\text{TP} + \#\text{FN}\right) = \#\text{TP}/\left\lvert D^+\right\rvert$
#Changes	$\left\lvert D^{+-}\right\rvert s_p\left(D^{+-}\right) + \left\lvert D^{-+}\right\rvert s_n\left(D^{-+}\right) + \left\lvert D^{+-}\right\rvert s_p\left(D^{+-}\right) + \left\lvert D^{-+}\right\rvert s_n\left(D^{-+}\right)$
Churn rate	$\#\text{Changes}/\left(\left\lvert D^{++}\right\rvert + \left\lvert D^{+-}\right\rvert + \left\lvert D^{-+}\right\rvert + \left\lvert D^{--}\right\rvert\right)$
Fairness constraint	$s_p\left(D^A\right) \geq \kappa s_p\left(D^B\right)$, where $\kappa \in [0, 1]$
Egregious example constraint	$s_p\left(D^+\right) \geq \kappa$ and/or $s_n\left(D^-\right) \leq \kappa$ for a dataset D of egregious examples, where $\kappa \in [0, 1]$

$$s_p\left(D; w, b\right) = \tfrac{1}{\left\lvert D\right\rvert}\textstyle\sum_{x \in D} \mathbf{1}\left(\left\langle w, x\right\rangle - b\right), \qquad s_n\left(D; w, b\right) = s_p\left(D; -w, -b\right) \tag{1}$$

where $\mathbf{1}$ is an indicator function that is 1 if its argument is positive, 0 otherwise. In words, $s_p(D; w, b)$ and $s_n(D; w, b)$ denote the proportion of positive or negative predictions, respectively, that f makes on D. Table 2 specifies how the metrics of Section 1 can be expressed in terms of the s_ps and s_ns.

We propose handling these goals by minimizing an ℓ^2-regularized positive linear combination of prediction rates on different datasets, subject to upper-bound constraints on other positive linear combinations of such prediction rates:

$$\begin{aligned} \underset{w \in \mathbb{R}^d, b \in \mathbb{R}}{\text{minimize}} \quad & \textstyle\sum_{i=1}^k \left(\alpha_i^{(0)} s_p(D_i; w, b) + \beta_i^{(0)} s_n(D_i; w, b)\right) + \frac{\lambda}{2}\left\|w\right\|_2^2 \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^k \left(\alpha_i^{(j)} s_p(D_i; w, b) + \beta_i^{(j)} s_n(D_i; w, b)\right) \leq \gamma^{(j)} \quad j \in \{1, \ldots, m\} \end{aligned} \tag{2}$$

where λ is the parameter on the ℓ^2 regularizer, there are k unlabeled datasets $D_1, \ldots, D_k$ and m constraints. The metrics minimized by the objective and bounded by the constraints are specified via the choices of the nonnegative coefficients $\alpha_i^{(0)}$, $\alpha_i^{(j)}$, $\beta_i^{(0)}$ and $\beta_i^{(j)}$ for the ith dataset and, where applicable, the jth constraint—a user should base these choices on Table 2. Note that because $s_p + s_n = 1$, it is possible to transform *any* linear combination of rates into an equivalent positive linear combination, plus a constant (see Appendix A[1] for an example).

We cannot optimize Problem 2 directly because the rate functions s_p and s_n are discontinuous. We can, however, work around this difficulty by following Cotter et al. [2013] and training a classifier that makes

[1]Appendices may be found in the supplementary material

Algorithm 1 Proposed majorization-minimization procedure for (approximately) optimizing the ramp version of Problem 2. Starting from an initial feasible solution $w^{(0)}$, b_0, we repeatedly find a convex upper bound problem that is tight at the current candidate solution, and optimize it to yield the next candidate. See Section 2.1 for details, and Section 2.2 for how one can perform the inner optimizations on line 3.

MajorizationMinimization $\left(w^{(0)}, b_0, T\right)$

1 For $t \in \{1, 2, \ldots, T\}$

2 Construct an instance of Problem 3 with $w' = w^{(t-1)}$ and $b' = b_{t-1}$

3 Optimize this convex optimization problem to yield $w^{(t)}$ and b_t

4 Return $w^{(t)}$, b_t

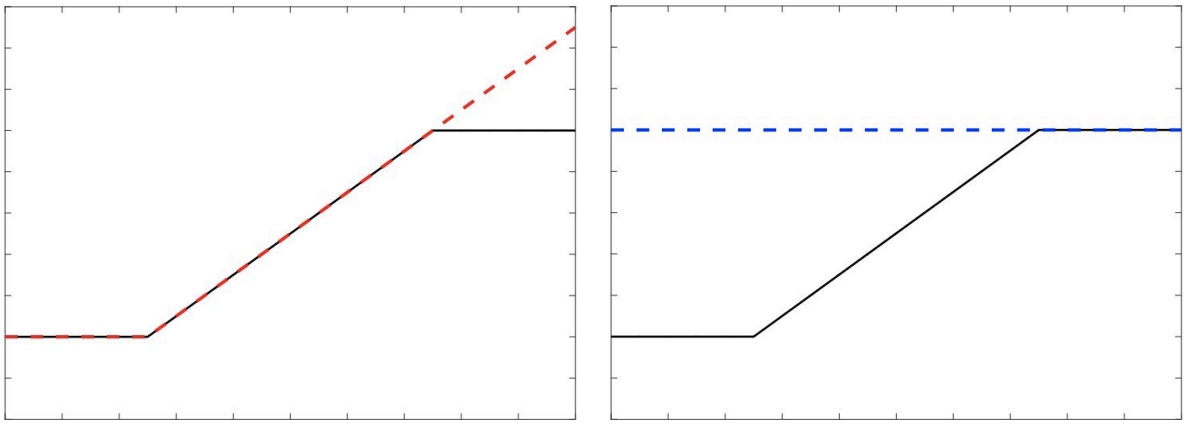

Figure 1: Convex upper bounds on the ramp function $\sigma(z) = \max\{0, \min\{1, 1/2 + z\}\}$. Notice that the hinge bound is tight for all $z \leq 1/2$, and the constant bound is tight for all $z \geq 1/2$.

randomized predictions based on the ramp function [Collobert et al., 2006]:

$$\sigma(z) = \max\{0, \min\{1, 1/2 + z\}\}$$

where the randomized classifier parameterized by w and b will make a positive prediction on x with probability $\sigma\left(\langle w, x\rangle - b\right)$, and will make a negative prediction otherwise. We can then define the *expected* positive and negative rates (with the expectation being taken w.r.t. the randomness of the classifier) on an unlabeled dataset D as:

$$r_p\left(D; w, b\right) = \tfrac{1}{|D|}\textstyle\sum_{x \in D} \sigma\left(\langle w, x\rangle - b\right), \qquad r_n\left(D; w, b\right) = r_p\left(D; -w, -b\right)$$

Substituting these expected rates for s_p and s_n gives a continuous (but non-convex) problem that we will henceforth refer to as the "ramp version" of Problem 2.

2.1 Optimizing the ramp problem

To address the non-convexity of the ramp version of our problem, we will iteratively optimize approximations, by, starting from an initial candidate solution, constructing a convex optimization problem upper-bounding the ramp version of Problem 2 that is *tight* at the current candidate, optimizing this convex problem to yield the next candidate, and repeating.

Our choice of a ramp for σ makes finding such tight convex upper bounds easy: both the hinge function $\max\{0, 1/2 + z\}$ and constant-1 function upper bound σ, with the former being tight for all $z \leq 1/2$, and the

latter for all $z \geq 1/2$ (see Figure 1). We'll therefore define the following upper bounds on σ and $1-\sigma$, with the additional parameter z' determining which of the two bounds (hinge or constant) will be used, such that the bounds will always be tight for $z = z'$:

$$\check{\sigma}_p(z; z') = \begin{cases} \max\{0, 1/2 + z\} & \text{if } z' \leq 1/2 \\ 1 & \text{otherwise} \end{cases}, \qquad \check{\sigma}_n(z; z') = \check{\sigma}_p(-z; -z')$$

Based upon these we define the following upper bounds on the expected rates:

$$\check{r}_p(D; w, b; w', b') = \frac{1}{|D|} \sum_{x \in D} \check{\sigma}_p(\langle w, x \rangle - b; \langle w', x \rangle - b')$$
$$\check{r}_n(D; w, b; w', b') = \frac{1}{|D|} \sum_{x \in D} \check{\sigma}_n(\langle w, x \rangle - b; \langle w', x \rangle - b')$$

which have the properties that both $\check{r}_p$ and $\check{r}_n$ are convex in w and b, are upper bounds on the original ramp-based rates:

$$\check{r}_p(D; w, b; w', b') \geq r_p(D; w, b) \qquad \text{and} \qquad \check{r}_n(D; w, b; w', b') \geq r_n(D; w, b)$$

and are tight at w', b':

$$\check{r}_p(D; w', b'; w', b') = r_p(D; w', b') \qquad \text{and} \qquad \check{r}_n(D; w', b'; w', b') = r_n(D; w', b')$$

Substituting these bounds into the ramp version of Equation 2 yields:

$$\begin{aligned} \underset{w \in \mathbb{R}^d, b \in \mathbb{R}}{\text{minimize}} \quad & \sum_{i=1}^k \left(\alpha_i^{(0)} \check{r}_p(D_i; w, b; w', b') + \beta_i^{(0)} \check{r}_n(D_i; w, b; w', b') \right) + \tfrac{\lambda}{2} \|w\|_2^2 \\ \text{s.t.} \quad & \sum_{i=1}^k \left(\alpha_i^{(j)} \check{r}_p(D_i; w, b; w', b') + \beta_i^{(j)} \check{r}_n(D_i; w, b; w', b') \right) \leq \gamma^{(j)} \quad j \in \{1, \ldots, m\} \end{aligned} \tag{3}$$

As desired, this problem upper bounds the ramp version of Problem 2, is tight at w', b', and is convex (because we only permit *positive* linear combinations of rates, and any positive linear combination of convex functions is convex).

Algorithm 1 contains our proposed (approximate) optimization procedure for solving Problem 3. Given an initial feasible solution, it's straightforward to verify inductively, using the fact that we construct tight convex upper bounds at every step, that every convex subproblem will have a feasible solution, every $w^{(t)}, b_t$ will be feasible w.r.t. the ramp version of Problem 2, and every pair $(w^{(t+1)}, b_{t+1})$ will have an objective function value that is no higher that that of $(w^{(t)}, b_t)$. In other words, no iteration can make negative progress.

The non-convexity of the ramp version of Problem 2 will cause Algorithm 1 to arrive at a suboptimal solution that depends on the initial $(w^{(0)}, b_0)$.

2.2 Optimizing the convex subproblems

Our approach for optimizing the constrained convex subproblems is based on the idea of adding Lagrange multipliers v over the constraints in Problem 3 to obtain the equivalent unconstrained problem:

$$\underset{v \geq 0}{\text{maximize}} \quad \digamma(v) = \min_{w,b} \Psi(w, b, v; w', b') \tag{4}$$

where the function:

$$\begin{aligned} \Psi(w, b, v; w', b') = & \sum_{i=1}^k \left(\left(\alpha_i^{(0)} + \sum_{j=1}^m v_j \alpha_i^{(j)} \right) \check{r}_p(D_i; w, b; w', b') \right. \\ & \left. + \left(\beta_i^{(0)} + \sum_{j=1}^m v_j \beta_i^{(j)} \right) \check{r}_n(D_i; w, b; w', b') \right) + \tfrac{\lambda}{2} \|w\|_2^2 - \sum_{j=1}^m v_j \gamma^{(j)} \end{aligned} \tag{5}$$

is convex in w and b, and concave in the multipliers v. For the purposes of this section, w' and b' are fixed constants.

Algorithm 2 Skeleton of a cutting-plane algorithm that solves Problem 4 to within ϵ. Here, $\mathcal{V} \subseteq \mathbb{R}^m$ is compact and convex and $l_0, u_0 \in \mathbb{R}$ are finite with $l_0 \leq \max_v \digamma(v) \leq u_0$. There are several options for the CutChooser function on line 8—please see Appendix C for details. The SVMOptimizer function returns $w^{(t)}$ and b_t approximately minimizing $\Psi(w, b, v^{(t)}; w', b')$, and a lower bound $l_t \leq \digamma(v)$ for which $u_t - l_t \leq \epsilon_t$ for u_t as defined on line 9.

CuttingPlane $(l_0, u_0, \mathcal{V}, \epsilon)$
1 Initialize $g^{(0)} \in \mathbb{R}^m$ to the all-zero vector
2 For $t \in \{1, 2, \dots\}$
3 Let $h_t(v) = \min_{s \in \{0,1,\dots,t-1\}} \left(u_s + \left\langle g^{(s)}, v - v^{(s)} \right\rangle\right)$
4 Let $L_t = \max_{s \in \{0,1,\dots,t-1\}} l_s$ and $U_t = \max_{v \in \mathcal{V}} h_t(v)$
5 If $U_t - L_t \leq \epsilon$ then
6 Let $s \in \{1, \dots, t-1\}$ be an index maximizing l_s
7 Return $w^{(s)}$, b_s, $v^{(s)}$
8 Let $v^{(t)}, \epsilon_t = \text{CutChooser}(h_t, L_t)$
9 Let $w^{(t)}, b_t, l_t = \text{SVMOptimizer}\left(v^{(t)}, L_t, h_t\left(v^{(t)}\right), \epsilon_t\right)$
10 Let $u_t = \Psi(w^{(t)}, b_t, v^{(t)}; w', b')$ and $g^{(t)} = \nabla_v \Psi(w^{(t)}, b_t, v^{(t)}; w', b')$

The key insight is that evaluating $\digamma(v)$ is, thanks to our use of hinge and constant upper-bounds on our ramp σ, equivalent to optimization of a support vector machine (SVM) with per-example weights—see Appendix D for details. This observation enables us to solve the saddle system in an inside-out manner. On the "inside", we optimize over (w, b) for a fixed v using an off-the-shelf SVM solver. On the "outside", the resulting (w, b)-optimizer is used as a component in an outer optimization over v. Notice that this outer optimization is very low-dimensional, since $v \in \mathbb{R}^m$, where m is the number of constraints.

Algorithm 2 contains a skeleton of the cutting-plane algorithm that we use for this outer optimization over v. Because this algorithm is intended to be used as an outer loop in a nested optimization routine, it does not expect that $\digamma(v)$ can be evaluated or differentiated exactly. Rather, it's based upon the idea of possibly making "shallow" cuts [Bland et al., 1981] by choosing a desired accuracy ϵ_t at each iteration, and expecting the SVMOptimizer to return a solution with suboptimality ϵ_t. More precisely, the SVMOptimizer function approximately evaluates $\digamma(v^{(t)})$ for a given fixed $v^{(t)}$ by constructing the corresponding SVM problem and finding a $(w^{(t)}, b_t)$ for which the primal and dual objective function values differ by at most ϵ_t.

After finding $(w^{(t)}, b_t)$, the SVMOptimizer then evaluates the dual objective function value of the SVM to determine l_t. The primal objective function value u_t and its gradient $g^{(t)}$ w.r.t. v (calculated on line 9 of Algorithm 2) define the cut $u_t + \left\langle g^{(t)}, v - v^{(t)} \right\rangle$. Notice that since $\Psi(w^{(t)}, b_t, v; w', b')$ is a linear function of v, it is equal to this cut function, which therefore upper-bounds $\min_{w,b} \Psi(w, b, v; w', b')$.

One advantage of this cutting-plane formulation is that typical CutChooser implementations will choose ϵ_t to be large in the early iterations, and will only shrink it to be ϵ or smaller once we're close to convergence. We leave the details of the analysis to the appendices—a summary can be found in Appendix E.

3 Related work

One strategy to satisfy *some* of the goals described in Section 1 is to first train an unconstrained classifier, and then adjust the decision threshold to satisfy the goals as a second step. Another standard approach is to reweight groups of examples as a preprocessing step—notice that, in our formulation, the Lagrange multipliers v play an identical role to such weights, with the difference being that they are dynamically chosen so as to satisfy constraints, rather than being provided by the user.

Collobert et al. [2006] also use a ramp loss as a relaxation of 0/1 loss (for optimizing accuracy), but do not consider constraints. The most related prior work is that of Gasso et al. [2011]. They also use an iterative ramp loss approximation, but only tackle the Neyman-Pearson problem, and their algorithm is less straightforward than that presented here. Gasso et al. [2011] compared their Neyman-Pearson classifier

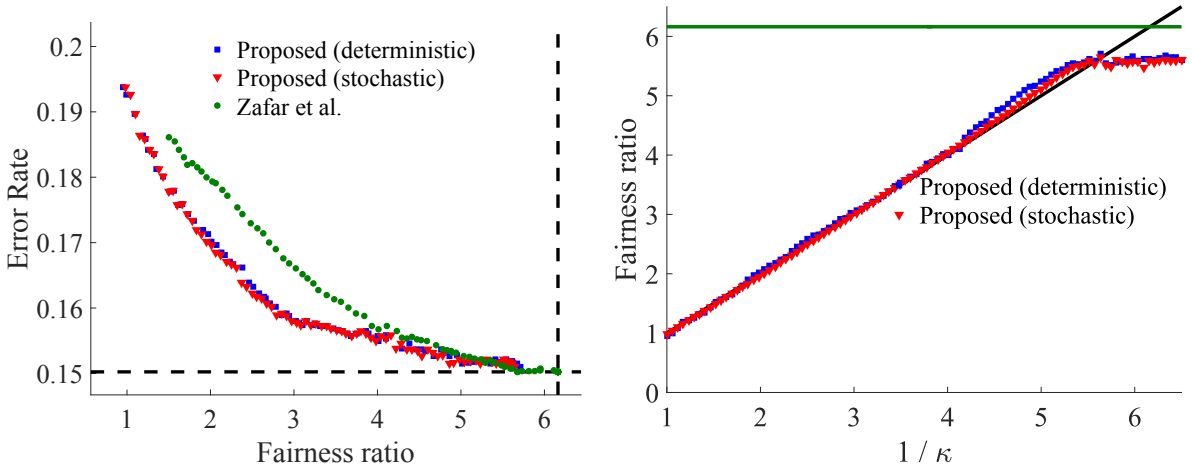

Figure 2: Blue dots: our proposal, with the classification functions' predictions being deterministically thresholded at zero. Red dots: same, but using the randomized classification rule described in Section 2. Green dots: Zafar et al. [2015]. Green line: unconstrained SVM. **(Left)** Test set error plotted vs. observed test set fairness ratio $s_p\left(D^M\right)/s_p\left(D^F\right)$. **(Right)** The $1/\kappa$ hyper-parameter used to specify the desired fairness in the proposed method, and the observed fairness ratios of our classifiers on the test data.

to that of Davenport et al. [2010], which differs in that it uses a hinge loss approximation instead of the ramp loss, and found with the ramp-loss they achieved similar or slightly better results with up to 10× less computation.

Narasimhan et al. [2015] considered optimizing for the F-measure and other quantities that can be written as concave functions of the TP and TN. In their proposed stochastic dual solver, they adaptively linearize concave functions of the rate functions (Equation 1). Joachims [2005] indirectly optimizes upper-bounds on functions of $s_p(D^+)$, $s_p(D^-)$, $s_n(D^+)$, $s_n(D^-)$ using a hinge loss approximation.

4 Experiments

We demonstrate the accuracy of the proposed algorithm for satisfying these goals on a benchmark dataset for fairness, and a real-world problem with churn and recall constraints.

4.1 Fairness

We compare training for fairness on the benchmark Adult dataset [2], the same dataset used by Zafar et al. [2015]. The 32 561 training and 16 281 testing examples, derived from the 1994 Census, are 123-dimensional and sparse. Each feature contains categorical attributes such as race, gender, education levels and relationship status. A positive class label means that individual's income exceeds 50k. Let D^M and D^F denote the sets of male and female examples. The number of positive labels in D^M is roughly six times that of D^F. The goal is to train a classifier that respects the fairness constraint $s_p\left(D^M\right) \leq s_p\left(D^F\right)/\kappa$. for a parameter $\kappa \in (0, 1]$ (where $\kappa = 0.8$ corresponds to the 80% rule mentioned in Section 1).

Our publicly-available `Julia` implementation[3] for these experiments uses `LIBLINEAR` [Fan et al., 2008] to implement the SVMOptimizer function, and does not include an unregularized bias b. The outer optimization over v does not use the m-dimensional cutting plane algorithm of Algorithm 2, instead using a simpler one-dimensional variant (observe that these experiments involve only one constraint).

[2] "a9a" from `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

[3] Link redacted for blind review

We compare to the method of Zafar et al. [2015], which proposed handling fairness with the constraint:

$$\langle w, \bar{x} \rangle \leq c, \qquad \bar{x} = \left|D^M\right|^{-1} \sum_{x \in D^M} x \; - \; \left|D^F\right|^{-1} \sum_{x \in D^F} x. \tag{6}$$

We optimized an SVM subject to this constraint (see Appendix B for details), for a range of c values.

Results in Figure 2 show the proposed method is much more accurate for any desired fairness, and achieves fairness ratios not reachable with the approach of Zafar et al. [2015] for any choice of c. It is also easier to control: the values of c in Zafar et al. [2015] do not have a clear interpretation, whereas κ is an effective proxy for the fairness ratio.

4.2 Churn

Our second set of experiments demonstrates meeting real-world requirements on a proprietary problem from a large internet services company : predicting whether a user interface element should be shown to a user, based on 31 informative features. We are given a currently-deployed model, which we refer to as the *baseline*. The goals are threefold: train a classifier that (i) has high accuracy, (ii) has no worse recall than the baseline, and (iii) has low churn w.r.t. the baseline.

We are given three datasets, D_1, D_2 and D_3, consisting of 131 840, 53 877 and 68 892 examples, respectively. The datasets D_1 and D_2 are hand-labeled, while D_3 is unlabeled. In addition, D_1 was chosen via active sampling, while D_2 and D_3 are sampled *i.i.d.* from the underlying data distribution. For all three datasets, we split out 80% for training and reserved 20% for testing. We address the three goals in the proposed framework by simultaneously training the classifier to minimize the number of errors on D_1 plus the number of false positives on D_2, subject to the constraints that the recall on D_2 be at least as high as the baseline recall (we're essentially performing Neyman-Pearson classification on D_2), and that the churn w.r.t. the baseline on D_3 be no larger than a given target parameter.

We found that 31-dimensional linear models were not capable of outperforming the baseline model. Instead, we use a fixed feature transformation Φ that maps each x to a roughly 30 000-dimensional feature vector, and train classifiers that are linear with respect to $\Phi(x)$.

These experiments use a proprietary `C++` implementation of Algorithm 2, using the combined SDCA and cutting plane approach of Appendix D to implement the inner optimizations over w and b, with the CutChooser helper functions being as described in Appendices C.1 and D.2.1. We performed 5 iterations of the majorization-minimization procedure of Algorithm 1.

The results in Figure 3 show the achieved churn and error rates on train and test sets plotted over a range of churn constraint values (blue line), compared to training an SVM without constraints and then changing its decision threshold to achieve the desired recall (green line). When using deterministic thresholding of the learned classifier (the blue curves, which significantly outperformed randomized classification–the red curves–in this experiment), the proposed method achieves lower churn and better accuracy for all targeted churn rates, while also meeting the recall constraint.

As expected, the empirical churn is extremely close to the targeted churn on the training set for the stochastic relaxation (red dashed line, and black line, top left plot), but less so on the 20% held-out test set (top right plot). We hypothesize this disparity is due to overfitting, as the classifier has 30 000 parameters, and D_3 is relatively small. However, except for the lowest targeted churn, the actual classifier churn (blue line, top plots) is substantially lower than the targeted churn.

References

K. Ball. An elementary introduction to modern convex geometry. *Flavors of Geometry*, 31:1–58, 1997.

D. Biddle. *Adverse Impact and Test Validation: A Practitioner's Guide to Valid and Defensible Employment Testing*. Gower, 2005.

R. G. Bland, D. Goldfarb, and M. J. Todd. Feature article—the ellipsoid method: A survey. *Operations Research*, 29(6):1039–1091, November 1981.

S. Boyd and L. Vandenberghe. Localization and cutting-plane methods, April 2011. Stanford EE 364b lecture notes.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

R. Collobert, F. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *ICML*, 2006.

A. Cotter, S. Shalev-Shwartz, and N. Srebro. Learning optimally sparse support vector machines. In *ICML*, pages 266–274, 2013.

M. Davenport, R. G. Baraniuk, and C. D. Scott. Tuning support vector machines for minimax and Neyman-Pearson classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

G. Gasso, A. Pappaionannou, M. Spivak, and L. Bottou. Batch and online learning algorithms for nonconvex Neyman-Pearson classification. *ACM Transactions on Intelligent Systems and Technology*, 2011.

B. Grünbaum. Partitions of mass-distributions and convex bodies by hyperplanes. *Pacific Journal of Mathematics*, 10(4):1257–1261, December 1960.

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

G. S. Mann and A. McCallum. Simple, robust, scalable semi-supervised learning with expectation regularization. In *ICML*, 2007.

H. Narasimhan, P. Kar, and P. Jain. Optimizing non-decomposable performance measures: a tale of two classes. In *ICML*, 2015.

A. Nemirovski. Lecture notes: Efficient methods in convex programming. 1994. URL `http://www2.isye.gatech.edu/~nemirovs/Lect_EMCO.pdf`.

C. D. Scott and R. D. Nowak. A Neyman-Pearson approach to statistical learning. *IEEE Transactions on Information Theory*, 2005.

S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss. *JMLR*, 14 (1):567–599, Feb. 2013.

S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. *Mathematical Programming*, 127(1):3–30, March 2011.

M. S. Vuolo and N. B. Levy. Disparate impact doctrine in fair housing. *New York Law Journal*, 2013.

M. B. Zafar, I. Valera, M. G. Rodriguez, and K. P. Gummadi. Fairness constraints: A mechanism for fair classification. In *ICML Workshop on Fairness, Accountability, and Transparency in Machine Learning*, 2015.

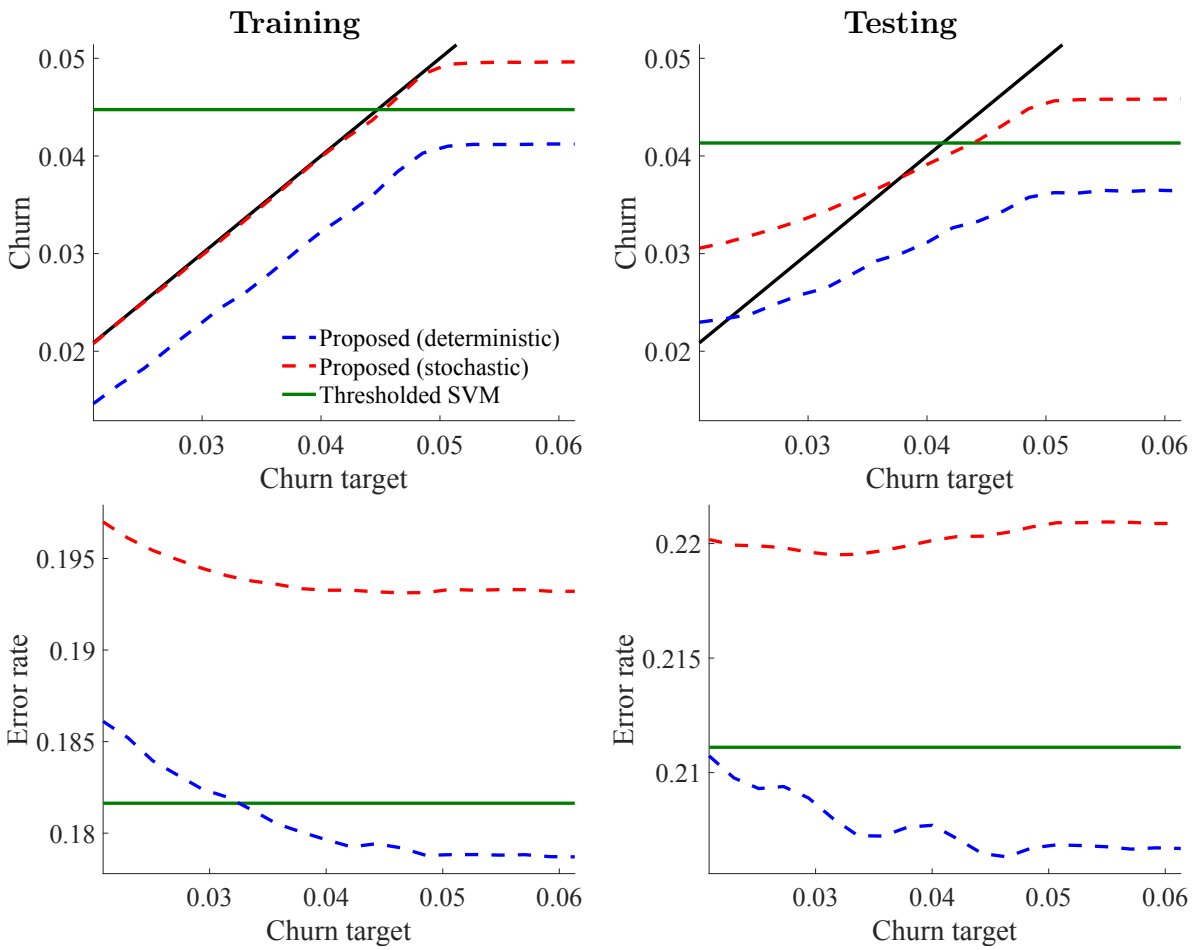

Figure 3: Blue: our proposal, with the classification functions' predictions being deterministically thresholded at zero. Red: same, but using the randomized classification rule described in Section 2. Green: unconstrained SVM trained on $D_1 \cup D_2$, then thresholded (by shifting the bias b) to satisfy the recall constraint on D_2. **(Top)** Observed churn (vertical axis) vs. the churn target used during training (horizontal axis), on the train (left) and test (right) splits of the unlabeled dataset D_3. **(Bottom)** Empirical error rates (vertical axis) vs the churn target, on the train (left) and test (right) splits of the union $D_1 \cup D_2$ of the two labeled datasets.

Table 3: Some ratio metrics (Appendix A), which are metrics that can be written as ratios of linear combinations of rates. #Wins and #Losses are actually linear combination metrics, but are needed for the other definitions.

Metric	**Expression**
Precision	$\#\text{TP}/\left(\#\text{TP}+\#\text{FP}\right)$
F score	$2\text{Precision}\cdot\text{Recall}/\left(\text{Precision}+\text{Recall}\right)$
#Wins	$\left\|D^{+-}\right\| s_p\left(D^{+-}\right)+\left\|D^{-+}\right\| s_n\left(D^{-+}\right)$
#Losses	$\left\|D^{++}\right\| s_n\left(D^{+-}\right)+\left\|D^{--}\right\| s_p\left(D^{-+}\right)$
Churn rate	$\#\text{Changes}/\left(\left\|D^{++}\right\|+\left\|D^{+-}\right\|+\left\|D^{-+}\right\|+\left\|D^{--}\right\|\right)$
Win/loss Ratio	#Wins/#Losses
Win/change Ratio	#Wins/#Changes

A Ratio metrics

Problem 2 minimizes an objective function and imposes upper-bound constraints on metrics that are written as linear combinations of positive and negative rates—we refer to such as "linear combination metrics". Some metrics of interest, however, cannot be written in this form. One important subclass are the so-called "ratio metrics", which are *ratios* of linear combinations of rates. Examples of ratio metrics are precision, F-score, win/loss ratio and win/change ratio (recall is a linear combination metric, since its denominator is a constant).

Ratio metrics may not be used directly in the objective of Problem 2, but can be included in constraints by multiplying through by the denominator, then shifting the constraint coefficients to be non-negative. For example, the constraint that precision must be greater than 90% can be expressed as follows:

$$\begin{aligned}
\left|D^{+}\right| s_p\left(D^{+}\right) &\geq 0.9\left(\left|D^{+}\right| s_p\left(D^{+}\right)+\left|D^{-}\right| s_p\left(D^{-}\right)\right)\\
0.1\left|D^{+}\right| s_p\left(D^{+}\right)-0.9\left|D^{-}\right| s_p\left(D^{-}\right) &\geq 0\\
-0.1\left|D^{+}\right| s_p\left(D^{+}\right)+0.9\left|D^{-}\right| s_p\left(D^{-}\right) &\leq 0\\
0.1\left|D^{+}\right| s_n\left(D^{+}\right)+0.9\left|D^{-}\right| s_p\left(D^{-}\right) &\leq 0.1\left|D^{+}\right|,
\end{aligned}$$

where we used the fact that $s_p\left(D^{+}\right)+s_n\left(D^{+}\right)=1$ on the last line—this is an example of a fact that we noted in Section 2: since positive and negative rates must sum to one, it is possible to write any linear combination of rates as a positive linear combination, plus a constant.

Multiplying through by the denominator is fine for the indicator version of Problem 2, but a natural question is whether, by using a stochastic classifier and optimizing the ramp version, we're doing the "right thing" in expectation. The answer is: only partly. Since the expectation of a ratio is not the ratio of expectations, e.g. a precision constraint in our original problem (Problem 2) becomes only a constraint on a precision-like quantity (the ratio of the expectations of the precision's numerator and denominator) in our relaxed problem.

B Summary Examples

We note that the constraints of Zafar et al. [2015] can be interpreted as a relaxation of the constraint $-c \leq s_p(D^A;w)-s_p(D^B;w)\leq c$ under the linear approximation

$$s_p(D;w,b)\approx\frac{1}{|D|}\sum_{x\in D}\left(\langle w,x\rangle-b\right)$$

giving:

$$s_p(D^A;w,b)-s_p(D^B;w,b)\approx\frac{1}{|D^A|}\sum_{x\in D^A}\left(\langle w,x\rangle-b\right)-\frac{1}{|D^B|}\sum_{x\in D^B}\left(\langle w,x\rangle-b\right)=\langle w,\bar{x}\rangle$$

where $\bar{x}$ is defined as in Equation 6. We can therefore simulate the approach of Zafar et al. [2015] within our framework by adding the constraints:

$$\langle w, \bar{x} \rangle \leq c \iff \max\{0, 1 - \langle w, \bar{x} \rangle\} \leq c + 1$$
$$c \leq \langle w, \bar{x} \rangle \iff \max\{0, 1 + \langle w, \bar{x} \rangle\} \leq c + 1$$

and solving the hinge constrained optimization problem described in Problem 3. Going further, we could implement these constraints as egregious examples using the constraint:

$$\langle w, \bar{x} \rangle \leq c \iff \left\langle w, \frac{1}{4c}\bar{x} \right\rangle \leq \frac{1}{4} \iff \frac{1}{2} + \left\langle w, \frac{1}{4c}\bar{x} \right\rangle \leq \frac{3}{4}$$
$$\iff \min\left\{ \max\left\{ \frac{1}{2} + \left\langle w, \frac{1}{4c}\bar{x} \right\rangle, 0 \right\}, 1 \right\} \leq \frac{3}{4} \iff r_p(\bar{x}) \leq \frac{3}{4}$$

permitting us to perform an analogue of their approximations in ramp form.

C Cutting plane algorithm

We'll now discuss some variants of Algorithm 2. To simplify the presentation, we'll define:

$$\psi(v) = \min_{w,b} \Psi(w, b, v; w', b')$$

as the function that we are attempting to maximize. We assume that:

- $\mathcal{V} \subseteq \mathbb{R}^m$ is compact and convex.
- $\psi : \mathcal{V} \to \mathbb{R}$ is concave.
- ψ has a (not necessarily unique) maximizer $v^* = \text{argmax}_{v \in \mathcal{V}} \psi(v)$.

C.1 Maximization-based

We're primarily interested in proving convergence rates, and will do so in Appendix C.2. With that said, there is one easy-to-implement variant of Algorithm 2 for which we have not proved a convergence rate, but that we use in some of our experiments due to its simplicity:

Definition 1. ***(Maximization-based Algorithm 2)*** *CutChooser chooses* $v^{(t)} = \text{argmax}_{v \in \mathcal{V}} h_t(v)$ *and* $\epsilon_t = (U_t - L_t)/2$.

Observe that this $v^{(t)}$ can be found at the same time as U_t is computed, since both result from optimization of the same linear program. However, despite the ease of implementing this variant, we have not proved any convergence rates about it.

C.2 Center of mass-based

We'll now discuss a variant of Algorithm 2 that chooses $v^{(t)}$ and ϵ_t based on the center of mass of the "superlevel hypograph" determined by h_t and L_t. The hypograph of h_t is the set of $m+1$-dimensional points (v, z) for which $z \leq h_t(z)$, and the intersection of the hypograph with the half-space containing all points for which $z \geq L_t$ is what we call the "superlevel hypograph". Notice that, in the context of Algorithm 2, the superlevel hypograph defined by h_t and L_t corresponds to the set of pairs of candidate maximizers and their possible function values at the tth iteration. Because this variant is based on finding a cut center in the $m+1$-dimensional hypograph, rather than an m-dimensional level set (which is arguably more typical), this is an instance of what Boyd and Vandenberghe [2011] call an "epigraph cutting plane method".

Throughout this section, we will take μ to be the Lebesgue measure (either 1-dimensional, m-dimensional, or $m+1$-dimensional, depending on context). We also must define some notation for dealing with superlevel sets and hypographs. For a concave $f : \mathcal{V} \to \mathbb{R}$ and $y \in \mathbb{R}$, define:

$$L(f, y) = \{v \in \mathcal{V} \mid f(v) \geq y\}$$

as the superlevel set of f at y. Further define:

$$G(f, y) = \{(v, z) \in \mathcal{V} \times \mathbb{R} \mid f(v) \geq z \geq y\}$$

as the superlevel hypograph of f above y. With these definitions in place, we're ready to explicitly state the center of mass-based rule for the CutChooser function on line 8 of Algorithm 2:

Definition 2. ***(Center of mass-based Algorithm 2)*** *CutChooser takes $(v^{(t)}, z_t)$ to be the center of mass of $S(h_t, L_t)$, and define $\epsilon_t = (z_t - L_t)/2$.*

Our final bit of "setup" before getting to our results is to state two classic theorems, plus a corollary, which will be needed for our proofs. The first enables us to interpolate the areas of superlevel sets:

Theorem 1. *Suppose that the superlevel sets of a concave $f : \mathcal{V} \to \mathbb{R}$ at y_1 and y_2 are nonempty, and take $\gamma \in [0, 1]$. Then:*

$$(\mu(L(f, \gamma y_1 + (1-\gamma) y_2)))^{1/m} \geq \gamma (\mu(L(f, y_1)))^{1/m} + (1-\gamma) (\mu(L(f, y_2)))^{1/m}$$

Proof. This is the Brunn-Minkowski inequality [e.g. Ball, 1997]. □

This theorem has the immediate useful corollary:

Corollary 1. *Suppose that $f : \mathcal{V} \to \mathbb{R}$ is concave with a maximizer $v^* \in \mathcal{V}$, and that $\delta \geq 0$. Then:*

$$\left(\frac{\delta}{m+1}\right) \mu(L(f, f(v^*) - \delta)) \leq \mu(G(f, f(v^*) + \delta)) \leq \delta \mu(L(f, f(v^*) - \delta))$$

Proof. By Theorem 1 (lower-bounding the second term on the RHS by zero), for $0 \leq z \leq \delta$:

$$\mu(L(f, f(v^*) - z)) \geq \left(\frac{z}{\delta}\right)^m \mu(L(f, f(v^*) - \delta))$$

From which integrating $\mu(G(f, f(v^*) - \delta)) = \int_0^\delta \mu(L(f, f(v^*) - z))\, m\mu(z)$ yields the claimed lower bound. The upper bound follows immediately from the fact that the superlevel sets shrink as z increases (i.e. $\mu(L(f, z')) \leq \mu(L(f, z))$ for $z' \geq z$). □

The second classic result enables us to bound how much "progress" is made by a cut based on the center of mass of a superlevel hypograph:

Theorem 2. *Suppose that $S \subseteq \mathbb{R}^m$ is a convex set. If we let $z \in S$ be the center of mass of S, then for any half-space $H \ni z$:*

$$\frac{\mu(S \cap H)}{\mu(S)} \geq \left(\frac{m}{m+1}\right)^m \geq \frac{1}{e}$$

Proof. This is Theorem 2 of Grünbaum [1960]. □

With the preliminaries out of the way, we're ready to move on to our first result: bounding the volumes of the superlevel hypographs of our h_ts, assuming that we base our cuts on the centers of mass of the superlevel hypographs:

Lemma 1. *In the context of Algorithm 2, suppose that we choose $v^{(t)}$ and ϵ_t as in Definition 2. Then:*

$$\mu\left(G\left(h_{t+1}, L_{t+1}\right)\right) \leq \left(1 - \frac{1}{2e}\right) \mu\left(G\left(h_t, L_t\right)\right)$$

from which it follows that:

$$\mu\left(G\left(h_t, L_t\right)\right) \leq \left(1 - \frac{1}{2e}\right)^{t-1} \left(u_0 - l_0\right) \mu\left(\mathcal{V}\right)$$

for all t.

Proof. We'll consider two cases: $u_t \leq z_t$ and $u_t > z_t$, corresponding to making a "deep" or "shallow" cut, respectively.

Deep cut case: If $u_t \leq z_t$, then the hyperplane $u_t + \left\langle g^{(t)}, v - v^{(t)} \right\rangle$ passes below the center of mass of $S(h_t, L_t)$, implying by Theorem 2 that:

$$\mu\left(G\left(h_{t+1}, L_{t+1}\right)\right) \leq \mu\left(G\left(h_{t+1}, L_t\right)\right) \leq \left(1 - \frac{1}{e}\right) \mu\left(G\left(h_t, L_t\right)\right)$$

Shallow cut case: Now suppose that $u_t > z_t$. Applying Theorem 2 to the level cut $\{(v, z) \mid z \leq z_t\}$ at z_t:

$$\begin{aligned}
\frac{1}{e}\mu\left(G\left(h_t, L_t\right)\right) &\leq \int_{L_t}^{z_t} \mu\left(\{v \in \mathcal{V} \mid h_t\left(v\right) \geq z\}\right) d\mu(z) \\
&\leq \int_{L_t}^{(z_t + L_t)/2} \mu\left(\{v \in \mathcal{V} \mid h_t\left(v\right) \geq z\}\right) d\mu(z) \\
&\quad + \int_{(z_t + L_t)/2}^{z_t} \mu\left(\{v \in \mathcal{V} \mid h_t\left(v\right) \geq z\}\right) d\mu(z)
\end{aligned}$$

Since h_t is concave, its superlevel sets shrink for larger z, so the first integral on the RHS above is larger than the second, implying that:

$$\frac{1}{2e}\mu\left(G\left(h_t, L_t\right)\right) \leq \int_{L_t}^{(z_t + L_t)/2} \mu\left(\{v \in \mathcal{V} \mid h_t\left(v\right) \geq z\}\right) d\mu(z)$$

The fact that $\epsilon_t = (z_t - L_t)/2$ implies that $l_t > (z_t + L_t)/2$, so $L_{t+1} > (z_t + L_t)/2$, and:

$$\frac{1}{2e}\mu\left(G\left(h_t, L_t\right)\right) \leq \int_{L_t}^{L_{t+1}} \mu\left(\{v \in \mathcal{V} \mid h_t\left(v\right) \geq z\}\right) d\mu(z)$$

showing that we will cut off at least a $1/2e$-proportion of the total volume, completing the proof of the first claim.

The second claim follows immediately by iterating the first, and observing that $\mu\left(G\left(h_1, L_1\right)\right) = \left(u_0 - l_0\right) \mu\left(\mathcal{V}\right)$. □

The above result shows that the volumes of the superlevel hypographs of the h_ts shrink at an exponential rate. However, our actual stopping condition (line 5 of Algorithm 2) depends not on the volume, but rather the "height" $U_t - L_t$, so we would prefer a bound on this height, rather than the volume. We find such a bound in the (proof of the) following lemma, which establishes how many iterations must elapse before the stopping condition is satisfied:

Lemma 2. *In the context of Algorithm 2, suppose that we choose $v^{(t)}$ and ϵ_t as in Definition 2. Then there is a iteration count T_ϵ satisfying:*

$$T_\epsilon = O\left(m \ln\left(\frac{u_0 - l_0}{\epsilon}\right) + \ln\left(\frac{\mu\left(\mathcal{V}\right)}{\mu\left(L\left(\psi, l_0\right)\right)}\right)\right)$$

such that, if $t \geq T_\epsilon$, then $U_t - L_t \leq \epsilon$. Hence, Algorithm 2 will terminate after T_ϵ iterations.

Proof. By Corollary 1:

$$\mu\left(G\left(h_t, L_t\right)\right) \geq \left(\frac{U_t - L_t}{m+1}\right) \mu\left(L\left(h_t, L_t\right)\right)$$

If $L_t \leq \psi\left(v^*\right) - \epsilon$, then $\mu\left(L\left(h_t, L_t\right)\right) \geq \mu\left(L\left(h_t, \psi\left(v^*\right) - \epsilon\right)\right)$ because h_t is concave. If $L_t > \psi\left(v^*\right) - \epsilon$, then by Theorem 1:

$$\mu\left(L\left(h_t, L_t\right)\right) \geq \left(\frac{U_t - L_t}{U_t - \psi\left(v^*\right) + \epsilon}\right)^m \mu\left(L\left(h_t, \psi\left(v^*\right) - \epsilon\right)\right)$$

In either case, $L_t \leq \psi\left(v^*\right)$ by definition, and we'll assume that $U_t - L_t > \epsilon$ (this will lead to a contradiction), so:

$$\mu\left(G\left(h_t, L_t\right)\right) \geq 2^{-m} \left(\frac{U_t - L_t}{m+1}\right) \mu\left(L\left(h_t, \psi\left(v^*\right) - \epsilon\right)\right)$$

Applying Lemma 1 yields that:

$$\left(1 - \frac{1}{2e}\right)^{t-1} \left(u_0 - l_0\right) \mu\left(\mathcal{V}\right) \geq 2^{-m} \left(\frac{U_t - L_t}{m+1}\right) \mu\left(L\left(h_t, \psi\left(v^*\right) - \epsilon\right)\right)$$

Next observe that, by Theorem 1:

$$\mu\left(L\left(h_t, \psi\left(v^*\right) - \epsilon\right)\right) \geq \left(\frac{U_t - \psi\left(v^*\right) + \epsilon}{U_t - l_0}\right)^m \mu\left(L\left(h_t, l_0\right)\right) \geq \left(\frac{\epsilon}{u_0 - l_0}\right)^m \mu\left(L\left(\psi, l_0\right)\right)$$

Combining the previous two equations gives:

$$U_t - L_t \leq \left(1 - \frac{1}{2e}\right)^{t-1} (m+1) \left(\frac{2}{\epsilon}\right)^m \left(u_0 - l_0\right)^{m+1} \left(\frac{\mu\left(\mathcal{V}\right)}{\mu\left(L\left(\psi, l_0\right)\right)}\right)$$

Simplifying this inequality yields that, if we have performed the claimed number of iterations, then $U_t - L_t \leq \epsilon$ (this contradicts our earlier assumption that $U_t - L_t > \epsilon$, so this is technically a proof by contradiction). □

The second term in the bound on T_ϵ measures how closely $\mathcal{V}$ matches with the set of all points z on which $\psi\left(z\right)$ exceeds our initial lower bound l_0. Observe that if $l_0 \leq \psi\left(v\right)$ for all $v \in \mathcal{V}$, then $\mu\left(L\left(\psi, l_0\right)\right) = \mu\left(\mathcal{V}\right)$, and this term will vanish.

Bounding the number of cutting-plane iterations that will be performed is not enough to establish how quickly our procedure will converge, since we rely on performing an inner SVM optimizations with target suboptimality ϵ_t, and the runtime of these inner optimizations naturally will depend on the magnitudes of the ϵ_ts, which are bounded in our final lemma:

Lemma 3. *In the context of Algorithm 2, suppose that we choose $v^{(t)}$ and ϵ_t as in Definition 2. Then:*

$$\epsilon_t \geq \frac{U_t - L_t}{2e\left(m+1\right)}$$

and in particular, for all t (before termination):

$$\epsilon_t \geq \frac{\epsilon}{2e\left(m+1\right)}$$

since we terminate as soon as $U_t - L_t \leq \epsilon$.

Proof. Because h_t is concave:

$$\mu\left(G\left(h_t, L_t\right)\right) - \mu\left(G\left(h_t, z_t\right)\right) \leq \left(z_t - L_t\right) \mu\left(L\left(h_t, L_t\right)\right)$$

where z_t is as in Lemma 1. By Corollary 1, $\mu\left(L\left(h_t, L_t\right)\right) \leq \frac{m+1}{U_t - L_t} \mu\left(G\left(h_t, L_t\right)\right)$, which combined with the above inequality gives that:

$$\frac{\mu\left(G\left(h_t, L_t\right)\right) - \mu\left(G\left(h_t, z_t\right)\right)}{\mu\left(G\left(h_t, L_t\right)\right)} \leq \frac{z_t - L_t}{U_t - L_t} \left(m+1\right)$$

By Theorem 2, the LHS is at least $1/e$, and $z_t - L_t = 2\epsilon_t$, giving the claimed result. □

D SVM optimization

We'll now move onto a discussion of how we propose implementing the SVMOptimizer of Algorithm 2. The easier-to-analyze approach, based on an inner SDCA optimization over w [Shalev-Shwartz and Zhang, 2013] and an outer cutting plane optimization over b (Algorithm 3), will be described in Appendices D.1 and D.2. The easier-to-implement version, which simply calls an off-the-shelf SVM solver, will be described in Appendix D.4.

D.1 SDCA w-optimization

To simplify the presentation, we're going to begin by reformulating Equation 5 in such a way that all of the datasets are "mashed together", with the coefficients being defined on a per-example basis, rather than per-dataset. To this end, for fixed w' and b', we define, for every $i \in \{1, \ldots, k\}$ and every $x \in D_i$:

$$\check{\alpha}_{i,x}^{(0)} = \begin{cases} \alpha_i^{(0)} & \text{if } \langle w', x \rangle - b' \leq 1/2 \\ 0 & \text{otherwise} \end{cases}$$

$$\check{\beta}_{i,x}^{(0)} = \begin{cases} \beta_i^{(0)} & \text{if } \langle w', x \rangle - b' \geq -1/2 \\ 0 & \text{otherwise} \end{cases}$$

This takes care of the loss coefficients. For the constraint coefficients, define:

$$\check{\alpha}_{i,x}^{(j)} = \begin{cases} \alpha_i^{(j)} & \text{if } \langle w', x \rangle - b' \leq 1/2 \\ 0 & \text{otherwise} \end{cases}$$

$$\check{\beta}_{i,x}^{(j)} = \begin{cases} \beta_i^{(j)} & \text{if } \langle w', x \rangle - b' \geq -1/2 \\ 0 & \text{otherwise} \end{cases}$$

and finally, we need to handle the constraint upper bounds:

$$\begin{aligned} \check{\gamma}^{(j)} =& \gamma^{(j)} - \sum_{i=1}^{k} \frac{1}{|D_i|} \Big(\alpha_i^{(j)} \left| \{ x \in D_i \mid \langle w', x \rangle - b' > 1/2 \} \right| \\ & + \beta_i^{(j)} \left| \{ x \in D_i \mid \langle w', x \rangle - b' < -1/2 \} \right| \Big) \end{aligned}$$

Observe that the $\check{\alpha}_{i,x}^{(0)}$s, $\check{\beta}_{i,x}^{(0)}$s, $\check{\alpha}_{i,x}^{(j)}$s, $\check{\beta}_{i,x}^{(j)}$s, and $\check{\gamma}^{(j)}$s all have implicit dependencies on w' and b'. In terms of these definitions, the Ψ defined in Equation 5 can be written as:

$$\begin{aligned} \Psi\left(w, b, v; w', b'\right) =& \sum_{i=1}^{k} \frac{1}{|D_i|} \sum_{x \in D_i} \left(\left(\check{\alpha}_{i,x}^{(0)} + \sum_{j=1}^{m} v_j \check{\alpha}_{i,x}^{(j)} \right) \max\left\{ 0, \frac{1}{2} + \left(\langle w, x \rangle - b \right) \right\} \right. \\ & \left. + \left(\check{\beta}_{i,x}^{(0)} + \sum_{j=1}^{m} v_j \check{\beta}_{i,x}^{(j)} \right) \max\left\{ 0, \frac{1}{2} - \left(\langle w, x \rangle - b \right) \right\} \right) \\ & + \frac{\lambda}{2} \|w\|_2^2 - \sum_{j=1}^{m} v_j \check{\gamma}^{(j)} \end{aligned}$$

This formulation makes it clear that minimizing Ψ as a function of w and b is equivalent to optimizing an SVM, since Ψ is just a positive linear combination of hinge losses, plus a ℓ^2 regularizer, plus a term that does not depend on w or b. Since Ψ can have both "positive" and "negative" hinge losses associated with the same

example, however, it's slightly simpler to combine both hinge losses together into a single piecewise linear per-example loss, rather than decomposing it into two separate hinges:

$$\ell_{i,x}(z) = \check{\alpha}_{i,x} \max\left\{0, \frac{1}{2} + z\right\} + \check{\beta}_{i,x} \max\left\{0, \frac{1}{2} - z\right\}$$

where:

$$\check{\alpha}_{i,x} = \frac{n}{|D_i|}\left(\check{\alpha}_{i,x}^{(0)} + \sum_{j=1}^{m} v_j \check{\alpha}_{i,x}^{(j)}\right) \quad \text{and} \quad \check{\beta}_{i,x} = \frac{n}{|D_i|}\left(\check{\beta}_{i,x}^{(0)} + \sum_{j=1}^{m} v_j \check{\beta}_{i,x}^{(j)}\right)$$

Here, $n = \sum_{i=1}^{k} |D_i|$ is the total number of examples across all of the datasets—we introduced the n factor here so that Ψ will be written in terms of the *average* loss (rather than the *total* loss). Although it is not represented explicitly in our notation, it should be emphasized that $\ell_{i,x}$ implicitly depends on v, w' and b'.

As the sum of two hinges, the $\ell_{i,x}$s are Lipschitz continuous in z, with the Lipschitz constant being:

$$L = \max_{i \in \{1,\ldots,k\}} \frac{n}{|D_i|}\left(\left(\alpha_i^{(0)} + \beta_i^{(0)}\right) + V \sum_{j=1}^{m}\left(\alpha_i^{(j)} + \beta_i^{(j)}\right)\right) \tag{7}$$

where $V = \max_{j \in \{1,\ldots,m\}} v_j$ is a uniform upper bound on the magnitudes of the Lagrange multipliers associated with the constraints. Notice that, if the datasets are comparable in size, then $n/|D_i|$ will be on the order of k, so L will typically not be as large as the n-dependence of its definition would appear to imply.

We may now write Ψ in terms of the loss functions $\ell_{i,x}$:

$$\Psi(w, b, v; w', b') = \frac{1}{n}\sum_{i=1}^{k}\sum_{x \in D_i} \ell_{i,x}(\langle w, x\rangle - b) + \frac{\lambda}{2}\|w\|_2^2 - \sum_{j=1}^{m} v_j \check{\gamma}^{(j)}$$

This is the form considered by Shalev-Shwartz and Zhang [2013], so we may apply SDCA:

Theorem 3. *If we use SDCA [Shalev-Shwartz and Zhang, 2013] to optimize Equation 8 for fixed b and v, then we will find a suboptimal solution with duality gap ϵ'' after performing at most:*

$$T_{\epsilon''} = O\left(\max\left\{0, n \ln\left(\frac{\lambda n}{L^2 X^2}\right)\right\} + n + \frac{L^2 X^2}{\lambda \epsilon''}\right)$$

iterations, where $X = \max_{i \in \{1,\ldots,k\}} \max_{x \in D_i} \|x\|_2$ is a uniform upper bound on the norms of the training examples.

Proof. This is Theorem 2 of Shalev-Shwartz and Zhang [2013]. □

SDCA works by, rather than directly minimizing Ψ over w, instead maximizing the following over the dual variables ξ:

$$\Psi^*(\xi, b, v; w', b') = \tag{8}$$
$$-\frac{1}{n}\sum_{i=1}^{k}\sum_{x \in D_i} \ell_{i,x}^*(\xi_{i,x}) - \frac{1}{2\lambda}\left\|\frac{1}{n}\sum_{i=1}^{k}\sum_{x \in D_i} \xi_{i,x} x\right\|_2^2 - \frac{1}{n}\sum_{i=1}^{k}\sum_{x \in D_i} \xi_{i,x} b - \sum_{j=1}^{m} v_j \check{\gamma}^{(j)}$$

using stochastic coordinate ascent, where:

$$w = -\frac{1}{\lambda n}\sum_{i=1}^{k}\sum_{x \in D_i} \xi_{i,x} x$$

is the primal solution w corresponding to a given set of dual variables ξ, and:

$$\ell_{i,x}^*(\xi_{i,x}) = \frac{1}{2}\left|\xi_{i,x} - \check{\alpha}_{i,x} + \check{\beta}_{i,x}\right| - \frac{1}{2}\left(\check{\alpha}_{i,x} + \check{\beta}_{i,x}\right)$$

is the Fenchel conjugate of $\ell_{i,x}$, and is defined for $-\check{\beta}_{i,x} \leq \xi_{i,x} \leq \check{\alpha}_{i,x}$ (these bounds become box constraints on the ξs of Equation 8).

Algorithm 3 Skeleton of a cutting-plane algorithm that finds a $b \in \mathcal{B}$ minimizing (to within ϵ) $\min_w \Psi(w, b, v; w', b')$, where $\mathcal{B} \subseteq \mathbb{R}$ is a closed bounded interval. It is assumed that $l_0, u_0 \in \mathbb{R}$ are finite, and that $l_0 \leq \max_v \min_w \Psi(w, b, v; w', b') \leq u_0$. The u'_0 increase that is "maybe" performed on line 2, and the CutChooser function on line 9, are discussed in Appendix D.2. The SDCAOptimizer function is as described in Appendix D.1.

SVMOptimizer $(v, l'_0, u'_0, \epsilon')$

1 Initialize $g'_0 \in \mathbb{R}$ to zero
2 *Maybe* update $u'_0 = u'_0 + (u'_0 - l'_0)$ // *needed for Lemma 4, but might not help in practice*
3 For $t \in \{1, 2, \dots\}$
4 Let $h'_t(b) = \max_{s \in \{0,1,\dots,t-1\}} (l'_s + g'_s (b - b_s))$
5 Let $L'_t = \min_{b \in \mathcal{B}} h'_t(b)$ and $U'_t = \min_{s \in \{0,1,\dots,t-1\}} u'_s$
6 If $U'_t - L'_t \leq \epsilon'$ then
7 Let $s \in \{1, \dots, t-1\}$ be an index minimizing u'_s
8 Return $w^{(s)}$, b_s, L'_t
9 Let $b_t, \epsilon_t = \text{CutChooser}(h'_t, U'_t)$
10 Let $\xi^{(t)}, w^{(t)} = \text{SDCAOptimizer}(b_t, v, \epsilon'_t)$
11 Let $u'_t = \Psi(w^{(t)}, b_t, v; w', b')$
12 Let $l'_t = \Psi^*(\xi^{(t)}, b_t, v; w', b')$ and $g'_t = \frac{\partial}{\partial'_b} \Psi^*(\xi^{(t)}, b_t, v; w', b')$

D.2 Cutting plane b-optimization

Having described in the previous section how we may optimize over w for fixed b and v using SDCA, we now move on to the problem of creating the SVMOptimizer needed by Algorithm 2, which must optimize over both w and b.

Many linear SVM optimizers do not natively handle an unregularized bias parameter b, and this has long been recognized as a potential issue. For example, Shalev-Shwartz et al. [2011] suggest using Pegasos to perform inner optimizations over w, and a bisection-based outer optimization over b. Our proposal is little different from this, except that Algorithm 3, rather than using bisection, optimizes over b using essentially the same cutting plane algorithm as we used in Algorithm 2. Our two cutting-plane algorithms are essentially identical, except that optimizing over b is a minimization problem (over v is maximization), and we might increase u_0 on line 2 for a technical reason (it will be needed by the proof of Lemma 4, but is probably not helpful in practice).

D.2.1 Minimization-based

Perhaps the easiest-to-implement version of Algorithm 3 is based on the idea of simply solving for the minimizer of h'_t at every iteration.

Definition 3. ***(Minimization-based Algorithm 3)*** *Do* not *increase u'_0 on line 2, and have CutChooser choose $b_t = \text{argmin}_{b \in \mathcal{B}} h'_t(b)$ and $\epsilon'_t = (U_t - L_t)/2$.*

As was the case in Appendix C.1, we have no convergence rates for this version.

D.3 Center of mass-based

Essentially the same center of mass-based approach as was described in Appendix C.2 can be used in this setting, except that we must find the center of mass of a 2-dimensional sublevel epigraph, rather than an $m+1$-dimensional superlevel hypograph:

Definition 4. ***(Center of mass-based Algorithm 3)*** Do *increase u'_0 on line 2, have CutChooser take (b_t, z_t) to be the center of mass of $\{(b, z) \mid h'_t(b) \leq z \leq U'_t\}$, and choose $\epsilon'_t = (U'_t - z_t)/2$.*

Due to the similarity between Algorithms 3 and 2, we can simply recycle the results of Appendix C.2, with the troublesome second term in the bound of Lemma 2 removed by combining the "maybe" portion of Algorithm 3 with the Lipschitz continuity of Ψ as a function of b:

Lemma 4. *In the context of Algorithm 3, suppose that we choose b_t and ϵ'_t as in Definition 4. Then there is a iteration count $T_{\epsilon'}$ satisfying:*

$$T_{\epsilon'} = O\left(\ln\left(\frac{LB\left(u'_0 - l'_0\right)}{\epsilon'}\right)\right)$$

such that, if $t \geq T_{\epsilon'}$, then $U'_t - L'_t \leq \epsilon'$, where $\mathcal{B} = [-B, B]$ is the set of allowed biases and L is as in Equation 7. Hence, Algorithm 3 will terminate after $T_{\epsilon'}$ iterations.

Proof. Starting from (and adapting) the final equation in the proof of Lemma 2:

$$\begin{aligned} U'_t - L'_t \leq & 32\left(1 - \frac{1}{2e}\right)^{t-1}\left(\frac{1}{\epsilon'}\right)\left(u'_0 - l'_0\right)^2 \\ & \cdot \left(\frac{B}{\mu\left(\{b \in [-B, B] \mid \min_{w \in \mathbb{R}^d} \Psi\left(w, b, v; w', b'\right) \leq u'_0\}\right)}\right) \end{aligned}$$

Observe that, as a function of b, $\Psi\left(w, b, v; w', b'\right)$ is L-Lipschitz. Hence, if we let $w^* \in \mathbb{R}^d, b^* \in [-B, B]$ be the optimal weight and bias, then:

$$\mu\left(\{b \in [-B, B] \mid \Psi\left(w^*, b, v; w', b'\right) \leq u'_0\}\right) \geq \min\left\{2B, \frac{u'_0 - b^*}{L}\right\}$$

Since $\min_{w \in \mathbb{R}^d} \Psi\left(w, b, v; w', b'\right) \leq \Psi\left(w^*, b, v; w', b'\right)$, it follows that:

$$U'_t - L'_t \leq 32\left(1 - \frac{1}{2e}\right)^{t-1}\left(\frac{1}{\epsilon'}\right)\left(u'_0 - l'_0\right)^2 \max\left\{\frac{1}{2}, \frac{LB}{u'_0 - b^*}\right\}$$

This is the reason that we increased u'_0 on line 2 of Algorithm 3, since doing so has the result that $u'_0 - b^* \geq u'_0 - l'_0$, so:

$$U'_t - L'_t \leq 32\left(1 - \frac{1}{2e}\right)^{t-1}\left(\frac{1}{\epsilon'}\right)\left(u'_0 - l'_0\right) \max\left\{\frac{u'_0 - l'_0}{2}, LB\right\}$$

The same reasoning as was used in the proof of Lemma 2 then gives the claimed bound on $T_{\epsilon'}$. □

In addition to the above result, the obvious analogue of Lemma 3 holds as well:

Lemma 5. *In the context of Algorithm 3, suppose that we choose b_t and ϵ'_t as in Definition 4. Then:*

$$\epsilon'_t \geq \frac{U'_t - L'_t}{2e}$$

and in particular, for all t (before termination):

$$\epsilon'_t \geq \frac{\epsilon'}{2e}$$

since we terminate as soon as $U'_t - L'_t \leq \epsilon'$.

Proof. Same as Lemma 3. □

In Appendix E, we'll combine these results with those of Appendices D.1 and C to bound the overall convergence rate of Algorithm 2.

D.4 Kernelization

The foregoing discussion covers the case in which we wish to learn a linear classifier, and use an SVM optimizer (SDCA) that doesn't handle an unregularized bias. It's clear that we could freely substitute another linear SVM optimizer for SDCA, as long as it finds both a primal and dual solution, enabling us to calculate the lower and upper bounds required by Algorithm 2.

Our technique is easily kernelized—the resulting algorithm simply depends on inner kernel SVM optimizations, rather than linear SVM optimizations. SDCA can be used in the kernel setting, but the per-iteration cost increases from $O(d)$ arithmetic operations to $O(n)$ kernel evaluations, where n is the total size of all of the datasets. Kernel-specific optimizers, such as `LIBSVM` [Chang and Lin, 2011], will generally work better than SDCA in practice, since they typically have the same per-iteration cost, but each iteration is "smarter". More importantly, such optimizers usually jointly optimize over w and b, eliminating the need for Algorithm 3. Hence, an implementation based on such an optimizer is the simplest version of our proposed approach, since it would be nothing but Algorithms 1 and 2, with the SVMOptimizer helper function being a call to an off-the-shelf solver.

E Overall convergence rates

We may now combine the results in Appendices C and D into one bound on the overall convergence rate of Algorithm 2, assuming that we use Algorithm 3, rather than an off-the-shelf SVM solver, to implement the SVMOptimizer. First, for the center of mass-based versions:

Theorem 4. *Assuming that SVMOptimizer is implemented as in Algorithm 3, with the CutChooser functions in Algorithms 2 and 3 being implemented using the center of mass (as in Definitions 2 and 4), and assuming that we have access to an oracle function for finding the center of mass of compact convex polytopes, then Algorithm 2 will perform:*

$$O\left(m \ln\left(\frac{u_0 - l_0}{\epsilon}\right) + \ln\left(\frac{\mu\left(\mathcal{V}\right)}{A\left(\psi, l_0\right)}\right)\right)$$

iterations, each of which contains a single call to Algorithm 3, with each such call requiring:

$$O\left(\ln\left(LBm\right)\right)$$

iterations, each of which contains a single call to SDCAOptimizer, with each such call requiring:

$$O\left(\max\left\{0, n \ln\left(\frac{\lambda n}{L^2 X^2}\right)\right\} + n + \frac{L^2 X^2 m}{\lambda \epsilon}\right)$$

iterations, each of which requires $O(d)$ arithmetic operations.

Proof. Follows immediately from combining Lemmas 2, 3, 4 and 5 with Theorem 3, and using the fact that the bounds L_t and U_t are passed from Algorithm 2 to Algorithm 3 as $u_0' = U_t$ and $l_0' = L_t$. □

In terms of only n, m, d and ϵ, dropping all of the other factors , the overall cost of finding an ϵ-suboptimal solution to Problem 3 is therefore $\tilde{O}\left(dnm \ln\left(1/\epsilon\right) + dm^2/\epsilon\right)$ total arithmetic operations in the inner SDCA optimizers, $\tilde{O}\left(m \ln\left(1/\epsilon\right)\right)$ calls to the center of mass oracles in Algorithms 2 and 3, and another $\tilde{O}\left(m \ln\left(1/\epsilon\right)\right)$ calls to a linear programming oracle for finding U_t in Algorithm 2 and L_t in Algorithm 3.

It must be pointed out that our reliance on a center of mass oracle is unrealistic, since finding the center of mass is a computationally difficult problem [Nemirovski, 1994, Section 3.2]. With that said, we hope that these results can provide a basis for future work.