Efficient Projections onto the ℓ_1-Ball for Learning in High Dimensions

John Duchi JDUCHI@CS.STANFORD.EDU
Google, Mountain View, CA 94043

Shai Shalev-Shwartz SHAI@TTI-C.ORG
Toyota Technological Institute, Chicago, IL, 60637

Yoram Singer SINGER@GOOGLE.COM
Tushar Chandra TUSHAR@GOOGLE.COM
Google, Mountain View, CA 94043

Abstract

We describe efficient algorithms for projecting a vector onto the ℓ_1-ball. We present two methods for projection. The first performs exact projection in $O(n)$ expected time, where n is the dimension of the space. The second works on vectors k of whose elements are perturbed outside the ℓ_1-ball, projecting in $O(k \log(n))$ time. This setting is especially useful for online learning in sparse feature spaces such as text categorization applications. We demonstrate the merits and effectiveness of our algorithms in numerous batch and online learning tasks. We show that variants of stochastic gradient projection methods augmented with our efficient projection procedures outperform interior point methods, which are considered state-of-the-art optimization techniques. We also show that in online settings gradient updates with ℓ_1 projections outperform the exponentiated gradient algorithm while obtaining models with high degrees of sparsity.

1. Introduction

A prevalent machine learning approach for decision and prediction problems is to cast the learning task as penalized convex optimization. In penalized convex optimization we seek a set of parameters, gathered together in a vector $\mathbf{w}$, which minimizes a convex objective function in $\mathbf{w}$ with an additional penalty term that assesses the complexity of $\mathbf{w}$. Two commonly used penalties are the 1-norm and the square of the 2-norm of $\mathbf{w}$. An alternative but mathematically equivalent approach is to cast the problem as a *constrained* optimization problem. In this setting we seek a minimizer of the objective function while constraining the solution to have a bounded norm. Many recent advances in statistical machine learning and related fields can be explained as convex optimization subject to a 1-norm constraint on the vector of parameters $\mathbf{w}$. Imposing an ℓ_1 constraint leads to notable benefits. First, it encourages sparse solutions, *i.e* a solution for which many components of $\mathbf{w}$ are zero. When the original dimension of $\mathbf{w}$ is very high, a sparse solution enables easier interpretation of the problem in a lower dimension space. For the usage of ℓ_1-based approach in statistical machine learning see for example (Tibshirani, 1996) and the references therein. Donoho (2006b) provided sufficient conditions for obtaining an optimal ℓ_1-norm solution which is sparse. Recent work on compressed sensing (Candes, 2006; Donoho, 2006a) further explores how ℓ_1 constraints can be used for recovering a sparse signal sampled below the Nyquist rate. The second motivation for using ℓ_1 constraints in machine learning problems is that in some cases it leads to improved generalization bounds. For example, Ng (2004) examined the task of PAC learning a sparse predictor and analyzed cases in which an ℓ_1 constraint results in better solutions than an ℓ_2 constraint.

In this paper we re-examine the task of minimizing a convex function subject to an ℓ_1 constraint on the norm of the solution. We are particularly interested in cases where the convex function is the average loss over a training set of m examples where each example is represented as a vector of high dimension. Thus, the solution itself is a high-dimensional vector as well. Recent work on ℓ_2 constrained optimization for machine learning indicates that gradient-related projection algorithms are more efficient in approaching a solution of good generalization than second-order algorithms when the number of examples and

Appearing in *Proceedings of the 25^{th} International Conference on Machine Learning*, Helsinki, Finland, 2008.

the dimension are large. For instance, Shalev-Shwartz et al. (2007) give recent state-of-the-art methods for solving large scale support vector machines. Adapting these recent results to projection methods onto the ℓ_1 ball poses algorithmic challenges. While projections onto ℓ_2 balls are straightforward to implement in linear time with the appropriate data structures, projection onto an ℓ_1 ball is a more involved task. The main contribution of this paper is the derivation of gradient projections with ℓ_1 domain constraints that can be performed almost as fast as gradient projection with ℓ_2 constraints.

Our starting point is an efficient method for projection onto the probabilistic simplex. The basic idea is to show that, after sorting the vector we need to project, it is possible to calculate the projection exactly in linear time. This idea was rediscovered multiple times. It was first described in an abstract and somewhat opaque form in the work of Gafni and Bertsekas (1984) and Bertsekas (1999). Crammer and Singer (2002) rediscovered a similar projection algorithm as a tool for solving the dual of multiclass SVM. Hazan (2006) essentially reuses the same algorithm in the context of online convex programming. Our starting point is another derivation of Euclidean projection onto the simplex that paves the way to a few generalizations. First we show that the same technique can also be used for projecting onto the ℓ_1-ball. This algorithm is based on sorting the components of the vector to be projected and thus requires $O(n \log(n))$ time. We next present an improvement of the algorithm that replaces sorting with a procedure resembling median-search whose expected time complexity is $O(n)$.

In many applications, however, the dimension of the feature space is very high yet the number of features which attain non-zero values for an example may be very small. For instance, in our experiments with text classification in Sec. 7, the dimension is two million (the bigram dictionary size) while each example has on average one-thousand non-zero features (the number of unique tokens in a document). Applications where the dimensionality is high yet the number of "on" features in each example is small render our second algorithm useless in some cases. We therefore shift gears and describe a more complex algorithm that employs red-black trees to obtain a linear dependence on the number of non-zero features in an example and only logarithmic dependence on the full dimension. The key to our construction lies in the fact that we project vectors that are the sum of a vector in the ℓ_1-ball and a sparse vector—they are "almost" in the ℓ_1-ball.

In conclusion to the paper we present experimental results that demonstrate the merits of our algorithms. We compare our algorithms with several specialized interior point (IP) methods as well as general methods from the literature for solving ℓ_1-penalized problems on both synthetic and real data (the MNIST handwritten digit dataset and the Reuters RCV1 corpus) for batch and online learning. Our projection based methods outperform competing algorithms in terms of sparsity, and they exhibit faster convergence and lower regret than previous methods.

2. Notation and Problem Setting

We start by establishing the notation used throughout the paper. The set of integers 1 through n is denoted by $[n]$. Scalars are denoted by lower case letters and vectors by lower case bold face letters. We use the notation $\mathbf{w} \succ b$ to designate that all of the components of $\mathbf{w}$ are greater than b. We use $\|\cdot\|$ as a shorthand for the Euclidean norm $\|\cdot\|_2$. The other norm we use throughout the paper is the 1-norm of the vector, $\|\mathbf{v}\|_1 = \sum_{i=1}^n |v_i|$. Lastly, we consider order statistics and sorting vectors frequently throughout this paper. To that end, we let $v_{(i)}$ denote the i^{th} order statistic of $\mathbf{v}$, that is, $v_{(1)} \geq v_{(2)} \geq \ldots \geq v_{(n)}$ for $\mathbf{v} \in \mathbb{R}^n$.

In the setting considered in this paper we are provided with a convex function $L : \mathbb{R}^n \to \mathbb{R}$. Our goal is to find the minimum of $L(\mathbf{w})$ subject to an ℓ_1-norm constraint on $\mathbf{w}$. Formally, the problem we need to solve is

$$\underset{\mathbf{w}}{\text{minimize}}\, L(\mathbf{w}) \quad \text{s.t.} \quad \|\mathbf{w}\|_1 \leq z \ . \tag{1}$$

Our focus is on variants of the projected subgradient method for convex optimization (Bertsekas, 1999). Projected subgradient methods minimize a function $L(\mathbf{w})$ subject to the constraint that $\mathbf{w} \in X$, for X convex, by generating the sequence $\{\mathbf{w}^{(t)}\}$ via

$$\mathbf{w}^{(t+1)} = \Pi_X \left(\mathbf{w}^{(t)} - \eta_t \nabla^{(t)} \right) \tag{2}$$

where $\nabla^{(t)}$ is (an unbiased estimate of) the (sub)gradient of L at $\mathbf{w}^{(t)}$ and $\Pi_X(\mathbf{x}) = \operatorname{argmin}_{\mathbf{y}} \{\|\mathbf{x} - \mathbf{y}\| \mid \mathbf{y} \in X\}$ is Euclidean projection of $\mathbf{x}$ onto X. In the rest of the paper, the main algorithmic focus is on the projection step (computing an unbiased estimate of the gradient of $L(\mathbf{w})$ is straightforward in the applications considered in this paper, as is the modification of $\mathbf{w}^{(t)}$ by $\nabla^{(t)}$).

3. Euclidean Projection onto the Simplex

For clarity, we begin with the task of performing Euclidean projection onto the positive simplex; our derivation naturally builds to the more efficient algorithms. As such, the most basic projection task we consider can be formally described as the following optimization problem,

$$\underset{\mathbf{w}}{\text{minimize}}\, \frac{1}{2}\|\mathbf{w} - \mathbf{v}\|_2^2 \quad \text{s.t.} \quad \sum_{i=1}^n w_i \ = \ z \ , \ \ w_i \geq 0 \ . \tag{3}$$

When $z = 1$ the above is projection onto the probabilistic simplex. The Lagrangian of the problem in Eq. (3) is

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\zeta}) \;=\; \frac{1}{2}\|\mathbf{w} - \mathbf{v}\|^2 + \theta\left(\sum_{i=1}^{n} w_i - z\right) - \boldsymbol{\zeta}\cdot\mathbf{w} \;\;,$$

where $\theta \in \mathbb{R}$ is a Lagrange multiplier and $\boldsymbol{\zeta} \in \mathbb{R}^n_+$ is a vector of non-negative Lagrange multipliers. Differentiating with respect to w_i and comparing to zero gives the optimality condition, $\frac{d\mathcal{L}}{dw_i} = w_i - v_i + \theta - \zeta_i = 0$. The complementary slackness KKT condition implies that whenever $w_i > 0$ we must have that $\zeta_i = 0$. Thus, if $w_i > 0$ we get that

$$w_i \;=\; v_i - \theta + \zeta_i \;=\; v_i - \theta \;. \tag{4}$$

All the non-negative elements of the vector $\mathbf{w}$ are tied via a single variable, so knowing the indices of these elements gives a much simpler problem. Upon first inspection, finding these indices seems difficult, but the following lemma (Shalev-Shwartz & Singer, 2006) provides a key tool in deriving our procedure for identifying non-zero elements.

Lemma 1. *Let $\mathbf{w}$ be the optimal solution to the minimization problem in Eq. (3). Let s and j be two indices such that $v_s > v_j$. If $w_s = 0$ then w_j must be zero as well.*

Denoting by I the set of indices of the non-zero components of the sorted optimal solution, $I = \{i \in [n] : v_{(i)} > 0\}$, we see that Lemma 1 implies that $I = [\rho]$ for some $1 \le \rho \le n$. Had we known ρ we could have simply used Eq. (4) to obtain that

$$\sum_{i=1}^{n} w_i = \sum_{i=1}^{n} w_{(i)} = \sum_{i=1}^{\rho} w_{(i)} = \sum_{i=1}^{\rho} \left(v_{(i)} - \theta\right) = z$$

and therefore

$$\theta = \frac{1}{\rho}\left(\sum_{i=1}^{\rho} v_{(i)} - z\right) \;\;. \tag{5}$$

Given θ we can characterize the optimal solution for $\mathbf{w}$ as

$$w_i \;=\; \max\left\{v_i - \theta \;,\; 0\right\} \;\;. \tag{6}$$

We are left with the problem of finding the optimal ρ, and the following lemma (Shalev-Shwartz & Singer, 2006) provides a simple solution once we sort $\mathbf{v}$ in descending order.

Lemma 2. *Let $\mathbf{w}$ be the optimal solution to the minimization problem given in Eq. (3). Let $\boldsymbol{\mu}$ denote the vector obtained by sorting $\mathbf{v}$ in a descending order. Then, the number of strictly positive elements in $\mathbf{w}$ is*

$$\rho(z, \boldsymbol{\mu}) \;=\; \max\left\{j \in [n] \;:\; \mu_j - \frac{1}{j}\left(\sum_{r=1}^{j} \mu_r - z\right) > 0\right\} \;\;.$$

The pseudo-code describing the $O(n \log n)$ procedure for solving Eq. (3) is given in Fig. 1.

INPUT: A vector $\mathbf{v} \in \mathbb{R}^n$ and a scalar $z > 0$

Sort $\mathbf{v}$ into $\boldsymbol{\mu} : \mu_1 \ge \mu_2 \ge \ldots \ge \mu_p$

Find $\rho = \max\left\{j \in [n] : \mu_j - \frac{1}{j}\left(\sum_{r=1}^{j} \mu_r - z\right) > 0\right\}$

Define $\theta = \frac{1}{\rho}\left(\sum_{i=1}^{\rho} \mu_i - z\right)$

OUTPUT: $\mathbf{w}$ s.t. $w_i = \max\{v_i - \theta \;,\; 0\}$

Figure 1. Algorithm for projection onto the simplex.

4. Euclidean Projection onto the ℓ_1-Ball

We next modify the algorithm to handle the more general ℓ_1-norm constraint, which gives the minimization problem

$$\underset{\mathbf{w}\in\mathbb{R}^n}{\text{minimize}} \;\; \|\mathbf{w} - \mathbf{v}\|_2^2 \quad \text{s.t.} \quad \|\mathbf{w}\|_1 \le z \;\;. \tag{7}$$

We do so by presenting a reduction to the problem of projecting onto the simplex given in Eq. (3). First, we note that if $\|\mathbf{v}\|_1 \le z$ then the solution of Eq. (7) is $\mathbf{w} = \mathbf{v}$. Therefore, from now on we assume that $\|\mathbf{v}\|_1 > z$. In this case, the optimal solution must be on the boundary of the constraint set and thus we can replace the inequality constraint $\|\mathbf{w}\|_1 \le z$ with an equality constraint $\|\mathbf{w}\|_1 = z$. Having done so, the sole difference between the problem in Eq. (7) and the one in Eq. (3) is that in the latter we have an additional set of constraints, $\mathbf{w} \ge 0$. The following lemma indicates that each non-zero component of the optimal solution $\mathbf{w}$ shares the sign of its counterpart in $\mathbf{v}$.

Lemma 3. *Let $\mathbf{w}$ be an optimal solution of Eq. (7). Then, for all i, $w_i\, v_i \ge 0$.*

Proof. Assume by contradiction that the claim does not hold. Thus, there exists i for which $w_i\, v_i < 0$. Let $\hat{\mathbf{w}}$ be a vector such that $\hat{w}_i = 0$ and for all $j \ne i$ we have $\hat{w}_j = w_j$. Therefore, $\|\hat{\mathbf{w}}\|_1 = \|\mathbf{w}\|_1 - |w_i| \le z$ and hence $\hat{\mathbf{w}}$ is a feasible solution. In addition,

$$\begin{aligned} \|\mathbf{w} - \mathbf{v}\|_2^2 - \|\hat{\mathbf{w}} - \mathbf{v}\|_2^2 &= (w_i - v_i)^2 - (0 - v_i)^2 \\ &= w_i^2 - 2w_i v_i > w_i^2 > 0 \;\;. \end{aligned}$$

They are symmetry.

We thus constructed a feasible solution $\hat{\mathbf{w}}$ which attains an objective value smaller than that of $\mathbf{w}$. This leads us to the desired contradiction. □

Based on the above lemma and the symmetry of the objective, we are ready to present our reduction. Let $\mathbf{u}$ be a vector obtained by taking the absolute value of each component of $\mathbf{v}$, $u_i = |v_i|$. We now replace Eq. (7) with

$$\underset{\boldsymbol{\beta}\in\mathbb{R}^n}{\text{minimize}} \;\; \|\boldsymbol{\beta} - \mathbf{u}\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{\beta}\|_1 \le z \;\text{ and }\; \boldsymbol{\beta} \ge 0 \;\;. \tag{8}$$

Once we obtain the solution for the problem above we construct the optimal of Eq. (7) by setting $w_i = \text{sign}(v_i)\,\beta_i$.

INPUT A vector $\mathbf{v} \in \mathbb{R}^n$ and a scalar $z > 0$
INITIALIZE $U = [n]$ $s = 0$ $\rho = 0$
WHILE $U \neq \phi$
  PICK $k \in U$ at random
  PARTITION U:
    $G = \{j \in U \mid v_j \geq v_k\}$
    $L = \{j \in U \mid v_j < v_k\}$
  CALCULATE $\Delta\rho = |G|$; $\Delta s = \sum_{j \in G} v_j$
  IF $(s + \Delta s) - (\rho + \Delta\rho)v_k < z$
    $s = s + \Delta s$; $\rho = \rho + \Delta\rho$; $U \leftarrow L$
  ELSE
    $U \leftarrow G \setminus \{k\}$
  ENDIF
SET $\theta = (s - z)/\rho$
OUTPUT $\mathbf{w}$ s.t. $v_i = \max\{v_i - \theta\ ,\ 0\}$

Figure 2. Linear time projection onto the simplex.

5. A Linear Time Projection Algorithm

In this section we describe a more efficient algorithm for performing projections. To keep our presentation simple and easy to follow, we describe the projection algorithm onto the simplex. The generalization to the ℓ_1 ball can straightforwardly incorporated into the efficient algorithm by the results from the previous section (we simply work in the algorithm with a vector of the absolute values of $\mathbf{v}$, replacing the solution's components w_i with $\text{sign}(v_i) \cdot w_i$).

For correctness of the following discussion, we add another component to $\mathbf{v}$ (the vector to be projected), which we set to 0, thus $v_{n+1} = 0$ and $v_{(n+1)} = 0$. Let us start by examining again Lemma 2. The lemma implies that the index ρ is the largest integer that still satisfies $v_{(\rho)} - \frac{1}{\rho}\left(\sum_{r=1}^{\rho} v_{(r)} - z\right) > 0$. After routine algebraic manipulations the above can be rewritten in the following somewhat simpler form:

$$\sum_{i=1}^{\rho} \left(v_{(i)} - v_{(\rho)}\right) < z \ \text{ and } \ \sum_{i=1}^{\rho+1} \left(v_{(i)} - v_{(\rho+1)}\right) \geq z. \quad (9)$$

Given ρ and $v_{(\rho)}$ we slightly rewrite the value θ as follows,

$$\theta = \frac{1}{\rho}\left(\sum_{j:v_j \geq v_{(\rho)}} v_j - z\right) \ . \quad (10)$$

The task of projection can thus be distilled to the task of finding θ, which in turn reduces to the task of finding ρ and the pivot element $v_{(\rho)}$. Our problem thus resembles the task of finding an order statistic with an additional complicating factor stemming from the need to compute summations (while searching) of the form given by Eq. (9). Our efficient projection algorithm is based on a modification of the randomized median finding algorithm (Cormen et al., 2001). The algorithm computes partial sums just-in-time and has expected linear time complexity.

The algorithm identifies ρ and the pivot value $v_{(\rho)}$ without sorting the vector $\mathbf{v}$ by using a divide and conquer procedure. The procedure works in rounds and on each round either eliminates elements shown to be strictly smaller than $v_{(\rho)}$ or updates the partial sum leading to Eq. (9). To do so the algorithm maintains a set of unprocessed elements of $\mathbf{v}$. This set contains the components of $\mathbf{v}$ whose relationship to $v_{(\rho)}$ we do not know. We thus initially set $U = [n]$. On each round of the algorithm we pick at random an index k from the set U. Next, we partition the set U into two subsets G and L. G contains all the indices $j \in U$ whose components $v_j > v_k$; L contains those $j \in U$ such that v_j is smaller. We now face two cases related to the current summation of entries in $\mathbf{v}$ greater than the hypothesized $v_{(\rho)}$ (i.e. v_k). If $\sum_{j:v_j \geq v_k}(v_j - v_k) < z$ then by Eq. (9), $v_k \geq v_{(\rho)}$. In this case we know that all the elements in G participate in the sum defining θ as given by Eq. (9). We can discard G and set U to be L as we still need to further identify the remaining elements in L. If $\sum_{j:v_j \geq v_k}(v_j - v_k) \geq z$ then the same rationale implies that $v_k < v_{(\rho)}$. Thus, all the elements in L are smaller than $v_{(\rho)}$ and can be discarded. In this case we can remove the set L and v_k and set U to be $G \setminus \{k\}$. The entire process ends when U is empty.

Along the process we also keep track of the sum and the number of elements in $\mathbf{v}$ that we have found thus far to be no smaller than $v_{(\rho)}$, which is required in order not to recalculate partial sums. The pseudo-code describing the efficient projection algorithm is provided in Fig. 2. We keep the set of elements found to be greater than $v_{(\rho)}$ only *implicitly*. Formally, at each iteration of the algorithm we maintain a variable s, which is the sum of the elements in the set $\{v_j : j \notin U, v_j \geq v_{(\rho)}\}$, and overload ρ to designate the cardinality of the this set throughout the algorithm. Thus, when the algorithms exits its main while loop, ρ is the maximizer defined in Lemma 1. Once the while loop terminates, we are left with the task of calculating θ using Eq. (10) and performing the actual projection. Since $\sum_{j:v_j \geq \mu_\rho} v_j$ is readily available to us as the variable s, we simply set θ to be $(s - z)/\rho$ and perform the projection as prescribed by Eq. (6).

Though omitted here for lack of space, we can also extend the algorithms to handle the more general constraint that $\sum a_i|w_i| \leq z$ for $a_i \geq 0$.

6. Efficient Projection for Sparse Gradients

Before we dive into developing a new algorithm, we remind the reader of the iterations the minimization algorithm takes from Eq. (2): we generate a sequence $\{\mathbf{w}^{(t)}\}$

INPUT A balanced tree $\mathcal{T}$ and a scalar $z > 0$
INITIALIZE $v^\star = \infty$, $\rho^* = n + 1$, $s^* = z$
CALL PIVOTSEARCH(root($\mathcal{T}$), 0, 0)
PROCEDURE PIVOTSEARCH(v, ρ, s)
  COMPUTE $\hat{\rho} = \rho + r(v)$; $\hat{s} = s + \sigma(v)$
  IF $\hat{s} < v\hat{\rho} + z$ // $v \geq$ pivot
    IF $v^\star > v$
      $v^\star = v$; $\rho^\star = \hat{\rho}$; $s^\star = \hat{s}$
    ENDIF
    IF leaf$_{\mathcal{T}}(v)$
      RETURN $\theta = (s^\star - z)/\rho^\star$
    ENDIF
    CALL PIVOTSEARCH(left$_{\mathcal{T}}(v), \hat{\rho}, \hat{s}$)
  ELSE // $v <$ pivot
    IF leaf$_{\mathcal{T}}(v)$
      RETURN $\theta = (s^\star - z)/\rho^\star$
    ENDIF
    CALL PIVOTSEARCH(right$_{\mathcal{T}}(v), \rho, s$)
  ENDIF
ENDPROCEDURE

Figure 3. Efficient search of pivot value for sparse feature spaces.

by iterating

$$\mathbf{w}^{(t+1)} = \Pi_W \left(\mathbf{w}^{(t)} + \mathbf{g}^{(t)} \right)$$

where $\mathbf{g}^{(t)} = -\eta_t \nabla^{(t)}$, $W = \{\mathbf{w} \mid \|\mathbf{w}\|_1 \leq z\}$ and Π_W is projection onto this set.

In many applications the dimension of the feature space is very high yet the number of features which attain a non-zero value for each example is very small (see for instance our experiments on text documents in Sec. 7). It is straightforward to implement the gradient-related updates in time which is proportional to the number of non-zero features, but the time complexity of the projection algorithm described in the previous section is linear in the dimension. Therefore, using the algorithm verbatim could be prohibitively expensive in applications where the dimension is high yet the number of features which are "on" in each example is small. In this section we describe a projection algorithm that updates the vector $\mathbf{w}^{(t)}$ with $\mathbf{g}^{(t)}$ and scales linearly in the number of non-zero entries of $\mathbf{g}^{(t)}$ and only *logarithmically* in the total number of features (*i.e.* non-zeros in $\mathbf{w}^{(t)}$).

The first step in facilitating an efficient projection for sparse feature spaces is to represent the projected vector as a "raw" vector $\mathbf{v}$ by incorporating a global shift that is applied to each non-zero component. Specifically, each projection step amounts to deducting θ from each component of $\mathbf{v}$ and thresholding the result at zero. Let us denote by θ_t the shift value used on the t^{th} iteration of the algorithm and by Θ_t the cumulative sum of the shift values, $\Theta_t = \sum_{s \leq t} \theta_s$. The representation we employ enables us to perform the step in which we deduct θ_t from all the elements of the vector *implicitly*, adhering to the goal of performing a sub-linear number of operations. As before, we assume that the goal is to project onto the simplex. Equipped with these variables, the j^{th} component of the projected vector after t projected gradient steps can be written as $\max\{v_j - \Theta_t, 0\}$.

The second substantial modification to the core algorithm is to keep only the *non-zero* components of the weight vector in a red-black tree (Cormen et al., 2001). The red-black tree facilitates an efficient search for the pivot element ($v_{(\rho)}$) in time which is logarithmic in the dimension, as we describe in the sequel. Once the pivot element is found we implicitly deduct θ_t from all the non-zero elements in our weight vector by updating Θ_t. We then remove all the components that are less than $v_{(\rho)}$ (i.e. less than Θ_t); this removal is efficient and requires only logarithmic time (Tarjan, 1983).

The course of the algorithm is as follows. After t projected gradient iterations we have a vector $\mathbf{v}^{(t)}$ whose non-zero elements are stored in a red-black tree $\mathcal{T}$ and a global deduction value Θ_t which is applied to each non-zero component just-in-time, *i.e.* when needed. Therefore, each non-zero weight is accessed as $v_j - \Theta_t$ while $\mathcal{T}$ does not contain the zero elements of the vector. When updating $\mathbf{v}$ with a gradient, we modify the vector $\mathbf{v}^{(t)}$ by adding to it the gradient-based vector $\mathbf{g}^{(t)}$ with k non-zero components. This update is done using k deletions (removing v_i from $\mathcal{T}$ such that $g_i^{(t)} \neq 0$) followed by k re-insertions of $v_i' = (v_i + g_i^{(t)})$ into $\mathcal{T}$, which takes $O(k \log(n))$ time. Next we find in $O(\log(n))$ time the value of θ_t. Fig. 3 contains the algorithm for this step; it is explained in the sequel. The last step removes all elements of the new raw vector $\mathbf{v}^{(t)} + \mathbf{g}^{(t)}$ which become zero due to the projection. This step is discussed at the end of this section.

In contrast to standard tree-based search procedure, to find θ_t we need to find a pair of consecutive values in $\mathbf{v}$ that correspond to $v_{(\rho)}$ and $v_{(\rho+1)}$. We do so by keeping track of the smallest element that satisfies the left hand side of Eq. (9) while searching based on the condition given on the right hand side of the same equation. $\mathcal{T}$ is keyed on the values of the un-shifted vector $\mathbf{v}_t$. Thus, all the children in the left (right) sub-tree of a node v represent values in $\mathbf{v}_t$ which are smaller (larger) than v. In order to efficiently find θ_t we keep at each node the following information: (a) The value of the component, simply denoted as v. (b) The number of elements in the right sub-tree rooted at v, denoted $r(v)$, including the node v. (c) The sum of the elements in the right sub-tree rooted at v, denoted $\sigma(v)$, including the value v itself. Our goal is to identify the pivot element $v_{(\rho)}$ and its index ρ. In the previous section we described a simple condition for checking whether an element in $\mathbf{v}$ is greater or smaller than the pivot value. We now rewrite this expression yet one more time. A component with value v is *not*

smaller than the pivot iff the following holds:

$$\sum_{j:v_j \geq v} v_j > |\{j : v_j \geq v\}| \cdot v + z \; . \tag{11}$$

The variables in the red-black tree form the infrastructure for performing efficient recursive computation of Eq. (11). Note also that the condition expressed in Eq. (11) still holds when we do *not* deduct Θ_t from all the elements in $\mathbf{v}$.

The search algorithm maintains recursively the number ρ and the sum s of the elements that have been shown to be greater or equal to the pivot. We start the search with the root node of $\mathcal{T}$, and thus initially $\rho = 0$ and $s = 0$. Upon entering a new node v, the algorithm checks whether the condition given by Eq. (11) holds for v. Since ρ and s were computed for the parent of v, we need to incorporate the number and the sum of the elements that are larger than v itself. By construction, these variables are $r(v)$ and $\sigma(v)$, which we store at the node v itself. We let $\hat{\rho} = \rho + r(v)$ and $\hat{s} = s + \sigma(v)$, and with these variables handy, Eq. (11) distills to the expression $\hat{s} < v\hat{\rho} + z$. If the inequality holds, we know that v is either larger than the pivot or it may be the pivot itself. We thus update our current hypothesis for μ_ρ and ρ (designated as $v^\star$ and $\rho^\star$ in Fig. 3). We continue searching the left sub-tree ($\text{left}_{\mathcal{T}}(v)$) which includes all elements smaller than v. If inequality $\hat{s} < v\hat{\rho} + z$ does not hold, we know that $v < \mu_\rho$, and we thus search the right subtree ($\text{right}_{\mathcal{T}}(v)$) and keep ρ and s intact. The process naturally terminates once we reach a leaf, where we can also calculate the correct value of θ using Eq. (10).

Once we find θ_t (if $\theta_t \geq 0$) we update the global shift, $\Theta_{t+1} = \Theta_t + \theta_t$. We need to discard all the elements in $\mathcal{T}$ smaller than Θ_{t+1}, which we do using Tarjan's (1983) algorithm for splitting a red-black tree. This step is logarithmic in the total number of non-zero elements of $\mathbf{v}_t$. Thus, as the additional variables in the tree can be updated in constant time as a function of a node's child nodes in $\mathcal{T}$, each of the operations previously described can be performed in logarthmic time (Cormen et al., 2001), giving us a total update time of $O(k \log(n))$.

7. Experiments

We now present experimental results demonstrating the effectiveness of the projection algorithms. We first report results for experiments with synthetic data and then move to experiments with high dimensional natural datasets.

In our experiment with synthetic data, we compared variants of the projected subgradient algorithm (Eq. (2)) for ℓ_1-regularized least squares and ℓ_1-regularized logistic regression. We compared our methods to a specialized coordinate-descent solver for the least squares problem due to Friedman et al. (2007) and to very fast interior point

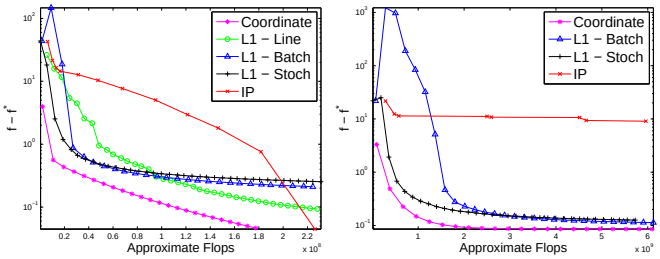

Figure 4. Comparison of methods on ℓ_1-regularized least squares. The left has dimension $n = 800$, the right $n = 4000$

methods for both least squares and logistic regression (Koh et al., 2007; Kim et al., 2007). The algorithms we use are batch projected gradient, stochastic projected subgradient, and batch projected gradient augmented with a backtracking line search (Koh et al., 2007). The IP and coordinate-wise methods both solve regularized loss functions of the form $f(\mathbf{w}) = L(\mathbf{w}) + \lambda\|\mathbf{w}\|_1$ rather than having an ℓ_1-domain constraint, so our objectives are not directly comparable. To surmount this difficulty, we first minimize $L(\mathbf{w}) + \lambda\|\mathbf{w}\|_1$ and use the 1-norm of the resulting solution $\mathbf{w}^*$ as the constraint for our methods.

To generate the data for the least squares problem setting, we chose a $\mathbf{w}$ with entries distributed normally with 0 mean and unit variance and randomly zeroed 50% of the vector. The data matrix $X \in \mathbb{R}^{m \times n}$ was random with entries also normally distributed. To generate target values for the least squares problem, we set $\mathbf{y} = X\mathbf{w} + \boldsymbol{\nu}$, where the components of $\boldsymbol{\nu}$ were also distributed normally at random. In the case of logistic regression, we generated data X and the vector $\mathbf{w}$ identically, but the targets y_i were set to be $\text{sign}(\mathbf{w} \cdot \mathbf{x}_i)$ with probability 90% and to $-\text{sign}(\mathbf{w} \cdot \mathbf{x}_i)$ otherwise. We ran two sets of experiments, one each for $n = 800$ and $n = 4000$. We also set the number of examples m to be equal to n. For the subgradient methods in these experiments and throughout the remainder, we set $\eta_t = \eta_0/\sqrt{t}$, choosing η_0 to give reasonable performance. (η_0 too large will mean that the initial steps of the gradient method are not descent directions; the noise will quickly disappear because the step sizes are proportional to $1/\sqrt{t}$).

Fig. 4 and Fig. 5 contain the results of these experiments and plot $f(\mathbf{w}) - f(\mathbf{w}^*)$ as a function of the number of floating point operations. From the figures, we see that the projected subgradient methods are generally very fast at the outset, getting us to an accuracy of $f(\mathbf{w}) - f(\mathbf{w}^*) \leq 10^{-2}$ quickly, but their rate of convergence slows over time. The fast projection algorithms we have developed, however, allow projected-subgradient methods to be very competitive with specialized methods, even on these relatively small problem sizes. On higher-dimension data sets interior point methods are infeasible or very slow. The rightmost graphs in Fig. 4 and Fig. 5 plot $f(\mathbf{w}) - f(\mathbf{w}^*)$ as functions of floating point operations for least squares and logistic re-

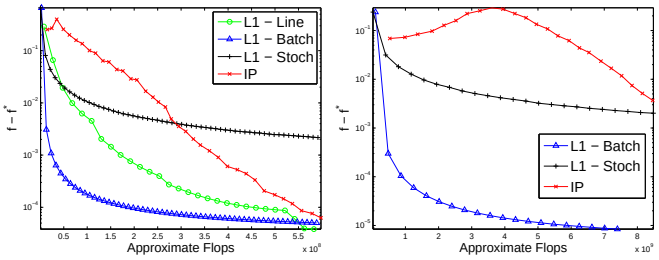

Figure 5. Comparison of methods on ℓ_1-regularized logistic regression. The left has dimension $n = 800$, the right $n = 4000$

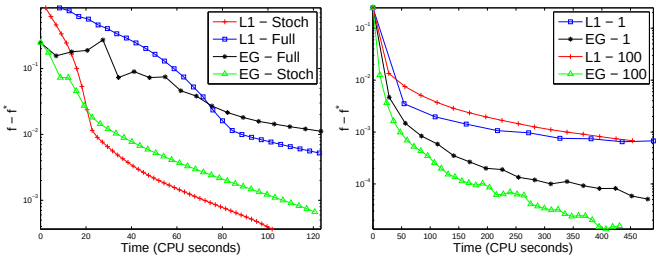

Figure 6. EG and projected subgradient methods on RCV1.

gression with dimension $n = 4000$. These results indicate that in high dimensional feature spaces, the asymptotically faster convergence of IP methods is counteracted by their quadratic dependence on the dimension of the space.

We also ran a series of experiments on two real datasets with high dimensionality: the Reuters RCV1 Corpus (Lewis et al., 2004) and the MNIST handwritten digits database. The Reuters Corpus has 804,414 examples; with simple stemming and stop-wording, there are 112,919 unigram features and 1,946,684 bigram features. With our preprocessing, the unigrams have a sparsity of 1.2% and the bigrams have sparsity of .26%. We performed ℓ_1-constrained binary logistic regression on the CCAT category from RCV1 (classifying a document as corporate/industrial) using unigrams in a batch setting and bigrams in an online setting. The MNIST dataset consists of 60,000 training examples and a 10,000 example test set and has 10-classes; each image is a gray-scale 28×28 image, which we represent as $\mathbf{x}_i \in \mathbb{R}^{784}$. Rather than directly use the input $\mathbf{x}_i$, however, we learned weights $\mathbf{w}_j$ using the following Kernel-based "similarity" function for each class $j \in \{1, \ldots, 10\}$:

$$k(\mathbf{x}, j) = \sum_{i \in S} w_{ji} \sigma_{ji} K(\mathbf{x}_i, \mathbf{x}), \; \sigma_{ji} = \begin{cases} 1 & \text{if } y_i = j \\ -1 & \text{otherwise.} \end{cases}$$

In the above, K is a Gaussian kernel function, so that $K(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2/25)$, and S is a 2766 element support set. We put an ℓ_1 constraint on each $\mathbf{w}_j$, giving us the following multiclass objective with dimension 27,660:

$$\begin{array}{rl} \text{minimize}_{\mathbf{w}} & \frac{1}{m} \sum_{i=1}^m \log \left(1 + \sum_{r \neq y_i} e^{k(\mathbf{x}_i, r) - k(\mathbf{x}_i, y_i)}\right) \\ \text{s.t.} & \|\mathbf{w}_j\|_1 \leq z, \mathbf{w}_j \succeq 0. \end{array} \tag{12}$$

As a comparison to our projected subgradient methods on real data, we used a method known in the literature as either entropic descent, a special case of mirror descent (Beck & Teboulle, 2003), or exponentiated gradient (EG) (Kivinen & Warmuth, 1997). EG maintains a weight vector $\mathbf{w}$ subject to the constraint that $\sum_i w_i = z$ and $\mathbf{w} \succeq 0$; it can easily be extended to work with negative weights under a 1-norm constraint by maintaining two vectors $\mathbf{w}^+$ and $\mathbf{w}^-$. We compare against EG since it works well in very high dimensional spaces, and it very quickly identifies and shrinks weights for irrelevant features (Kivinen & Warmuth, 1997). At every step of EG we update

$$w_i^{(t+1)} = \frac{w_i^{(t)} \exp\left(-\eta_t \nabla_i f(\mathbf{w}^{(t)})\right)}{Z_t} \tag{13}$$

where Z_t normalizes so that $\sum_i w_i^{(t+1)} = z$ and $\nabla_i f$ denotes the i^{th} entry of the gradient of f, the function to be minimized. EG can actually be viewed as a projected subgradient method using generalized relative entropy ($D(\mathbf{x}\|\mathbf{y}) = \sum_i x_i \log \frac{x_i}{y_i} - x_i + y_i$) as the distance function for projections (Beck & Teboulle, 2003). We can replace $\nabla_i f$ with $\hat{\nabla}_i f$ in Eq. (13), an unbiased estimator of the gradient of f, to get stochastic EG. A step size $\eta_t \propto 1/\sqrt{t}$ guarantees a convergence rate of $O(\sqrt{\log n/T})$. For each experiment with EG, however, we experimented with learning rates proportional to $1/t$, $1/\sqrt{t}$, and constant, as well as different initial step-sizes; to make EG as competitive as possible, we chose the step-size and rate for which EG performed best on each individual test..

Results for our batch experiments learning a logistic classifier for CCAT on the Reuters corpus can be seen in Fig. 6. The figure plots the binary logistic loss of the different algorithms minus the optimal log loss as a function of CPU time. On the left side Fig. 6, we used projected gradient descent and stochastic gradient descent using 25% of the training data to estimate the gradient, and we used the algorithm of Fig. 2 for the projection steps. We see that ℓ_1-projections outperform EG both in terms of convergence speed and empirical log-loss. On the right side of the figure, we performed stochastic descent using only 1 training example or 100 training examples to estimate the gradient, using Fig. 3 to project. When the gradient is sparse, updates for EG are $O(k)$ (where k is the number of non-zeros in the gradient), so EG has a run-time advantage over ℓ_1-projections when the gradient is very sparse. This advantage can be seen in the right side of Fig. 6.

For MNIST, with dense features, we ran a similar series of tests to those we ran on the Reuters Corpus. We plot the multiclass logistic loss from Eq. (12) over time (as a function of the number gradient evaluations) in Fig. 7. The left side of Fig. 7 compares EG and gradient descent using

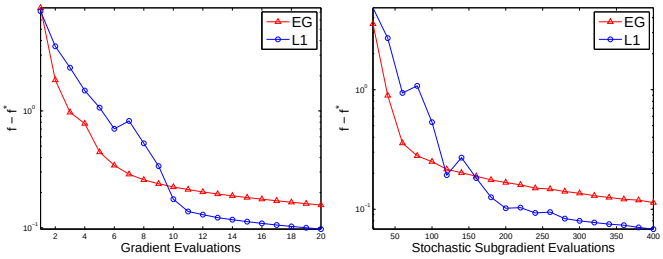

Figure 7. MNIST multiclass logistic loss as a function of the number of gradient steps. The left uses true gradients, the right stochastic subgradients.

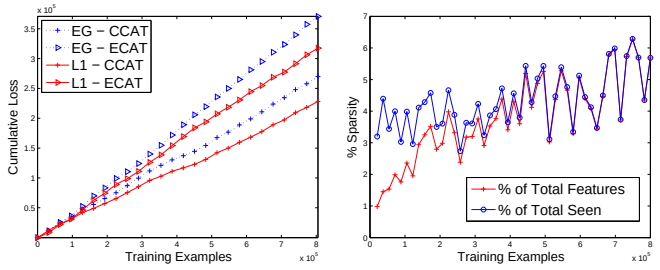

Figure 8. Online learning of bigram classifier on RCV1. Left is the cumulative loss, right shows sparsity over time.

the true gradient while the right figure compares stochastic EG and stochastic gradient descent using only 1% of the training set to estimate the gradient. On top of outperforming EG in terms of convergence rate and loss, the ℓ_1-projection methods also gave sparsity, zeroing out between 10 and 50% of the components of each class vector $\mathbf{w}_j$ in the MNIST experiments, while EG gives no sparsity.

As a last experiment, we ran an online learning test on the RCV1 dataset using bigram features, comparing ℓ_1-projections to using decreasing step sizes given by Zinkevich (2003) to exponential gradient updates. The ℓ_1-projections are computationally feasible because of algorithm 3, as the dimension of our feature space is nearly 2 million (using the expected linear-time algorithm of Fig. 2 takes 15 times as long to compute the projection for the sparse updates in online learning). We selected the bound on the 1-norm of the weights to give the best online regret of all our experiments (in our case, the bound was 100). The results of this experiment are in Fig. 8. The left figure plots the cumulative log-loss for the CCAT and ECAT binary prediction problems as a function of the number of training examples, while the right hand figure plots the sparsity of the ℓ_1-constrained weight vector both as a function of the dimension and as a function of the number of features actually seen. The ℓ_1-projecting learner maintained an active set with only about 5% non-zero components; the EG updates have no sparsity whatsoever. Our online ℓ_1-projections outperform EG updates in terms of the online regret (cumulative log-loss), and the ℓ_1-projection updates also achieve a classification error rate of 11.9% over all the examples on the CCAT task and 14.9% on ECAT (versus more than 15% and 20% respectively for EG).

Acknowledgments

We thank the anonymous reviewers for their helpful and insightful comments.

References

Beck, A., & Teboulle, M. (2003). Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, *31*, 167–175.

Bertsekas, D. (1999). *Nonlinear programming*. Athena Scientific.

Candes, E. J. (2006). Compressive sampling. *Proc. of the Int. Congress of Math., Madrid, Spain*.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms*. MIT Press.

Crammer, K., & Singer, Y. (2002). On the learnability and design of output codes for multiclass problems. *Machine Learning*, *47*.

Donoho, D. (2006a). Compressed sensing. *Technical Report, Stanford University*.

Donoho, D. (2006b). For most large underdetermined systems of linear equations, the minimal ℓ_1-norm solution is also the sparsest solution. *Comm. Pure Appl. Math. 59*.

Friedman, J., Hastie, T., & Tibshirani, R. (2007). Pathwise coordinate optimization. *Annals of Applied Statistics*, *1*, 302–332.

Gafni, E., & Bertsekas, D. P. (1984). Two-metric projection methods for constrained optimization. *SIAM Journal on Control and Optimization*, *22*, 936–964.

Hazan, E. (2006). Approximate convex optimization by online game playing. Unpublished manuscript.

Kim, S.-J., Koh, K., Lustig, M., Boyd, S., & Gorinevsky, D. (2007). An interior-point method for large-scale ℓ_1-regularized least squares. *IEEE Journal on Selected Topics in Signal Processing*, *4*, 606–617.

Kivinen, J., & Warmuth, M. (1997). Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, *132*, 1–64.

Koh, K., Kim, S.-J., & Boyd, S. (2007). An interior-point method for large-scale ℓ_1-regularized logistic regression. *Journal of Machine Learning Research*, *8*, 1519–1555.

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

Ng, A. (2004). Feature selection, l_1 vs. l_2 regularization, and rotational invariance. *Proceedings of the Twenty-First International Conference on Machine Learning*.

Shalev-Shwartz, S., & Singer, Y. (2006). Efficient learning of label ranking by soft projections onto polyhedra. *Journal of Machine Learning Research, 7 (July)*, 1567–1599.

Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for SVM. *Proceedings of the 24th International Conference on Machine Learning*.

Tarjan, R. E. (1983). *Data structures and network algorithms*. Society for Industrial and Applied Mathematics.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *J. Royal. Statist. Soc B.*, *58*, 267–288.

Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. *Proceedings of the Twentieth International Conference on Machine Learning*.