Dual coordinate solvers for large-scale structural SVMs

Deva Ramanan
UC Irvine

This manuscript describes a method for training linear SVMs (including binary SVMs, SVM regression, and structural SVMs) from large, out-of-core training datasets. Current strategies for large-scale learning fall into one of two camps; batch algorithms which solve the learning problem given a finite datasets, and online algorithms which can process out-of-core datasets. The former typically requires datasets small enough to fit in memory. The latter is often phrased as a stochastic optimization problem [4, 15]; such algorithms enjoy strong theoretical properties but often require manual tuned annealing schedules, and may converge slowly for problems with large output spaces (e.g., structural SVMs). We discuss an algorithm for an "intermediate" regime in which the data is too large to fit in memory, but the active constraints (support vectors) are small enough to remain in memory. In this case, one can design rather efficient learning algorithms that are as stable as batch algorithms, but capable of processing out-of-core datasets. We have developed such a MATLAB-based solver and used it to train a series of recognition systems [19, 7, 21, 12] for articulated pose estimation, facial analysis, 3D object recognition, and action classification, all with publicly-available code. This writeup describes the solver in detail.

Approach: Our approach is closely based on data-subsampling algorithms for collecting hard examples [9, 10, 6], combined with the dual coordinate quadratic programming (QP) solver described in liblinear [8]. The latter appears to be current fastest method for learning linear SVMs. We make two extensions (1) We show how to generalize the solver to other types of SVM problems such as (latent) structural SVMs (2) We show how to modify it to behave as a partially-online algorithm, which only requires access to small amounts of data at a time.

Overview: Sec. 1 describes a general formulation of an SVM problem that encompasses many standard tasks such as multi-class classification and (latent) structural prediction. Sec. 2 derives its dual QP, and Sec. 3 describes a dual coordinate descent optimization algorithm. Sec. 4 describes modifications for optimizing in an online fashion, allowing one to learn near-optimal models with a single pass over large, out-of-core datasets. Sec. 5 briefly touches on some theoretical issues that are necessary to ensure convergence. Finally, Sec. 6 and Sec. 7 describe modifications to our basic formulation to accommodate non-negativity constraints and flexible regularization schemes during learning.

1 Generalized SVMs

We first describe a general formulation of a SVM which encompasses various common problems such as binary classification, regression, and structured prediction. Assume we are given training data where the i^{th} example is described by a set of N_i vectors $\{x_{ij}\}$ and a set of N_i scalars $\{l_{ij}\}$, where j varies from 1 to N_i. We wish to solve the following optimization problem:

$$\underset{w}{\operatorname{argmin}}\, L(w) = \frac{1}{2}||w||^2 + \sum_i \max_{j \in N_i}(0, l_{ij} - w^T x_{ij}) \tag{1}$$

We can write the above optimization as the following quadratic program (QP):

$$\begin{aligned} &\underset{w,\xi \geq 0}{\operatorname{argmin}} \frac{1}{2}||w||^2 + \sum_i^N \xi_i \\ &\text{s.t.} \qquad \forall i, j \in N_i \qquad w^T x_{ij} \geq l_{ij} - \xi_i \end{aligned} \tag{2}$$

Eq. (1) and its QP variant (2) is a general form that encompasses binary SVMs, multiclass SVMs[5], SVM regression [16], structural SVMs [17, 18] the convex variant of latent SVMs [9, 10] and the convex variant of latent structural SVMs [20]. Below, we show how to derive the various special cases. We then describe a general-purpose methodology for efficiently optimizing the above special cases.

1.1 Binary classification

A linearly-parametrized binary classifier predicts a binary label for an input x:

$$\text{Label}(x) = \{w^T x > 0\} \tag{3}$$

The associated learning problem is defined by a dataset of labeled examples $\{x_i, y_i\}$, where $x_i \in \mathcal{R}^N, y_i \in \{-1, 1\}$:

$$\begin{aligned} &\underset{\beta}{\operatorname{argmin}} \frac{1}{2}||\beta||^2 + C\sum_i \xi_i \\ &\text{s.t.} \quad y_i(\beta^T x_i + b) \geq 1 - \xi_i \end{aligned} \tag{4}$$

One can convert the above into (4) with the following: first append a constant value v to each feature to model the bias term with $x_i' = (x_i, v)$ where $v = 1$ is the typical choice. This allows us to write $\beta^T x_i + b = w^T x_i'$ where $w = (\beta, b)$. We then multiply in the class label y_i and slack scaling C into each feature x', yielding $x_{ij} = (Cy_i x_i, Cy_i)$, where $j \in \{1\}$, $N_i = 1$ and $l_{ij} = C$.

Bias term: The above mapping does not precisely correspond to (1) because the bias b is now regularized. This means the learning objective function will prefer biases closer to 0, while (4) does not favor any particular bias. This may or may not be desired. In practice, one can decrease the effect of regularization by appending a large constant value v. For example, a model learned with $v = 100$ will tend to produce larger effective biases b than $v = 1$. In the limit that $v \to \infty$, (4) does map directly to (2). We later describe a modification to (1) that allows for arbitrary (but finite) regularization factors for individual parameters. For the description of subsequent SVM problems, we omit the bias term for notational simplicity.

Margin-rescaling: The above formulation can easily handle example-specific margins: for example, we may require that certain "prototypical" positives score higher than 2 rather than the standard margin of 1. This can be done by defining $l_{ij} = \text{margin}_i$, where margin_i is the margin associated with example i. In the literature, this modification is sometimes known as margin-rescaling. We will see that margin rescaling is one core component of structural SVM problems.

Cost-sensitive examples (slack-rescaling): The above formulation can be easily extended to cost-sensitive SVMs by defining $l_{ij} = C_i$ and $x_{ij} = (C_i y_i x_i, C_i y_i)$. This is sometimes known as slack rescaling. For example, one could define a different cost penalty for positive versus negative examples. Such class-specific costs have been shown be useful for learning classifiers from imbalanced datasets [1]. For the description of subsequent SVM problems, we omit any slack rescaling term (C or C_i) for notational simplicity, though they can always be incorporated by scaling x_{ij} and l_{ij}.

1.2 Multiclass SVMs

A linearly-parametrized multiclass predictor produces a class label for x with the following:

$$\text{Label}(x) = \underset{j \in \{1 \ldots K\}}{\operatorname{argmax}} w_j^T x$$

The associated learning problem is defined by a dataset $\{x_i, y_i\}$ where $x_i \in \mathcal{R}^N$ and $y_i \in \{1, 2, \ldots K\}$. There exist many approaches to multiclass prediction that reduce the problem to a series of binary prediction problems (say, by training K 1-vs-all predictors or K^2 pairwise predictors). Each of the core binary prediction problems can be written as (4), and so can be directly mapped to (2). Here, we describe the particular multiclass formulation from [5] which requires an explicit mapping:

$$\underset{w,\xi \geq 0}{\operatorname{argmin}} \frac{1}{2} \sum_j ||w_j||^2 + \sum_i \xi_i \tag{5}$$

$$\text{s.t.} \forall i, j \neq y_i \quad w_{y^i}^T x_i - w_j^T x_i \geq \text{loss}(y_i, j) - \xi_i \tag{6}$$

The above formulation states that for example i, the score of the true class y_i should dominate the score of any other class j by $\text{loss}(y_i, j)$; if not, we should pay the difference (the slack). For example, given a multi-class problem where the class labels are car, bus, and person, one may wish to penalize mistakes that label a car as a person higher than those that label a car as a bus. In the general setting, this can be specified with a loss function $\text{loss}(y_i, j)$ that specifies the cost of labeling class y_i as class j. The original formulation from [5] defined a 0-1 loss where $\text{loss}(y_i, j) = 0$ for $j = y_i$ and $\text{loss}(y_i, j) = 1$ for $j \neq y_i$. Finally, we have omitted an explicit class-specific bias term b_j in the above formulation, but one can apply the same trick of appending a constant value to feature x_i.

The above form can be massaged into (2) by the following: let us define $w = (w_1, \ldots, w_K)$ as a NK-long vector of concatenated class-specific weights w_j, and $\phi(x_i, j)$ as a NK-length sparse vector with N non-zero entries corresponding to the interval given by class j. These two definitions allow us to write $w_j^T x_i = w^T \phi(x_i, j)$. This in turn allows us to define $x_{ij} = \phi(x_i, y_i) - \phi(x_i, j)$, which then maps the above into (1), where $N_i = (K - 1)$.

1.3 Structural SVMs

A linearly-parametrized structural predictor produces a label of the form

$$\text{Label}(x) = \underset{y \in Y}{\operatorname{argmax}}\, w^T \phi(x_i, y)$$

where Y represents a (possibly exponentially-large) structured output space. The associated learning problem is given by a dataset $\{x_i, y_i\}$ where $x_i \in \mathcal{R}^N$ and $y_i \in Y$:

$$\underset{w,\xi \geq 0}{\operatorname{argmin}} \frac{1}{2} ||w||^2 + \sum_i \xi_i \tag{7}$$

$$\text{s.t.} \forall i, h \in Y \quad w^T \phi(x_i, y_i) - w^T \phi(x_i, h) \geq \text{loss}(y_i, h) - \xi_i \tag{8}$$

One can define $N_i = |Y|, x_{ij} = \phi(x_i, y_i) - \phi(x_i, j)$ and $l_{ij} = \text{loss}(y_i, j)$, where $j = h$ is interpreted as an index into the output space Y.

1.4 Latent SVMs

A latent SVM produces a binary prediction by searching over a latent variable

$$\text{Label}(x) = \{\max_{z \in Z} w \cdot (x, z) > 0\} \tag{9}$$

where Z represents a (possibly exponentially-large) latent space. In latent-SVM learning, and in particular, the convex optimization stage of coordinate descent [9], each training example is given by $\{x_i, z_i, y_i\}$ where $y_i \in \{-1, 1\}$, and z_i are latent variables specified for positive examples:

$$\underset{w,\xi \geq 0}{\operatorname{argmin}} \frac{1}{2} ||w||^2 + \sum_i \xi_i \tag{10}$$

$$\text{s.t.} \forall i \in \text{pos} \qquad w^T \phi(x_i, z_i) \geq 1 - \xi_i \tag{11}$$

$$\text{s.t.} \forall i \in \text{neg}, g \in Z \qquad w^T \phi(x_i, g) \leq -1 + \xi_i \tag{12}$$

One can map this to the above problem with the following: for $i \in \text{pos}$, $N_i = 1, x_{ij} = \phi(x_i, z_i), l_{ij} = 1$. For $i \in \text{neg}$, $N_i = |Z|$, $x_{ij} = -\phi(x_i, j), l_{ij} = -1$ where $j = g$.

1.5 Latent structural SVMs

One can extend the above model to the latent structural case, where the predictor behaves as follows:

$$\text{Label}(x) = \operatorname*{argmax}_{y \in Y} \left[\max_{z \in Z} w^T \phi(x, y, z) \right]$$

The associated learning problem is defined by a dataset $\{x_i, z_i, y_i\}$ where $y_i \in Y$ is a structured label rather than a binary one. In this scenario, the analogous convex step of "coordinate descent" corresponds the optimizing the following optimization:

$$\begin{aligned} &\operatorname*{argmin}_{w, \xi \geq 0} \frac{1}{2}||w||^2 + \sum_i \xi_i \\ \text{s.t.} \forall i, h \in Y, g \in Z, \quad &w^T \phi(x_i, y_i, z_i) - w^T \phi(x_i, h, g) \geq \text{loss}(y_i, h, g) - \xi_i \end{aligned} \tag{13}$$

This can be mapped to our general formulation by defining $N_i = |Y||Z|$, $x_{ij} = \phi(x_i, y_i, z_i) - \phi(x_i, j)$ for $j \in Y \times Z$.

1.6 Regression

A linear regressor makes the following predictions

$$\text{Label}(x) = w^T x$$

The associated SVM regression problem is specified by a dataset $\{x_i, y_i\}$ where $x_i \in \mathcal{R}^N$ and $y_i \in \mathcal{R}$:

$$\begin{aligned} &\operatorname*{argmin}_{w, \xi \geq 0} \frac{1}{2}||w||^2 + \sum_i (\xi_i + \xi_i^*) \\ \text{s.t.} \forall i, \quad &w^T x_i \geq y_i - \epsilon - \xi_i \\ &w^T x_i \leq y_i + \epsilon + \xi_i^* \end{aligned} \tag{14}$$

The above constraints can be converted to the from (2) by doubling the number of constraints by defining $(x_i', y_i') = (x_i, y_i - \epsilon)$ and $(x_{2i}', y_{2i}') = (-x_i, -y_i - \epsilon)$ and $N_i = N_{2i} = 1$.

Summary: In this section, we have shown that many previously-proposed SVM problems can be written as instances of the generic problem in (1), which can be written as the quadratic program (QP) in (2).

2 Dual quadratic program (QP)

In this section, we will derive the dual of the general QP from (2), which is repeated here for completeness:

$$\begin{aligned} &\operatorname*{argmin}_{w, \xi \geq 0} \frac{1}{2}||w||^2 + \sum_i^N \xi_i \\ \text{s.t.} \quad &\forall i, j \in N_i \quad w^T x_{ij} \geq l_{ij} - \xi_i \end{aligned} \tag{15}$$

The above generalizes a traditional SVM learning formulation in several ways. Firstly, each example x_{ij} comes equipped with its own margin l_{ij}. We shall see that this involves a relatively small modification to the QP solver. A more significant modification is that slack variables are now *shared* across linear constraints. If each example x_{ij} had its own slack variable ξ_{ij}, then (15) would be structurally equivalent to a standard SVM and amenable to standard QP optimization techniques. Intuitively, with independent slack variables,

the set of examples x_{ij} could contribute N_i "dollars" to the loss. This could be a problem when N_i is exponentially-large (as is the case for structured output spaces). By sharing slacks, the set of examples can only contribute at most one "dollar" to the loss.

To further analyze the effect of shared slacks, we derive the dual QP by writing down the associated Lagrangian:

$$L(w, \xi, \alpha, \mu) = \frac{1}{2}||w||^2 + \sum_i \xi_i - \sum_{ij} \alpha_{ij}(w \cdot x_{ij} - l_{ij} + \xi_i) - \sum_i \mu_i \xi_i \tag{16}$$

By strong duality

$$\min_{w,\xi} \left[\max_{\alpha \geq 0, \mu \geq 0} L(w, \alpha, \mu) \right] = \max_{\alpha \geq 0, \mu \geq 0} \left[\min_{w,\xi} L(w, \alpha, \mu) \right] \tag{17}$$

We take the derivative of the Lagrangian with respect to the primal variables to get the KKT conditions:

$$\frac{\partial L(w, \alpha, \mu)}{\partial w} = 0 \quad \rightarrow \quad w = \sum_{ij} \alpha_{ij} x_{ij} \tag{18}$$

$$\frac{\partial L(w, \alpha, \mu)}{\partial \xi_i} = 0 \quad \rightarrow \quad \sum_j \alpha_{ij} \leq 1 \quad \forall i \tag{19}$$

We write the dual of the QP in (2) as

$$F(\alpha) = -\frac{1}{2} \sum_{ij,kl} \alpha_{ij} x_{ij}^T x_{kl} \alpha_{kl} + \sum_{ij} l_{ij} \alpha_{ij} \tag{20}$$

$$\text{s.t.} \quad \forall i, \quad \sum_j \alpha_{ij} \leq 1 \tag{21}$$

$$\forall i, j \in N_i, \quad \alpha_{ij} \geq 0$$

We wish to maximize (20) over the dual variables α. We can further analyze the nature of the optimal solution by considering *complementary slackness* conditions, which states that either a primal constraint is active, or its corresponding dual Lagrangian multiplier is 0:

$$\forall i, j \in N_i \quad s.t \quad \alpha_{ij} > 0, \quad w^T x_{ij} = l_{ij} - \xi_i \tag{22}$$

The above condition states that all examples with non-zero alpha (support vectors) associated with a single i will incur the same slack loss ξ_i. In order words, these examples correspond to "ties" in the maximization over $\max_j(0, l_{ij} - w \cdot x_{ij})$ from (1). At the optimal dual solution, the linear constraint from (21) delicately balances the influence of support vectors associated with i to ensure they all pay the same slack loss. If each example x_{ij} had its own slack variable ξ_{ij}, (21) would be replaced by independent "box constraints" $\alpha_{ij} \leq 1$ for all ij. Box constraints are simpler to deal with because they decouple across the dual variables. Indeed, we show that the linear constraints considerable complicate the optimization problem.

3 Batch optimization

We now describe efficient training algorithms for solving dual QPs of the general form from (20). We begin by describing a solver that operates in a batch setting, requiring access to all training examples. The fastest current batch solver for linear SVMs appears to be liblinear [8], which is a dual coordinate descent method. A naive implementation of a dual solver would require maintaining a $N \times N$ kernel matrix. The innovation of liblinear is the realization that one can implicitly represent the kernel matrix for linear SVMs by maintaining the primal weight vector w, which is much smaller. We show a similar insight can be used to design efficient dual solvers for generalized SVMS of the form from (15).

To derive the modified optimization, let us first try to naively apply dual coordinate descent to optimizing (20): let us pick a single dual variable α_{ij}, and update it holding all other α's fixed. This reduces to maximizing a 1-D quadratic function subject to box constraints. This appears easy at first; solve for the maximum of the quadratic and clip the solution to lie within the box constraint. We solve for the offset a that maximizes:

$$\begin{aligned} F(\alpha + a\delta_{ij}) =& \frac{1}{2} h_{ij} a^2 + g_{ij} a + \text{constant} \\ \text{s.t.} \quad & 0 \leq \alpha_i + a \leq 1 \end{aligned} \tag{23}$$

where $\alpha_i = \sum_j \alpha_{ij}$, $h_{ij} = -x_{ij}^T x_{ij}$ (which can be precomputed), and the gradient can be efficiently computed using w:

$$\begin{aligned} g_{ij} &= l_{ij} - \sum_{kl} x_{ij}^T x_{kl} \alpha_{kl} \\ &= l_{ij} - w^T x_{ij} \end{aligned} \tag{24}$$

There are four scenarios one can encounter when attempting to maximize (23):

1. $g_{ij} = 0$, in which case α_{ij} is optimal for the current α.
2. $g_{ij} < 0$, in which case decreasing α_{ij} will increase the dual.
3. $g_{ij} > 0$ and $\alpha_i < 1$, in which case increasing α_{ij} will increase the dual.
4. $g_{ij} > 0$ and $\alpha_i = 1$, in which case increasing α_{ij} may increase the dual.

For the first three scenarios, the linear inequality constraint from (21) is not active. This means that we can apply "standard" dual-coordinate updates that maximize (23) in closed form:

$$a^* = \min\Big(\max\big(-\alpha_i, \frac{g_{ij}}{h_{ij}}\big), 1 - \alpha_i\Big) \tag{25}$$

yielding the update rule

$$a_{ij} := a_{ij} + a^* \tag{26}$$

For the last scenario, we would like to increase α_{ij} but cannot because of the active linear constraint $\sum_j \alpha_{ij} = 1$. Let us select another dual variable $(\alpha_{ik}, k \neq j)$ that shares this constraint to possibly decrease. We would like to find the offset a that maximizes:

$$\begin{aligned} F(\alpha + a\delta_{ij} - a\delta_{ik}) &= \frac{1}{2} h' a^2 + g' a + \text{constant} \\ \text{s.t.} \quad 0 \leq \alpha_{ij} + a \leq 1 \quad &\text{and} \quad 0 \leq \alpha_{ik} - a \leq 1 \end{aligned} \tag{27}$$

where $h' = h_{ij} + h_{ik} - 2x_{ij}^T x_{ij}$ and $g' = g_{ij} - g_{ik}$. Any value of a will ensure that the linear constraint is satisfied. The above maximization can be computed in closed form:

$$\begin{aligned} a^* &= \min\Big(\max\big(a_0, \frac{g_{ij}}{h_{ij}}\big), a_1\Big) \quad \text{where} \\ a_0 = -\max(\alpha_{ij}, 1 - \alpha_{ik}) \quad &\text{and} \quad a_1 = \min(\alpha_{ik}, 1 - \alpha_{ij}) \end{aligned} \tag{28}$$

which yields the following coordinate updates:

$$\alpha_{ij} := \alpha_{ij} + a^* \quad \text{and} \quad \alpha_{ik} := \alpha_{ik} - a^* \tag{29}$$

3.1 Tracking w and α_i

In order to enable efficient computation of the gradient for the next coordinate step, we track the change in w using the KKT condition (**??**):

$$w := w + a^* x_{ij} \quad \text{for single dual update; e.g., conditions 1-3 hold} \tag{30}$$

$$w := w + a^*(x_{ij} - x_{ik}) \quad \text{for pairwise dual update; e.g., condition 4 holds} \tag{31}$$

Similarly, we can track the change in α_i from (23) :

$$\alpha_i := \alpha_i + a^* \tag{32}$$

which only needs to updated for the single dual update.

3.2 Dual coordinate-descent

We provide pseudocode for our overall batch optimization algorithm below.

Input: $\{x_{ij}, l_{ij}\}$
Output: w

$\forall ij, \quad \alpha_{ij} := 0, \alpha_i := 0, w := 0$; `// Initialize variables (if not passed as arguments)`
Repeat
  Randomly pick a dual variable α_{ij};
  Compute gradient g_{ij} from (24);
  if $g_{ij} > \epsilon$ *and* $\alpha_i = 1$ **then** `// Find another variable if linear constraint is active`
    Randomly pick another dual variable with same i (α_{ik} for $k \neq j$);
    Compute a^* with (28) ;
    Update $\alpha_{ij}, \alpha_{ik}, w$ with (29),(30)
  else if $|g_{ij}| > \epsilon$ **then** `/* Else update single dual variable */`
    Compute a^* with (25);
    Update α_{ij}, α_i, w with (26), (32),(30)
  end

Algorithm 1: *Optimize*$(\{x_{ij}, l_{ij}\})$ performs batch optimization of a fixed dataset using multiple passes of dual coordinate descent. We also define a variant that can be "hot-started" with an existing set of dual variables, and optimized until some tolerance threshold *tol* is met *Optimize*$(\{x_{ij}, l_{ij}, a_{ij}\}, tol)$.

Random sampling: One may question the need for random sampling; why not iterate over dual variables "in order"? The answer is that in practice, neighboring examples x_{ij} will tend to be correlated (e.g., consider examples extracted from overlapping sliding windows in an image). In the extreme case, consider two identical training examples x_1 and x_2. After performing a dual update on x_1, x_1 will usually score better, and often pass the margin test under the newly-updated w. If we immediately visit x_2, it will also pass the margin test and w will not be updated. However, assume we first visit an uncorrelated example (that does trigger w to be updated) and then visit x_2. This allows us to effectively "revisit" x_1 in a single pass over our data. Hence a single (but randomly permuted) pass of coordinate descent effectively mimics multiple passes of (sequential) coordinate descent over correlated datasets.

Speed: In practice, we apply our randomized batch algorithm by performing a large number of sequential passes over random permutations of a fixed dataset. During initial iterations, dual variables tend to be small and the linear inequality constraints in (21) are not active. During such updates, our solver will update a single dual variable at a time using (26), and essentially is fast as liblinear. In later passes, the linear constraint tends to be active, and the solver is slower because update from (26) requires computing a dot product between two feature vectors with shared slack variables. In theory, one could cache these dot products in a reduced kernel matrix (since one needs to only store dot products between examples with

shared slacks, rather than all N^2 examples). We found that computing them on-the-fly is still rather fast, and simplifies code.

Convergence: With enough passes, the batch algorithm is guaranteed to converge (Sec. 5). In the following, we provide a practical stopping criteria for convergence (within some tolerance). In the next section on online-learning, we will make use of such a tolerance to manage computation to repeated calls of a dynamic QP-solver. To define our stopping criteria, we closely follow the duality-based stopping criteria described in [11]. Let $OPT = \min_w L(w)$, the optimal primal objective function value from (1). Let us consider a candidate solution to the dual problem specified by a particular setting of dual variables $\alpha = \{\alpha_{ij}\}$. We can compute a lower bound on OPT by with $F(\alpha)$, since all dual solutions are a lower bound on OPT (by strong duality). We can also compute the associated primal weight vector $w(\alpha) = \sum_{ij} \alpha_{ij} x_{ij}$. We know that $L(w(\alpha))$ must be an upper bound on OPT, since OPT is the minimal possible primal objective over all w:

$$LB \leq OPT \leq UB \qquad \text{where} \tag{33}$$

$$LB = F(\alpha) = -\frac{1}{2} w(\alpha)^T w(\alpha) + l(\alpha) \tag{34}$$

$$UB = L(w(\alpha)) = \frac{1}{2} w(\alpha)^T w(\alpha) + \sum_{ij} \max_j (0, l_{ij} - w(\alpha)^T x_{ij}) \tag{35}$$

$$w(\alpha) = \sum_{ij} \alpha_{ij} x_{ij}$$

$$l(\alpha) = \sum_{ij} l_{ij} \alpha_{ij}$$

It is straightforward to track changes to the lower bound from (34) by modifying lines 8 and 11 from Alg. 1 to maintain a running estimate of $l(\alpha)$ as dual variables are sequentially updated. The upper bound cannot be easily tracked, and instead has to be computed by passing over the entire set of data to compute the loss from (35) for a particular dual solution α. In practice, one can simply update the upper bound occasionally, after a large number of dual updates. Once the upper and lower bound are found to lie within some tolerance tol, the batch algorithm terminates. Sec. 5 suggests that given sufficient iterations, the bounds must meet. The full interface to our batch algorithm is $Optimize(\{x_{ij}, l_{ij}, \alpha_{ij}\}, tol)$.

Approximate upper-bound: We now describe an approximate upper bound that is easily tracked given a single sequential pass over a fixed dataset. Recall that our batch algorithm performs a large number of sequential passes over random permutations of our fixed dataset. The intuition behind our approximate upper bound is that we can use gradients g_{ij}, computed during dual updates from (24) to approximate the loss $l_{ij} - w(\alpha)^T x_{ij}$:

$$UB' = \frac{1}{2} w(\alpha)^T w(\alpha) + \sum_{ij} \max_j (0, g_{ij}(\alpha^t)) \tag{36}$$

With some abuse of notation, we write $g_{ij}(\alpha^t)$ to explicitly denote the fact that gradients are computed with a changing set of dual variables at step t of coordinate descent. If no dual variables are updated during a single sequential pass over the fixed dataset, then $\alpha^t = \alpha$ and the approximation is exact $UB' = UB$. In general we find that $UB' > UB$ since the loss due to a data example $l_{ij} - w(\alpha)^T x_{ij}$ typically decreases after optimizing the dual variable associated with that data example. In practice, we keep track of this approximate upper bound, and whenever the tolerance criteria is satisfied with respect to this approximation, we compute the true upper-bound and perform another sequential pass if the true tolerance is not met. We find this speeds up batch optimization to convergence (up to tol) by factor of 2, compared to explicitly recomputing the true upper-bound after each sequential pass.

4 Online learning

In this section, we describe an efficient algorithm for online optimization, given large streaming datasets.

One-pass coordinate-descent: The batch algorithm from Alg.1 can be "trivially" turned into a online algorithm by performing a single, sequential pass over a streaming dataset. Given a new data point i, one can sample a random $j \in N_i$ (or select j with the largest gradient, as in LaRank [2]) and compute its associated dual variable α_{ij}. If the linear constraint is active, one can sample α_{ik} at random and update both variables. This online algorithm has two notable properties. (1) This optimization step is guaranteed to not decrease the dual objective function. This is in contrast to online algorithms such as the perceptron or stochastic gradient descent that may take steps in the wrong direction. (2) The algorithm never requires the computation of the kernel matrix, and instead maintains an estimate of the primal variable w. This means that the storage requirement is constant with respect to the number of training examples, rather than quadratic (as required for a kernel matrix).

Exploration vs optimization: A crucial question in terms of convergence time is the order of optimization of the dual variables α_{ij}. The above one-pass sequential algorithm takes an online perspective, where one continually explores new training examples. As w is being learned, at some point many (if not most) new examples will be "easy" and pass the margin test. For such cases, $\alpha_{ij} = 0$ and $g_{ij} \leq 0$, implying that the given dual coordinate step does not trigger an update, making learning inefficient. The LaRank algorithm [2] makes the observation that it is beneficial to revisit examples with a non-zero alpha, since they are more likely to trigger an update to their dual value. This can be implemented by maintaining a cache of "hard examples" [10], or support vectors and routinely optimizing over them while exploring new examples. This is basis for much of the literature on both batch and online SVMs, through the use of heuristics such as active-set selection [8, 18] and new-process/optimization steps [2]. We find that the precise scheduling of the optimization over new points (which we call *exploration*) versus existing support vectors (which we call *optimization*) is crucial.

Scheduling strategies: One scheduling strategy is to continually *explore* a new data point, analogous to the one-pass algorithm described above. Instead, LaRank suggests a exploring/optimization ratio of 10:1; revisit 10 examples from the cache for every new training example. Still another popular approach is to *explore* new data points by adding them to cache up to some fixed memory limit, and then *optimize* the cache to convergence and repeat. The hard-negative mining solver of [10] does exactly this. We make a number of observations to derive our strategy. First, successive calls to a dual solver can be hot-started from the previous dual solution, making them quite cheap. Secondly, it is advantageous to behave like an online algorithm (*explore*) during initial stages of learning, and behave more batch-like (*optimize*) during later stages of learning when the model is close to convergence. We propose here a novel on-line cutting plane algorithm [13, 14] that maintains running estimates of lower and upper bounds of the primal objective that naturally trade off the exploration of new data points versus the optimization of existing support vectors.

Online duality-gap: At any point in time during learning, we have a cache of examples and associated dual variables in memory $\{x_{ij}, l_{ij}, \alpha_{ij} : ij \in A\}$. Together, these completely specify a primal weight vector $w(\alpha) = \sum_{ij \in A} \alpha_{ij} x_{ij}$. We must decide between two choices; we can either further optimize α over the current examples in memory, or we can query for a brand new, unseen datapoint. From one perspective, the current cache of examples specifies a well-defined finite QP, and we may as well optimize that QP to completion. This would allow us to define a good w that would, in-turn, allow us to collect more relevant hard examples in the future. From another perspective, we might always choose to optimize with respect to new data (*explore*) rather than optimize over an example that we have already seen. We posit that one should ideally make the choice that produces the greatest increase in dual objective function value. Since it is difficult to compute the potential increase, we adopt the following strategy: we always choose to *explore* a new, unseen data point, unless the duality gap for the current QP solution is large (above *tol*), in which case we would rather *optimize* over the current cache to reduce the duality gap. We efficiently track the duality

gap in an online fashion with the following streaming algorithm:

Input: Streaming dataset $\{x_{ij}, l_{ij}\}$ and *tol*
Output: w

$A = \{\}$;
$w := 0$; $UB := 0$; $LB := 0$; // Initialize variables
for $i = 1 : \infty$ **do**
  Consider new example $x_i = \{x_{ij} : j \in N_i\}$; // Explore new example
  Compute j with maximum gradient $\max_j g_{ij}$ from (24);
  if $g_{ij} > 0$ **then** // Add to cache if it violates margin
    $UB := UB + g_{ij}$;
    $\alpha_{ij} := 0$;
    $A := A \cap (ij)$
  end
  if *UB - LB* $>$ *tol* **then** // Optimize over cache if duality gap is violated
    $(w, \alpha_A, LB, UB) = \text{Optimize}(\{x_A, l_A, \alpha_A\}, tol)$;
    $A := A \setminus \{ij : \alpha_{ij} = 0\}$; // Remove non-support-vectors from cache
  end
end

Algorithm 2: The above online algorithm performs one pass of learning across a dataset by maintaining a cache of examples indices $A = \{(ij)\}$. At any point in time, the associated dual variables $\{\alpha_{ij}\}$ encode the optimal model w for the QP defined by the cached examples, up to the duality gap *tol*.

Convergence: To examine the convergence of the online algorithm, let us make a distinction between two QP problems. The cached-QP is the QP defined by the current set of examples in cache A. The full-QP is the QP defined by the possibly infinitely-large set of examples in the full dataset. During the online algorithm, UB and LB are always upper and lower bounds on the cached-QP. LB is also a lower bound on the full-QP. One can derive this by setting dual variables for all examples not in the cache to 0, and scoring the resulting full-α vector under the dual objective, which must be $F(\alpha) = LB$. Crucially though, UB is *not* an upper bound on the full-QP. This makes intuitive sense; it is hard to upper bound our loss without seeing all the data. Hence convergence for the online algorithm cannot be strictly guaranteed. However, if we apply the online algorithm by cycling over the dataset multiple times, one can ensure convergence with similar arguments to our batch optimization algorithm. Moreover, after learning a weight vector $w(\alpha_A)$, we can verify that it is optimal by computing a true upper bound $L(w(\alpha_A))$. This can be computed with a single, out-of-core pass over the entire dataset. We find that in practice, a single pass through large datasets often suffices for convergence (that can be explicitly verified with an additional single pass).

5 Theoretical guarantees

In this section, we briefly point to some theoretical analysis that is necessary to ensure to show that the batch and cyclical-online version of our algorithm will converge. [3] describe theoretical guarantees about convex programming algorithms that rely on successive direction searches, just as our coordinate descent algorithm does. Such an analysis can be used to show that our coordinate descent algorithm is at the global optimum once no improvement to the dual objective can be made. In order to show this, one must prove that for any small step along any direction that is feasible (in the convex set defined by our constraints) from the current α, the value of the dual objective decreases. [3] prove that is suffices to show that no improvement can be made along a set of "witness" directions that form a basis for the feasible set of directions.

Joint optimization: If no constraints are active at a given α, the coordinate axes do define a basis set. This means that for examples i for which the linear constraint $\sum_j \alpha_{ij} \leq 1$ is not active, it suffices to ensure that the dual cannot be improved by independently perturbing each dual variable α_{ij}. However, this is not true for examples i with active linear constraints; its possible that no improvement can be made by taking a

step along any dual variable, but an improvement *can* be made by jointly optimizing a pair of dual variables. This necessitates the need for the switch clauses enumerating possibly active constraints, and precisely the reason why shared slacks in a structural SVM require joint optimization over pairs of dual variables.

Cyclic optimization: Consider an algorithm that randomly samples update directions with any distribution such that all feasible directions can be drawn with non-zero probability; [3] show that such an algorithm probably convergences to the optimum, within some specified tolerance, in finite time. Our batch algorithm and cyclic variant of our online algorithm satisfy this premise because they consider directions along each dual variable, as well as linear combinations of linearly-constrained variables.

6 Non-negativity constraints

We now describe a simple modification to our proposed algorithms that accept non-negativity constraints.

$$\begin{aligned} L(w,\xi) =& \frac{1}{2}||w||^2 + \sum_i \xi_i && (37)\\ \text{s.t.} \quad & w^T x_{ij} > l_{ij} - \xi_i \\ & \xi_i \geq 0 \\ & w_k \geq 0 \quad \forall k \in N && (38) \end{aligned}$$

We can define the Lagrangian as

$$L(w,b,\xi,\alpha,\mu,\beta_k) = \frac{1}{2}||w||^2 + \sum_i \xi_i - \sum_{ij} \alpha_{ij}(w \cdot x_{ij} - l_{ij} + \xi_i) - \sum_i \mu_i \xi_i - \beta \cdot w \tag{39}$$

By strong duality

$$\min_{w,b,\xi} \Big[\max_{\alpha \geq 0, \mu \geq 0, \beta \geq 0} L(w,b,\alpha,\mu) \Big] = \max_{\alpha \geq 0, \mu \geq 0, \beta \geq 0} \Big[\min_{w,b,\xi} L(w,b,\alpha,\mu) \Big] \tag{40}$$

We take the derivative of the Lagrangian with respect to the primal variables to get the KKT conditions:

$$w = \sum_{ij} \alpha_{ij} x_{ij} + \beta \tag{41}$$

$$\sum_j \alpha_{ij} \leq 1 \quad \forall i \tag{42}$$

We can write the dual of the QP in (37) as

$$\begin{aligned} F(\alpha,\beta) =& -\frac{1}{2}||\sum_{ij} \alpha_{ij} x_{ij} + \beta||^2 + \sum_{ij} l_{ij}\alpha_{ij} && (43)\\ \text{s.t.} \quad & \sum_j \alpha_{ij} \leq 1 \\ & \alpha_{ij} \geq 0 \end{aligned}$$

We iterate between optimizing a single α_i holding β fixed, followed by optimizing β. One can show this is equivalent to zero-ing our negative parameters during dual updates:

1. Update α_{ij}, w with a coordinate descent update.
2. Update β by $w[k] = \max(w[k], 0), \forall k \in \{1 \dots N\}$.

7 Flexible regularization

This section will describe a method for using the aforementioned solver to solve a more general SVM problem with a Gaussian regularization or "prior" on w given by (μ, Σ):

$$\begin{aligned} \underset{w,\xi}{\operatorname{argmin}} & \frac{1}{2}||(w - w_0)R||^2 + \sum_i \xi_i \\ \text{s.t.} \quad & w^T x_{ij} > l_{ij} - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{44}$$

where $w_0 = \mu$, and $R = \Sigma^{-1/2}$. We can massage (44) into (2) with the substitution $\hat{w} = (w - w_0)R$:

$$\begin{aligned} \underset{\hat{w},\xi}{\operatorname{argmin}} & \frac{1}{2}||\hat{w}||^2 + \sum_i \xi_i \\ \text{s.t.} \quad & \hat{w}^T \hat{x}_{ij} > \hat{l}_{ij} - \xi_i \\ & \xi_i \geq 0 \\ \text{where } & \hat{w} = (w - w_0)R \\ & \hat{x}_{ij} = R^{-1} x_{ij} \\ & \hat{l}_{ij} = l_{ij} - w_o \cdot x_{ij} \end{aligned} \tag{45}$$

We assume that Σ is full rank, implying that R^{-1} exists. An important special case is given by a diagonal matrix Σ, which corresponds to an arbitrary regularization of each parameter associated with a particular feature. This is useful, for example, when regularizing a feature vector constructed from heterogeneous features (such as appearance features, spatial features, and biases). After solving for the re-parametrized weight vector $\hat{w}$ by optimizing the QP from (45), one can recover the score of the original weight vector with the following:

$$w \cdot x_{ij} = (\hat{w} + w_0 R) \cdot \hat{x}_{ij} \tag{46}$$

8 Conclusion

We have described a dual coordinate solver for solving general SVM problems (including multiclass, structural, and latent variations) with out-of-core, or even streaming datasets. The ideas described here are implemented in publicly available solvers released in [19, 7, 21, 12]. We gladly acknowledge co-authors for numerous discussions and debugging efforts.

References

[1] Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz. Applying support vector machines to imbalanced datasets. In *Machine Learning: ECML 2004*, pages 39–50. Springer, 2004. 2

[2] A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with LaRank. In *ICML*, pages 89–96. ACM, 2007. 9

[3] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *The Journal of Machine Learning Research*, 6:1579–1619, 2005. 10, 11

[4] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20:161–168, 2008. 1

[5] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292, 2002. 2, 3

[6] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005*, volume 1, 2005. 1

[7] Chaitanya Desai and Deva Ramanan. Detecting actions, poses, and objects with relational phraselets. *Computer Vision–ECCV 2012*, pages 158–172, 2012. 1, 12

[8] R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, and C.J. Lin. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008. 1, 5, 9

[9] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. *Computer Vision and Pattern Recognition, Anchorage, USA, June*, 2008. 1, 2, 3

[10] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part based models. *IEEE TPAMI*, 32(9), 2010. 1, 2, 9

[11] V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *Proceedings of the 25th international conference on Machine learning*, pages 320–327. ACM New York, NY, USA, 2008. 8

[12] Mohsen Hejrati and Deva Ramanan. Analyzing 3d objects in cluttered images. In *Advances in Neural Information Processing Systems*, pages 602–610, 2012. 1, 12

[13] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM New York, NY, USA, 2006. 9

[14] T. Joachims, T. Finley, and Chun-Nam Yu. Cutting-plane training of structural svms. *Machine Learning*, to appear,Joachims/etal/09a. 9

[15] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, pages 807–814. ACM, 2007. 1

[16] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004. 2

[17] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. *Advances in neural information processing systems*, 16, 2003. 2

[18] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*. ACM New York, NY, USA, 2004. 2, 9

[19] Y. Yang and D. Ramanan. Articulated pose estimation with flexible mixtures-of-parts. In *CVPR*. http://phoenix.ics.uci.edu/software/pose/. 1, 12

[20] C.N.J. Yu and T. Joachims. Learning structural svms with latent variables. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1169–1176. ACM, 2009. 2

[21] Xiangxin Zhu and D Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2879–2886. IEEE, 2012. http://www.ics.uci.edu/~xzhu/face/. 1, 12