Taylor & Francis
Taylor & Francis Group

Journal of Computational and Graphical Statistics

ISSN: 1061-8600 (Print) 1537-2715 (Online) Journal homepage: http://www.tandfonline.com/loi/ucgs20

Fast Algorithms for Large-Scale Generalized Distance Weighted Discrimination

Xin Yee Lam, J. S. Marron, Defeng Sun & Kim-Chuan Toh

To cite this article: Xin Yee Lam, J. S. Marron, Defeng Sun & Kim-Chuan Toh (2018) Fast Algorithms for Large-Scale Generalized Distance Weighted Discrimination, Journal of Computational and Graphical Statistics, 27:2, 368-379, DOI: 10.1080/10618600.2017.1366915

To link to this article: https://doi.org/10.1080/10618600.2017.1366915

JOURNAL OF COMPUTATIONAL AND GRAPHICAL STATISTICS
2018, VOL. 27, NO. 2, 368–379
https://doi.org/10.1080/10618600.2017.1366915

Fast Algorithms for Large-Scale Generalized Distance Weighted Discrimination

Xin Yee Lam[a], J. S. Marron [b], Defeng Sun[c], and Kim-Chuan Toh[d]

[a]Department of Mathematics, National University of Singapore, Singapore; [b]Department of Statistics and Operations Research, University of North Carolina, Chapel Hill, NC; [c]The Hong Kong Polytechnic University, Hong Kong; [d]Department of Mathematics and Institute of Operations Research and Analytics, National University of Singapore, Singapore

ABSTRACT

High-dimension-low-sample size statistical analysis is important in a wide range of applications. In such situations, the highly appealing discrimination method, support vector machine, can be improved to alleviate data piling at the margin. This leads naturally to the development of distance weighted discrimination (DWD), which can be modeled as a second-order cone programming problem and solved by interior-point methods when the scale (in sample size and feature dimension) of the data is moderate. Here, we design a scalable and robust algorithm for solving large-scale generalized DWD problems. Numerical experiments on real datasets from the UCI repository demonstrate that our algorithm is highly efficient in solving large-scale problems, and sometimes even more efficient than the highly optimized LIBLINEAR and LIBSVM for solving the corresponding SVM problems. Supplementary material for this article is available online.

ARTICLE HISTORY
Received February 2017
Revised May 2017

KEYWORDS
Convergent multi-block ADMM; Data piling; Support vector machine

1. Introduction

We consider the problem of finding a (kernelized) linear classifier for a training dataset $\{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$ and the class label $y_i \in \{-1, 1\}$ for all $i = 1, \ldots, n$. Here n is the sample size (the number of data vectors available) and d is the feature dimension. By far, the most popular and successful method for getting a good linear classifier from the training data is the support vector machine (SVM), originally proposed by Vapnik (1995). Indeed, it has been demonstrated in Fernández-Delgado et al. (2014) that the kernel SVM is one of the best performers in the pool of 179 commonly used classifiers. Despite its success, it has been observed in Marron, Todd, and Ahn (2007) that SVM may suffer from a "data-piling" problem in the high-dimension-low-sample size (HDLSS) setting (where the sample size n is smaller than the feature dimension d). The authors proposed a new linear classifier, called "distance weighted discrimination" (DWD), as a superior alternative to the SVM. DWD has become a workhorse method for a number of statistical tasks, including data visualization (Marron and Alonso 2014), hypothesis testing linked with visualization in very high dimensions (Wei et al. 2016), and adjustment for data biases and heterogeneity (Benito et al. 2004; Liu et al. 2009).

It is well known that there is a strong need for efficient HDLSS methods for the settings where d is large, say in the order of 10^4–10^5, especially in the area of genetic molecular measurements (usually having a small sample size, where many gene level features have been measured), chemometrics (typically a small sample of high-dimensional spectra), and medical image analysis (a small sample of 3-d shapes represented by high-dimensional vectors). On the other hand, given the advent of a huge volume of data collectible through various sources, especially from the Internet, it is also important for us to consider the case where the sample size n is large, while the feature dimension may be moderate. Thus in this article, we are interested in the problem of finding a linear classifier through DWD for data instances where n and/or d are large.

In Marron, Todd, and Ahn (2007), DWD is formulated as a second-order cone programming (SOCP) problem, and the resulting model is solved by using a primal-dual interior-point method for SOCP problems implemented in the software SDPT3 (Toh, Todd, and Tutuncu 1999). However, the IPM-based solver employed for DWD in Marron, Todd, and Ahn (2007) is computationally very expensive for solving problems where n or d is large, thus making it impractical for large-scale problems. A recent approach to overcome such a computational bottleneck has appeared in Wang and Zou (2015), where the authors proposed a novel reformulation of the primal DWD model, which consists of minimizing a highly nonlinear convex objective function subject to a ball constraint. The resulting problem is solved via its dual and an MM (minimization-majorization) algorithm is designed to compute the Lagrangian dual function for each given dual multiplier. The algorithm appears to be quite promising in theory for solving large-scale DWD problems. However, the current numerical experiments and implementation of the proposed algorithm in Wang and Zou (2015) are preliminary and limited to small-scale data instances, and it appears that substantial work must be done to make it efficient for large-scale instances. Hence, it is premature to compare our proposed algorithm with the one in Wang and Zou (2015).

The main contribution of this article is to design a new method for solving large-scale DWD problems, where we

CONTACT Kim-Chuan Toh mattohkc@nus.edu.sg Department of Mathematics and Institute of Operations Research and Analytics, National University of Singapore, 10 Lower Kent Ridge Road, Singapore 119076.

Supplementary materials for this article are available online. Please go to www.tandfonline.com/r/JCGS.

target to solve a problem with the sample size $n \approx 10^4$–10^6 and/or the dimension $d \approx 10^4$–10^5. Our method is a convergent three-block semiproximal alternating direction method of multipliers (ADMM), which is designed based on the recent advances in research on convergent multi-block ADMM-type methods (Sun, Toh, and Yang 2015; Li, Sun, and Toh 2016; Chen, Sun, and Toh 2017) for solving convex composite quadratic conic programming problems.

The classical ADMM is initially proposed for solving a two-block convex optimization problem with a collection of coupling linear constraints. Over the years, there have been many variations of ADMM proposed and applied to a great variety of optimization problems. A natural modification is to extend the original ADMM from two-block to multi-block settings. However, in Chen et al. (2016), it was shown that the directly extended ADMM may not be convergent. Thus, it is necessary to make some modifications to the directly extended ADMM to get a convergent algorithm. In Sun, Toh, and Yang (2015), the authors proposed a semiproximal ADMM for solving a convex conic programming problem with three blocks of variables and four types of constraints. The algorithm is a convergent modification of the ADMM with an additional inexpensive step in each iteration. In Li, Sun, and Toh (2016), the authors proposed a Schur complement-based (SCB) convergent semiproximal ADMM for solving a multi-block linearly constrained convex programming problem whose objective function is the sum of two proper closed convex functions plus an arbitrary number of convex quadratic or linear functions. One of the key contributions in Li, Sun, and Toh (2016) is the discovery of the Schur complement-based decomposition method, which allows the multi-block subproblems to be solved efficiently while ensuring the convergence of the algorithm. More recently, Li, Sun, and Toh (2015) generalized the SCB decomposition method in Li, Sun, and Toh (2016) to the *inexact* symmetric Gauss–Seidel decomposition method, which provides an elegant framework and simpler derivation. Based on this previous research, in Chen, Sun, and Toh (2017), the authors proposed an *inexact* symmetric Gauss–Seidel-based multi-block semiproximal ADMM for solving a class of high-dimensional convex composite conic optimization problems, which has been demonstrated to have much better performance than the possibly nonconvergent directly extended ADMM in solving high-dimensional convex quadratic semidefinite programming problems.

Inspired by the above works, we propose a convergent three-block semiproximal ADMM, which is a modification of the inexact sGS-ADMM algorithm designed in Chen, Sun, and Toh (2017) to solve the DWD model. The first contribution we make is in reformulating the primal formulation of the *generalized* DWD model (using the terminology from Wang and Zou 2015) and adapting the powerful inexact sGS-ADMM framework for solving the reformulated problem. This is in contrast to numerous SVM algorithms that are primarily designed for solving the dual formulation of the SVM model. The second contribution we make is in designing highly efficient techniques to solve the subproblems in each of the inexact sGS-ADMM iterations. If n or d is moderate, then the complexity at each iteration is $O(nd) + O(n^2)$ or $O(nd) + O(d^2)$, respectively. If both n and d are large, then we employ the conjugate gradient iterative method for solving the large linear systems of equations involved. We also devise various strategies to speed up the practical performance of the sGS-ADMM algorithm in solving large-scale instances (with the largest instance having $n = 256{,}000$ and $d \approx 3 \times 10^6$) of DWD problems with real datasets from the UCI machine learning repository (Lichman 2013). We should emphasize that the key in achieving high efficiency in our algorithm depends very much on the intricate numerical techniques and sophisticated implementation we have developed.

Finally, we conduct extensive numerical experiments to evaluate the performance of our proposed algorithm against a few other alternatives. Relative to the primal-dual interior-point method used in Marron, Todd, and Ahn (2007), our algorithm is vastly superior in terms of computational time and memory usage in solving large-scale problems, where our algorithm can be a few thousands times faster. By exploiting all the highly efficient numerical techniques we have developed in the implementation of the sGS-ADMM algorithm for solving the generalized DWD problem, we can also get an efficient implementation of the possibly nonconvergent directly extended ADMM for solving the same problem. On the tested problems, our algorithm generally requires fewer iterations compared to the directly extended ADMM even when the latter is convergent. On quite a few instances, the directly extended ADMM actually requires many more iterations than our proposed algorithm to solve the problems. We also compare the efficiency of our algorithm in solving the generalized DWD problem against the highly optimized LIBLINEAR (Fan et al. 2008) and LIBSVM (Chang and Lin 2011) in solving the corresponding dual SVM problem. Surprisingly, our algorithm can even be more efficient than LIBSVM in solving large-scale problems even though the DWD model is more complex, and on some instances, our algorithm is 50–100 times faster. Our DWD model is also able to produce the best test (or generalization) errors compared to LIBLINEAR and LIBSVM among the tested instances.

The remaining parts of this article are organized as follows. In Section 2, we present the DWD formulation in full detail. In Section 3, we propose our inexact sGS-based ADMM method for solving large-scale DWD problems. We also discuss some essential computational techniques used in our implementation. We report our numerical results in Section 4. We will also compare the performance of our algorithm to other solvers on the same datasets in this particular section. Finally, we conclude the article in Section 5.

Notation. We denote the two-norm of a vector x by $\|x\|$, and the Frobenius norm of a matrix M by $\|M\|_F$. The inner product of two vectors x, y is denoted by $\langle x, y\rangle$. If S is a symmetric positive semidefinite matrix, then we denote the weighted norm of a vector x with the weight matrix S by $\|x\|_S := \sqrt{\langle x, Sx\rangle}$.

2. Generalized Distance Weighted Discrimination

This section gives details on the optimization problems underlying the distance weighted discrimination. Let (x_i, y_i), $i = 1, \ldots, n$, be the training data where $x_i \in \mathbb{R}^d$ is the feature vector and $y_i \in \{+1, -1\}$ is its corresponding class label. We let $X \in \mathbb{R}^{d\times n}$ be the matrix whose columns are the x_i's, and $y = [y_1, \ldots, y_n]^T$. In linear discrimination, we attempt to separate the vectors in the two classes by a hyperplane $H = \{x \in \mathbb{R}^d \mid w^T x + \beta = 0\}$, where $w \in \mathbb{R}^d$ is the unit normal and $|\beta|$ is its

distance to the origin. Given a point $z \in \mathbb{R}^d$, the signed distance between z and the hyperplane H is given by $w^T z + \beta$. For binary classification where the label $y_i \in \{-1, 1\}$, we want

$$y_i\left(\beta + x_i^T w\right) \geq 1 - \xi_i \quad \forall\, i = 1, \ldots, n,$$

where we have added a slack variable $\xi \geq 0$ to allow the possibility that the positive and negative data points may not be separated cleanly by the hyperplane. In matrix-vector notation, we need

$$r \;:=\; Z^T w + \beta y + \xi \;\geq\; \mathbf{1}, \tag{1}$$

where $Z = X\mathrm{diag}(y)$ and $\mathbf{1} \in \mathbb{R}^n$ is the vector of ones.

In SVM, w and β are chosen by maximizing the minimum residual, that is,

$$\max\{\delta - C\langle \mathbf{1}, \xi\rangle \mid Z^T w + \beta y + \xi \geq \delta \mathbf{1},\ \xi \geq 0,\ w^T w \leq 1\}, \tag{2}$$

where $C > 0$ is a tuning parameter to control the level of penalization on ξ. For the DWD approach introduced in Marron, Todd, and Ahn (2007), w and β are chosen instead by minimizing the sum of reciprocals of the r_i's, that is,

$$\min\left\{\sum_{i=1}^n \frac{1}{r_i} + C\langle \mathbf{1}, \xi\rangle \mid r = Z^T w + \beta y + \xi,\ r > 0,\ \xi \geq 0,\ w^T w \leq 1,\ w \in \mathbb{R}^d\right\}. \tag{3}$$

Detailed discussions on the connections between the DWD model (3) and the SVM model (2) can be found in Marron, Todd, and Ahn (2007). The DWD optimization problem (3) is shown to be equivalent to a second-order cone programming problem in Marron, Todd, and Ahn (2007) and hence it can be solved by interior-point methods such as those implemented in the solver SDPT3 (Toh, Todd, and Tutuncu 1999).

Here, we design an algorithm that is capable of solving large-scale generalized DWD problems of the following form:

$$\min\left\{\Phi(r, \xi) := \sum_{i=1}^n \theta_q(r_i) + C\langle e, \xi\rangle \mid Z^T w + \beta y + \xi - r = 0,\ \|w\| \leq 1,\ \xi \geq 0\right\}, \tag{4}$$

where $e \in \mathbb{R}^n$ is a given positive vector such that $\|e\|_\infty = 1$ (the last condition is for the purpose of normalization). The exponent q can be any given positive number, though the values of most interest are likely to be $q = 0.5, 1, 2, 4$, and $\theta_q(r_i)$ is the function defined by

$$\theta_q(t) = \frac{1}{t^q} \quad \text{if } t > 0, \text{ and } \quad \theta_q(t) = \infty \quad \text{if } t \leq 0.$$

Observe that in addition to allowing for a general exponent q in (4), we also allow for a nonuniform weight $e_i > 0$ in the penalty term for each ξ_i. By a simple change of variables and modification of the data vector y, (4) can also include the case where the terms in $\sum_{i=1}^n \frac{1}{r_i^q}$ are weighted nonuniformly. For brevity, we omit the details.

Proposition 1. Let $\kappa = \frac{q+1}{q} q^{\frac{1}{q+1}}$. The dual of problem (4) is given as follows:

$$-\min_{\alpha}\left\{\Psi(\alpha) := \|Z\alpha\| - \kappa \sum_{i=1}^n \alpha_i^{\frac{q}{q+1}} \mid 0 \leq \alpha \leq Ce,\ \langle y, \alpha\rangle = 0\right\}. \tag{5}$$

Proof. Consider the Lagrangian function associated with (4):

$$\begin{aligned} &L(r, w, \beta, \xi; \alpha, \eta, \lambda) \\ &\quad = \textstyle\sum_{i=1}^n \theta_q(r_i) + C\langle e, \xi\rangle - \langle \alpha, Z^T w + \beta y + \xi - r\rangle \\ &\qquad + \frac{\lambda}{2}(\|w\|^2 - 1) - \langle \eta, \xi\rangle \\ &\quad = \textstyle\sum_{i=1}^n \theta_q(r_i) + \langle r, \alpha\rangle + \langle \xi, Ce - \alpha - \eta\rangle - \beta\langle y, \alpha\rangle \\ &\qquad - \langle w, Z\alpha\rangle + \frac{\lambda}{2}(\langle w, w\rangle - 1), \end{aligned}$$

where $r \in \mathbb{R}^n$, $w \in \mathbb{R}^d$, $\beta \in \mathbb{R}$, $\xi \in \mathbb{R}^n$, $\alpha \in \mathbb{R}^n$, $\lambda, \eta \geq 0$. Now

$$\inf_{r_i}\{\theta_q(r_i) + \alpha_i r_i\} = \begin{cases} \kappa\, \alpha_i^{\frac{q}{q+1}} & \text{if } \alpha_i \geq 0 \\ -\infty & \text{if } \alpha_i < 0 \end{cases}$$

$$\inf_{w}\{-\langle Z\alpha, w\rangle + \frac{\lambda}{2}\|w\|^2\} = \begin{cases} -\frac{1}{2\lambda}\|Z\alpha\|^2 & \text{if } \lambda > 0 \\ 0 & \text{if } \lambda = 0, Z\alpha = 0 \\ -\infty & \text{if } \lambda = 0, Z\alpha \neq 0 \end{cases}$$

$$\inf_{\xi}\{\langle \xi, Ce - \alpha - \eta\rangle\} = \begin{cases} 0 & \text{if } Ce - \alpha - \eta = 0 \\ -\infty & \text{otherwise} \end{cases},$$

$$\inf_{\beta}\{-\beta\langle y, \alpha\rangle\} = \begin{cases} 0 & \text{if } \langle y, \alpha\rangle = 0 \\ -\infty & \text{otherwise} \end{cases}.$$

Let $F_D = \{\alpha \in \mathbb{R}^n \mid 0 \leq \alpha \leq Ce, \langle y, \alpha\rangle = 0\}$. Hence,

$$\min_{r, w, \beta, \xi} L(r, w, \beta, \xi; \alpha, \eta, \lambda) = \begin{cases} \kappa \sum_{i=1}^n \alpha_i^{\frac{q}{q+1}} & \text{if } \lambda > 0, \alpha \in F_D, \\ \quad -\frac{1}{2\lambda}\|Z\alpha\|^2 - \frac{\lambda}{2}, & \\ \kappa \sum_{i=1}^n \alpha_i^{\frac{q}{q+1}}, & \text{if } \lambda = 0, Z\alpha = 0, \alpha \in F_D, \\ -\infty, & \text{if } \lambda = 0, Z\alpha \neq 0, \alpha \in F_D, \text{ or } \alpha \notin F_D. \end{cases}$$

Now for $\alpha \in F_D$, we have

$$\max_{\lambda \geq 0, \eta \geq 0}\{\min_{r, w, \beta, \xi} L(r, w, \beta, \xi; \alpha, \eta, \lambda)\} = \kappa \sum_{i=1}^n \alpha_i^{\frac{q}{q+1}} - \|Z\alpha\|.$$

From here, we get the required dual problem. □

It is straightforward to show that the feasible regions of (4) and (5) both have nonempty interiors. Thus, optimal solutions for both problems exist and they satisfy the following KKT (Karush–Kuhn–Tucker) optimality conditions:

$$\begin{aligned} &Z^T w + \beta y + \xi - r = 0, && \langle y, \alpha\rangle = 0, \\ &r > 0,\ \alpha > 0, \quad \alpha \leq Ce, \quad \xi \geq 0, && \langle Ce - \alpha, \xi\rangle = 0, \\ &\alpha_i = \tfrac{q}{r_i^{q+1}},\ i = 1, \ldots, n, && \text{either } w = \tfrac{Z\alpha}{\|Z\alpha\|}, \text{ or} \\ & && Z\alpha = 0,\ \|w\|^2 \leq 1. \end{aligned} \tag{6}$$

Let $(r^*, \xi^*, w^*, \beta^*)$ and α^* be an optimal solution of (4) and (5), respectively. Next we analyze some properties of the optimal solution. In particular, we show that the optimal solution α^* is bounded away from 0.

Proposition 2. There exists a positive δ such that $\alpha_i^* \geq \delta \ \ \forall\, i = 1, \ldots, n$.

Proof. For convenience, let $F_P = \{(r, \xi, w, \beta) \mid Z^T w + \beta y + \xi - r = 0, \|w\| \leq 1, \xi \geq 0\}$ be the feasible region of (4). Since $(\mathbf{1}, \mathbf{1}, 0, 0) \in F_P$, we have that

$$\begin{aligned} Ce_{\min}\xi_i^* &\leq C\langle e, \xi^*\rangle \leq \Phi(r^*, \xi^*, w^*, \beta^*) \leq \Phi(\mathbf{1}, \mathbf{1}, 0, 0) \\ &= n + C\textstyle\sum_{i=1}^n e_i \quad \forall\, i = 1, \ldots, n, \end{aligned}$$

where $e_{\min} = \min_{1\leq i\leq n}\{e_i\}$. Hence, we have $0 \leq \xi^* \leq \varrho\mathbf{1}$, where $\varrho := \frac{n+C\sum_{i=1}^n e_i}{Ce_{\min}}$.

Next, we establish a bound for $|\beta^*|$. Suppose $\beta^* > 0$. Consider an index i such that $y_i = -1$. Then $0 < \beta^* = Z_i^T w^* + \xi_i^* - r_i^* \leq \|Z_i\|\|w^*\| + \xi_i^* \leq K + \varrho$, where Z_i denotes the ith column of Z, $K = \max_{1\leq j\leq n}\{\|Z_j\|\}$. On the other hand, if $\beta^* < 0$, then we consider an index k such that $y_k = 1$, and $0 < -\beta^* = Z_k^T w^* + \xi_k^* - r_k^* \leq K + \varrho$. To summarize, we have that $|\beta^*| \leq K + \varrho$.

Now we can establish an upper bound for r^*. For any $i = 1, \ldots, n$, we have that

$$r_i^* = Z_i^T w^* + \beta^* y_i + \xi_i^* \leq \|Z_i\|\|w^*\| + |\beta^*| + \xi_i^* \leq 2(K + \varrho).$$

From here, we get $\alpha_i^* = \frac{q}{(r_i^*)^{q+1}} \geq \delta := \frac{q}{(2K+2\varrho)^{q+1}} \quad \forall\, i = 1, \ldots, n$. This completes the proof of the proposition. □

3. An Inexact SGS-Based ADMM for Large-Scale DWD Problems

We can rewrite the model (4) as

$$\min\left\{\sum_{i=1}^n \theta_q(r_i) + C\langle e, \xi\rangle + \delta_B(w) + \delta_{\mathbb{R}^n_+}(\xi) \mid Z^T w + \beta y + \xi - r = 0, \quad w \in \mathbb{R}^d, \ r, \xi \in \mathbb{R}^n\right\},$$

where $B = \{w \in \mathbb{R}^d \mid \|w\| \leq 1\}$. Here, both $\delta_B(w)$ and $\delta_{\mathbb{R}^n_+}(\xi)$ are infinity indicator functions. In general, an infinity indicator function over a set $\mathcal{C}$ is defined by

$$\delta_{\mathcal{C}}(x) := \begin{cases} 0, & \text{if } x \in \mathcal{C}; \\ +\infty, & \text{otherwise.} \end{cases}$$

The model above is a convex minimization problem with three nonlinear blocks. By introducing an auxiliary variable $u = w$, we can reformulate it as

$$\begin{aligned} \min\ & \sum_{i=1}^n \theta_q(r_i) + C\langle e, \xi\rangle + \delta_B(u) + \delta_{\mathbb{R}^n_+}(\xi) \\ \text{s.t.}\ & Z^T w + \beta y + \xi - r = 0 \\ & D(w - u) = 0, \quad w, u \in \mathbb{R}^d, \ \beta \in \mathbb{R}, \ r, \xi \in \mathbb{R}^n, \end{aligned} \tag{7}$$

where $D \in \mathbb{R}^{d\times d}$ is a given positive scalar multiple of the identity matrix, which is introduced for the purpose of scaling the variables.

For a given parameter $\sigma > 0$, the augmented Lagrangian function associated with (7) is given by

$$\begin{aligned} &L_\sigma(r, w, \beta, \xi, u; \alpha, \rho) \\ &= \sum_{i=1}^n \theta_q(r_i) + C\langle e, \xi\rangle + \delta_B(u) + \delta_{\mathbb{R}^n_+}(\xi) \\ &\quad + \frac{\sigma}{2}\|Z^T w + \beta y + \xi - r - \sigma^{-1}\alpha\|^2 \\ &\quad + \frac{\sigma}{2}\|D(w - u) - \sigma^{-1}\rho\|^2 - \frac{1}{2\sigma}\|\alpha\|^2 - \frac{1}{2\sigma}\|\rho\|^2. \end{aligned}$$

The algorithm that we will design later is based on recent progress in algorithms for solving multi-block convex conic programming. In particular, our algorithm is designed based on the inexact ADMM algorithm in Chen, Sun, and Toh (2017) and we made essential use of the inexact symmetric Gauss–Seidel decomposition theorem in Li, Sun, and Toh (2016) to solve the subproblems arising in each iteration of the algorithm.

We can view (7) as a linearly constrained nonsmooth convex programming problem with three blocks of variables grouped as (w, β), r, (u, ξ). The template for our inexact sGS-based ADMM is described next. Note that the subproblems need not be solved exactly as long as they satisfy some prescribed accuracy.

Algorithm 1. An inexact sGS-ADMM for solving (7).

Let $\{\varepsilon_k\}$ be a summable sequence of nonnegative nonincreasing numbers. Given an initial iterate $(r^0, w^0, \beta^0, \xi^0, u^0)$ in the feasible region of (7), and (α^0, ρ^0) in the dual feasible region of (7), choose a $d \times d$ symmetric positive semidefinite matrix $\mathcal{T}$, and perform the following steps in each iteration.

Step 1a. Compute

$$(\bar{w}^{k+1}, \bar{\beta}^{k+1}) \approx \operatorname{argmin}_{w,\beta}\left\{L_\sigma(r^k, w, \beta, \xi^k, u^k; \alpha^k, \rho^k) + \frac{\sigma}{2}\|w - w^k\|_{\mathcal{T}}^2\right\}.$$

In particular, $(\bar{w}^{k+1}, \bar{\beta}^{k+1})$ is an approximate solution to the following $(d+1)\times(d+1)$ linear system of equations:

$$\underbrace{\begin{bmatrix} ZZ^T + D^2 + \mathcal{T} & Zy \\ (Zy)^T & y^T y \end{bmatrix}}_{A}\begin{bmatrix} w \\ \beta \end{bmatrix} = \bar{h}^k := \begin{bmatrix} -Z(\xi^k - r^k - \sigma^{-1}\alpha^k) + D^2 u^k + D(\sigma^{-1}\rho^k) + \mathcal{T} w^k \\ -y^T(\xi^k - r^k - \sigma^{-1}\alpha^k) \end{bmatrix}. \tag{8}$$

We require the residual of the approximate solution $(\bar{w}^{k+1}, \bar{\beta}^{k+1})$ to satisfy

$$\|\bar{h}^k - A[\bar{w}^{k+1}; \bar{\beta}^{k+1}]\| \leq \varepsilon_k. \tag{9}$$

Step 1b. Compute $r^{k+1} \approx \operatorname{argmin}_{r\in\mathbb{R}^n} L_\sigma(r, \bar{w}^{k+1}, \bar{\beta}^{k+1}, \xi^k, u^k; \alpha^k, \rho^k)$. Specifically, by observing that the objective function in this subproblem is actually separable in r_i for $i = 1, \ldots, n$, we can compute r_i^{k+1} as follows:

$$\begin{aligned} r_i^{k+1} &\approx \arg\min_{r_i}\left\{\theta_q(r_i) + \frac{\sigma}{2}\left\|r_i - c_i^k\right\|^2\right\} \\ &= \arg\min_{r_i>0}\left\{\frac{1}{r_i^q} + \frac{\sigma}{2}\left\|r_i - c_i^k\right\|^2\right\} \quad \forall\, i = 1, \ldots, n, \end{aligned} \tag{10}$$

where $c^k = Z^T\bar{w}^{k+1} + y\bar{\beta}^{k+1} + \xi^k - \sigma^{-1}\alpha^k$. The details on how the above one-dimensional problems are solved will be given later. The solution r_i^{k+1} is deemed to be sufficiently accurate if

$$\left| -\frac{q}{(r_i^{k+1})^{q+1}} + \sigma\left(r_i^{k+1} - c_i^k\right)\right| \le \varepsilon_k/\sqrt{n} \quad \forall\, i = 1, \ldots, n.$$

Step 1c. Compute

$$(w^{k+1}, \beta^{k+1}) \approx \operatorname{argmin}_{w,\beta}\left\{L_\sigma\left(r^{k+1}, w, \beta, \xi^k, u^k; \alpha^k, \rho^k\right) + \frac{\sigma}{2}\|w - w^k\|_{\mathcal{T}}^2\right\},$$

which amounts to solving the linear system of equations (8) but with r^k in the right-hand side vector $\bar{h}^k$ replaced by r^{k+1}. Let h^k be the new right-hand side vector. We require the approximate solution to satisfy the accuracy condition that

$$\|h^k - A[w^{k+1}; \beta^{k+1}]\| \le 5\varepsilon_k.$$

Observe that the accuracy requirement here is more relaxed than that stated in (9) of Step 1a. The reason for doing so is that one may hope to use the solution $(\bar{w}^{k+1}, \bar{\beta}^{k+1})$ computed in Step 1a as an approximate solution for the current subproblem. If $(\bar{w}^{k+1}, \bar{\beta}^{k+1})$ indeed satisfies the above accuracy condition, then one can simply set $(w^{k+1}, \beta^{k+1}) = (\bar{w}^{k+1}, \bar{\beta}^{k+1})$ and the cost of solving this new subproblem can be saved.

Step 2. Compute $(u^{k+1}, \xi^{k+1}) = \operatorname{argmin}_{u,\xi} L_\sigma(r^{k+1}, w^{k+1}, \beta^{k+1}, \xi, u; \alpha^k, \rho^k)$. By observing that the objective function is actually separable in u and ξ, we can compute u^{k+1} and ξ^{k+1} separately as follows:

$$\begin{aligned} u^{k+1} &= \arg\min\left\{\delta_B(u) + \frac{\sigma}{2}\|D(u - g^k)\|^2\right\} \\ &= \begin{cases} g^k & \text{if } \|g^k\| \le 1 \\ g^k/\|g^k\| & \text{otherwise} \end{cases}, \\ \xi^{k+1} &= \Pi_{\mathbb{R}^n_+}(r^{k+1} - Z^T w^{k+1} - y\beta^{k+1} \\ &\quad + \sigma^{-1}\alpha^k - \sigma^{-1}Ce), \end{aligned}$$

where $g^k = w^{k+1} - \sigma^{-1}D^{-1}\rho^k$, and $\Pi_{\mathbb{R}^n_+}(\cdot)$ denotes the projection onto $\mathbb{R}^n_+$.

Step 3. Compute

$$\begin{aligned} \alpha^{k+1} &= \alpha^k - \tau\sigma(Z^T w^{k+1} + y\beta^{k+1} + \xi^{k+1} - r^{k+1}), \\ \rho^{k+1} &= \rho^k - \tau\sigma D(w^{k+1} - u^{k+1}), \end{aligned}$$

where $\tau \in (0, (1+\sqrt{5})/2)$ is the steplength, which is typically chosen to be 1.618.

In our implementation of Algorithm 1, we choose the summable sequence $\{\varepsilon_k\}_{k\ge 0}$ to be $\varepsilon_k = c/(k+1)^{1.5}$ where c is a constant that is inversely proportional to $\|Z\|_F$. Next we discuss the computational cost of Algorithm 1. As we shall see later, the most computationally intensive steps in each iteration of the above algorithm are in solving the linear systems of equations of the form (8) in Steps 1a and 1c. The detailed analysis of their computational costs will be presented in Section 3.3. All the other steps can be done in at most $O(n)$ or $O(d)$ arithmetic operations, together with the computation of $Z^T w^{k+1}$, which costs $2dn$ operations if we do not take advantage of any possible sparsity in Z.

3.1. Convergence Results

We have the following convergence theorem for the inexact sGS-ADMM, established by Chen, Sun, and Toh (2017, Theorem 1). This theorem guarantees the convergence of our algorithm to optimality, as a merit over the possibly nonconvergent directly extended semiproximal ADMM.

Theorem 1. Suppose that the system (6) has at least one solution. Let $\{(r^k, w^k, \beta^k, \xi^k, u^k; \alpha^k, \rho^k)\}$ be the sequence generated by the inexact sGS-ADMM in Algorithm 1. Then the sequence $\{(r^k, w^k, \beta^k, \xi^k, u^k)\}$ converges to an optimal solution of problem (7) and the sequence $\{(\alpha^k, \rho^k)\}$ converges to an optimal solution to the dual of problem (7).

Proof. To apply the convergence result in Chen, Sun, and Toh (2017), we need to express (7) as follows:

$$\begin{aligned} \min\big\{p(r) + f(r, w, \beta) + q(\xi, u) + g(\xi, u) \mid\ & A_1^* r + A_2^*[w; \beta] \\ & + B^*[\xi; u] = 0\big\}, \end{aligned} \tag{11}$$

where

$$\begin{aligned} p(r) &= \sum_{i=1}^n \theta_q(r_i), \\ f(r, w, \beta) &\equiv 0, \\ q(\xi, u) &= \delta_B(u) + C\langle e, \xi\rangle + \delta_{\mathbb{R}^n_+}(\xi), \\ g(\xi, u) &\equiv 0, \end{aligned}$$

$$A_1^* = \begin{pmatrix} -I \\ 0 \end{pmatrix},\ A_2^* = \begin{pmatrix} Z^T & y \\ D & 0 \end{pmatrix},\ B^* = \begin{pmatrix} I & 0 \\ 0 & -D \end{pmatrix}.$$

Next we need to consider the following matrices:

$$\begin{aligned} &\begin{pmatrix} A_1 \\ A_2 \end{pmatrix}(A_1^*,\ A_2^*) + \begin{pmatrix} 0 & 0 & 0 \\ 0 & \mathcal{T} & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} I & [-Z^T, -y] \\ [-Z^T, -y]^T & M \end{pmatrix}, \quad BB^* = \begin{pmatrix} I & 0 \\ 0 & D^2 \end{pmatrix}, \end{aligned}$$

where

$$M = \begin{pmatrix} ZZ^T + D^2 + \mathcal{T} & Zy \\ (Zy)^T & y^T y \end{pmatrix} \succ 0.$$

One can show that M is positive definite by using the Schur complement lemma. With the conditions that $M \succ 0$ and $BB^* \succ 0$, the conditions in Proposition 4.2 of Chen, Sun, and Toh (2017) are satisfied, and hence the convergence of Algorithm 1 follows by using Theorem 1 in Chen, Sun, and Toh (2017). □

We note here that the convergence analysis in Chen, Sun, and Toh (2017) is highly nontrivial. But it is motivated by the proof for the simpler case of an exact semiproximal ADMM that is available in Appendix B of the article by Fazel et al. (2013). In that article, one can see that the convergence proof is based on the descent property of a certain function, while the augmented Lagrangian function itself does not have such a descent property.

3.2. Numerical Computation of the Subproblem (10) in Step 1b

In the presentation of Algorithm 1, we have described how the subproblem in each step can be solved except for the subproblem (10) in Step 1b. Now we discuss how it can be solved. Observe that for each i, we need to solve a one-dimensional problem of the form

$$\min\left\{\varphi(s) := \frac{1}{s^q} + \frac{\sigma}{2}(s-a)^2 \mid s > 0\right\}, \tag{12}$$

where a is given. It is easy to see that $\varphi(\cdot)$ is a convex function and it has a unique minimizer in the domain $(0, \infty)$. The optimality condition for (12) is given by

$$s - a = \frac{q\sigma^{-1}}{s^{q+1}},$$

where the unique minimizer s^* is determined by the intersection of the line $s \mapsto s - a$ and the curve $s \mapsto \frac{q\sigma^{-1}}{s^{q+1}}$ for $s > 0$. We propose to use Newton's method to find the minimizer, and the template is given as follows. Given an initial iterate s^0, perform the following iterations:

$$s_{k+1} = s_k - \varphi'(s_k)/\varphi''(s_k) = s_k\left(\frac{q(q+2)\sigma^{-1} + as_k^{q+1}}{q(q+1)\sigma^{-1} + s_k^{q+2}}\right), \quad k = 0, 1, \ldots$$

Since $\varphi''(s^*) > 0$, Newton's method would have a local quadratic convergence rate, and we would expect it to converge in a small number of iterations, say less than 20, if a good initial point s^0 is given. In solving the subproblem (10) in Step 1b, we always use the previous solution r_i^k as the initial point to warm-start Newton's method. If a good initial point is not available, one can use the bisection technique to find one. In our tests, this technique was however never used.

Observe that the computational cost for solving the subproblem (10) in Step 1b is $O(n)$ if Newton's method converges within a fixed number of iterations (say 20) for all $i = 1, \ldots, n$. Indeed, in our experiments, the average number of Newton iterations required to solve (12) for each of the instances is less than 10.

3.3. Efficient Techniques to Solve the Linear System (8)

Observe that in each iteration of Algorithm 1, we need to solve a $(d+1) \times (d+1)$ linear system of Equation (8) with the same coefficient matrix A. For large-scale problems where n and/or d are large, this step would constitute the most expensive part of the algorithm. To solve such a linear system efficiently, we design different techniques to solve it, depending on the dimensions n and d. We consider the following cases.

3.3.1. The Case Where $d \ll n$ and d is Moderate

This is the most straightforward case where we set $\mathcal{T} = 0$, and we solve (8) by computing the Cholesky factorization of the coefficient matrix A. The cost of computing A is $2nd^2$ arithmetic operations. Assuming that A is stored, then we can compute its Cholesky factorization at the cost of $O(d^3)$ operations, which needs only to be performed once at the very beginning of Algorithm 1. After that, whenever we need to solve the linear system (8), we compute the right-hand-side vector at the cost of 2nd operations and solve two $(d+1) \times (d+1)$ triangular systems of linear equations at the cost of $2d^2$ operations.

3.3.2. The Case Where $n \ll d$ and n is Moderate

In this case, we also set $\mathcal{T} = 0$. But solving the large $(d+1) \times (d+1)$ system of linear equations (8) requires more thought. To avoid inverting the high-dimensional matrix A directly, we make use of the Sherman–Morrison–Woodbury formula to get A^{-1} by inverting a much smaller $(n+1) \times (n+1)$ matrix as shown in the following proposition.

Proposition 3. The coefficient matrix A can be rewritten as follows:

$$A = \widehat{D} + UEU^T, \quad U = \begin{bmatrix} Z & 0 \\ y^T & \|y\| \end{bmatrix}, \quad E = \mathrm{diag}(I_n, -1), \tag{13}$$

where $\widehat{D} = \mathrm{diag}(D, \|y\|^2)$. It holds that

$$A^{-1} = \widehat{D}^{-1} - \widehat{D}^{-1}UH^{-1}U^T\widehat{D}^{-1}, \tag{14}$$

where

$$H = E^{-1} + U^T\widehat{D}^{-1}U = \begin{bmatrix} I_n + Z^TD^{-1}Z + yy^T/\|y\|^2 & y/\|y\| \\ y^T/\|y\| & 0 \end{bmatrix}. \tag{15}$$

Proof. It is easy to verify that (13) holds and we omit the details. To get (14), we only need to apply the Sherman–Morrison–Woodbury formula in Golub and Loan (1996, p. 50) to (13) and perform some simplifications. □

Note that in making use of (14) to compute $A^{-1}\bar{h}^k$, we need to find H^{-1}. A rather cost effective way to do so is to express H as follows and use the Sherman–Morrison–Woodbury formula to find its inverse:

$$H = J + \bar{y}\bar{y}^T, \quad J = \mathrm{diag}(I_n + Z^TD^{-1}Z, -1), \quad \bar{y} = [y/\|y\|;\ 1].$$

With the above expression for H, we have that

$$H^{-1} = J^{-1} - \frac{1}{1 + \bar{y}^TJ^{-1}\bar{y}}(J^{-1}\bar{y})(J^{-1}\bar{y})^T.$$

Thus to solve (8), we first compute the $n \times n$ matrix $I_n + Z^TD^{-1}Z$ in (15) at the cost of $2dn^2$ operations. Then we compute its Cholesky factorization at the cost of $O(n^3)$ operations. (Observe that even though we are solving a $(d+1) \times (d+1)$ linear system of equations for which $d \gg n$, we only need to compute the Cholesky factorization of a much smaller $n \times n$ matrix.) Also, we need to compute $J^{-1}\bar{y}$ at the cost of $O(n^2)$ operations by using the previously computed Cholesky factorization. These computations only need to be performed once at the beginning of Algorithm 1. After that, whenever we need to solve a linear system of the form (8), we can compute $\bar{h}^k$ at the cost of $2nd$ operations, and then make use of (14) to get $A^{-1}\bar{h}^k$ by solving two $n \times n$ triangular systems of linear equations at the cost of $2n^2$ operations, and performing two matrix-vector multiplications involving Z and Z^T at a total cost of $4nd$ operations. To summarize, given the Cholesky factorization of the first diagonal block of H, the cost of solving (8) via (14) is $6nd + 2n^2$ operations.

3.3.3. The Case Where d and n are Both Large

The purpose of introducing the proximal term $\frac{1}{2}\|w - w^k\|_{\mathcal{T}}^2$ in Steps 1a and 1c is to make the computation of the solutions of the subproblems easier. However, one should note that adding the proximal term typically will make the algorithm converge more slowly, and the deterioration will become worse for larger $\|\mathcal{T}\|$. Thus in practice, one would need to strike a balance between choosing a symmetric positive semidefinite matrix $\mathcal{T}$ to make the computation easier while not slowing down the algorithm by too much.

In our implementation, we first attempt to solve the subproblem in Step 1a (similarly for 1c) without adding a proximal term by setting $\mathcal{T} = 0$. In particular, we solve the linear system (8) by using a preconditioned symmetric quasi-minimal residual (PSQMR) iterative solver (Freund 1997) when both n and d are large. Basically, it is a variant of the Krylov subspace method similar to the idea in GMRES (Saad 2003). For more details on the PSQMR algorithm, the reader is referred to the Appendix. In each step of the PSQMR solver, the main cost is in performing the matrix-vector multiplication with the coefficient matrix A, which costs $4nd$ arithmetic operations. As the number of steps taken by an iterative solver to solve (8) to the required accuracy (9) is dependent on the conditioning of A, in the event that the solver requires more than 50 steps to solve (8), we would switch to adding a suitable nonzero proximal term $\mathcal{T}$ to make the subproblem in Step 1a easier to solve.

The most common and natural choice of $\mathcal{T}$ to make the subproblem in Step 1a easy to solve is to set $\mathcal{T} = \lambda_{\max} I - ZZ^T$, where $\lambda_{\max}$ denotes the largest eigenvalue of ZZ^T. In this case, the corresponding linear system (8) is very easy to solve. More precisely, for the linear system in (8), we can first compute $\bar{\beta}^{k+1}$ via the Schur complement equation in a single variable followed by computing $\bar{w}^k$ as follows:

$$\begin{aligned} \big(y^T y - (Zy)^T(\lambda_{\max} I + D)^{-1}(Zy)\big)\beta \\ = \bar{h}_{d+1}^k - (Zy)^T(\lambda_{\max} I + D)^{-1}\bar{h}_{1:d}^k, \\ \bar{w}^{k+1} = (\lambda_{\max} I + D)^{-1}(\bar{h}_{1:d}^k - (Zy)\bar{\beta}^{k+1}), \end{aligned} \tag{16}$$

where $\bar{h}_{1:d}^k$ denotes the vector extracted from the first d components of $\bar{h}^k$. In our implementation, we pick a $\mathcal{T}$, which is less conservative than the above natural choice as follows. Suppose we have computed the first ℓ largest eigenvalues of ZZ^T such that $\lambda_1 \geq \cdots \geq \lambda_{\ell-1} > \lambda_\ell$, and their corresponding orthonormal set of eigenvectors, $v_1, \ldots, v_\ell$. We pick $\mathcal{T}$ to be

$$\mathcal{T} = \lambda_\ell I + \textstyle\sum_{i=1}^{\ell-1}(\lambda_i - \lambda_\ell) v_i v_i^T - ZZ^T, \tag{17}$$

which can be proved to be positive semidefinite by using the spectral decomposition of ZZ^T. In practice, one would typically pick ℓ to be a small integer, say 10, and compute the first ℓ largest eigenvalues and their corresponding eigenvectors via variants of the Lanczos method. The most expensive step in each iteration of the Lanczos method is a matrix-vector multiplication, which requires $O(d^2)$ operations. In general, the cost of computing the first few largest eigenvalues of ZZ^T is much cheaper than that of computing the full eigenvalue decomposition. In MATLAB, such a computation can be done by using the routine `eigs`. To solve (8), we need the inverse of $ZZ^T + D + \mathcal{T}$. Fortunately, when $D = \mu I_d$, it can easily be inverted with

$$\begin{aligned} (ZZ^T + D + \mathcal{T})^{-1} = (\mu + \lambda_\ell)^{-1} I_d + \textstyle\sum_{i=1}^{\ell-1}((\mu + \lambda_i)^{-1} \\ - (\mu + \lambda_\ell)^{-1}) v_i v_i^T. \end{aligned}$$

One can then compute $\bar{\beta}^k$ and $\bar{w}^k$ as in (16) with $(\lambda_{\max} I + D)^{-1}$ replaced by the above inverse.

4. Experiments

In this section, we test the performance of our inexact sGS-ADMM method on several publicly available datasets. The numerical results presented in the subsequent subsections are obtained from a computer with processor specifications: Intel(R) Xeon(R) CPU E5-2670 @ 2.5GHz (2 processors) and 64GB of RAM, running on a 64-bit Windows Operating System.

4.1. Tuning the Penalty Parameter

In the DWD model (7), we see that it is important to make a suitable choice of the penalty parameter C. In Marron, Todd, and Ahn (2007), it was noted that a reasonable choice for the penalty parameter when the exponent $q = 1$ is a large constant divided by the square of a typical distance between the x_i's, where the typical distance, dist, is defined as the median of the pairwise Euclidean distances between classes. We found out that in a more general case, C should be inversely proportional to dist^{q+1}. On the other hand, we observed that a good choice of C also depends on the sample size n and the dimension of features d. In our numerical experiments, we empirically set the value of C to be $10^{q+1} \max\big\{1, \frac{10^{q-1}\log(n)\max\{1000, d\}^{\frac{1}{3}}}{\text{dist}^{q+1}}\big\}$, where $\log(\cdot)$ is the natural logarithm.

4.2. Scaling of Data

A technique that is very important in implementing ADMM-based methods in practice to achieve fast convergence is the data scaling technique. Empirically, we have observed that it is good to scale the matrix Z in (7) so that the magnitude of all the blocks in the equality constraint would be roughly the same. Here, we choose the scaling factor to be $Z_{\text{scale}} = \sqrt{\|X\|_F}$, where $\|\cdot\|_F$ is the Frobenius norm. Hence, the optimization model in (7) becomes

$$\begin{aligned} \min \quad & \textstyle\sum_{i=1}^n \frac{1}{r_i^q} + C\langle e, \xi\rangle + \delta_{\widetilde{B}}(\tilde{u}) + \delta_{\mathbb{R}^n_+}(\xi) \\ \text{s.t.} \quad & \widetilde{Z}^T\tilde{w} + \beta y + \xi - r = 0, \qquad r > 0, \\ & D(\tilde{w} - \tilde{u}) = 0, \quad \tilde{w}, \tilde{u} \in \mathbb{R}^d, \quad r, \xi \in \mathbb{R}^n, \end{aligned} \tag{18}$$

where $\widetilde{Z} = \frac{Z}{Z_{\text{scale}}}$, $\tilde{w} = Z_{\text{scale}} w$, $\tilde{u} = Z_{\text{scale}} u$, and $\widetilde{B} = \{\tilde{w} \in \mathbb{R}^d \mid \|\tilde{w}\| \leq Z_{\text{scale}}\}$. Therefore, if we have computed an optimal solution $(r^*, \tilde{w}^*, \beta^*, \xi^*, \tilde{u}^*)$ of (18), then $(r^*, Z_{\text{scale}}^{-1}\tilde{w}^*, \beta^*, \xi^*, Z_{\text{scale}}^{-1}\tilde{u}^*)$ would be an optimal solution of (7).

4.3. Stopping Condition for Inexact sGS-ADMM

We measure the accuracy of an approximate optimal solution $(r, w, \beta, \xi, u, \alpha, \rho)$ for (18) based on the KKT optimality conditions (6) by defining the following relative residuals:

$$\eta_{C_1} = \frac{|y^T\alpha|}{1+C}, \quad \eta_{C_2} = \frac{|\xi^T(Ce-\alpha)|}{1+C}, \quad \eta_{C_3} = \frac{\|\alpha-s\|^2}{1+C} \text{ with } s_i = \frac{q}{r_i^{q+1}},$$
$$\eta_{P_1} = \frac{\|\widetilde{Z}^T\tilde{w}+\beta y+\xi-r\|}{1+C}, \quad \eta_{P_2} = \frac{\|D(\tilde{w}-\tilde{u})\|}{1+C}, \quad \eta_{P_3} = \frac{\max\{\|\tilde{w}\|-Z_{\text{scale}},0\}}{1+C},$$
$$\eta_{D_1} = \frac{\|\min\{0,\alpha\}\|}{1+C}, \quad \eta_{D_2} = \frac{\|\max\{0,\alpha-Ce\}\|}{1+C},$$

where Z_{scale} is a scaling factor, which has been discussed in the last subsection. Additionally, we calculate the relative duality gap by

$$\eta_{\text{gap}} := \frac{|\text{obj}_{\text{primal}} - \text{obj}_{\text{dual}}|}{1 + |\text{obj}_{\text{primal}}| + |\text{obj}_{\text{dual}}|},$$

where $\text{obj}_{\text{primal}} = \sum_{i=1}^n \frac{1}{r_i^q} + C\langle e, \xi\rangle$, $\text{obj}_{\text{dual}} = \kappa \sum_{i=1}^n \alpha_i^{\frac{q}{q+1}} - Z_{\text{scale}}\|\widetilde{Z}\alpha\|$, with $\kappa = \frac{q+1}{q} q^{\frac{1}{q+1}}$. We should emphasize that although for machine learning problems, a high accuracy solution is usually not required, it is important however to use the KKT optimality conditions as the stopping criterion to find a moderately accurate solution to design a robust solver.

We terminate the solver when $\max\{\eta_P, \eta_D\} < 10^{-5}$, $\min\{\eta_C, \eta_{\text{gap}}\} < \sqrt{10^{-5}}$, and $\max\{\eta_C, \eta_{\text{gap}}\} < 0.05$. Here, $\eta_C = \max\{\eta_{C_1}, \eta_{C_2}, \eta_{C_3}\}$, $\eta_P = \max\{\eta_{P_1}, \eta_{P_2}, \eta_{P_3}\}$, and $\eta_D = \max\{\eta_{D_1}, \eta_{D_2}\}$. Furthermore, the maximum number of iterations is set to be 2000.

4.4. Adjustment of Lagrangian Parameter σ

Based upon some preliminary experiments, we set our initial Lagrangian parameter σ to be $\sigma_0 = \min\{10C, n\}^q$, where q is the exponent in (7), and adapt the following strategy to update σ to improve the convergence speed of the algorithm in practice:

Step 1. Set $\chi = \frac{\eta_P}{\eta_D}$, where η_P and η_D are defined in Section 4.3;

Step 2. If $\chi > \theta$, set $\sigma_{k+1} = \zeta\sigma_k$; elseif $\frac{1}{\chi} > \theta$, set $\sigma_{k+1} = \frac{1}{\zeta}\sigma_k$.

Here, we empirically set θ to be 5 and ζ to be 1.1. Nevertheless, if we have either $\eta_P \ll \eta_D$ or $\eta_D \ll \eta_P$, then we would increase ζ accordingly, say 2.2 if $\max\{\chi, \frac{1}{\chi}\} > 500$ or 1.65 if $\max\{\chi, \frac{1}{\chi}\} > 50$.

4.5. Performance of the sGS-ADMM on UCI Datasets

In this subsection, we test our algorithm on instances from the UCI data repository (Lichman 2013). The datasets we have chosen here are all classification problems with two classes. However, the size for each class may not be balanced. To tackle the case of uneven class proportions, we use the weighted DWD model discussed by Qiao et al. (2010). Specifically, we consider the model (4) using $e = \mathbf{1}$ and the term $\sum_{i=1}^n 1/r_i^q$ is replaced by $\sum_{i=1}^n \tau_i^q/r_i^q$, with the weights τ_i given as

$$\tau_i = \begin{cases} \frac{\tau_-}{\max\{\tau_+,\tau_-\}} & \text{if } y_i = +1 \\ \frac{\tau_+}{\max\{\tau_+,\tau_-\}} & \text{if } y_i = -1 \end{cases},$$

where $\tau_\pm = (|n_\pm|K^{-1})^{\frac{1}{1+q}}$. Here, $n_\pm$ is the number of data points with class label ± 1, respectively, and $K := n/\log(n)$ is a normalizing factor.

Table 1 presents the number of iterations and runtime required, as well as training error produced when we perform our inexact sGS-ADMM algorithm to solve 16 datasets. Here, the running time is the total time spent in reading the training data and in solving the DWD model. The timing for getting the best penalty parameter C is excluded. The results are generated using the exponent $q = 1$. In the table, "psqmr" is the iteration count for the preconditioned symmetric quasi-minimal residual method for solving the linear system (8). A "0" for "psqmr" means that we are using a direct solver as mentioned in Sections 3.3.1 and 3.3.2. Under the column "double" in Table 1, we also record the number of iterations for which the extra Step 1c is executed to ensure the convergence of Algorithm 1.

Denote the index set $S = \{i \mid y_i[\text{sgn}(\beta + x_i^T w)] \le 0, i = 1, \ldots, n\}$ for which the data instances are categorized wrongly, where $\text{sgn}(x)$ is the sign function. The training and testing errors are both defined by $\frac{|S|}{n} \times 100\%$, where $|S|$ is the cardinality of the set S.

Our algorithm is capable of solving all the datasets, even when the size of the data matrix is huge. In addition, for data with unbalanced class size, such as w7a and w8a, our algorithm is able to produce a classifier with small training error.

Table 1. The performance of our inexact sGS-ADMM method on the UCI datasets.

Data	n	d	C	Iter	Time (s)	psqmr|double	Train-error (%)
a8a	22,696	123	6.27e + 02	201	2.28	0|201	15.10
a9a	32,561	123	6.49e + 02	201	2.31	0|201	14.93
covtype	581,012	54	3.13e + 03	643	104.34	0|191	23.74
gisette	6000	4972	1.15e + 04	101	39.69	0| 49	0.17
gisette-scale	6000	4956	1.00e + 02	201	59.73	0|201	0.00
ijcnn1	35,000	22	4.23e + 03	401	3.16	0|401	7.77
mushrooms	8124	112	3.75e + 02	81	1.09	0| 81	0.00
real-sim	72,309	20958	1.55e + 04	210	47.69	875|210	1.45
w7a	24,692	300	5.95e + 02	701	4.84	0|701	1.17
w8a	49,749	300	6.36e + 02	906	9.43	0|906	1.20
rcv1	20,242	44505	9.18e + 03	81	9.18	234| 49	0.63
leu	38	7129	1.00e + 02	489	2.84	0|489	0.00
prostate	102	6033	1.00e + 02	81	3.19	0| 81	0.00
farm-ads	4143	54877	3.50e + 03	81	6.92	792| 81	0.14
dorothea	800	88120	1.00e + 02	51	4.44	0| 51	0.00
url-svm	256,000	685896	1.18e + 06	121	294.55	364|121	0.01

4.6. Comparison with Other Solvers

In this subsection, we compare our inexact sGS-ADMM method for solving (4) via (7) with the primal-dual interior-point method implemented by Toh, Todd, and Tutuncu (1999) and used by Marron, Todd, and Ahn (2007). We also compare our method with the directly extended (semiproximal) ADMM (using the aggressive step-length 1.618) even though the latter's convergence is not guaranteed. Note that the directly extended ADMM we have implemented here follows exactly the same design used for sGS-ADMM, except that we switch off the additional Step 1c in the Algorithm 1. We should emphasize that our directly extended ADMM is not a simple adaption of the classical ADMM, but instead incorporates all the sophisticated techniques we have developed for sGS-ADMM.

We will report our computational results for two different values of the exponent, $q = 1$ and $q = 2$, in Tables 2 and 3, respectively.

Table 2 reports the runtime, number of iterations required as well as the training error of three different solvers for solving the UCI datasets. We can observe that the interior point method is almost always the slowest to achieve optimality compared to the other two solvers, despite requiring the least number of iterations, especially when the sample size n is large. The inefficiency of the interior-point method is caused by its need to solve an $n \times n$ linear system of equations in each iteration, which could be very expensive if n is large. In addition, it cannot solve the DWD problem where n is huge due to the excessive computer memory needed to store the large $n \times n$ matrix.

On the other hand, our inexact sGS-ADMM method outperforms the directly extended (semiproximal) ADMM for 9 out of 16 cases in terms of runtime. For the other cases, we are only slower by a relatively small margin. Furthermore, when our algorithm outperforms the directly extended ADMM, it often shortens the runtime by a large margin. In terms of number of iterations, for 14 out of 16 cases, the directly extended ADMM requires at least the same number of iterations as our inexact sGS-ADMM method. We can say that our algorithm is remarkably efficient and it further possesses a convergence guarantee. In contrast, the directly extended ADMM is not guaranteed to converge although it is also very efficient when it does converge. We can observe that the directly extended ADMM sometimes would take many more iterations to solve a problem compared to our inexact sGS-ADMM, especially for the instances in Table 3, possibly because the lack of a convergence guarantee makes it difficult for the method to find a sufficiently accurate approximate optimal solution.

To summarize, our inexact sGS-ADMM method is an efficient yet convergent algorithm for solving the primal form of the DWD model. It is also able to solve large-scale problems, which cannot be handled by the interior point method.

Table 3 reports the runtime, number of iterations required as well as the training error of three different solvers for solving the UCI datasets for the case when $q = 2$. Again, we can see that the interior point method is almost always the slowest to converge to optimality.

Our sGS-ADMM algorithm outperforms the directly extended ADMM algorithm in 12 out of 16 datasets in terms of runtime. In terms of the number of iterations, it has the best performance among almost all the datasets. On the other hand, for eight datasets, the number of iterations required by the directly extended ADMM hits the maximum iterations allowed, probably implying nonconvergence of the method. For the interior point method, it takes an even longer time to solve the problems compared to the case when $q = 1$. This is due to an increase in the number of constraints generated in the second-order cone programming formulation of the DWD model with $q = 2$.

The numerical result we obtained in this case is consistent with the one we obtained for the case $q = 1$. This further shows the merit of our algorithm in a more general setting. We could also expect the similar result when the exponent is 4 or 8.

4.7. Comparison with LIBSVM and LIBLINEAR

In this subsection, we will compare the performance of our DWD model to the state-of-the-art model support vector machine (SVM). We apply our sGS-ADMM algorithm as the DWD model and use LIBSVM in Chang and Lin (2011) as

Table 2. Comparison between the performance of our inexact sGS-ADMM, directly extended ADMM "directADMM," and the interior point method "IPM" on the UCI datasets. A "*" next to the error in the table means that the problem set cannot be solved properly by the respective solver; "-" means the algorithm cannot solve the dataset due to insufficient computer memory.

exponent $q = 1$				sGS-ADMM			directADMM			IPM		
Data	n	d	C	Time (s)	Iter	Error (%)	Time (s)	Iter	Error (%)	Time (s)	Iter	Error (%)
a8a	22,696	123	6.27e + 02	2.28	201	15.10	2.01	219	15.10	1321.20	47	15.09
a9a	32,561	123	6.49e + 02	2.31	201	14.93	2.12	258	14.93	2992.60	43	14.93
covtype	581,012	54	3.13e + 03	104.34	643	23.74	100.26	700	23.74	—	—	—
gisette	6000	4972	1.15e + 04	39.69	101	0.17	33.14	70	0.25	2403.40	62	20.50*
gisette-scale	6000	4956	1.00e + 02	59.73	201	0.00	152.05	2000	0.00	49800.58	84	17.80*
ijcnn1	35,000	22	4.23e + 03	3.16	401	7.77	2.95	501	7.77	1679.34	34	7.77
mushrooms	8124	112	3.75e + 02	1.09	81	0.00	1.42	320	0.00	136.67	50	0.00
real-sim	72,309	20,958	1.55e + 04	47.69	210	1.45	89.55	702	1.42	—	—	—
w7a	24,692	300	5.95e + 02	4.84	701	1.17	8.52	2000	1.16	4474.13	53	1.17
w8a	49,749	300	6.36e + 02	9.43	906	1.20	13.58	2000	1.16	—	—	—
rcv1	20,242	44,505	9.18e + 03	9.18	81	0.63	16.07	245	0.63	9366.41	43	0.17
leu	38	7129	1.00e + 02	2.84	489	0.00	5.35	2000	0.00	1.33	11	0.00
prostate	102	6033	1.00e + 02	3.19	81	0.00	4.13	2000	0.00	7.73	19	0.00
farm-ads	4143	54,877	3.50e + 03	6.92	81	0.14	4.20	61	0.14	642.79	54	0.24
dorothea	800	88,120	1.00e + 02	4.44	51	0.00	37.31	2000	0.00	15.53	23	0.00
url-svm	256,000	685,896	1.18e + 06	294.55	121	0.01	264.52	195	0.00	—	—	—

Table 3. Same as Table 2 but for $q = 2$.

Exponent $q = 2$				sGS-ADMM			directADMM			IPM		
Data	n	d	C	Time (s)	Iter	Error (%)	Time (s)	Iter	Error (%)	Time (s)	Iter	Error (%)
a8a	22,696	123	1.57e + 04	2.77	248	15.10	2.97	533	15.11	7364.34	45	15.17
a9a	32,561	123	1.62e + 04	2.92	279	14.94	5.78	1197	14.94	16402.05	48	14.95
covtype	581,012	54	1.52e + 05	81.63	368	23.74	275.29	2000	23.74	—	—	—
gisette	6000	4972	1.01e + 06	39.72	101	0.03	28.42	81	0.03	2193.74	55	0.00
gisette-scale	6000	4956	1.00e + 03	88.89	439	0.00	147.80	2000	1.20	31827.33	53	0.00
ijcnn1	35,000	22	2.69e + 05	2.76	233	7.94	4.80	1056	7.87	1959.68	38	7.98
mushrooms	8124	112	7.66e + 03	1.59	301	0.00	3.63	2000	0.17	594.33	52	0.00
real-sim	72,309	20,958	1.10e + 06	43.20	180	1.51	23.56	181	1.51	—	—	—
w7a	24,692	300	1.44e + 04	3.96	473	1.15	7.83	2000	2.69	5448.78	48	1.18
w8a	49,749	300	1.54e + 04	7.07	543	1.13	13.57	2000	2.68	—	—	—
rcv1	20,242	44,505	4.69e + 05	9.47	81	1.16	7.54	101	1.16	8562.76	47	0.24*
leu	38	7129	1.00e + 03	1.72	204	0.00	4.52	2000	0.00	2.82	23	0.00
prostate	102	6033	1.00e + 03	3.32	111	0.00	3.87	2000	0.00	8.33	21	0.00
farm-ads	4143	54,877	5.21e + 04	37.79	401	0.19	8.31	146	0.19	343.13	43	0.27
dorothea	800	88,120	1.00e + 03	4.39	51	0.00	31.49	2000	0.00	16.84	25	0.00
url-svm	256,000	685,896	5.49e + 07	452.81	205	0.02	587.46	490	0.02	—	—	—

well as LIBLINEAR in Fan et al. (2008) to implement the SVM model. LIBSVM is a general solver for solving SVM models with different kernels; while LIBLINEAR is a solver highly specialized in solving SVM with linear kernels. LIBLINEAR is a fast linear classifier; in particular, we would apply it to the dual of L2-regularized L1-loss support vector classification problem. We would like to emphasize that the solution given by LIBSVM using linear kernel and that given by LIBLINEAR is not exactly the same. This may be due to the reason that LIBLINEAR has internally preprocessed the data and assumes that there is no bias term in the model.

The parameters used in LIBSVM are chosen to be the same as in Ito, Takeda, and Toh (2015), whereas for LIBLINEAR, we make use of the default parameter $C = 1$ for all the datasets.

Table 4 shows the runtime and number of iterations needed for solving the binary classification problem via the DWD and SVM models, respectively, on the UCI datasets. It also gives the training and testing classification error produced by the three algorithms. Note that the training time excludes the time for computing the best penalty parameter C. The stopping tolerance for all algorithms is set to be 10^{-5}. For LIBLINEAR, we observed that the default maximum number of iteration is breached for many datasets. Thus, we increase the maximum number of iteration from the default 1000 to 20,000.

In terms of runtime, LIBLINEAR is almost always the fastest to solve the problem, except for the two largest datasets (rcv1,url-svm) for which our algorithm is about two to three times faster. Note that the maximum iteration is reached for these two datasets. On the other hand, LIBSVM is almost always the slowest solver. It may only be faster than sGS-ADMM for small datasests (three cases). Our algorithm can be 50–100 times faster than LIBSVM when solving large data instances. Furthermore, LIBSVM may have the risk of not being able to handle extremely large-scaled datasets. For example, it cannot solve the biggest dataset (url-svm) within 24 hr.

In terms of training and testing error, we may observe from the table that the DWD and SVM models produced comparable training classification errors, although there are some discrepancies due to the differences in the models and penalty parameters used. On the other hand, the testing errors vary across different solvers. For most datasets, the DWD model (solved by sGS-ADMM) produced smaller (sometimes much

Table 4. Comparison between the performance of our inexact sGS-ADMM on DWD model with LIBLINEAR and LIBSVM on SVM model. n_{test} is the size of testing sample, Err_{tr} is the percentage of training error, while Err_{test} is that of the testing error. "-" means the result cannot be obtained within 24 hr, and "/" means test sets are not available.

				DWD via sGS-ADMM				SVM via LIBLINEAR				SVM via LIBSVM			
Data	n	d	n_{test}	Time	Iter	Err_{tr}	Err_{test}	Time	Iter	Err_{tr}	Err_{test}	Time	Iter	Err_{tr}	Err_{test}
a8a	22,696	123	9865	2.28	201	15.10	14.67	0.69	20,000	15.32	14.87	50.45	34,811	15.44	14.80
a9a	32,561	123	16,281	2.31	201	14.93	15.19	0.79	6475	15.01	15.02	93.91	25,721	15.24	15.03
covtype	581,012	54	/	104.34	643	23.74	/	23.53	20,000	23.69	/	19641.19	224,517	23.70	/
gisette	6000	4972	1000	39.69	101	0.17	3.00	4.34	132	0.00	10.70	85.93	14,818	0.40	14.30
gisette-scale	6000	4956	1000	59.73	201	0.00	2.50	33.07	484	0.00	2.30	152.27	15,738	0.40	2.20
ijcnn1	35,000	22	91,701	3.16	401	7.77	7.82	0.55	11,891	8.70	8.35	27.04	10,137	9.21	8.76
mushrooms	8124	112	/	1.09	81	0.00	/	0.19	326	0.00	/	0.64	1003	0.00	/
real-sim	72,309	20,958	/	47.69	210	1.45	/	7.56	1190	1.07	/	3846.07	52,591	1.37	/
w7a	24,692	300	25,057	4.84	701	1.17	1.30	0.43	1689	1.28	10.12	14.79	62,830	1.37	1.38
w8a	49,749	300	14,951	9.43	906	1.20	1.31	2.45	13,095	1.18	9.64	63.27	124,373	1.36	1.43
rcv1	20,242	44,505	677,399	9.18	81	0.63	5.12	33.98	20,000	0.15	5.17	819.32	33,517	0.62	13.82
leu	38	7129	34	2.84	489	0.00	5.88	0.30	27	0.00	20.59	0.59	184	0.00	17.65
prostate	102	6033	/	3.19	81	0.00	/	2.73	353	0.00	/	2.54	1063	0.00	/
farm-ads	4143	54,877	/	6.92	81	0.14	/	6.11	5364	0.14	/	10.34	15,273	0.14	/
dorothea	800	88,120	350	4.44	51	0.00	5.14	0.83	11	0.00	8.86	7.48	4553	0.00	7.71
url-svm	256,000	685,896	/	294.55	121	0.01	/	660.39	20,000	0.01	/	—	—	—	—

smaller) testing errors than the other algorithms (eight cases); whereas the SVM model (solved by LIBLINEAR) produced the worst testing errors among all algorithms (five cases). The discrepancy between the testing errors given by LIBSVM and LIBLINEAR may be due to the different treatment of the bias term in-built into the algorithms.

It is reasonable to claim that our algorithm is more efficient than the extremely fast solver LIBLINEAR in solving large data instances even though our algorithm is designed for the more complex DWD model compared to the simpler SVM model. Moreover, our algorithm for solving the DWD model is able to produce testing errors, which are generally better than those produced by LIBLINEAR for solving the SVM model.

5. Conclusion

In this article, by making use of the recent advances in ADMM from the work in Sun, Toh, and Yang (2015), Li, Sun, and Toh (2016), and Chen, Sun, and Toh (2017), we proposed a convergent three-block inexact symmetric Gauss-Seidel-based semiproximal ADMM algorithm for solving large-scale DWD problems. We applied the algorithm successfully to the primal formulation of the DWD model and designed highly efficient routines to solve the subproblems arising in each of the inexact sGS-ADMM iterations. Numerical experiments for the cases when the exponent equals to 1 and 2 demonstrated that our algorithm is capable of solving large-scale problems, even when the sample size and/or the feature dimension is huge. In addition, it is also highly efficient while guaranteeing the convergence to optimality. As a conclusion, we have designed an efficient method for solving the binary classification model through DWD.

Appendix: The PSQMR Algorithm

PSQMR algorithm: In the following we shall present a brief version of the PSQMR method to solve the linear system $Ax = b$. The following is a simplified algorithm without preconditioner.

Given initial iterate x_0, define each of the following: $r_0 = b - Ax_0$; $q_0 = r_0$; $\tau_0 = \|q_0\|$; $\theta_0 = 0$; $d_0 = 0$.

Then for each iteration k until convergent condition is met, compute

$$
\begin{aligned}
r_k &= r_{k-1} - \frac{r_{k-1}^T r_{k-1}}{q_{k-1}^T A q_k} A q_k \\
\theta_k &= \frac{\|r_k\|}{\tau_{k-1}} \\
c_k &= \frac{1}{\sqrt{1+\theta_k^2}} \\
\tau_k &= \tau_{k-1}\theta_k c_k \\
d_k &= c_k^2 \theta_{k-1}^2 d_{k-1} + c_k^2 \frac{r_{k-1}^T r_{k-1}}{q_{k-1}^T A q_k} A q_k \\
x_k &= x_{k-1} + d_k \\
q_k &= r_k + \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}} q_{k-1}.
\end{aligned}
$$

The final x_k obtained is an approximated solution to the linear system.

Supplementary Materials

Matlab package: MATLAB-package containing codes to perform the sGS-ADMM algorithm for the DWD model described in the article. The package also contains datasets used for experiment purpose in the article. It could be downloaded from *http://www.math.nus.edu.sg/~mattohkc/DWDLarge.zip* or accessed on the publisher's website.

R package: R-package containing codes to perform the sGS-ADMM algorithm for the DWD model described in the article. It would be available at CRAN.

ORCID

J. S. Marron http://orcid.org/0000-0003-2000-1476

References

Benito, M., Parker, J., Du, Q., Wu, J., Xiang, D., Perou, C. M., and Marron, J. S. (2004), "Adjustment of Systematic Microarray Data Biases," *Bioinformatics*, 20, 105–114. [368]

Chang, C.-C., and Lin, C.-J. (2011), "LIBSVM: A Library for Support Vector Machines," *ACM Transactions on Intelligent Systems and Technology*, 2, 27 p. [369,376]

Chen, C., He, B., Ye, Y., and Yuan, X. (2016), "The Direct Extension of ADMM for Multi-Block Convex Minimization Problems is Not Necessarily Convergent," *Mathematical Programming*, 155, 57–79. [369]

Chen, L., Sun, D. F., and Toh, K. C. (2017), "An Efficient Inexact Symmetric Gauss-Seidel Based Majorized ADMM for High-Dimensional Convex Composite Conic Programming," *Mathematical Programming*, 161, 237–270. [369,371,372,378]

Fan, P.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008), "LIBLINEAR: A Library for Large Linear Classification," *Journal of Machine Learning Research*, 9, 1871–1874. [369,377]

Fazel, M., Pong, T. K., Sun, D. F., and Tseng, P. (2013), "Hankel Matrix Rank Minimization With Applications to System Identification and Realization," *SIAM Journal of Matrix Analysis and Applications*, 34, 946–977. [372]

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014), "Do We Need Hundreds of Classifiers to Solve Real World Classification Problems?" *Journal of Machine Learning Research*, 15, 3133–3181. [368]

Freund, R. W. (1997), "Preconditioning of Symmetric, But Highly Indefinite Linear Systems," in *Proceedings of 15th IMACS World Congress on Scientific Computation Modelling and Applied Mathematics*, Germany: Berlin, pp. 551–556. [374]

Golub, G. H., and Loan, C. F. V. (1996), *Matix Computation* (3rd ed.), Baltimore, MD: John Hopkins University Press. [373]

Ito, N., Takeda, A., and Toh, K. C. (2017), "A Fast United Classification Algorithm Based on Accelerated Proximal Gradient Method," *Journal of Machine Learning Research*, 18, 1–49. [377]

Li, X. D., Sun, D. F., and Toh, K. C. (2015), "Qsdpnal: A Two-Phase Newton-CG Proximal Augmented Lagrangian Method for Convex Quadratic Semidefinite Programming Problems," arXiv:1512.08872. [369]

——— (2016), "A Schur Complement Based Semi-Proximal ADMM for Convex Quadratic Conic Programming and Extensions," *Mathematical Programming*, 155, 333–373. [369,371,378]

Lichman, M. (2013), *UCI Machine Learning Repository*, School of Information and Computer Sciences, University of California, Irvine. Available at *http://archive.ics.uci.edu/ml* [369,375]

Liu, X., Parker, J., Fan, C., Perou, C. M., and Marron, J. S. (2009), "Visualization of Cross-Platform Microarray Normalization," in *Batch Effects and Noise in Microarray Experiments: Sources and Solutions*, ed. A. Scherer, Chichester, UK: Wiley, pp. 167–181. [368]

Marron, J. S., and Alonso, A. M. (2014), "Overview of Object Oriented Data Analysis," *Biometrical Journal*, 56, 732–753. [368]

Marron, J. S., Todd, M. J., and Ahn, J. (2007), "Distance Weighted Discrimination," *Journal of the American Statistical Association*, 102, 1267–1271. [368,369,370,374,376]

Qiao, X., Zhang, H. H., Liu, Y., Todd, M. J., and Marron, J. S. (2010), "Weighted Distance Weighted Discrimination and Its Asymptotic Properties," *Journal of the American Statistical Association*, 105, 401–414. [375]

Saad, Y. (2003), *Iterative Methods for Sparse Linear Systems* (2nd ed.), Philadelphia, PA: Society for Industrial and Applied Mathematics. [374]

Sun, D. F., Toh, K. C., and Yang, L. Q. (2015), "A Convergent 3-Block Semi-Proximal Alternating Direction Method of Multipliers for Conic Programming With 4-Type Constraints," *SIAM Journal on Optimization*, 25, 882–915. [369,378]

Toh, K. C., Todd, M. J., and Tutuncu, R. H. (1999), "SDPT3 – A Matlab Software Package for Semidefinite Programming," *Optimization Methods and Software*, 11, 545–581. [368,370,376]

Vapnik, V. (1995), *The Nature of Statistical Learning Theory*, New York: Springer-Verlag. [368]

Wang, B., and Zou, H. (2018), "Another Look at Distance-weighted Discrimination," *Journal of the Royal Statistical Society*, Series B, 80, 177–198. [368,369]

Wei, S., Lee, C., Wichers, L., and Marron, J. S. (2016), "Direction-Projection-Permutation for High Dimensional Hypothesis Tests," *Journal of Computational and Graphical Statistics*, 25, 549–569. [368]