Pattern Anal Applic
DOI 10.1007/s10044-015-0485-z

Proximal gradient method for huberized support vector machine

Yangyang Xu · Ioannis Akrotirianakis · Amit Chakraborty

Received: date / Accepted: date

Abstract The Support Vector Machine (SVM) has been used in a wide variety of classification problems. The original SVM uses the hinge loss function, which is non-differentiable and makes the problem difficult to solve in particular for regularized SVMs, such as with ℓ_1-regularization. This paper considers the Huberized SVM (HSVM), which uses a differentiable approximation of the hinge loss function. We first explore the use of the Proximal Gradient (PG) method to solving binary-class HSVM (B-HSVM) and then generalize it to multi-class HSVM (M-HSVM). Under strong convexity assumptions, we show that our algorithm converges linearly. In addition, we give a finite convergence result about the support of the solution, based on which we further accelerate the algorithm by a two-stage method. We present extensive numerical experiments on both synthetic and real datasets which demonstrate the superiority of our methods over some state-of-the-art methods for both binary- and multi-class SVMs.

Keywords Support Vector Machine · Proximal Gradient method · Binary and multiclass classification problems · Huberized hinge loss · Elastic net · Linear convergence.

Yangyang Xu
Department of Computational and Applied Mathematics, Rice University, Houston, TX.
E-mail: yangyang.xu@rice.edu

Ioannis Akrotirianakis
Siemens Corporate Research, Princeton, NJ.
E-mail: ioannis.akrotirianakis@siemens.com

Amit Chakraborty
Siemens Corporate Research, Princeton, NJ.
E-mail: amit.chakraborty@siemens.com

1 Introduction

The original linear support vector machine (SVM) aims to find a hyperplane that separates a collection of data points. Since it was proposed in [7], it has been widely used for binary classifications, such as texture classification [24], gene expression data analysis [5], face recognition [18], to name a few. Mathematically, given a set of samples $\{\mathbf{x}_i\}_{i=1}^n$ in p-dimensional space and each $\mathbf{x}_i$ attached with a label $y_i \in \{+1, -1\}$, the linear SVM learns a hyperplane $(\mathbf{w}^*)^\top \mathbf{x} + b^* = 0$ from the training samples. A new data point $\mathbf{x}$ can be categorized into the "+1" or "−1" class by inspecting the sign of $(\mathbf{w}^*)^\top \mathbf{x} + b^*$.

The binary-class SVM (B-SVM) has been generalized to multicategory classifications to tackle problems that have data points belonging to more than two classes. The initially proposed multi-class SVM (M-SVM) methods construct several binary classifiers, such as "one-against-all' [3], "one-against-one" [25], and "directed acyclic graph SVM" [36]. These M-SVMs may suffer from data imbalance, namely, some classes have much fewer data points than others, which can result in inaccurate predictions. One alternative is to put all the data points together in one model, which results in the so-called "all-together" M-SVMs. The "all-together" M-SVMs train multi-classifiers by considering a single large optimization problem. An extensive comparison of different M-SVMs can be found in [20].

In this paper, we consider both B-SVM and M-SVM. More precisely, we consider the binary-class huberized SVM (B-HSVM) (see (2) below) for B-SVM and the "all-together" multi-class HSVM (M-HSVM) (see (3) below) for M-SVM. The advantage of HSVM over classic SVM with hinge loss is the continuous differentiability of its loss function, which enables the use

of the "fastest" first-order method: Proximal Gradient (PG) method [34,1] (see the overview in section 2.1). We demonstrate that the PG method is in general much faster than existing methods for (regularized) B-SVM and M-SVM while yielding comparable prediction accuracies. In addition, extensive numerical experiments are done to compare our method to state-of-the-art ones for both B-SVM and M-SVM on synthetic and also benchmark datasets. Statistical comparison is also performed to show the difference between the proposed method and other compared ones.

1.1 Related work

B-HSVM appears to be first considered[1] by Wang *et al* in [44]. They demonstrate that B-HSVM can perform better than the original unregularized SVM and also ℓ_1-regularized SVM (i.e., (1) with $\lambda_2 = 0$) for microarray classification.

A closely related model to B-HSVM is the elastic net regularized SVM [43]

$$\min_{b,\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}\left[1-y_i(b+\mathbf{x}_i^\top\mathbf{w})\right]_+ + \lambda_1\|\mathbf{w}\|_1 + \frac{\lambda_2}{2}\|\mathbf{w}\|_2^2, \quad (1)$$

where $[1-t]_+ = \max(0, 1-t)$ is the hinge loss function. The ℓ_1-norm regularizer has the ability to perform continuous shrinkage and automatic variable selection at the same time [40], and the ℓ_2-norm regularizer can help to reduce the variance of the estimated coefficients and usually results in satisfactory prediction accuracy, especially in the case where there are many correlated variables. The elastic net inherits the benefits of both ℓ_1 and ℓ_2-norm regularizers and can perform better than either of them alone, as demonstrated in [52].

Note that (1) uses non-differentiable hinge loss function while B-HSVM uses differentiable huberized hinge loss function. The differentiability makes B-HSVM relatively easier to solve. A path-following algorithm was proposed by Wang *et. al* [44] for solving B-HSVM. Their algorithm is not efficient in particular for large-scale problems, since it needs to track the disappearance of variables along a regularization path. Recently, Yang and Zou [48] proposed a Generalized Coordinate Descent (GCD) method, which was, in most cases, about 30 times faster than the path-following algorithm. However, the GCD method needs to compute the gradient of the loss function of B-HSVM after each coordinate update, which makes the algorithm slow.

[1] Strictly speaking, we add the term $\frac{\lambda_3}{2}b^2$ in (2). This modification is similar to that used in [22], and the extra term usually does not affect the prediction but makes PG method converge faster.

M-HSVM has been considered in [30], which generalizes the work [44] on B-HSVM to M-HSVM and also makes a path-following algorithm. However, their algorithm could be even worse since it also needs to track the disappearance of variables along a regularization path and M-HSVM often involves more variables than those of B-HSVM. Hence, it is not suitable for large-scale problems either.

Similar to M-HSVM, several other models have been proposed to train multiple classifiers by solving one single large optimization problem to handle the multi-category classification, such as the ℓ_1-norm regularized M-SVM in [42] and the ℓ_∞-norm regularized M-SVM in [50]. Again, these models use the non-differentiable hinge loss function and are relatively more difficult than M-HSVM to solve. There are also methods that use binary-classifiers to handle multicategory classification problems including "one-against-all" [3], "one-against-one" [25], and "directed acyclic graph SVM" [36]. The work [15] makes a thorough review of methods using binary-classifiers for multi-category classification problems and gives extensive experiments on various applications. In a follow up and more recent paper [14] the same authors present a dynamic classifier weighting method which deals with the limitations introduced by the non-competent classifiers in the one-versus-one classification strategy. In order to dynamically weigh the outputs of the individual classifiers, they use the nearest neighbor of every class from the instance that needs to be classified. Furthermore in [26] the authors propose a new approach for building multi-class classifiers based on the notion of data-clustering in the feature space. For the derived clusters they construct one-class classifiers which can be combined to solve complex classification problems.

One key advantage of our algorithm is its scalability with the training sample size, and thus it is applicable for large-scale SVMs. While preparing this paper, we note that some other algorithms are also carefully designed for handling the large-scale SVMs, including the block coordinate Frank-Wolfe method [28] and the stochastic alternating direction method of multiplier [35]. In addition, Graphics Processing Unit (GPU) computing has been utilized in [31] to run multiple training tasks in parallel to accelerate the cross validation procedure. Furthermore, different variants of SVMs have been proposed for specific applications such as the Value-at-Risk SVM for stability to outliers [41], the structural twin SVM to contain prior domain knowledge [37], the hierarchical SVM for customer churn prediction [6] and the ellipsoidal SVM for outlier detection in wireless sensor networks [51]. Finally in [8] a two-ellipsoid kernel decomposition is proposed for the efficient training of

SVMs. In order to avoid the use of SOCP techniques, introduced by the ellipsoids, the authors transform the data using a matrix which is determined from the sum of the classes' covariances. With that transformation it is possible to use classical SVM, and their method can be incorporated into existing SVM libraries.

1.2 Contributions

We develop an efficient PG method to solve the B-HSVM

$$\min_{b,\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} \phi_H\left(y_i(b+\mathbf{x}_i^\top \mathbf{w})\right)+\lambda_1\|\mathbf{w}\|_1+\frac{\lambda_2}{2}\|\mathbf{w}\|_2^2+\frac{\lambda_3}{2}b^2, \tag{2}$$

and the "all-together" M-HSVM

$$\begin{aligned}\min_{\mathbf{b},\mathbf{W}}\ & \tfrac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{J} a_{ij}\phi_H(b_j+\mathbf{x}_i^\top \mathbf{w}_j)+\lambda_1\|\mathbf{W}\|_1 \\ & +\tfrac{\lambda_2}{2}\|\mathbf{W}\|_F^2+\tfrac{\lambda_3}{2}\|\mathbf{b}\|^2, \\ \text{s.t. } & \mathbf{We}=\mathbf{0}, \mathbf{e}^\top\mathbf{b}=0.\end{aligned} \tag{3}$$

In (2), $y_i \in \{+1,-1\}$ is the label of $\mathbf{x}_i$, and

$$\phi_H(t)=\begin{cases}0, & \text{for } t>1,\\ \frac{(1-t)^2}{2\delta}, & \text{for } 1-\delta<t\le 1,\\ 1-t-\frac{\delta}{2}, & \text{for } t\le 1-\delta,\end{cases}$$

is the huberized hinge loss function which is continuously differentiable. In (3), $y_i \in \{1,\dots,J\}$ is the i-th label, $\|\mathbf{W}\|_1=\sum_{i,j}|w_{ij}|$, $a_{ij}=1$ if $y_i\neq j$ and $a_{ij}=0$ otherwise, $\mathbf{e}$ denotes the vector with all one's, and $\mathbf{w}_j$ denotes the j-th column of $\mathbf{W}$. The constraints $\mathbf{We}=\mathbf{0}$ and $\mathbf{e}^\top\mathbf{b}=0$ in (3) are imposed to eliminate redundancy in $\mathbf{W},\mathbf{b}$ and also are necessary to make the loss function

$$\ell_M=\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{J} a_{ij}\phi_H(b_j+\mathbf{x}_i^\top\mathbf{w}_j)$$

Fisher-consistent [29].

We choose the PG methodology because it requires only first-order information and converges fast. As shown in [34,1], it is an optimal first-order method. Note that the objectives in (2) and (3) have non-smooth terms and are not differentiable. Hence, direct gradient or second-order methods such as the interior point method are not applicable.

We speed up the algorithm by using a two-stage method, which detects the support of the solution at the first stage and solves a lower-dimensional problem at the second stage. For large-scale problems with sparse features, the two-stage method can achieve more than 5-fold acceleration. We also analyze the convergence of PG method under fairly general settings and get similar results as those in [34,38].

In addition, we compare the proposed method to state-of-the-art ones for B-SVM and M-SVM on both synthetic and benchmark datasets. Extensive numerical experiments demonstrate that our method can outperform other compared ones in most cases. Statistical tests are also performed to show significant differences between the compared methods.

1.3 Structure of the paper

The rest of the paper is organized as follows. In section 2, we overview the PG method and then apply it to (2) and (3). In addition, assuming strong convexity, we show linear convergence of the PG method under fairly general settings. Numerical results are given in section 3. Finally, section 4 concludes the paper.

2 Algorithms and convergence analysis

In this section, we first give an overview of the PG method. Then we apply it to (2) and (3). We complete this section by showing that under strong convexity assumptions the PG method possesses linear convergence.

2.1 Overview of the PG method

Consider convex composite optimization problems in the form of

$$\min_{\mathbf{u}\in\mathcal{U}} \xi(\mathbf{u})\equiv\xi_1(\mathbf{u})+\xi_2(\mathbf{u}), \tag{4}$$

where $\mathcal{U}\subset\mathbb{R}^m$ is a convex set, ξ_1 is a differentiable convex function with Lipschitz continuous gradient (that is, there exists $L>0$ such that $\|\nabla\xi_1(\bar{\mathbf{u}})-\nabla\xi_1(\tilde{\mathbf{u}})\|\le L\|\bar{\mathbf{u}}-\tilde{\mathbf{u}}\|$, for all $\bar{\mathbf{u}},\tilde{\mathbf{u}}\in\mathcal{U}$), and ξ_2 is a proper closed convex function. The PG method for solving (4) iteratively updates the solution by

$$\mathbf{u}^k=\arg\min_{\mathbf{u}\in\mathcal{U}} Q(\mathbf{u},\hat{\mathbf{u}}^{k-1}) \tag{5}$$

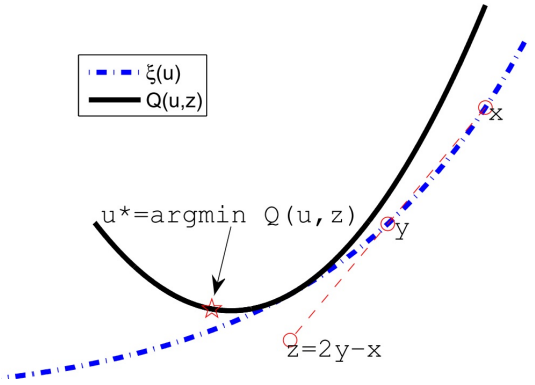
where

$$\begin{aligned}Q(\mathbf{u},\hat{\mathbf{u}}^{k-1})=\ & \xi_1(\hat{\mathbf{u}}^{k-1})+\langle\nabla\xi_1(\hat{\mathbf{u}}^{k-1}),\mathbf{u}-\hat{\mathbf{u}}^{k-1}\rangle \\ & +\tfrac{L_k}{2}\|\mathbf{u}-\hat{\mathbf{u}}^{k-1}\|^2+\xi_2(\mathbf{u}),\end{aligned}$$

$L_k>0$ and $\hat{\mathbf{u}}^{k-1}=\mathbf{u}^{k-1}+\omega_{k-1}(\mathbf{u}^{k-1}-\mathbf{u}^{k-2})$ for some nonnegative $\omega_{k-1}\le 1$. The extrapolation technique can significantly accelerate the algorithm.

When L_k is the Lipschitz constant of $\nabla\xi_1$, it can easily be shown that $\xi(\mathbf{u})\le Q(\mathbf{u},\hat{\mathbf{u}}^{k-1})$, and thus this

Fig. 1 Simple illustration of PG method: $Q(u,z)$ is a majorization approximation of ξ at z, which is an extrapolated point of x and y. The new iterate u^* is the minimizer of Q.



method is a kind of majorization minimization, as illustrated in Figure 1. With appropriate choice of ω_{k-1} and L_k, the sequence $\{\xi(\mathbf{u}^k)\}$ converges to the optimal value $\xi^* = \xi(\mathbf{u}^*)$. Nesterov [34], and Beck and Teboulle [1] showed, independently, that if $\omega_{k-1} \equiv 0$ and L_k is taken as the Lipschitz constant of $\nabla\xi_1$, then $\{\xi(\mathbf{u}^k)\}$ converges to ξ^* with the rate $O(1/k)$, namely, $\xi(\mathbf{u}^k) - \xi(\mathbf{u}^*) \le O(1/k)$. In addition, using carefully designed ω_{k-1}, they were able to show that the convergence rate can be increased to $O(1/k^2)$, which is the optimal rate of first-order methods for general convex problems [33].

2.2 PG method for binary-class HSVM

We first write the B-HSVM problem (2) into the general form of (4). Let

$$\begin{cases} f_i(b,\mathbf{w}) = \phi_H\left(y_i(b+\mathbf{x}_i^\top\mathbf{w})\right), \text{ for } i=1,\cdots,n,\\ f(b,\mathbf{w}) = \frac{1}{n}\sum_{i=1}^n f_i(b,\mathbf{w}),\\ g(b,\mathbf{w}) = \lambda_1\|\mathbf{w}\|_1 + \frac{\lambda_2}{2}\|\mathbf{w}\|^2 + \frac{\lambda_3}{2}b^2. \end{cases}$$

Then (2) can be written as

$$\min_{b,\mathbf{w}} F(b,\mathbf{w}) \equiv f(b,\mathbf{w}) + g(b,\mathbf{w}). \tag{6}$$

For convenience, we use the following notation in the rest of this section

$$\mathbf{u} = (b;\mathbf{w}), \qquad \mathbf{v}_i = (y_i; y_i\mathbf{x}_i). \tag{7}$$

Proposition 1 *The function f defined above is differentiable and convex, and its gradient ∇f is Lipschitz continuous with constant*

$$L_f = \frac{1}{n\delta}\sum_{i=1}^n y_i^2(1+\|\mathbf{x}_i\|^2). \tag{8}$$

The proof of this proposition involves standard arguments and can be found in the appendix.

Now, we are ready to apply PG to (2). Define the *proximal operator* for a given extended real-valued convex function $h(\mathbf{u})$ on $\mathbb{R}^m$ by

$$\text{prox}_h(\mathbf{v}) = \arg\min_{\mathbf{u}} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|^2 + h(\mathbf{u}).$$

Replacing ξ_1 and ξ_2 in (5) with f and g, respectively, we obtain the update

$$\begin{cases} b^k = \frac{L_k\hat{b}^{k-1} - \nabla_b f(\hat{\mathbf{u}}^{k-1})}{L_k+\lambda_3},\\ \mathbf{w}^k = \frac{1}{L_k+\lambda_2}\mathcal{S}_{\lambda_1}\left(L_k\hat{\mathbf{w}}^{k-1} - \nabla_{\mathbf{w}} f(\hat{\mathbf{u}}^{k-1})\right), \end{cases} \tag{9}$$

where $\mathcal{S}_\nu(\cdot)$ is a component-wise shrinkage operator defined by $\mathcal{S}_\nu(t) = \text{sign}(t)\max(|t|-\nu, 0)$.

In (9), we dynamically update L_k by

$$L_k = \min(\eta^{n_k}L_{k-1}, L_f), \tag{10}$$

where $\eta > 1$ is a pre-selected constant, L_f is defined in (8) and n_k is the smallest integer such that the following condition is satisfied

$$\begin{aligned} &f(\text{prox}_{h_k}(\mathbf{v}^{k-1}))\\ &\le f(\hat{\mathbf{u}}^{k-1}) + \left\langle \nabla f(\hat{\mathbf{u}}^{k-1}), \text{prox}_{h_k}(\mathbf{v}^{k-1}) - \hat{\mathbf{u}}^{k-1}\right\rangle\\ &\quad + \frac{L_k}{2}\left\|\text{prox}_{h_k}(\mathbf{v}^{k-1}) - \hat{\mathbf{u}}^{k-1}\right\|^2, \end{aligned} \tag{11}$$

where $h_k(\mathbf{u}) = \frac{1}{L_k}g(\mathbf{u})$, $\mathbf{v}^{k-1} = \hat{\mathbf{u}}^{k-1} - \frac{1}{L_k}\nabla f(\hat{\mathbf{u}}^{k-1})$ and $\hat{\mathbf{u}}^{k-1} = \mathbf{u}^{k-1} + \omega_{k-1}(\mathbf{u}^{k-1} - \mathbf{u}^{k-2})$ for some weight $\omega_{k-1} \le 1$. The inequality in (11) is necessary to make the algorithm convergent (see Lemma 2 in the appendix). It guarantees sufficient decrease, and the setting $L_k = L_f$ will make it satisfied. In the case where L_f becomes unnecessarily large, a smaller L_k can speed up the convergence. To make the overall objective non-increasing, we re-update $\mathbf{u}^k$ by setting $\hat{\mathbf{u}}^{k-1} = \mathbf{u}^{k-1}$ in (9) whenever $F(\mathbf{u}^k) > F(\mathbf{u}^{k-1})$. *This non-increasing monotonicity is very important, since we observe that the PG method may be numerically unstable without this modification. In addition, it significantly accelerates the PG method; see Table 1 below.*

Algorithm 1 summarizes our discussion. We name it as B-PGH.

2.3 PG method for multi-class HSVM

In this section we generalize the PG method for solving multi-class classification problems. Denote $\mathbf{U} = (\mathbf{b};\mathbf{W})$. Let

$$G(\mathbf{b},\mathbf{W}) = \lambda_1\|\mathbf{W}\|_1 + \frac{\lambda_2}{2}\|\mathbf{W}\|_F^2 + \frac{\lambda_3}{2}\|\mathbf{b}\|^2,$$
$$\mathcal{U} = \{(\mathbf{b},\mathbf{W}) : \mathbf{W}\mathbf{e} = \mathbf{0}, \mathbf{e}^\top\mathbf{b} = 0\}.$$

Algorithm 1 Proximal gradient method for B-HSVM (B-PGH)

Input: sample-label pairs $(\mathbf{x}_i, y_i), i = 1, \cdots, n$; parameters $\lambda_1, \lambda_2, \lambda_3, \delta$.
Initialization: choose $\mathbf{u}^0 = (b^0; \mathbf{w}^0)$, $\mathbf{u}^{-1} = \mathbf{u}^0$; compute L_f from (8) and choose $\eta > 1$ and $L_0 \le L_f$; set $k = 1$.
while *Not converged* **do**
  Let $\hat{\mathbf{u}}^{k-1} = \mathbf{u}^{k-1} + \omega_{k-1}(\mathbf{u}^{k-1} - \mathbf{u}^{k-2})$ for some $\omega_{k-1} \le 1$;
  Update L_k according to (10) and $\mathbf{u}^k$ according to (9);
  if $F(\mathbf{u}^k) > F(\mathbf{u}^{k-1})$ **then**
    Re-update $\mathbf{u}^k$ according to (9) with $\hat{\mathbf{u}}^{k-1} = \mathbf{u}^{k-1}$;
  end if
  Let $k = k + 1$.
end while

Then we can write (3) as

$$\min_{\mathbf{U}\in\mathcal{U}} H(\mathbf{U}) \equiv \ell_M(\mathbf{U}) + G(\mathbf{U}) \tag{12}$$

Similar to Proposition 1, we can show that $\nabla_{\mathbf{U}}\ell_M$ is Lipschitz continuous with constant

$$L_m = \frac{J}{n\delta}\sum_{i=1}^{n}(1 + \|\mathbf{x}_i\|^2). \tag{13}$$

The proof is essentially the same as that of Proposition 1, and we do not repeat it.

Now we are ready to apply the PG method to (3) or equivalently (12). We update $\mathbf{U}$ iteratively by solving the $\mathbf{b}$-subproblem

$$\begin{aligned}\mathbf{b}^k = \underset{\mathbf{e}^\top\mathbf{b}=0}{\arg\min}\langle\nabla_{\mathbf{b}}\ell_M(\hat{\mathbf{U}}^{k-1}), \mathbf{b} - \hat{\mathbf{b}}^{k-1}\rangle \\ + \tfrac{L_k}{2}\|\mathbf{b} - \hat{\mathbf{b}}^{k-1}\|^2 + \tfrac{\lambda_3}{2}\|\mathbf{b}\|^2\end{aligned} \tag{14}$$

and $\mathbf{W}$-subproblem

$$\begin{aligned}\mathbf{W}^k = \underset{\mathbf{We}=\mathbf{0}}{\arg\min}\langle\nabla_{\mathbf{W}}\ell_M(\hat{\mathbf{U}}^{k-1}), \mathbf{W} - \hat{\mathbf{W}}^{k-1}\rangle \\ + \tfrac{L_k}{2}\|\mathbf{W} - \hat{\mathbf{W}}^{k-1}\|^2 + \lambda_1\|\mathbf{W}\|_1 + \tfrac{\lambda_2}{2}\|\mathbf{W}\|^2,\end{aligned} \tag{15}$$

where L_k is chosen in the same way as (11).

Problem (14) is relatively easy and has a closed form solution. Let $\mathbf{P} = \begin{bmatrix}\mathbf{I}\\ -\mathbf{e}^\top\end{bmatrix} \in \mathbb{R}^{J\times(J-1)}$ and $\bar{\mathbf{b}} \in \mathbb{R}^{J-1}$ be the vector consisting of the first $J-1$ components of $\mathbf{b}$, where $\mathbf{I}$ denotes the identity matrix. Substituting $\mathbf{b} = \mathbf{P}\bar{\mathbf{b}}$ to (14) gives the solution

$$\bar{\mathbf{b}}^k = \frac{1}{\lambda_3 + L_k}(\mathbf{P}^\top\mathbf{P})^{-1}\mathbf{P}^\top\big(L_k\hat{\mathbf{b}}^{k-1} - \nabla_{\mathbf{b}}\ell_M(\hat{\mathbf{U}}^{k-1})\big).$$

Hence, the update in (14) can be explicitly written as

$$\mathbf{b}^k = \mathbf{P}\bar{\mathbf{b}}^k, \tag{16}$$

where $\bar{\mathbf{b}}^k$ is defined in the above equation.

Problem (15) can be further decomposed into p independent small problems. Each of them involves only one row of $\mathbf{W}$ and can be written in the following form

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w} - \mathbf{z}\|^2 + \lambda\|\mathbf{w}\|_1, \text{ s.t. } \mathbf{e}^\top\mathbf{w} = 0, \tag{17}$$

where $\lambda = \frac{\lambda_1}{L_k+\lambda_2}$ and $\mathbf{z}^\top$ is the i-th row-vector of the matrix $\frac{L_k}{L_k+\lambda_2}\hat{\mathbf{W}}^{k-1} - \frac{1}{L_k+\lambda_2}\nabla_{\mathbf{W}}\ell_M(\hat{\mathbf{U}}^{k-1})$ for the i-th small problem. The coexistence of the non-smooth term $\|\mathbf{w}\|_1$ and the equality constraint $\mathbf{e}^\top\mathbf{w} = 0$ makes (17) a difficult optimization problem to solve. Next we describe a new efficient yet simple method for solving (17) using its dual problem, defined by

$$\begin{aligned}\max_{\sigma}\ \gamma(\sigma) \equiv \tfrac{1}{2}\|\mathcal{S}_\lambda(\mathbf{z} - \sigma) - \mathbf{z}\|^2 + \lambda\|\mathcal{S}_\lambda(\mathbf{z} - \sigma)\|_1 \\ + \sigma\mathbf{e}^\top\mathcal{S}_\lambda(\mathbf{z} - \sigma).\end{aligned} \tag{18}$$

Since (17) is strongly convex, $\gamma(\sigma)$ is concave and continuously differentiable. Hence, the solution σ^* of (18) can be found by solving the single-variable equation $\gamma'(\sigma) = 0$. It is easy to verify that $\gamma'(z_{\min} - \lambda) > 0$ and $\gamma'(z_{\max} + \lambda) < 0$, so the solution σ^* lies between $z_{\min} - \lambda$ and $z_{\max} + \lambda$, where $z_{\min}$ and $z_{\max}$ respectively denote the minimum and maximum components of $\mathbf{z}$. In addition, note that $\mathcal{S}_\lambda(\mathbf{z} - \sigma)$ is piece-wise linear about σ, and the breakpoints are at $z_i \pm \lambda$, $i = 1, \cdots, J$. Hence σ^* must be in $[v_l, v_{l+1}]$ for some l, where $\mathbf{v}$ is the sorted vector of $(\mathbf{z} - \lambda; \mathbf{z} + \lambda)$ in the increasing order. Therefore, to solve (18), we first obtain $\mathbf{v}$, then search the interval that contains σ^* by checking the sign of $\gamma'(\sigma)$ at the breakpoints, and finally solve $\gamma'(\sigma) = 0$ within that interval. Algorithm 2 summarizes our method for solving (18).

Algorithm 2 Exact method for solving (18)

Input: $(\mathbf{z}, \lambda)$ with $\mathbf{z}$ in J-dimensional space and $\lambda > 0$.
Let $\mathbf{v} = [\mathbf{z} - \lambda; \mathbf{z} + \lambda] \in \mathbb{R}^{2J}$ and sort $\mathbf{v}$ as $v_1 \le v_2 \le \cdots \le v_{2J}$; set $l = J$.
while $\gamma'(v_l) \cdot \gamma'(v_{l+1}) > 0$ **do**
  If $\gamma'(v_l) > 0$, set $l = l + 1$; else $l = l - 1$.
end while
Solve $\gamma'(\sigma) = 0$ within $[v_l, v_{l+1}]$ and output the solution σ^*.

After determining σ^*, we can obtain the solution of (17) by setting $\mathbf{w}^* = \mathcal{S}_\lambda(\mathbf{z} - \sigma^*)$. Algorithm 3 summarizes the main steps of the PG method for efficiently solving (3). We name it as M-PGH.

2.4 Convergence results

Instead of analyzing the convergence of Algorithms 1 and 3, we do the analysis of the PG method for (4)

Algorithm 3 Proximal gradient method for M-HSVM (M-PGH)

1: **Input:** sample-label pairs $(\mathbf{x}_i, y_i), i = 1, \cdots, n$ with each $y_i \in \{1, \cdots, J\}$; parameters $\lambda_1, \lambda_2, \lambda_3, \delta$.
2: **Initialization:** choose $\mathbf{U}^0$, $\mathbf{U}^{-1} = \mathbf{U}^0$; compute L_m in (13) and choose $\eta > 1$ and $L_0 \le L_m$; set $k = 1$.
3: **while** Not converged **do**
4: Let $\hat{\mathbf{U}}^{k-1} = \mathbf{U}^{k-1} + \omega_{k-1}(\mathbf{U}^{k-1} - \mathbf{u}^{k-2})$ for some $\omega_{k-1} \le 1$;
5: Choose L_k in the same way as (10);
6: Update $\mathbf{b}^k$ by (16);
7: **for** $i = 1, \cdots, p$ **do**
8: Set $\mathbf{z}$ to the i-th row of
9: $\frac{L_k}{L_k+\lambda_2}\hat{\mathbf{W}}^{k-1} - \frac{1}{L_k+\lambda_2}\nabla_{\mathbf{W}}\ell_M(\hat{\mathbf{U}}^{k-1})$;
10: Solve (18) by Algorithm 2 with input $(\mathbf{z}, \frac{\lambda_1}{L_k+\lambda_2})$ and let σ^* be the output;
11: Set the i-th row of $\mathbf{W}^k$ to be $\mathcal{S}_\lambda(\mathbf{z} - \sigma^*)$;
12: **end for**
13: **if** $H(\mathbf{U}^k) > H(\mathbf{U}^{k-1})$ **then**
14: Re-update $\mathbf{b}^k$ and $\mathbf{W}^k$ according to (16) and (15) with $\hat{\mathbf{U}}^{k-1} = \mathbf{U}^{k-1}$;
15: **end if**
16: Let $k = k + 1$.
17: **end while**

with general ξ_1 and ξ_2 and regard Algorithms 1 and 3 as special cases. Throughout our analysis, we assume that ξ_1 is a differentiable convex function and $\nabla\xi_1$ is Lipschitz continuous with Lipschitz constant L. We also assume that ξ_2 is strongly convex[2] with constant $\mu > 0$, namely,

$$\xi_2(\mathbf{u}) - \xi_2(\mathbf{v}) \ge \langle \mathbf{g}_\mathbf{v}, \mathbf{u} - \mathbf{v}\rangle + \frac{\mu}{2}\|\mathbf{u} - \mathbf{v}\|^2,$$

for any $\mathbf{g}_\mathbf{v} \in \partial\xi_2(\mathbf{v})$ and $\mathbf{u}, \mathbf{v} \in \mathrm{dom}(\xi_2)$, where $\mathrm{dom}(\xi_2) = \{\mathbf{u} \in \mathbb{R}^m : \xi_2(\mathbf{u}) < \infty\}$ denotes the domain of ξ_2.

Similar results have been shown by Nesterov [34] and Schmidt *et al* [38]. However, our analysis fits to more general settings. Specifically, we allow dynamical update of the Lipschitz parameter L_k and an acceptable interval of the parameters ω_k. On the other hand, [34,38] either require L_k to be fixed to the Lipschitz constant of ξ_1 or require specific values for the ω_k's. In addition, neither of [34,38] do the re-update step as in line 7 of Algorithm 1 or line 13 of Algorithm 3.

We tested the PG method under different settings on synthetic datasets, generated in the same way as described in section 3.1.1. Our goal is to demonstrate the practical effect that each setting has on the overall performance of the PG method. Table 1 summarizes the numerical results, which show that PG method under our settings converges significantly faster than that under the settings of [34,38].

[2] Without strong convexity of ξ_2, we only have sublinear convergence as shown in [1].

To analyze the PG method under fairly general settings, we use the so-called Kurdyka-Łojasiewicz (KL) inequality, which has been widely applied in non-convex optimization. Our results show that the KL inequality can also be applied and simplify the analysis in convex optimization. Extending the discussion of the KL inequality is beyond the scope of this paper and therefore we refer the interested readers to [46,32,27,2,47] and the references therein.

Our main result is summarized as follows.

Theorem 1 *Let $\{\mathbf{u}^k\}$ be the sequence generated by (5) with $L_k \le L$ and $\hat{\mathbf{u}}^{k-1} = \mathbf{u}^{k-1} + \omega_{k-1}(\mathbf{u}^{k-1} - \mathbf{u}^{k-2})$ for some $\omega_{k-1} \le \sqrt{\frac{L_{k-1}}{L_k}}$ such that (24) holds. In addition, we make $\{\xi(\mathbf{u}^k)\}$ nonincreasing by re-updating $\mathbf{u}^k$ with $\hat{\mathbf{u}}^{k-1} = \mathbf{u}^{k-1}$ in (5) whenever $\xi(\mathbf{u}^k) > \xi(\mathbf{u}^{k-1})$. Then $\mathbf{u}^k$ R-linearly converges to the unique minimizer $\mathbf{u}^*$ of (4), namely, there exist positive constants C and $\tau < 1$ such that*

$$\|\mathbf{u}^k - \mathbf{u}^*\| \le C\tau^k, \ \forall k \ge 0. \tag{19}$$

The proof of this theorem is given in the Appendix due to its technical complexity. Using the results of Theorem 1, we establish the convergence results of Algorithms 1 and 3 in the following corollary.

Corollary 1 *Let $\{\mathbf{u}^k\}$ and $\{\mathbf{U}^k\}$ be the sequences generated by Algorithms 1 and 3 with $\lambda_2, \lambda_3 > 0$ and $\omega_{k-1} \le \sqrt{\frac{L_{k-1}}{L_k}}$. Then $\{\mathbf{u}^k\}$ and $\{\mathbf{U}^k\}$ R-linearly converge to the unique solutions of (2) and (3), respectively.*

Remark 1 If one of λ_2 and λ_3 vanishes, we only have sublinear convergence by some appropriate choice of ω_k. The results can be found in [1].

2.5 Two-stage accelerated method for large-scale problems

Most cost of Algorithm 1 at iteration k occurs in the computation of $\nabla f(\hat{\mathbf{u}}^{k-1})$ and the evaluation of $F(\mathbf{u}^k)$. Let $\mathbf{V} = (\mathbf{v}_1, \cdots, \mathbf{v}_n)^\top$ where $\mathbf{v}_1, \cdots, \mathbf{v}_n$ are defined in (7). To compute $\nabla f(\hat{\mathbf{u}}^{k-1})$, we need to have $\mathbf{V}\hat{\mathbf{u}}^{k-1}$ and $\nabla f_i(\hat{\mathbf{u}}^{k-1}), i = 1, \cdots, n$, which costs $O(np)$ in total. Evaluating $F(\mathbf{u}^k)$ needs $\mathbf{V}\mathbf{u}^k$ which costs $O(np)$. To save the computing time, we store both $\mathbf{V}\mathbf{u}^{k-1}$ and $\mathbf{V}\mathbf{u}^k$ so that $\mathbf{V}\hat{\mathbf{u}}^k$ can be obtained by $\mathbf{V}\hat{\mathbf{u}}^k = \mathbf{V}\mathbf{u}^k + \omega_k(\mathbf{V}\mathbf{u}^k - \mathbf{V}\mathbf{u}^{k-1})$. This way, we only need to compute $\mathbf{V}\mathbf{u}^k$ once during each iteration, and the total computational cost is $O(Tnp)$ where T is the total number of iterations.

As p is large and the solution of (2) is sparse, namely, only a few features are relevant, we can further reduce the cost of Algorithm 1 by switching from the original

Table 1 Performance of the PG method with different settings: [38] sets $L_k = L_f$ for all k; neither of [34,38] makes $F(\mathbf{x}_k)$ non-increasing. The data used in these tests is generated in the same way as described in section 3.1.1, and here we set the correlation parameter $\rho = 0$.

Problems	Our settings			Settings of [34]			Settings of [38]		
(n, p, s)	#iter	time	obj.	#iter	time	obj.	#iter	time	obj.
(3000, 300, 30)	34	0.06	1.0178e-1	135	0.22	1.0178e-1	475	0.75	1.0178e-1
(2000, 20000, 200)	91	4.34	8.0511e-2	461	21.77	8.0511e-2	2000	99.05	8.0511e-2

high-dimensional problem to a lower-dimensional one. More precisely, we first run Algorithm 1 with $L_k = L_f$ and $\omega_k = 0$ until the support of $\mathbf{w}^k$ remains almost unchanged. Then we reduce (2) to a lower-dimensional problem by confining $\mathbf{w}$ to the detected support, namely, all elements out of the detected support are kept *zero*. Finally, Algorithm 1 is employed again to solve the lower-dimensional problem. The two-stage method for (2) is named as B-PGH-2, and its solidness rests on the following lemma, which can be shown in a similar way as the proof of Lemma 5.2 in [17].

Lemma 1 *Let $\{(b^k, \mathbf{w}^k)\}$ be the sequence generated by Algorithm 1 with $L_k = L_f$ and $\omega_k = 0$ starting from any $\mathbf{u}^0 = (b^0, \mathbf{w}^0)$. Suppose $\mathbf{u}^* = (b^*, \mathbf{w}^*)$ is the unique solution of (2) with $\lambda_3 > 0$. Let*

$$Q(b, \mathbf{w}) = f(b, \mathbf{w}) + \tfrac{\lambda_2}{2}\|\mathbf{w}\|^2 + \tfrac{\lambda_3}{2}b^2,$$
$$h_i(\mathbf{w}) = w_i - \tfrac{1}{L_f+\lambda_2}\nabla_{w_i} Q(\mathbf{u}),$$

and

$$\mathcal{I} = \{i : |\nabla_{w_i} Q(\mathbf{u}^*)| < \lambda_1\},\ \mathcal{E} = \{i : |\nabla_{w_i} Q(\mathbf{u}^*)| = \lambda_1\},$$

where w_i is the ith component of $\mathbf{w}$. Then $\mathrm{supp}(\mathbf{w}^*) \subset \mathcal{E}$ and $w_i^* = 0, \forall i \in \mathcal{I}$, where $\mathrm{supp}(\mathbf{w}^*)$ denotes the support of $\mathbf{w}^*$. In addition, $w_i^k = 0, \forall i \in \mathcal{I}$ and $\mathrm{sign}(h_i(\mathbf{w}^k)) = \mathrm{sign}(h_i(\mathbf{w}^*)), \forall i \in \mathcal{E}$ for all but at most finite iterations.

Suppose the cardinality of the solution support is s. Then the total computational cost of the two-stage method B-PGH-2 is $O(T_1 np + T_2 ns)$, where T_1, T_2 are the numbers of iterations in the first and second stages, respectively. Numerically, we found that $\mathrm{supp}(\mathbf{w}^*)$ could be detected in several iterations, namely, T_1 is usually small. When $s \ll p$, B-PGH-2 can be significantly faster than B-PGH, as demonstrated by our experiments in Section 3. In the same way, Algorithm 3 can be accelerated by a two-stage method. We omit the analysis since it can be derived by following the same steps.

3 Numerical Experiments

In the first part of this section, we compare B-PGH, described in Algorithm 1, with two very recent binary-class SVM solvers using ADMM [49] and GCD[3] [48] on both synthetic and real data. ADMM solves model (1) whereas both B-PGH and GCD solve model (2). In the second part, we compare the performance of five different multi-class SVMs which are: the model defined by (3), the ℓ_1-regularized M-SVM in [42], the ℓ_∞-regularized M-SVM in [50], the "one-vs-all" (OVA) method [3], and the Decision Directed Acyclic Graph (DDAG) method [36]. We use M-PGH[4], described in Algorithm 3, to solve (3) and Sedumi [39] for the ℓ_1 and ℓ_∞-regularized M-SVMs. Sedumi is called through CVX [21]. All the tests were performed on a computer having an i7-620m CPU and 3-GB RAM and running 32-bit Windows 7 and Matlab 2010b.

3.1 Binary-class SVM

For B-PGH, the parameters η and L_0 were set to $\eta = 1.5$ and $L_0 = \frac{2L_f}{n}$, respectively. We chose

$$\omega_{k-1} = \min\left(\frac{t_{k-1}-1}{t_k}, \sqrt{\frac{L_{k-1}}{L_k}}\right),$$

where $t_0 = 1$ and $t_k = \frac{1}{2}\left(1 + \sqrt{1 + 4t_{k-1}^2}\right)$. The starting point was chosen to be a *zero* vector. We stop B-PGH if both of the following conditions are satisfied during three consecutive iterations

$$\frac{F_{k-1} - F_k}{1 + F_{k-1}} \le tol, \qquad \frac{\|\mathbf{u}^{k-1} - \mathbf{u}^k\|}{1 + \|\mathbf{u}^{k-1}\|} \le tol, \tag{20}$$

where $\mathbf{u}^k = (b^k; \mathbf{w}^k)$ and $F_k = F(\mathbf{u}^k)$. The stopping tolerance was set to $tol = 10^{-6}$ for B-PGH and GCD and $tol = 10^{-5}$ for ADMM since $tol = 10^{-6}$ was too stringent. For B-PGH-2, we took $tol = 10^{-3}$ at the first stage and $tol = 10^{-6}$ at the second stage. The penalty parameters for ADMM were set to $\mu_1 = \frac{100}{n}$ and $\mu_2 = 50$ as suggested in [49]. All the other parameters for ADMM and GCD were set to their default values.

[3] Our algorithm and ADMM are both implemented in MATLAB, while the code of GCD is written in R. To be fair, we re-wrote the code of GCD and implemented it in MATLAB.

[4] The paper [30] uses a path-following method to solve (3). However, its code is not publicly available.

3.1.1 Synthetic data

We generated n samples in $\mathbb{R}^p$ with one half in the "+1" class and the other half in the "−1" class. Specifically, each sample in the "+1" class was generated according to the Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and each sample in the "−1" class according to $\mathcal{N}(-\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The mean vector

$$\boldsymbol{\mu} = (\underbrace{1, \cdots, 1}_{s}, \underbrace{0, \cdots, 0}_{p-s})^\top,$$

and the covariance matrix

$$\boldsymbol{\Sigma} = \begin{bmatrix} \rho \mathbf{1}_{s\times s} + (1-\rho)\mathbf{I}_{s\times s} & \mathbf{0}_{s\times(p-s)} \\ \mathbf{0}_{(p-s)\times s} & \mathbf{I}_{(p-s)\times(p-s)} \end{bmatrix},$$

where $\mathbf{1}_{s\times s}$ is the matrix of size $s \times s$ with all *one*'s, $\mathbf{0}$ is the matrix with all *zero*'s, and $\mathbf{I}_{s\times s}$ is an identity matrix of size $s \times s$. The first s variables are relevant for classification and the rest ones being noise. This simulated data was also used in [44,49]. We tested the speed of the algorithms on different sets of dimension (n, p, s) and the correlation ρ. The smoothing parameter for B-PGH and GCD was set to $\delta = 1$ in this subsection, and $\lambda_3 = \lambda_2$ is set in (2). Table 2 lists the average running time (sec) of 10 independent trials. For each run, we averaged the time over 25 different pairs of (λ_1, λ_2). From the results, we can see that B-PGH was consistently faster than GCD and over 50 times faster than ADMM. In addition, the two-stage accelerated algorithm B-PGH-2 (see section 2.5) was significantly faster than B-PGH when p was large and $s \ll p$.

We also tested the prediction accuracy and variable selection of the algorithms. The problem dimension was set to $n = 50, p = 300, s = 20$. The optimal pair of (λ_1, λ_2) was selected from a large grid by 10-fold cross validation. We use n_t for the number of selected relevant variables and n_f for the number of selected noise variables. In addition, we use "accu." for the prediction accuracy. The average results of 500 independent runs corresponding to $\rho = 0$ and $\rho = 0.8$ are shown in Table 3. During each run, the algorithms were compared on a test set of 1000 samples. From the table, we can see that B-PGH achieved similar accuracy to that by GCD, and they both performed better than ADMM, especially in the case of $\rho = 0.8$. In addition, B-PGH-2 obtained similar solutions as B-PGH.

3.1.2 Medium-scale real data

In this subsection, we compare B-PGH, GCD and ADMM on medium-scale real data (see Table 4). The first seven datasets are available from the LIBSVM dataset[5] and the last three from Tom Mitchells neuroinformatics research group[6]. Both **rcv1** and **realsim** have large feature dimensions but only about 0.2% nonzeros. **colon**, **duke** and **leuk** are datasets of gene expression profiles for colon cancer, breast cancer and leukemia, respectively. The original dataset of **colon** consists of 62 samples, and we randomly chose 30 of them for training and the rest for tesing. **gisette** is a hand-writing digit recognition problem from NIPS 2003 Feature Selection Challenge. The training set for **gisette** is a random subset of the original 6000 samples, and the testing set contains all of the original 1000 samples. **rcv1** is a collection of manually cetegorized news wires from Reuters. Both the training and tesing instances for **sub-rcv1** are randomly sub-sampled from the original training and tesing samples. **realsim** contains UseNet articles from four discussion groups, for simulated auto racing, simulated aviation, real autos and real aviation. The original dataset of **realsim** has 72,309 samples, and we randomly sub-sampled 500 instances for training and 1,000 instances for tesing. **fMRIa**, **fMRIb**, and **fMRIc** are functional MRI (fMRI) data of brain activities when the subjects are presented with pictures and text paragraphs.

Table 4 The size and data type of the tested real datasets

Dataset	#training	#testing	#features	Type
australian	200	490	14	Dense
colon	30	32	2,000	Dense
duke	32	10	7,129	Dense
gisette	500	1,000	5,000	Dense
leuk	38	34	7,129	Dense
sub-rcv1	500	1,000	47,236	Sparse
sub-realsim	500	1,000	20,958	Sparse
fMRIa	30	10	1715	Dense
fMRIb	30	10	1874	Dense
fMRIc	30	10	1888	Dense

We fixed $\lambda_2 = \lambda_3 = 1$ since the algorithms appeared not sensitive to λ_2 and λ_3. The optimal λ_1's were tuned by 10-fold cross-validation on training sets. The smoothing parameter was set to $\delta = 1$ for both B-PGH and GCD. All the other settings are the same as those in the previous test. The results are shown in Table 5. For comparison, we also report the prediction accuracies by LIBLINEAR [11] with L_1-regularized L_2-loss. From the results, we can see that B-PGH is significantly faster than GCD and ADMM, and it also gives the best prediction accuracy. B-PGH-2 was fastest and achieved the same accuracy as B-PGH except for **gisette**. We want to mention that GCD can give the same accuracy as B-PGH but it needs to run much longer time, and ADMM can rarely achieve the same accuracy even if it runs longer since it solves a different model.

[5] http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets

[6] http://www.cs.cmu.edu/∼tom/fmri.html

Table 2 Running time (in seconds) of B-PGH, GCD and ADMM. Each result is the average of 10 independent trials*. For each run, the time is averaged over 25 different pairs of (λ_1, λ_2).

* For $n = 2000, p = 20000$, the time for ADMM is over one half hour.

Problems	B-PGH		B-PGH-2		GCD		ADMM	
(n, p, s)	$\rho = 0$	$\rho = 0.8$	$\rho = 0$	$\rho = 0.8$	$\rho = 0$	$\rho = 0.8$	$\rho = 0$	$\rho = 0.8$
(500, 50, 5)	0.0060	0.0081	0.0059	0.0071	0.0116	0.0192	0.6354	0.8641
(2000, 100, 10)	0.0176	0.0256	0.0173	0.0218	0.0848	0.1469	2.4268	3.5340
(50, 300, 30)	0.0126	0.0162	0.0099	0.0113	0.0338	0.0409	0.6819	1.1729
(100, 500, 50)	0.0179	0.0242	0.0117	0.0152	0.0727	0.0808	1.4879	2.5482
(200, 2000, 100)	0.0720	0.1227	0.0301	0.0446	0.5653	0.4735	7.9985	12.998
(2000, 20000, 200)	5.7341	8.5379	1.1543	1.7531	32.721	30.558	——	——

Table 3 Classification accuracies and variable selections of B-PGH, GCD and ADMM. All results are the averages of 500 independent runs, each of which tests on a 1000 tesing set. The numbers in the parentheses are the corresponding standard errors.

Algorithms	$\rho = 0$			$\rho = 0.8$		
	n_t	n_f	accu.(%)	n_t	n_f	accu.(%)
B-PGH	20.0(0.1)	0.1(0.4)	100(0.000)	19.9(0.3)	7.3(3.9)	86.6(0.011)
B-PGH-2	20.0(0.1)	0.1(0.3)	100(0.000)	19.9(0.4)	7.5(4.1)	86.6(0.011)
GCD	20.0(0.1)	0.1(0.3)	100(0.000)	19.9(0.4)	7.4(3.8)	86.4(0.011)
ADMM	19.0(0.9)	2.9(1.8)	100(0.000)	17.2(1.5)	23.1(6.0)	85.7(0.013)

Table 5 Comparison results of B-PGH, GCD, ADMM and LIBLINEAR on real data. The best prediction accuracy for each dataset is highlighted in **bold** and the best running time (sec) in *italics*.

Dataset	B-PGH			B-PGH-2			GCD			ADMM			LIBLINEAR
	accu(%)	supp	time	accu(%)	supp	time	accu(%)	supp	time	accu(%)	supp	time	accu(%)
australian	**87.4**	11	*0.01*	**87.4**	11	*0.01*	86.7	10	0.02	86.9	14	1.08	85.7
colon	**84.4**	89	*0.04*	**84.4**	89	0.05	**84.4**	89	0.38	**84.4**	118	1.48	81.3
duke	**90**	118	0.20	**90**	118	*0.10*	**90**	112	0.93	**90**	171	3.11	80
gisette	**92.9**	977	1.99	92.7	946	*1.94*	92.6	959	17.61	92.8	1464	218.5	91.7
leuk	**91.2**	847	0.19	**91.2**	846	*0.15*	82.4	716	3.10	82.4	998	2.35	**91.2**
sub-rcv1	**84.8**	1035	0.06	**84.8**	1035	*0.05*	83.3	1035	3.46	82.3	1776	2.61	80.1
sub-realsim	**92.9**	1134	0.04	**92.9**	1134	*0.02*	91.9	1134	2.96	92.8	1727	1.61	90.9
fMRIa	**90**	141	0.07	**90**	141	*0.06*	70	130	5.31	70	203	0.57	70
fMRIb	**100**	1098	0.11	**100**	1108	0.07	90	180	2.26	**100**	1767	*0.03*	70
fMRIc	**100**	1827	0.10	**100**	1825	0.08	80	1324	2.05	90	1882	*0.06*	50

3.1.3 Statistical comparison

We also performed the statistical comparison of B-PGH, GCD, and ADMM. Following [9], we did the Wilcoxon signed-ranks test[7] [45] and Friedman test [12,13] to see if the differences of the compared methods are significant. The former test is for pair-wise comparison and the latter one for multiple comparison. Specifically, for two different methods, denote d_i as the difference of their score (e.g., prediction accuracy) on the i-th dataset and rank d_i's based on their absolute value. Let

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}}, \tag{21}$$

where N is the number of datasets, $T = \min(R^+, R^-)$, and

$$R^+ = \sum_{d_i>0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i),$$
$$R^- = \sum_{d_i<0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i).$$

Table 6 shows the pair-wise z-values and the corresponding p-values of the five compared methods. At p-value < 0.05, we find that there is no significant differences between B-PGH and B-PGH-2 and neither between GCD and ADMM, and any other pair of methods make significant difference.

The Friedman statistic can be calculated by

$$\chi_F^2 = \frac{12N}{K(K+1)}\left[\sum_{j=1}^{K} AR_j - \frac{K(K+1)^2}{4}\right], \tag{22}$$

[7] The Wilcoxon signed-ranks test is in general better than the paired t-test as demonstrated in [9].

Table 6 z-value (above diagonal) and p-value (below diagnal) of Wilcoxon signed-ranks test of B-PGH, GCD, ADMM and LIBLINEAR on real datasets.

Methods	B-PGH	B-PGH-2	GCD	ADMM	LIBLINEAR
B-PGH	–	-0.5096	-2.6502	-2.4973	-2.7521
B-PGH-2	0.6102	–	-2.6502	-2.0896	-2.7521
GCD	0.0080	0.0080	–	-1.4780	-2.0386
ADMM	0.0126	0.0366	0.1394	–	-2.0386
LIBLINEAR	0.0060	0.0060	0.0414	0.0414	–

Table 7 Friedman ranking of B-PGH, GCD, ADMM and LIBLINEAR according to their prediction accuaries on real datasets.

Dataset	B-PGH	B-PGH-2	GCD	ADMM	LBLINEAR
australian	1.5	1.5	4	3	5
colon	2.5	2.5	2.5	2.5	5
duke	2.5	2.5	2.5	2.5	5
gisette	1	2	4	3	5
leuk	2	2	4.5	4.5	2
sub-rcv1	1.5	1.5	3	4	5
sub-realsim	1.5	1.5	4	3	5
fMRIa	1.5	1.5	4	4	4
fMRIb	2	2	4	2	5
fMRIc	1.5	1.5	4	3	5
Average rank	1.75	1.85	3.65	3.15	4.6
χ_F^2-value $= 23.56$, p-value $= 9.78 \times 10^{-5}$					

where K is the number of compared methods, $AR_j = \sum_{i=1}^N r_i^j$ denotes the average rank of the j-th method, and r_i^j is the rank of the j-th method on the i-th dataset. Table 7 shows the ranks of the compared methods according to their prediction accuracies, where average ranks are applied to ties. The p-value indicates significant difference among the five methods at p-value $<$ 0.05.

We further proceeded with a post-hoc test through Holm's step-down procedure [19]. Assuming the p-values for compared classifiers to the control one are ordered as $p_1 \le p_2 \le \ldots \le p_{K-1}$, Holm's step-down procedure starts from the largest one and compares it to $\frac{\alpha}{(K-1)}$, where α is the target p-value. If $p_1 < \frac{\alpha}{(K-1)}$, it rejects the corresponding null-hypothesis and compares p_2 to $\frac{\alpha}{(K-2)}$, and so on. We set B-PGH as the control classifier and used $(R_1 - R_j)/\sqrt{\frac{K(K+1)}{6N}}$ to find the p-value for the j-th method compared to the control one. The p-values are 0.8875, 0.0072, 0.0477, and 5.57×10^{-5} respectively for B-PGH-2, GCD, ADMM, and LIBLINEAR. Hence, at $\alpha = 0.05$, B-PGH made significant difference with GCD and LIBLINEAR but not with B-PGH-2 or ADMM, and at $\alpha = 0.10$, B-PGH made significant difference with all other methods except B-PGH-2.

3.1.4 Large-scale real data

This subsection compares B-PGH, GCD and ADMM on two large-scale datasets. The first one is **rcv1**, part of which has been tested in section 3.1.2. It has 20,242 training samples, and each sample has 47,236 features. We use the same 1,000 samples as in section 3.1.2 for testing. The second dataset contains all the 72,309 samples of **realsim**, part of which is used in section 3.1.2, and each sample has 20,958 features. We randomly selected 20,000 samples for testing and the rest for training. Both the two datasets are highly sparse. For all algorithms, we fixed $b = 0$ since we observed that using the bias term would affect the prediction accuracy. For B-PGH and GCD, we set $\delta = 1$. The best values of λ_1 and λ_2 were searched from a large grid by 10-fold cross-validation. The parameters for B-PGH and GCD were set the same as above. ADMM suffered from memory problem since it needs to explicitly form the matrix $\mathbf{X}^\top\mathbf{X}$, which is dense even though $\mathbf{X}$ is sparse, where the i-th column of $\mathbf{X}$ was formed by the i-th data point $\mathbf{x}_i$. Hence, we did not report the results of ADMM. The results by B-PGH and GCD are shown in Table 8, where we also reported the prediction accuracy by LIBLINEAR for comparison. From the results we can conclude that both or our algorithms, B-PGH and B-PGH-2, are significantly faster (almost 400 times) than the GCD method. In addition, the accuracy of B-PGH and B-PGH-2 is very similar to that of GCD.

3.1.5 Effects of the smoothing parameter δ

We tested how δ affected B-PGH and GCD on the real datasets used in section 3.1.2. Since the cost reduction by B-PGH-2 was not significant for these datasets as shown in Table 5, we did not include it in this test. All parameters were set to the same values as in section 3.1.2 except for δ, which varied between 0.1 and 0.01. The running time and prediction accuracies are shown in Table 9. Comparing the results with those in Table 5, we find that the algorithms tend to give more accurate predictions. However, the accuracy corresponding to $\delta = 0.01$ is hardly improved over $\delta = 0.1$. In addition, the solutions have more nonzeros, i.e., more features are selected. For these reasons, we do not recommend to choose very small δ. We observed that $\delta \in [0.1, 1]$ was fine in all our tests. Furthermore, comparing the columns that show the time in Tables 5 and 9 we observe that the efficiency of the GCD method was greatly affected by different values of δ. In most cases, GCD can become significantly slow with small δ's. On the contrary, the efficiency of B-PGH was seldom affected by different values of δ.

3.2 Multi-class SVM

This subsection tests the performance of M-PGH for solving (3) on a synthetic dataset, eight benchmark

Table 8 Comparison of the accuracy and time (sec) for B-PGH and GCD on the large-scale real datasets **rcv1** and **realsim**.

Dataset	B-PGH			B-PGH-2			GCD			LIBLINEAR
	accu(%)	supp.	time	accu(%)	supp.	time	accu(%)	supp.	time	accu(%)
rcv1	**100**	2188	9.72	**100**	2159	*9.54*	99.7	4253	8384.57	99.5
realsim	96.7	3506	*16.81*	96.7	3429	18.75	96.9	5092	8028.62	**97.0**

Table 9 Performance of B-PGH and GCD on real data with different values of the smoothing parameter ($\delta = 0.1, 0.01$) in the huberized loss function ϕ_H.

Dataset	$\delta = 0.1$						$\delta = 0.01$					
	B-PGH			GCD			B-PGH			GCD		
	accu(%)	supp	time	accu(%)	supp	time	accu(%)	supp	time	accu(%)	supp	time
australian	85.3	11	*0.01*	85.3	11	0.12	85.5	11	*0.01*	86.1	11	0.24
colon	84.4	109	*0.07*	84.4	106	1.16	84.4	123	*0.14*	84.4	118	4.98
duke	90	147	*0.30*	90	158	4.98	90	344	*0.40*	90	181	10.3
gisette	93.1	1394	*2.74*	93.2	1525	4.83	93.0	2781	*3.25*	92.7	1788	19.2
leuk	91.2	1006	*0.57*	88.2	748	18.5	91.2	3678	*0.65*	91.2	970	18.5
sub-rcv1	85.1	1040	*0.03*	85.1	1040	4.71	85.1	1040	*1.16*	85.1	1040	10.7
sub-realsim	93.2	1145	*0.02*	93.2	1145	3.07	93.2	1145	*0.03*	93.2	1145	8.92
fMRIa	90	156	*0.18*	80	149	28.35	90	237	*0.28*	90	217	28.35
fMRIb	100	1335	*0.26*	80	386	2.27	100	1874	*0.23*	100	1399	1.48
fMRIc	100	1856	*0.21*	100	1269	2.19	100	1888	*0.24*	100	1888	2.56

datasets, and also two microarray datasets. The parameters of M-PGH were set in the same way as those of B-PGH except we set $L_0 = \frac{L_m}{nJ}$, where L_m is given in (13). For all the tests, $\delta = 1$ was set in (3).

3.2.1 Synthetic data

We compare model (3) solved using M-PGH, the ℓ_1-regularized M-SVM [42], and the ℓ_∞-regularized M-SVM [50] on a four-class example with each sample in p-dimensional space. The data in class j was generated from the mixture of Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j), j = 1, 2, 3, 4$. The mean vectors and covariance matrices are $\boldsymbol{\mu}_2 = -\boldsymbol{\mu}_1, \boldsymbol{\mu}_4 = -\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_2 = \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_4 = \boldsymbol{\Sigma}_3$, which take the form of

$$\boldsymbol{\mu}_1 = (\underbrace{1, \cdots, 1}_{s}, \underbrace{0, \cdots, 0}_{p-s})^\top,$$

$$\boldsymbol{\mu}_3 = (\underbrace{0, \cdots, 0}_{s/2}, \underbrace{1, \cdots, 1}_{s}, \underbrace{0, \cdots, 0}_{p-3s/2})^\top,$$

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} \rho \mathbf{1}_{s\times s} + (1-\rho)\mathbf{I}_{s\times s} & \mathbf{0}_{s\times(p-s)} \\ \mathbf{0}_{(p-s)\times s} & \mathbf{I}_{(p-s)\times(p-s)} \end{bmatrix},$$

$$\boldsymbol{\Sigma}_3 = \begin{bmatrix} \mathbf{I}_{\frac{s}{2}\times\frac{s}{2}} & \mathbf{0}_{\frac{s}{2}\times s} & \mathbf{0}_{\frac{s}{2}\times(p-\frac{3s}{2})} \\ \mathbf{0}_{s\times\frac{s}{2}} & \rho\mathbf{1}_{s\times s} + (1-\rho)\mathbf{I}_{s\times s} & \mathbf{0}_{s\times(p-\frac{3s}{2})} \\ \mathbf{0}_{(p-\frac{3s}{2})\times\frac{s}{2}} & \mathbf{0}_{(p-\frac{3s}{2})\times s} & \mathbf{I}_{(p-\frac{3s}{2})\times(p-\frac{3s}{2})} \end{bmatrix}.$$

This kind of data has been tested in section 3.1.1 for binary classifications. We took $p = 500, s = 30$ and $\rho = 0, 0.8$ in this test. The best parameters for all three models were tuned by first generating 100 training samples and another 100 validation samples. Then we compared the three different models with the selected parameters on 100 randomly generated training samples and 20,000 random testing samples. The comparison was independently repeated 100 times. The performance of different models and algorithms were measured by prediction accuracy, running time (sec), the number of incorrect zeros (IZ), the number of nonzeros in each column (NZ1, NZ2, NZ3, NZ4).

Table 10 summarizes the average results, where we can see that M-PGH is very efficient in solving (3). In addition we observe that (3) tends to give the best predictions.

3.2.2 Benchmark data

In this subsection, we compare M-PGH to two popular methods for multicategory classification by binary-classifier. The first one is the "one-vs-all" (OVA) method [3] coded in LIBLINEAR library and another one the Decision Directed Acyclic Graph (DDAG) method[8] [36]. We compared them on eight sets of benchmark data, all of which are available from the LIBSVM dataset. The problem statistics are shown in Table 11. The original dataset of **connect4** has 67,557 data points. We randomly chose 500 for training and 1,000 for testing. All 2,000 data points in the training set of **dna** were used, and we randomly chose 500 for training and the rest for testing. **glass** has 214 data points, and we randomly chose 164 for training and another 50 for testing.

[8] The code of DDAG is available from http://theoval.cmp.uea.ac.uk/svm/toolbox/

Table 10 Results of different models solved by M-PGH and Sedumi on a four-class example with synthetic data. The numbers in the parentheses are corresponding standard errors. Highest predictions are highlighted.

models	accu.(%)	time	IZ	NZ1	NZ2	NZ3	NZ4
	$\rho = 0$						
(3) by M-PGH	**96.7**(0.007)	*0.017*	29.43	28.59	29.19	28.78	29.14
ℓ_1-regularized [42] by Sedumi	83.0(0.018)	3.56	59.6	29.3	28.7	29.3	28.9
ℓ_∞-regularized [50] by Sedumi	84.0(0.019)	20.46	33.2	49.3	49.3	49.3	49.3
	$\rho = 0.8$						
(3) by M-PGH	**78.4**(0.020)	*0.021*	35.15	26.93	27.29	26.41	26.75
ℓ_1-regularized [42] by Sedumi	64.8(0.024)	3.50	91.6	16.4	17.1	17.2	16.4
ℓ_∞-regularized [50] by Sedumi	67.2(0.015)	20.64	74.1	46.1	46.1	46.1	46.1

For **letter**, we randomly picked 1300 out of the original 15,000 training samples with 50 for each class for training and 500 out of the original 5,000 testing points for testing. The **poker** dataset has 25,010 training and 1 million testing data points. For each class, we randomly selected 50 out of each class for training except the 6th through 9th classes which have less than 50 samples and hence were all used. In addition, we randomly chose 100k points from the testing set for testing. **protein** has 17,766 training data points and 6,621 testing points. We randomly chose 1,500 from the training dataset for training and all the points in the testing dataset for testing. **usps** is a handwritten digit dataset consisting of 7291 training and 2007 testing digits from 0 to 9. We randomly picked 50 with 5 for each class out of the training set for training and 500 out of the testing set for testing. **wine** has 128 data points, and we randomly chose 50 for training and the rest for testing.

We fixed $\lambda_3 = 1$ in (3) for M-PGH and tuned λ_1, λ_2. DDAG has two parameters C, γ, which were tuned from a large grid of values. The parameters for OVA were set to their default values in the code of LIBLINEAR. We did the tests for 10 times independently. The average prediction accuracies[9] by the three different methods are reported in Table 12. From the results, we see that M-PGH performs consistently better than OVA except for **letter** and also comparable with DDAG. When the training samples are sufficiently many, DDAG can give higher prediction accuracies such as for **glass** and **letter**. However, it can perform badly if the training samples are few compared to the feature numbers such as for **dna** and also the tests in the next subsection. In addition, note that the **poker** dataset is imbalanced, and our "all-together" model (3) gives significantly higher accuracy than that of OVA. This suggests that "all-together" methods can perform better than "one-vs-all" in imbalanced datasets. We plan to investigate this further in a follow up paper.

[9] Our reported accuracies are lower than those in [20] because we used fewer training samples.

Table 11 Statistics of eight benchmark datasets.

dataset	#training	#testing	#feature	#class
connect4	500	1000	126	3
dna	500	1500	180	3
glass	164	50	9	6
letter	1300	500	16	26
poker	352	100k	10	10
protein	1500	6621	357	3
usps	50	500	256	10
wine	50	128	13	3

Table 12 Prediction accuracies (%) by different methods on benchmark data.

dataset	M-PGH	OVA	DDAG
connect4	**53.28**	51.20	53.07
dna	**92.82**	89.04	33.71
glass	53.00	51.40	**62.80**
letter	43.24	65.90	**80.96**
poker	**35.13**	14.29	24.05
protein	**60.22**	56.84	53.20
usps	74.44	73.46	**76.28**
wine	**96.64**	96.25	94.00

3.2.3 Application to microarray classification

This subsection applies M-PGH to microarray classifications. Two real data sets were used. One is the children cancer data set in [23], which used cDNA gene expression profiles and classified the small round blue cell tumors (SRBCTs) of childhood into four classes: neuroblastoma (NB), rhabdomyosarcoma (RMS), Burkitt lymphomas (BL) and the Ewing family of tumors (EWS). The other is the leukemia data set in [16], which used gene expression monitoring and classified the acute leukemias into three classes: B-cell acute lymphoblastic leukemia (B-ALL), T-cell acute lymphoblastic leukemia (T-ALL) and acute myeloid leukemia (AML). The original distributions of the two data sets are given in Table 13. Both the two data sets have been tested before on certain M-SVMs for gene selection; see [42,50] for example.

Table 13 Original distributions of SRBCT and leukemia data sets

Data set	SRBCT					leukemia			
	NB	RMS	BL	EWS	total	B-ALL	T-ALL	AML	total
Training	12	20	8	23	63	19	8	11	38
Testing	6	5	3	6	20	19	1	14	34

Each observation in the SRBCT dataset has dimension of $p = 2308$, namely, there are 2308 gene profiles. We first standardized the original training data in the following way. Let $\mathbf{X}^o = [\mathbf{x}_1^o, \cdots, \mathbf{x}_n^o]$ be the original data matrix. The standardized matrix $\mathbf{X}$ was obtained by

$$x_{gj} = \frac{x_{gj}^o - \text{mean}(x_{g1}^o, \cdots, x_{gn}^o)}{\text{std}(x_{g1}^o, \cdots, x_{gn}^o)}, \ \forall g, j.$$

Similar normalization was done to the original testing data. Then we selected the best parameters of each model by three-fold cross validation on the standardized training data. Finally, we put the standardized training and testing data sets together and randomly picked 63 observations for training and the remaining 20 ones for testing. The average prediction accuracy, running time (sec), number of nonzeros (NZ) and number of nonzero rows (NR) of 100 independent trials are reported in Table 14, from which we can see that all models give similar prediction accuracies. The ℓ_∞-regularized M-SVM gives denser solutions, and M-PGH is much faster than Sedumi.

The leukemia data set has $p = 7,129$ gene profiles. We standardized the original training and testing data in the same way as that in last test. Then we rank all genes on the standardized training data by the method used in [10]. Specifically, let $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_n]$ be the standardized data matrix. The relevance measure for gene g is defined as follows:

$$R(g) = \frac{\sum_{i,j} I(y_i = j)(m_g^j - m_g)}{\sum_{i,j} I(y_i = j)(x_{gi} - m_g^j)}, \ g = 1, \cdots, p,$$

where m_g denotes the mean of $\{x_{g1}, \cdots, x_{gn}\}$ and m_g^j denotes the mean of $\{x_{gi} : y_i = j\}$. According to $R(g)$, we selected the 3,571 most significant genes. Finally, we put the processed training and tesing data together and randomly chose 38 samples for training and the remaining ones for testing. The process was independently repeated 100 times. We compared all the above five different methods for M-SVMs. Table 14 summarizes the average results, which show that M-PGH is significantly faster than Sedumi and that model (3) gives comparable prediction accuracies with relatively denser solutions. In addition, we note that DDAG performs very badly possibly because the training samples are too few.

Table 15 z-value (above diagonal) and p-value (below diagnal) of Wilcoxon signed-ranks test of M-PGH, OVA, and DDAG on the eight benchmark and two microarray datasets.

Methods	M-PGH	OVA	DDAG
M-PGH	–	-1.7838	-1.2741
OVA	0.0745	–	-0.5606
DDAG	0.2026	0.5751	–

Table 16 Average ranks of M-PGH, OVA, and DDAG according to their classification accuracies on the eight benchmark and two microarray datasets

Methods	M-PGH	OVA	DDAG
Average rank	1.4	2.4	2.2
χ_F^2-value = 5.6, p-value = 0.0608			

3.2.4 Statistical comparison

As in section 3.1.3, we also performed statistical comparison of M-PGH, OVA, and DDAG on the benchmark datasets together with the two microarray datasets. Table 15 shows their z-values and corresponding p-values of the Wilcoxon signed-rank test. From the table, we see that there is no significant difference between any pair of the three methods at p-value < 0.05. However, M-PGH makes significant difference with OVA at p-value < 0.10. Average ranks of M-PGH, OVA, and DDAG according to their prediction accuracies on the 10 datasets are shown in Table 16 together with the Friedman statistic and p-value. Again, we see that there is no significant difference among the three methods at p-value < 0.05 but there is at p-value < 0.10. We further did a post-hoc test using the Holm's step-down procedure as in section 3.1.3. M-PGH was set as the control classifier. The p-values are 0.0253 and 0.0736 respectively for OVA and DDAG. Hence, M-PGH made significant differences with OVA and DDAG at p-value $= 0.10$.

4 Conclusions

SVMs have been popularly used to solve a wide variety of classification problems. The original SVM model uses the non-differentiable hinge loss function, which together with some regularizers like ℓ_1-term makes it difficult to develop simple yet efficient algorithms. We considered the huberized hinge loss function that is a

Table 14 Comparison of computational results for the different methods on SRBCT and leukemia datasets

Problems	SRBCT				leukemia			
	accu.(%)	time	NZ	NR	accu.(%)	time	NZ	NR
(3) by M-PGH	98.6(0.051)	*0.088*	220.44	94.43	**91.9**(0.049)	*0.241*	457.02	218.25
ℓ_1-regularized [42] by Sedumi	**98.9**(0.022)	13.82	213.67	96.71	85.2(0.063)	11.21	82.41	40.00
ℓ_∞-regularized [50] by Sedumi	97.9(0.033)	120.86	437.09	109.28	85.2(0.063)	169.56	82.41	40.00
OVA	96.2	—	—	—	81.5	—	—	—
DDAG	72.0	—	—	—	47.1	—	—	—

differentiable approximation of the original hinge loss. This allowed us to apply PG method to both binary-class and multi-class SVMs in an efficient and accurate way. In addition, we presented a two-stage algorithm that is able to solve very large-scale binary classification problems. Assuming strong convexity and under fairly general assumptions, we proved the linear convergence of PG method when applied in solving composite problems in the form of (4), special cases of which are the binary and multi-class SVMs. We performed a wide range of numerical experiments on both synthetic and real datasets, demonstrating the superiority of the proposed algorithms over some state-of-the-art algorithms for both binary and multi-class SVMs. In particular for large-scale problems, our algorithms are significantly faster than compared methods in all cases with comparable accuracy. Finally, our algorithms are more robust to the smoothing parameter δ in terms of CPU time.

Acknowledgements

The authors would like to thank four anonymous referees and the associate editor for their valuable comments and suggestions that improved the quality of our paper. The first author would also like to thank Professor Wotao Yin for his valuable discussion.

A Proof of Proposition 1

First, we derive the convexity of each f_i from the composition of the convex function ϕ_H and the linear transformation $y_i(b+\mathbf{x}_i^\top \mathbf{w})$ (e.g., see [4]). Thus, f is also convex since it is the sum of n convex functions. Secondly, it is easy to verify that

$$\phi'_H(t)=\begin{cases}0, & \text{for } t>1;\\ \frac{t-1}{\delta}, & \text{for } 1-\delta<t\le 1;\\ -1, & \text{for } t\le 1-\delta;\end{cases}$$

and it is Lipschitz continuous, namely, $|\phi'_H(t)-\phi'_H(s)|\le \frac{1}{\delta}|t-s|$, for any t,s.

Using the notations in (7) and by the chain rule, we have $\nabla f_i(\mathbf{u})=\phi'_H(\mathbf{v}_i^\top\mathbf{u})\mathbf{v}_i$, for $i=1,\cdots,n$ and

$$\nabla f(\mathbf{u})=\frac{1}{n}\sum_{i=1}^n \phi'_H(\mathbf{v}_i^\top\mathbf{u})\mathbf{v}_i. \tag{23}$$

Hence, for any $\mathbf{u},\hat{\mathbf{u}}$, we have

$$\begin{aligned}&\|\nabla f(\mathbf{u})-\nabla f(\hat{\mathbf{u}})\|\\ \le&\tfrac{1}{n}\textstyle\sum_{i=1}^n\|\nabla f_i(\mathbf{u})-\nabla f_i(\hat{\mathbf{u}})\|\\ =&\tfrac{1}{n}\textstyle\sum_{i=1}^n\left\|\left(\phi'(\mathbf{v}_i^\top\mathbf{u})-\phi'(\mathbf{v}_i^\top\hat{\mathbf{u}})\right)\mathbf{v}_i\right\|\\ \le&\tfrac{1}{n\delta}\textstyle\sum_{i=1}^n|\mathbf{v}_i^\top(\mathbf{u}-\hat{\mathbf{u}})|\|\mathbf{v}_i\|\\ \le&\tfrac{1}{n\delta}\textstyle\sum_{i=1}^n\|\mathbf{v}_i\|^2\|\mathbf{u}-\hat{\mathbf{u}}\|,\end{aligned}$$

which completes the proof.

B Proof of Theorem 1

We begin our analysis with the following lemma, which can be shown through essentially the same proof of Lemma 2.3 in [1].

Lemma 2 *Let $\{\mathbf{u}^k\}$ be the sequence generated by (5) with $L_k\le L$ such that for all $k\ge 1$,*

$$\begin{aligned}\xi_1(\mathbf{u}^k)\le\xi_1(\hat{\mathbf{u}}^{k-1})+\left\langle\nabla\xi_1(\hat{\mathbf{u}}^{k-1}),\mathbf{u}^k-\hat{\mathbf{u}}^{k-1}\right\rangle\\ +\tfrac{L_k}{2}\left\|\mathbf{u}^k-\hat{\mathbf{u}}^{k-1}\right\|^2.\end{aligned} \tag{24}$$

Then for all $k\ge 1$, it holds for any $\mathbf{u}\in\mathcal{U}$ that

$$\begin{aligned}\xi(\mathbf{u})-\xi(\mathbf{u}^k)\ge\tfrac{L_k}{2}\|\mathbf{u}^k-\hat{\mathbf{u}}^{k-1}\|^2+\tfrac{\mu}{2}\|\mathbf{u}-\mathbf{u}^k\|^2\\ +L_k\langle\hat{\mathbf{u}}^{k-1}-\mathbf{u},\mathbf{u}^k-\hat{\mathbf{u}}^{k-1}\rangle.\end{aligned} \tag{25}$$

We also need the following lemma, which is the KL property for strongly convex functions.

Lemma 3 *Suppose ξ is strongly convex with constant $\mu>0$. Then for any $\mathbf{u},\mathbf{v}\in\mathrm{dom}(\partial\xi)$ and $\mathbf{g_u}\in\partial\xi(\mathbf{u})$, we have*

$$\xi(\mathbf{u})-\xi(\mathbf{v})\le\frac{1}{\mu}\|\mathbf{g_u}\|^2. \tag{26}$$

Proof For any $\mathbf{g_u}\in\partial\xi(\mathbf{u})$, we have from the strong convexity of ξ and the Cauchy-Schwarz inequality that

$$\xi(\mathbf{u})-\xi(\mathbf{v})\le\langle\mathbf{g_u},\mathbf{u}-\mathbf{v}\rangle-\frac{\mu}{2}\|\mathbf{u}-\mathbf{v}\|^2\le\frac{1}{\mu}\|\mathbf{g_u}\|^2,$$

which completes the proof.

Global convergence

We first show $\mathbf{u}^k \to \mathbf{u}^*$. Letting $\mathbf{u} = \mathbf{u}^{k-1}$ in (25) gives

$$\begin{aligned}
&\xi(\mathbf{u}^{k-1}) - \xi(\mathbf{u}^k) \\
\geq &\frac{L_k}{2}\|\mathbf{u}^k - \hat{\mathbf{u}}^{k-1}\|^2 + \frac{\mu}{2}\|\mathbf{u}^{k-1} - \mathbf{u}^k\|^2 \\
&+ L_k\langle \hat{\mathbf{u}}^{k-1} - \mathbf{u}^{k-1}, \mathbf{u}^k - \hat{\mathbf{u}}^{k-1}\rangle \\
= &\frac{L_k}{2}\|\mathbf{u}^k - \mathbf{u}^{k-1}\|^2 + \frac{\mu}{2}\|\mathbf{u}^k - \mathbf{u}^{k-1}\|^2 \\
&- \frac{L_k}{2}\omega_{k-1}^2\|\mathbf{u}^{k-1} - \mathbf{u}^{k-2}\|^2 \\
\geq &\frac{L_k}{2}\|\mathbf{u}^k - \mathbf{u}^{k-1}\|^2 + \frac{\mu}{2}\|\mathbf{u}^k - \mathbf{u}^{k-1}\|^2 \qquad (27) \\
&- \frac{L_{k-1}}{2}\|\mathbf{u}^{k-1} - \mathbf{u}^{k-2}\|^2.
\end{aligned}$$

Summing up the inequality (27) over k, we have

$$\begin{aligned}
&\xi(\mathbf{u}^0) - \xi(\mathbf{u}^K) \\
\geq &\ \tfrac{\mu}{2}\textstyle\sum_{k=1}^K \|\mathbf{u}^{k-1} - \mathbf{u}^k\|^2 + \frac{L_K}{2}\|\mathbf{u}^K - \mathbf{u}^{K-1}\|^2,
\end{aligned}$$

which implies $\mathbf{u}^k - \mathbf{u}^{k-1} \to \mathbf{0}$ since $\xi(\mathbf{u}^k)$ is lower bounded.

Note that $\{\mathbf{u}^k\}$ is bounded since ξ is coercive and $\{\xi(\mathbf{u}^k)\}$ is upper bounded by $\xi(\mathbf{u}^0)$. Hence, $\{\mathbf{u}^k\}$ has a limit point $\bar{\mathbf{u}}$, so there is a subsequence $\{\mathbf{u}^{k_j}\}$ converging to $\bar{\mathbf{u}}$. Note $\mathbf{u}^{k_j+1}$ also converges to $\bar{\mathbf{u}}$ since $\mathbf{u}^k - \mathbf{u}^{k-1} \to \mathbf{0}$. Without loss of generality, we assume $L_{k_j+1} \to \bar{L}$. Otherwise, we can take a convergent subsequence of $\{L_{k_j+1}\}$. By (5), we have

$$\begin{aligned}
\mathbf{u}^{k_j+1} = \arg\min_{\mathbf{u}} &\xi_1(\hat{\mathbf{u}}^{k_j}) + \langle \nabla\xi_1(\hat{\mathbf{u}}^{k_j}), \mathbf{u} - \hat{\mathbf{u}}^{k_j}\rangle \\
&+ \tfrac{L_{k_j+1}}{2}\|\mathbf{u} - \hat{\mathbf{u}}^{k_j}\|^2 + \xi_2(\mathbf{u}).
\end{aligned}$$

Letting $j \to \infty$ in the above formula and observing $\hat{\mathbf{u}}^{k_j} \to \bar{\mathbf{u}}$ yield

$$\bar{\mathbf{u}} = \arg\min_{\mathbf{u}} \xi_1(\bar{\mathbf{u}}) + \langle \nabla\xi_1(\bar{\mathbf{u}}), \mathbf{u} - \bar{\mathbf{u}}\rangle + \tfrac{\bar{L}}{2}\|\mathbf{u} - \bar{\mathbf{u}}\|^2 + \xi_2(\mathbf{u}),$$

which indicates $-\nabla\xi_1(\bar{\mathbf{u}}) \in \partial\xi_2(\bar{\mathbf{u}})$. Thus $\bar{\mathbf{u}}$ is the minimizer of ξ.

Since $\{\xi(\mathbf{u}^k)\}$ is nonincreasing and lower bounded, it converges to $\xi(\bar{\mathbf{u}}) = \xi(\mathbf{u}^*)$. Noting $\frac{\mu}{2}\|\mathbf{u}^k - \mathbf{u}^*\|^2 \leq \xi(\mathbf{u}^k) - \xi(\mathbf{u}^*)$, we have $\mathbf{u}^k \to \mathbf{u}^*$.

Convergence rate

Next we go to show (19). Without loss of generality, we assume $\xi(\mathbf{u}^*) = 0$. Otherwise, we can consider $\xi - \xi(\mathbf{u}^*)$ instead of ξ. In addition, we assume $\xi(\mathbf{u}^k) > \xi(\mathbf{u}^*)$ for all $k \geq 0$ because if $\xi(\mathbf{u}^{k_0}) = \xi(\mathbf{u}^*)$ for some k_0, then $\mathbf{u}^k = \mathbf{u}^{k_0} = \mathbf{u}^*$ for all $k \geq k_0$.

For ease of description, we denote $\xi_k = \xi(\mathbf{u}^k)$. Letting $\mathbf{v} = \mathbf{u}^*$ in (26), we have

$$\sqrt{\xi_k} \leq \frac{1}{\sqrt{\mu}}\|\mathbf{g}_{\mathbf{u}^k}\|, \text{ for all } \mathbf{g}_{\mathbf{u}^k} \in \partial\xi(\mathbf{u}^k). \qquad (28)$$

Noting $-\nabla\xi_1(\hat{\mathbf{u}}^{k-1}) - L_k(\mathbf{u}^k - \hat{\mathbf{u}}^{k-1}) + \nabla\xi_1(\mathbf{u}^k) \in \partial\xi(\mathbf{u}^k)$ and $L_k \leq L$, we have for all $k \geq K$,

$$\sqrt{\xi_k} \leq \frac{2L}{\sqrt{\mu}}\left(\|\mathbf{u}^k - \mathbf{u}^{k-1}\| + \|\mathbf{u}^{k-1} - \mathbf{u}^{k-2}\|\right) \qquad (29)$$

Noting $\sqrt{\xi_k} - \sqrt{\xi_{k+1}} \geq \frac{\xi_k - \xi_{k+1}}{2\sqrt{\xi_k}}$ and using (27) yield

$$\begin{aligned}
&(L_{k+1} + \mu)\|\mathbf{u}^k - \mathbf{u}^{k+1}\|^2 \\
\leq &\ L_k\|\mathbf{u}^{k-1} - \mathbf{u}^k\|^2 + \tfrac{8L}{\sqrt{\mu}}\left(\sqrt{\xi_k} - \sqrt{\xi_{k+1}}\right)\|\mathbf{u}^k - \mathbf{u}^{k-1}\| \\
&+ \tfrac{8L}{\sqrt{\mu}}\left(\sqrt{\xi_k} - \sqrt{\xi_{k+1}}\right)\|\mathbf{u}^{k-1} - \mathbf{u}^{k-2}\|,
\end{aligned}$$

after rearrangements. Take $0 < \delta < \frac{1}{2}(\sqrt{L+\mu} - \sqrt{L})$. Using the inequalities $a^2 + b^2 \leq (a+b)^2$ and $\sqrt{ab} \leq ta + \frac{1}{4t}b$ for $a, b, t > 0$, we have from the above inequality that

$$\begin{aligned}
&\sqrt{L_{k+1} + \mu}\|\mathbf{u}^k - \mathbf{u}^{k+1}\| \\
\leq &\ \sqrt{L_k}\|\mathbf{u}^{k-1} - \mathbf{u}^k\| + \tfrac{2L}{\delta\sqrt{\mu}}\left(\sqrt{\xi_k} - \sqrt{\xi_{k+1}}\right) \\
&+ \delta\left(\|\mathbf{u}^k - \mathbf{u}^{k-1}\| + \|\mathbf{u}^{k-1} - \mathbf{u}^{k-2}\|\right).
\end{aligned}$$

Summing up the above inequality over k and noting $\|\mathbf{u}^k - \mathbf{u}^{k+1}\| \to 0, \xi_k \to 0$ yield

$$\begin{aligned}
&\textstyle\sum_{k=m}^{\infty}\left(\sqrt{L_{k+1} + \mu} - \sqrt{L_{k+1}} - 2\delta\right)\|\mathbf{u}^k - \mathbf{u}^{k+1}\| \\
\leq &\left(\sqrt{L} + 2\delta\right)\|\mathbf{u}^{m-1} - \mathbf{u}^m\| + \delta\|\mathbf{u}^{m-2} - \mathbf{u}^{m-1}\| + \tfrac{2L}{\delta\sqrt{\mu}}\sqrt{\xi_m},
\end{aligned}$$

which together with $\sqrt{L+\mu} - \sqrt{L} \leq \sqrt{L_k + \mu} - \sqrt{L_k}$ for all $k \geq 0$ implies

$$\begin{aligned}
&\textstyle\sum_{k=m}^{\infty}\|\mathbf{u}^k - \mathbf{u}^{k+1}\| \\
\leq &\ C_1\sqrt{\xi_m} + C_2\left(\|\mathbf{u}^{m-1} - \mathbf{u}^m\| + \|\mathbf{u}^{m-2} - \mathbf{u}^{m-1}\|\right),
\end{aligned} \qquad (30)$$

where

$$C_1 = \tfrac{2L}{\delta\sqrt{\mu}\left(\sqrt{L+\mu} - \sqrt{L} - 2\delta\right)},\ C_2 = \tfrac{\sqrt{L} + 2\delta}{\sqrt{L+\mu} - \sqrt{L} - 2\delta}.$$

Denote $S_m = \sum_{k=m}^{\infty}\|\mathbf{u}^k - \mathbf{u}^{k+1}\|$ and write (30) as

$$S_m \leq C_1\sqrt{\xi_m} + C_2(S_{m-2} - S_m),$$

which together with (29) gives

$$\begin{aligned}
S_m &\leq \left(C_1\tfrac{2L}{\sqrt{\mu}} + C_2\right)(S_{m-2} - S_m) \\
&= C_3(S_{m-2} - S_m), \text{ for all } m \geq 2.
\end{aligned}$$

Let $\tau = \sqrt{\frac{C_3}{1+C_3}}$ and $C = S_0$. Then we have $S_m \leq C\tau^m, \forall m \geq 0$. Note $\|\mathbf{u}^m - \mathbf{u}^*\| \leq S_m$, and thus we complete the proof.

References

1. A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.
2. J. Bolte, A. Daniilidis, and A. Lewis. The lojasiewicz inequality for nonsmooth subanalytic functions with applications to subgradient dynamical systems. *SIAM Journal on Optimization*, 17(4):1205–1223, 2007.
3. L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U.A. Muller, E. Sackinger, P. Simard, et al. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, volume 2, pages 77–82. IEEE, 1994.
4. S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, 2004.

5. M.P.S. Brown, W.N. Grundy, D. Lin, N. Cristianini, C.W. Sugnet, T.S. Furey, M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene expression data by using support vector machines. *Proceedings of the National Academy of Sciences*, 97(1):262, 2000.
6. Zhen-Yu Chen, Zhi-Ping Fan, and Minghe Sun. A hierarchical multiple kernel support vector machine for customer churn prediction using longitudinal behavioral data. *European Journal of Operational Research*, 223(2):461–472, 2012.
7. C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
8. W. M. Czarnecki and J. Tabor. Two ellipsoid support vector machines. *Expert Systems with Applications*, 41:8211–8224, 2014.
9. Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.
10. S. Dudoit, J. Fridlyand, and T.P. Speed. Comparison of discrimination methods for the classification of tumors using gene expression data. *Journal of the American statistical association*, 97(457):77–87, 2002.
11. Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.
12. Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.
13. Milton Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.
14. M. Galar, A. Fernandez, E. Barrenechea, and F. Herrera. Drcw-ovo: Distance-based relative competence weighting combination for one-vs-one strategy in multiclass problems. *Pattern Recognition*, 48:28–42, 2015.
15. Mikel Galar, Alberto Fernández, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recognition*, 44(8):1761–1776, 2011.
16. T.R. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J.P. Mesirov, H. Coller, M.L. Loh, J.R. Downing, M.A. Caligiuri, et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *science*, 286(5439):531–537, 1999.
17. E. T. Hale, W. Yin, and Y. Zhang. Fixed-point continuation for l_1-minimization: methodology and convergence. *SIAM J. Optim.*, 19(3):1107–1130, 2008.
18. B. Heisele, P. Ho, and T. Poggio. Face recognition with support vector machines: Global versus component-based approach. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 688–694. IEEE, 2001.
19. Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.
20. C.W. Hsu and C.J. Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.
21. CVX Research, Inc. CVX: Matlab software for disciplined convex programming, version 2.0 beta. *http://cvxr.com/cvx*, September 2012.
22. S.S. Keerthi and D. DeCoste. A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6(1):341, 2006.
23. J. Khan, J.S. Wei, M. Ringnér, L.H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C.R. Antonescu, C. Peterson, et al. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature medicine*, 7(6):673–679, 2001.
24. K.I. Kim, K. Jung, S.H. Park, and H.J. Kim. Support vector machines for texture classification. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(11):1542–1550, 2002.
25. Stefan Knerr, Léon Personnaz, and Gérard Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In *Neurocomputing*, pages 41–50. Springer, 1990.
26. B. Krawcczyk, M. Wozniak, and B. Cyganek. Clustering-based ensembles for one-class classification. *Information Sciences*, 264:182–195, 2014.
27. K. Kurdyka. On gradients of functions definable in o-minimal structures. In *Annales de l'institut Fourier*, volume 48, pages 769–784. Chartres: L'Institut, 1950-, 1998.
28. Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate frank-wolfe optimization for structural svms. *arXiv preprint arXiv:1207.4747*, 2012.
29. Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines. *Journal of the American Statistical Association*, 99(465):67–81, 2004.
30. J. Li and Y. Jia. Huberized multiclass support vector machine for microarray classification. *Acta Automatica Sinica*, 36(3):399–405, 2010.
31. Qi Li, Raied Salman, Erik Test, Robert Strack, and Vojislav Kecman. Parallel multitask cross validation for support vector machine using gpu. *Journal of Parallel and Distributed Computing*, 73(3):293–302, 2013.
32. S. Łojasiewicz. Sur la géométrie semi-et sous-analytique. *Ann. Inst. Fourier (Grenoble)*, 43(5):1575–1595, 1993.
33. Y. Nesterov. Introductory lectures on convex optimization. 87, 2004. A basic course.
34. Y. Nesterov. Gradient methods for minimizing composite objective function. *CORE Discussion Papers*, 2007.
35. Hua Ouyang, Niao He, Long Tran, and Alexander Gray. Stochastic alternating direction method of multipliers. In *Proceedings of the 30th International Conference on Machine Learning*, pages 80–88, 2013.
36. J.C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification. *Advances in neural information processing systems*, 12(3):547–553, 2000.
37. Zhiquan Qi, Yingjie Tian, and Yong Shi. Structural twin support vector machine for classification. *Knowledge-Based Systems*, 43:74–81, 2013.
38. M. Schmidt, N.L. Roux, and F. Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. *Arxiv preprint arXiv:1109.2415*, 2011.
39. J.F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4):625–653, 1999.
40. R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.
41. Peter Tsyurmasto, Michael Zabarankin, and Stan Uryasev. Value-at-risk support vector machine: stability to outliers. *Journal of Combinatorial Optimization*, pages 1–15, 2014.
42. L. Wang and X. Shen. On L_1-norm multiclass support vector machines. *Journal of the American Statistical Association*, 102(478):583–594, 2007.

43. L. Wang, J. Zhu, and H. Zou. The doubly regularized support vector machine. *Statistica Sinica*, 16(2):589, 2006.
44. L. Wang, J. Zhu, and H. Zou. Hybrid huberized support vector machines for microarray classification and gene selection. *Bioinformatics*, 24(3):412–419, 2008.
45. Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, pages 80–83, 1945.
46. Y. Xu and W. Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on Imaging Sciences*, 6(3):1758–1789, 2013.
47. Y. Xu and W. Yin. A globally convergent algorithm for nonconvex optimization based on block coordinate update. *Arxiv preprint arXiv:1410.1386*, 2014.
48. Yi Yang and Hui Zou. An efficient algorithm for computing the hhsvm and its generalizations. *Journal of Computational and Graphical Statistics*, 22(2):396–415, 2013.
49. G.B. Ye, Y. Chen, and X. Xie. Efficient variable selection in support vector machines via the alternating direction method of multipliers. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2011.
50. H. Zhang, Y. Liu, Y. Wu, and J. Zhu. Variable selection for the multicategory svm via adaptive sup-norm regularization. *Electronic Journal of Statistics*, 2:149–167, 2008.
51. Yang Zhang, Nirvana Meratnia, and Paul JM Havinga. Distributed online outlier detection in wireless sensor networks using ellipsoidal support vector machine. *Ad hoc networks*, 11(3):1062–1074, 2013.
52. J. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67(1):301–320, 2005.