A Unified Algorithm for Fitting Penalized Models with High Dimensional Data

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Yi Yang

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Advisor: Hui Zou

September, 2013

Acknowledgements

Writing this Ph.D. thesis would not have been possible without the valuable support of quite a number of persons. First of all I thank my advisor Hui Zou for his immense support, the inspirational meetings and discussions.

Thanks to my colleagues Teng Zhang, Wei Qian, Qing Mai, Yuwen Gu, Zhihua Su and Ying Nan. I very much enjoy working at the department. I am especially grateful to Teng Zhang for giving me tremendous amount of help. I am grateful to the committee members of my thesis, consisting of Yuhong Yang, Glen Meeden and Vipin Kumar, for investing their time to read and evaluate my Ph.D.thesis.

I want to thank my parents, Ling Bai and Xinmin Yang, for supporting my interest in statistics and always encouraging me to do what makes me happy (even though they probably still do not understand how hard science can make anyone happy).

Dedication

To my parents who nursing me with affections and love and their dedicated partnership for success in my life

Abstract

In the light of high dimensional problems, research on the penalized model has received much interest. Correspondingly, several algorithms have been developed for solving penalized high dimensional models. In this thesis, we propose fast and efficient unified algorithms for computing the solution path for a collection of penalized models. In particular, we study the algorithm for solving ℓ_1 penalized learning problems and the algorithm for solving group-lasso learning problems. These algorithm take advantage of a majorization-minimization trick to make each update simple and efficient. The algorithms also enjoy a proven convergence property. To demonstrate the generality of our algorithms, we further extend these algorithms on a class of elastic net penalized large margin classification methods and the elastic net penalized Cox's proportional hazards model. These algorithms have been implemented in three R packages `gglasso`, `gcdnet` and `fastcox`, which are publicly available from the Comprehensive R Archive Network (CRAN) at `http://cran.r-project.org/web/packages`. On simulated and real data, our algorithms consistently outperform the existing software in speed for computing penalized models and often delivers better quality solutions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

With the advances in modern technology, high-dimensional data frequently appear in fields such as medical and biological sciences, finances and economics, etc. The maximum partial likelihood estimator does not work well in the presence of high-dimensional covariates. Sparse penalized model offer a way to conduct continuous subset selection while address the overfitting issue for high dimensional problems in which the number of variables is much larger than the number of observations. In sparse penalized models many variables with zero coefficient estimates are automatically deleted, while nonzero coefficients left indicate the important variables. With sparsity, model selection can improve estimation accuracy as well as model interpretability.

There has been great amount of research work contributed in sparse penalized models. In some special models, solution path is piece-wise linear, for example: Least Angle Regression (LARS) for the lasso [1]. LARS-EN for the elastic net [2]. LARS-SVM for ℓ_1-penalized SVM (Zhu *et al.*, 2003). LARS for piecewise quadratic Loss+piecewise linear penalty (or vice versa) [3]. For some other important models, their solution paths are curved. Important examples include: ℓ_1-penalized logistic regression/Cox's regression.

Algorithms were also developed for computing curved ℓ_1 solution paths: [4] and [5] use the connection between Boosting algorithm and the lasso method. [6] proposed the predictor-corrector method of convex-optimization for computing solution path. [7] proposes a generalized Lars algorithm with the loss function approximated by a

quadratic spline.

Coordinate-wised optimization method works extremely well in solving penalized models. [8] proposed the shooting algorithm. [9] extended shooting for the lasso penalized logistic regression. [10] developed coordinate descent algorithms for the lasso penalized regression, also [11] [12] [13] developed the elastic net penalized GLMs where they used CD loop within a Newton-Ralphson loop for efficiently solving the solution path. The computational efficiency of coordinate descent is due to the fast speed to carry out a simple update on each coordinate; and taking advantage of the sparsity - many of the coefficients remains zero thus the algorithm can skip them and focus on updating the non-zeros, which saves tremendous computing time.

Here we propose a new unified algorithm called coordinate majorization descent (CMD) for computing the solution path for a collection of penalized models. The CMD has the advantages of CD. In addition, the CMD at some points resembles MM algorithms [14][15]. Thus CMD shares the virtue of MM algorithm [16]: (a) providing the convergence of CMD by the descent property of MM algorithms [17]. (b) avoiding large matrix inversions like one did in Newton-Raphson methods. (c) turning a non-differentiable problem into a smooth problem. Given certain conditions hold, it always guarantees convergence to the global solution of convex problems [18] and a local solution of non-convex problems. Most importantly it works for much larger class of penalized models. The possible loss functions can be the following:

- Least Squares
- Logistic regression
- Cox proportional hazards partial likelihood
- Squared SVM
- Huber loss for classification [19]

The penalty function in the model can be one of the following:

- Lasso [20] / Adaptive lasso [21]
- Elastic net [2] / Generalized elastic net [22]
- Grouped lasso [23]

1.2 A short introduction to sparse penalized models

The inputs we have are:

- A training data $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\intercal$, $\mathbf{y} = (y_1, \ldots, y_n)^\intercal$, where $\mathbf{x}_i \in \mathbb{R}^p$ and for regression $y_i \in \mathbb{R}$, for two-class classification $y \in \{\pm 1\}$.
- The design matrix $\mathbf{X} = (x_{ij})$ is standardized so that each column has mean 0 and variance 1.
- A convex nonnegative loss functional $L : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$.
- A nonnegative penalty functional $P : \mathbb{R}^p \to \mathbb{R}$, with $p(0) = 0$.

Consider estimating a sparse vector of coefficients $\boldsymbol{\beta}$ based on training data, through penalized empirical loss minimization

$$\hat{\boldsymbol{\beta}}(\lambda) = \underset{\boldsymbol{\beta} \in \mathbb{R}^p}{\operatorname{argmin}} \left[L(\mathbf{y}, \mathbf{X}\boldsymbol{\beta}) + \lambda P(\boldsymbol{\beta}) \right] \tag{1.1}$$

where $\lambda > 0$ is the regularization parameter; $\lambda = 0$ corresponds to no regularization and $\lim_{\lambda \to \infty} \hat{\boldsymbol{\beta}}(\lambda) = 0$. One often need to compute the solution at a fine grid of λ's in order to pick a data-driven optimal λ for fitting a 'best' final model.

$P(\boldsymbol{\beta})$ is a sparsity-inducing penalty to produce a sparse estimator, which is especially preferred when $p \gg n$. Some widely used regularization methods include the lasso, the elastic net and the grouped lasso penalty.

The lasso [20] is a very popular technique for high-dimensional modeling

$$\lambda P(\boldsymbol{\beta}) = \lambda ||\boldsymbol{\beta}||_1.$$

Lasso yields sparse estimates of $\boldsymbol{\beta}$ because it shrinks small lease squares estimates $\hat{\beta}_j^{ols}$'s toward exact zero. [24] proposed the elastic net penalty as an improved variant of the lasso for high-dimensional data when predictors are highly correlated. It connects the lasso penalty and the ridge penalty

$$\lambda P(\boldsymbol{\beta}) = \lambda ||\boldsymbol{\beta}||_1 + \frac{1}{2}\lambda_2 ||\boldsymbol{\beta}||_2 \ (\lambda_2 > 0).$$

The group-lasso [23] is a generalization of the lasso for doing group-wise variable selection. [23] motivated the group-wise variable selection problem. In such cases, one often

hope that a group of predictors should either be all included or excluded. Suppose that there are K groups and $\boldsymbol{\beta}^{(k)}$ denotes the coefficients of X_j's in kth group. Then the group lasso penalty is

$$\lambda P(\boldsymbol{\beta}) = \lambda \sum_{k=1}^{K} \|\boldsymbol{\beta}^{(k)}\|_2.$$

Many "modern" machine learning methods can be cast in the framework of penalized optimization [3]. In penalized regression problems, the loss function takes the form

$$L(\mathbf{y}, \mathbf{X}\boldsymbol{\beta}) = \sum_i l(y_i - \mathbf{x}_i^\intercal \boldsymbol{\beta})$$

where the residuals $y_i - \mathbf{x}_i^\intercal \boldsymbol{\beta}$ quantifies the discrepancy between an observation y_i and a linear predictor $\mathbf{x}_i^\intercal \boldsymbol{\beta}$. An example is the lasso penalized least squares:

$$\text{Least Squares :} \qquad \hat{\boldsymbol{\beta}}(\lambda) = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left[\frac{1}{2n} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\intercal \boldsymbol{\beta}) + \lambda ||\boldsymbol{\beta}||_1 \right]$$

In classification problems,

$$L(\mathbf{y}, \mathbf{X}\boldsymbol{\beta}) = \sum_i l(y_i \mathbf{x}_i^\intercal \boldsymbol{\beta})$$

where $(y_i \mathbf{x}_i^\intercal \boldsymbol{\beta})$ are margins for classification. Examples are the lasso penalized logistic regression and support vector machine using squared hinge loss or Huberized squared hinge loss with the lasso penalty.

$$\text{Logistic :} \qquad \hat{\boldsymbol{\beta}}(\lambda) = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \log \left(1 + e^{-y_i \mathbf{x}_i^\intercal \boldsymbol{\beta}} \right) + \lambda ||\boldsymbol{\beta}||_1$$

$$\text{Squared SVM :} \qquad \hat{\boldsymbol{\beta}}(\lambda) = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (1 - y_i \mathbf{x}_i^\intercal \boldsymbol{\beta})_+^2 + \lambda ||\boldsymbol{\beta}||_1$$

$$\text{Huberized SVM :} \qquad \hat{\boldsymbol{\beta}}(\lambda) = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} H_c(y_i \mathbf{x}_i^\intercal \boldsymbol{\beta}) + \lambda ||\boldsymbol{\beta}||_1$$

$$\text{where } H_c(t) = \begin{cases} 0 & t > 1 \\ (1-t)^2/2\delta & 1 - \delta < t \leq 1 \\ 1 - t - \delta/2 & t \leq 1 - \delta \end{cases}$$

Another well known example of penalized model is Cox's partial likelihood. Assume we have n training samples $(y_i, \mathbf{x}_i, d_i)_{i=1}^n$ where y_i is the survival time; $1 - d_i$ is the the censoring indicator: $d_i = 1$ indicates no censoring and $d_i = 0$ indicates right censoring; Denote by $t_1 < t_2 < \cdots < t_S$ the distinct failure times; let R_s be the risk set at time $t_s - 0$ and let k_s be the index of the failure at time t_s (meaning that patient k_s died at time t_s). Assume (1) noninformative censoring, (2) proportional hazards, then the negative log-partial-likelihood with the lasso regularization is

$$\text{Cox}'\text{s Model} \qquad \hat{\boldsymbol{\beta}}(\lambda) = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{n} \sum_{s=1}^{S} -\mathbf{x}_{k_s}^{\mathsf{T}} \boldsymbol{\beta} + \log \left(\sum_{k \in R_s} \exp(\mathbf{x}_k^{\mathsf{T}} \boldsymbol{\beta}) \right) + \lambda ||\boldsymbol{\beta}||_1$$

1.3 Thesis outline

This thesis consists of four papers to be found in Chapters 2-5.

In Chapter 2, we consider a family of coordinate majorization descent algorithms including the classical coordinate descent as a special case. The generalization is actually very straightforward. We simply replace each of the coordinate descent step with a coordinate majorization descent (CMD) operation and everything else in glmnet stays the same. Numerical experiments show that this simple CMD trick can lead to substantial improvement in speed when the predictors have moderate or high correlations.

In Chapter 3, we formulate the general group-lasso learning problem. We introduce the qudaratic majorization (QM) condition and show that many popular loss functions for regression and classification satisfy the QM condition. We then derive the BMD algorithm for solving the group-lasso model satisfying the QM condition and discuss some important implementation issues. Simulation and real data examples are also presented.

In Chapter 4, we review the HHSVM and then introduce the GCD algorithm for computing the HHSVM. We study the descent property of the GCD algorithm by making an intimate connection to the MM principle. The analysis motivates us to further consider a generic GCD algorithm for handling a class of elastic net penalized large margin classifiers. Numerical experiments are presented in the end.

In Chapter 5, we show how to combine the majorization-minimization principle and

coordinate descent into a new coordinate-majorization-descent algorithm (CMD). We further show how to integrate the strong rule and the CMD algorithm, which leads to the final cocktail algorithm for computing the solution paths of the elastic net penalized Cox's regression. Numerical examples are presented in the last section.

Finally, Chapter 6 discusses some potential future work.

Chapter 2

A Coordinate Majorization Descent Algorithm for ℓ_1 Penalized Learning

2.1 Chapter Overview

The glmnet package by [12] is an extremely fast implementation of the standard coordinate descent algorithm for solving ℓ_1 penalized learning problems. In this chapter, we consider a family of coordinate majorization descent algorithms for solving the ℓ_1 penalized learning problems by replacing each coordinate descent step with a coordinate-wise majorization descent operation. Numerical experiments show that this simple modification can lead to substantial improvement in speed when the predictors have moderate or high correlations.

2.2 Introduction

The lasso [20] is a very popular technique for high-dimensional modeling. A key contributor to the tremendous popularity of the lasso is the celebrated *Least Angle Regression* (LARS) algorithm proposed by [1]. LARS efficiently produces the piecewise linear solution paths of the lasso penalized least squares with the computational cost of a single

least squares fit. Another efficient algorithm for solving the lasso is the *cyclical coordinate descent* algorithm. [8] developed the first working coordinate descent algorithm for solving the lasso. Some recent papers have made coordinate descent a popular computational algorithm for sparse regression. See [25], [11], [10], among others.

[2] proposed the elastic net penalty as an improved variant of the lasso to better handle correlated variables and to stabilize the lasso solution paths. The ℓ_1 component of the elastic net is responsible for achieving sparsity. Hence one can regard the elastic net as a member of the ℓ_1 penalized methods. [2] developed the LARS-EN algorithm for computing the entire solution paths of the elastic net penalized least squares. [12] developed the glmnet package to implement a coordinate descent algorithm for fitting the entire lasso or elastic net regularization paths for generalized linear models. Numerical experiments in [12] showed that glmnet is faster than the other publicly available packages for solving the ℓ_1 penalized models.

2.3 Coordinate Majorization Descent

2.3.1 Review of glmnet

Because we present an improved glmnet algorithm, it is convenient and necessary to review some key elements of glmnet first. Consider the penalized least squares (PLS) problem. Given a training dataset with N observations (x_i, y_i) where x denotes a p dimensional predictor vector and y is a continuous response. Without loss of generality, let us assume that the predictors are standardized: $\sum_{i=1}^N x_{ij} = 0$, $\frac{1}{N}\sum_{i=1}^N x_{ij}^2 = 1$, for $j = 1, \ldots, p$. We use a linear function $\beta_0 + x^\intercal\beta$ to predict y. Define a penalized residual sum squares as follows

$$R(\beta_0, \beta) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \beta_0 - x_i^\intercal\beta)^2 + P_{\lambda,\alpha}(\beta), \tag{2.1}$$

where $P_{\lambda,\alpha}(\beta)$ is the elastic net penalty [2] and it is defined as

$$P_{\lambda,\alpha}(\beta) \quad = \lambda\textstyle\sum_{j=1}^p p_\alpha(\beta_j) = \quad \lambda\displaystyle\sum_{j=1}^{p}\left[\frac{1}{2}(1-\alpha)\beta_j^2 + \alpha|\beta_j|\right]. \tag{2.2}$$

Then the fitted model is obtained via $(\hat\beta, \hat\beta_0) = \underset{(\beta_0,\beta)\in\mathbb{R}^{p+1}}{\operatorname{argmin}} R(\beta_0, \beta)$. The elastic net with $\alpha = 1$ reduces to the lasso. When the predictors exhibit strong correlation, using some

Algorithm 1 The coordinate descent algorithm for the elastic net PLS

1. Initialize $(\tilde{\beta}_0, \tilde{\beta})$.
2. Cyclic coordinate descent, for $j = 1, 2, \ldots, p$: compute $r_i = y_i - \tilde{\beta}_0 - x_i^\intercal \tilde{\beta}$ and
$$\hat{\beta}_j = \frac{S\left(\frac{1}{N}\sum_{i=1}^N x_{ij} r_i + \tilde{\beta}_j, \lambda\alpha\right)}{1 + \lambda(1-\alpha)}.$$
3. Set $\tilde{\beta}_j = \hat{\beta}_j$.
4. Repeat steps $2 - 3$ until convergence of $\hat{\beta}$.

$\alpha < 1$ yields better prediction accuracy.

For each fixed λ, cyclic coordinate descent can be easily implemented for solving the elastic net. To keep our discussion concise, we refer interested readers to [12] for more details. We just discuss the main ideas. Let $r_i = y_i - \tilde{\beta}_0 - x_i^\intercal \tilde{\beta}$ be the current residual. To update the estimate for β_j we need to solve a univariate elastic net problem

$$\hat{\beta}_j \quad = \quad \underset{\beta_j}{\text{argmin}} R(\beta_j | \tilde{\beta}_0, \tilde{\beta}), \tag{2.3}$$

where

$$R(\beta_j | \tilde{\beta}_0, \tilde{\beta}) = \frac{1}{2}\left(\beta_j - \tilde{\beta}_j\right)^2 - \frac{1}{N}\sum_{i=1}^N r_i x_{ij}\left(\beta_j - \tilde{\beta}_j\right) + \lambda p_\alpha(\beta_j). \tag{2.4}$$

It turns out that (2.3) has a simple closed form solution [2]

$$\hat{\beta}_j = \frac{S\left(\frac{1}{N}\sum_{i=1}^N x_{ij} r_i + \tilde{\beta}_j, \lambda\alpha\right)}{1 + \lambda(1-\alpha)}, \tag{2.5}$$

where $S(z, t) = (|z| - t)_+ \text{sgn}(z)$. We next set $\tilde{\beta}_j = \hat{\beta}_j$ as the new estimate. The operation is sequentially conducted on each coordinate β_j till convergence. See Algorithm 1.

Algorithm 2 The CMD algorithm for the elastic net PLS

1. Initialize $(\tilde{\beta}_0, \tilde{\beta})$.
2. Cyclic coordinate descent, for $j = 1, 2, \ldots, p$: compute $r_i = y_i - \tilde{\beta}_0 - x_i^{\mathsf{T}}\tilde{\beta}$ and
$$\hat{\beta}_j^{\mathrm{B}} = \frac{S\left(\frac{1}{N}\sum_{i=1}^{N} x_{ij} r_i + f \cdot \tilde{\beta}_j, \lambda\alpha\right)}{f \cdot 1 + \lambda(1-\alpha)}. \quad (f \geq 1)$$
3. Set $\tilde{\beta}_j = \hat{\beta}_j^{\mathrm{B}}$.
4. Repeat steps $2 - 3$ until convergence of $\hat{\beta}$.

2.3.2 The majorization trick

We now introduce a family of generalized coordinate descent algorithms. We consider modifying the update formula (2.5) as follows

$$\hat{\beta}_j^{\mathrm{B}} = \frac{S\left(\frac{1}{N}\sum_{i=1}^{N} x_{ij} r_i + f \cdot \tilde{\beta}_j, \lambda\alpha\right)}{f \cdot 1 + \lambda(1-\alpha)} \quad (f \geq 1) \tag{2.6}$$

and hence Algorithm 1 becomes Algorithm 2.

Comparing (2.5) and (2.6), one can see that the generalization lies in an extra constant factor f. When $f = 1$, (2.6) reduces to (2.5). We will show that as long as f is greater or equal to 1, Algorithm 2 is guaranteed to converge. We have verified that with $f = 1$ glmnet2 and glmnet not only give exactly identical solutions but also use the same timing. Interestingly, when using some f greater than 1, Algorithm 2 can enjoy faster convergence than Algorithm 1. In numerical experiments presented in this chapter we set $f = 2$ unless stated otherwise.

The convergence property of Algorithm 1 comes from the fact that each operation by (2.5) minimizes the objective function along the jth coordinate direction, which is the basic idea of coordinate descent. We show that each operation by (2.6) at least decreases the objective function along the jth coordinate direction if $f > 1$. To appreciate this fact, we invoke the Majorization-Minimization (MM) principle [17, 16, 26] which can be regarded as a more generalized form of the famous Expectation-Maximization algorithm [14].

To apply the MM principle, define

$$Q(\beta_j) = \frac{1}{2} f \cdot \left(\beta_j - \tilde{\beta}_j\right)^2 - \frac{1}{N}\sum_{i=1}^{N} r_i x_{ij}\left(\beta_j - \tilde{\beta}_j\right) + \lambda p_\alpha(\beta_j). \tag{2.7}$$

Note that $\hat{\beta}_j^{\mathrm{B}}$ actually minimizes $Q(\beta_j)$, i.e.,

$$\hat{\beta}_j^{\mathrm{B}} = \underset{\beta_j}{\mathrm{argmin}} Q(\beta_j). \tag{2.8}$$

On the other hand, we have

$$Q(\beta_j) - R(\beta_j|\tilde{\beta}_0, \tilde{\beta}) = \frac{1}{2}(f-1)(\beta_j - \tilde{\beta}_j)^2. \tag{2.9}$$

Therefore, for any $f > 1$, $Q(\beta_j) > R(\beta_j|\tilde{\beta}_0, \tilde{\beta})$ unless $\beta_j = \tilde{\beta}_j$. Hence we have

$$\begin{aligned} R(\hat{\beta}_j^{\mathrm{B}}|\tilde{\beta}_0, \tilde{\beta}) &= Q(\hat{\beta}_j^{\mathrm{B}}) + R(\hat{\beta}_j^{\mathrm{B}}|\tilde{\beta}_0, \tilde{\beta}) - Q(\hat{\beta}_j^{\mathrm{B}}) && (2.10)\\ &\leq Q(\tilde{\beta}_j) && (2.11)\\ &= R(\tilde{\beta}_j|\tilde{\beta}_0, \tilde{\beta}). && (2.12) \end{aligned}$$

Obviously, $R(\hat{\beta}_j^{\mathrm{B}}|\tilde{\beta}_0, \tilde{\beta}) = R(\tilde{\beta}_j|\tilde{\beta}_0, \tilde{\beta})$ if and only if $\hat{\beta}_j^{\mathrm{B}} = \tilde{\beta}_j$. The above arguments show that Algorithm 2 retains the essential descent property of the original coordinate descent algorithm. So it is a genuine coordinate-wise descent algorithm. Because the MM principle is crucial to its descent property, Algorithm 2 is named coordinate majorization descent algorithm.

Unlike Algorithm 1, Algorithm 2 does not take the steepest descent step along each coordinate direction. This seems counterintuitive, as we usually want to decrease the objective function as much as we can at each iteration. In fact, when the predictors are uncorrelated, Algorithm 1 gives the exact solution after one cycle, while Algorithm 2 still needs to iterate. Thus we expect to see Algorithm 1 is faster than Algorithm 2 when the predictors are uncorrelated or nearly uncorrelated. However, in high dimensional data the predictors often have strong correlations or many moderate correlations. What we have found is that in such more complex situations Algorithm 2 can be substantially faster than Algorithm 1. In Section 3.4 we offer some explanation to this interesting phenomenon.

Algorithm 3 The coordinate descent algorithm for the elastic net PWLS

1. Initialize $(\tilde{\beta}_0, \tilde{\beta})$.

2. Cyclic coordinate descent, for $j = 1, 2, \ldots, p$: compute $r_i = y_i - \tilde{\beta}_0 - x_i^\intercal \tilde{\beta}$ and

$$\hat{\beta}_j = \frac{S\left(\frac{1}{N}\sum_{i=1}^N \omega_i x_{ij} r_i + \left(\frac{1}{N}\sum_{i=1}^N \omega_i x_{ij}^2\right)\tilde{\beta}_j, \lambda\alpha\right)}{\frac{1}{N}\sum_{i=1}^N \omega_i x_{ij}^2 + \lambda(1-\alpha)}. \tag{2.14}$$

3. Set $\tilde{\beta}_j = \hat{\beta}_j$.

4. Repeat steps $2 - 3$ until convergence of $\hat{\beta}$.

2.3.3 Penalized weighted least squares and logistic regression

Often we need to assign a weight ω_i (other than $1/N$) to each observation in least squares. For example, weighted least squares handles linear regression with heteroscadestic variance and iterative weighted least squares is a classical algorithm for fitting many statistical models. Both Algorithm 1 and Algorithm 2 can be easily modified to deal with the elastic net penalized weighted least squares (PWLS) in which the objective function is

$$\min_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} \frac{1}{2N}\sum_{i=1}^N \omega_i (y_i - \beta_0 - x_i^\intercal \beta)^2 + P_{\lambda,\alpha}(\beta). \tag{2.13}$$

Algorithm 3 computes that the solution to (2.13) [12]. Following the arguments in Section 2.2 we derive a coordinate majorization descent algorithm for solving (2.13). See Algorithm 4.

In a logistic regression model we have a binary response variable $Y = \{0, 1\}$ and assume

$$\Pr(Y = 1|x) = \frac{1}{1 + \exp(-\beta_0 + x^\intercal \beta)} = p_i.$$

We consider the elastic net penalized maximum likelihood estimate

$$(\hat{\beta}_0, \hat{\beta}) = \underset{(\beta_0,\beta)\in\mathbb{R}^{p+1}}{\operatorname{argmin}} \left[-\frac{1}{N}\sum_{i=1}^N \{y_i \log p_i + (1 - y_i)\log(1 - p_i)\} + P_{\lambda,\alpha}(\beta)\right]. \tag{2.16}$$

The un-penalized logistic regression is often solved by using the Newton-Raphson algorithm in many standard statistical packages. glmnet uses a similar strategy and it is so

Algorithm 4 The CMD algorithm for the elastic net PWLS

1. Initialize $(\tilde{\beta}_0, \tilde{\beta})$.

2. Cyclic coordinate descent, for $j = 1, 2, \ldots, p$: compute $r_i = y_i - \tilde{\beta}_0 - x_i^\intercal \tilde{\beta}$ and

$$\hat{\beta}_j^{\mathrm{B}} = \frac{S\left(\frac{1}{N}\sum_{i=1}^N \omega_i x_{ij} r_i + f \cdot \left(\frac{1}{N}\sum_{i=1}^N \omega_i x_{ij}^2\right)\tilde{\beta}_j, \lambda\alpha\right)}{f \cdot \frac{1}{N}\sum_{i=1}^N \omega_i x_{ij}^2 + \lambda(1-\alpha)}. \quad (f > 1) \tag{2.15}$$

3. Set $\tilde{\beta}_j = \hat{\beta}_j^{\mathrm{B}}$.

4. Repeat steps $2 - 3$ until convergence of $\hat{\beta}$.

far the fastest algorithm for computing the elastic net penalized logistic regression with high dimensional data. Basically, glmnet uses coordinate descent within the iterative re-weighted least squares loop to solve the penalized logistic regression problems. Let $(\tilde{\beta}_0, \tilde{\beta})$ be the current estimate in the iterative re-weighted least squares. As in the usual logistic regression, we define the following quantities

$$\eta_i = \tilde{\beta}_0 + x_i^\intercal \tilde{\beta}, \quad \tilde{p}_i = \frac{1}{1 + \exp\left(-\tilde{\eta}_i\right)},$$
$$z_i(\tilde{\eta}_i) = \tilde{\eta}_i + \frac{y_i - \tilde{p}_i}{\tilde{p}_i(1-\tilde{p}_i)}, \quad \omega_i(\tilde{\eta}_i) = \tilde{p}_i(1-\tilde{p}_i).$$

The Newton-Raphson algorithm finds the updated solution by solving

$$\min_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} \left\{ \frac{1}{2N}\sum_{i=1}^N \omega_i(\tilde{\eta}_i)(z_i(\tilde{\eta}_i) - \beta_0 - x_i^\intercal \beta)^2 + P_{\lambda,\alpha}(\beta) \right\}. \tag{2.17}$$

Glmnet calls Algorithm 3 to solve (2.17). The complete glmnet algorithm for penalized logistic regression is given in Algorithm 5. Our package glmnet2 is almost identical to glmnet except that we use Algorithm 4 to solve (2.17).

2.4 Numerical Experiments

Glmnet uses several tricks to boost its speed, including pathwise descent, warm start and active set convergence. For the sake of space we do not repeat the details of these

Algorithm 5 Glmnet and Glmnet2 for penalized logistic regression

1. Initialize $(\tilde{\beta}_0, \tilde{\beta})$, and set $\tilde{\eta}_i = \tilde{\beta}_0 + x_i^\intercal \tilde{\beta}$ for $i = 1, \ldots, N$.

2. Compute $z_i = z_i(\tilde{\eta}_i)$ and $\omega_i = \omega_i(\tilde{\eta}_i)$.

3. Glmnet calls Algorithm 3 and Glmnet 2 calls Algorithm 4 to solve

$$(\hat{\beta}_0, \hat{\beta}) = \underset{(\beta_0,\beta)\in\mathbb{R}^{p+1}}{\operatorname{argmin}} \frac{1}{2N} \sum_{i=1}^{N} \omega_i (z_i - \beta_0 - x_i^\intercal \beta)^2 + P_{\lambda,\alpha}(\beta) .$$

4. Set $\tilde{\beta} = \hat{\beta}, \tilde{\beta}_0 = \hat{\beta}_0$.

5. Repeat steps $2 - 4$ until convergence of $\hat{\beta}$.

tricks here. The readers are referred to [12] for the warn start trick, which deals with initial values for the iterative coordinate descent, and the active set trick. In the latest version of glmnet (version 1.7), glmnet further uses the strong rule trick [27]. In order for us to show that the timing difference between glmnet and glmnet2 is solely due to the extra factor f, we need to make sure that the two algorithms use the same implementation tricks. To do so, we took the core `Fortran` routines used in glmnet and added the extra f factor in those used for doing the soft-thresholding operation.

In glmnet version 1.7 the convergence criterion is $\max_j \left(\hat{\beta}_j^{\text{old}} - \hat{\beta}_j^{\text{new}}\right)^2 < \epsilon^2$. The same convergence criterion is used in glmnet2. In this section $\epsilon = 10^{-5}$. We compare the run times of glmnet and glmnet2. All timings were carried out on an Intel Core 2 Duo 2.4 GHz processor.

2.4.1 Simulated data

To fix idea, we use $f = 2$ in this subsection. We further explore the effect of f on timing in Section 2.4.3. Consider Friedman's model for timing comparison [12]. We simulated data with N observations and p predictors where each pair of predictors X_j and $X_{j'}$ have the same population correlation ρ, with ρ ranges from zero to 0.95. We tried

$(N = 5000, p = 100)$ and $(N = 100, p = 5000)$. The response variable was generated by

$$Y = \sum_{j=1}^{p} X_j \beta_j + k \cdot N(0, 1),$$

where $\beta_j = (-1)^j \exp(-(2j-1)/20)$ and k is set to make the signal-to-noise ratio equal 3. For logistic regression, we used the same simulation setup as above, except we define $p = 1/(1 + \exp(-Y))$ and generate a two class response Y' is generated with $\Pr(\mathrm{Y}' = 0) = p \quad \Pr(\mathrm{Y}' = +1) = 1 - p$. For each data set we computed its elastic net solution paths with $\alpha = 1$ and $\alpha = 0.5$ for 100 λ values. In Table 2.1 and 2.2 We report the average run time of glmnet and glmnet2 over 10 independent runs.

2.4.2 Real data

We also compared glmnet and glmnet2 on some benchmark data sets. See Table 2.3. Colon [28] and Prostate [29] are the typical examples of the $p \gg N$ data. In the other three data sets, Wisconsin Breast Cancer Diagnostic (WBCD) data; Ionosphere data and Sonar data [30], the original dimension is less than the sample size. We expanded the predictor set by including the second order polynomials and pairwise interactions of the original predictors. Then the expanded dimension becomes much larger or at least similar to the sample size; see row 2 of Table 2.3. We fit the elastic net penalized logistic regression model on each data set and used 10-fold cross-validation to choose α; see row 3 of Table 2.3. Fix $\alpha = \alpha_{CV}$. We compared the running time of glmnet and glmnet2. The relative speed improvement is defined as $(t_{\text{glmnet}} - t_{\text{glmnet2}})/t_{\text{glmnet2}}$. We can see than glmnet2 is noticeably faster than glmnet, especially on the WBCD and Sonar.

2.4.3 Exploring the factor size

We have fixed $f = 2$ in the previous numerical examples. Now we further explore the effect of f on timing. In Figure 2.1 we plotted the run time (in log scale) against f for different correlation levels varying from 0 to 0.95. The dotted vertical reference line in each panel indicates the run time of glmnet. We see that when $\rho = 0$ increasing f only slows down the convergence, which is expected. However, when the correlation becomes stronger ($\rho \geq 0.2$), the curve starts to have a valley and using some $f > 1$ can reduce

	Correlation					
	0	0.1	0.2	0.5	0.8	0.95
	Least Squares $\alpha = 1$					
	$N = 5000$, $p = 100$					
glmnet	0.0719	0.0746	0.0787	0.0988	0.1641	0.3144
glmnet2	0.0752	0.0757	0.0764	0.0833	0.1094	0.1921
	$N = 100$, $p = 5000$					
glmnet	0.2222	0.2339	0.2979	0.4606	0.7919	1.9016
glmnet2	0.2533	0.2519	0.2886	0.3758	0.5450	1.0735
	Logistics Regression $\alpha = 1$					
	$N = 5000$, $p = 100$					
glmnet	1.4282	1.5591	1.9760	4.1573	8.9447	35.5548
glmnet2	1.8754	1.9002	2.0551	2.6598	5.2532	14.5804
	$N = 100$, $p = 5000$					
glmnet	0.2756	0.2734	0.2938	0.3940	0.8790	1.6118
glmnet2	0.2744	0.2734	0.2876	0.3302	0.5328	0.9422

Table 2.1: Timings (in seconds) for **glmnet** and **glmnet2** in the elastic net penalized ($\alpha = 1$) regression and logistic regression. Total time for 100 λ values, averaged over 10 independent runs.

	Correlation					
	0	0.1	0.2	0.5	0.8	0.95
	Least Squares $\alpha = 0.5$					
	$N = 5000$, $p = 100$					
glmnet	0.0718	0.0750	0.0802	0.1003	0.1704	0.5112
glmnet2	0.0750	0.0755	0.0770	0.0859	0.1145	0.2526
	$N = 100$, $p = 5000$					
glmnet	0.2107	0.2189	0.2356	0.3669	0.7765	2.1528
glmnet2	0.2225	0.2285	0.2414	0.2861	0.4876	1.3335
	Logistics Regression $\alpha = 0.5$					
	$N = 5000$, $p = 100$					
glmnet	1.5438	1.6540	2.0519	4.3477	11.9141	37.5434
glmnet2	2.0086	2.0039	2.1666	2.9319	6.1041	16.8284
	$N = 100$, $p = 5000$					
glmnet	0.3193	0.3573	0.4661	0.6217	1.1428	2.0621
glmnet2	0.3176	0.3253	0.3456	0.4042	0.5980	1.3126

Table 2.2: Timings (in seconds) for **glmnet** and **glmnet2** in the elastic net penalized ($\alpha = 0.5$) regression and logistic regression. Total time for 100 λ values, averaged over 10 independent runs.

	Colon	Prostate	WBCD	Ionosphere	Sonar
N	62	102	569	351	208
p	2000	6033	495 (30)	560 (32)	1890 (60)
α_{CV}	0.6	0.5	0.6	0.4	0.4
Test Error	8.3%	5%	1.77%	2.86%	24.39%
glmnet	0.1166	0.3283	9.4039	0.5158	2.0828
glmnet2	0.0910	0.2938	4.9593	0.3667	1.0945
Improv. %	+28%	+11.7%	+89.6%	+40.6%	+90.3%

Table 2.3: Timings (in seconds) of **glmnet** and **glmnet2** for some real data, averaged over 10 runs.

the computing time. From Figure 2.1 it seems that 2 is a good default value for f. We also see that when the correlation is very high such as $\rho = 0.8$ or higher, $f = 4$ or $f = 6$ can even work slightly better than $f = 2$. On the other hand, using $f = 4$ or $f = 6$ can have much bigger loss in speed when correlation is low compared to using $f = 2$. It would be also interesting to decide f's value based on the empirical correlations. We tested this idea and did not find this strategy works noticeably better than just using $f = 2$.

2.4.4 Some explanation of the acceleration effect

We have shown by numerical experiments that using $f > 1$ in the CMD could lead to faster convergence than the ordinary coordinate descent using $f = 1$, especially when the predictors are highly correlated. Now we attempt to provide some explanation to this acceleration effect. To gain some insight, we consider a simpler case where we use the CMD to solve the ordinary least squares problem with p predictors, defined as $\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \frac{1}{2N} \left\| \mathbf{y} - \mathbf{x}\beta^T \right\|^2$. This model is a special point on the ℓ_1 penalized least squares solution path. Without loss of generality assume all predictors are standardized such that $\overline{x_j} = \frac{1}{N}\sum_{i=1}^N x_{ij} = 0$, $s_{x_j}^2 = \frac{1}{N}\sum_{i=1}^N (x_{ij} - \overline{x_j})^2 = 1$, for $j = 1, \cdots, p$. To simplify the analysis, we further assume that the pairwise sample correlation is a constant, i.e., $\frac{1}{N}\sum_{i=1}^N x_{ij}x_{ik} = \rho$ for $j, k \in \{1, \cdots, p\}$.

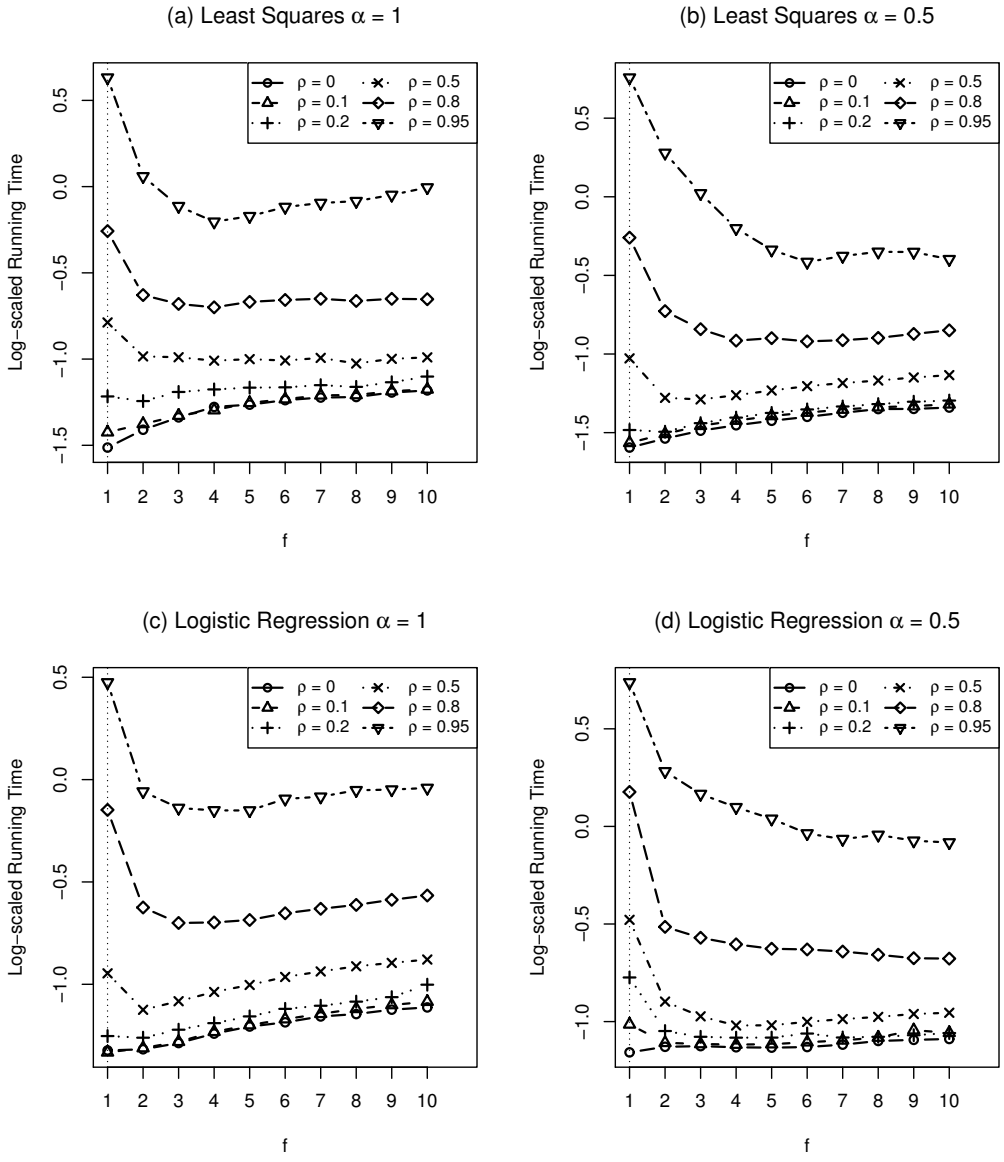

Figure 2.1: The running time of **glmnet2** for computing solution paths at 100 λs of the elastic net penalized regression and logistic regression with $\alpha = 1$ and $\alpha = 0.5$, averaged over 10 independent runs. The factor size f varies from 1 to 10. The data were generated from the simulation model in Section 2.4.1 with $N = 100$, $p = 5000$. Each curve corresponds to a different correlation level.

Given below is the CMD algorithm for the least squares regression problem:

1. Initialize $\tilde{\beta} = \beta^{(0)}$.

2. For $k = 1, 2, 3, \cdots$, iterate step 3 until convergence of $\tilde{\beta}$.

3. For $j = 1, \cdots, p$, fix $\tilde{\beta} = (\beta_1^{(k)}, \cdots, \beta_{j-1}^{(k)}, \beta_j^{(k-1)}, \beta_{j+1}^{(k-1)}, \cdots, \beta_p^{(k-1)})$, we update the j-th coordinate of $\tilde{\beta}$,

$$\begin{aligned} \tilde{\beta}_j^{(k)} &= \underset{\beta_j}{\operatorname{argmin}} \frac{1}{2} f(\beta_j - \beta_j^{(k-1)})^2 - \frac{1}{N} x_j^{\intercal} \left(y - \mathbf{x}\tilde{\beta}^{\intercal} \right) (\beta_j - \beta_j^{(k-1)}) \quad (2.18) \\ &= \tilde{\beta}(\underbrace{-\frac{\rho}{f}, \cdots, -\frac{\rho}{f}}_{j-1}, 1 - \frac{1}{f}, \underbrace{-\frac{\rho}{f}, \cdots, -\frac{\rho}{f}}_{p-j})^{\intercal} + \frac{x_j^{\intercal} y}{fN}. \quad (2.19) \end{aligned}$$

We have used the equal correlation assumption to simplify (18) to get (19).

Note that we can rewrite (19) in step 3 as follows

$$\tilde{\beta}^{\text{new}} = \tilde{\beta}\mathbf{W}_j + \upsilon_j, \tag{2.20}$$

where

$$\tilde{\beta}^{\text{new}} = \left(\beta_1^{(k)}, \cdots, \beta_{j-1}^{(k)}, \beta_j^{(k)}, \beta_{j+1}^{(k-1)}, \cdots, \beta_p^{(k-1)} \right), \qquad \upsilon_j = (0, \cdots, \frac{x_j^{\intercal} y}{fN}, \cdots, 0).$$

$$\mathbf{W}_j = \mathbf{I}_{p\times p} + \left[\mathbf{0}_{p\times(j-1)} \quad \mathbf{u}_j \quad \mathbf{0}_{p\times(N-j)} \right], \qquad \mathbf{u}_j = (u_{kj})_{p\times 1} = \begin{cases} -\frac{1}{f} & k = j \\ -\frac{\rho}{f} & k \neq j \end{cases}.$$

Using (2.20), we find that after a complete cycle from $j = 1$ to $j = p$ we can write

$$\tilde{\beta}^{(k)} = \tilde{\beta}^{(k-1)}\mathbf{A} + \mu, \tag{2.21}$$

where

$$\mathbf{A} = \prod_{j=1}^{p} \mathbf{W}_j, \qquad \mu = \sum_{s=1}^{p-1} \left(\upsilon_s \prod_{j=s+1}^{p} \mathbf{W}_j \right) + \upsilon_p. \tag{2.22}$$

Then by a simple transformation $\tilde{\gamma}^{(k)} = \tilde{\beta}^{(k)} + \omega$ where $\omega = (\mathbf{I} - \mathbf{A})^{-1}\mu$ we can express (21) in terms of $\tilde{\gamma}^{(k)}$ and $\tilde{\gamma}^{(k-1)}$ as follows

$$\gamma^{(k)} = \mathbf{A}\gamma^{(k-1)}, \tag{2.23}$$

which means that

$$\gamma^{(k)} = \mathbf{A}^k \gamma^{(0)}, \tag{2.24}$$

and

$$\left\|\gamma^{(k)}\right\| = \left\|\mathbf{A}^k \gamma^{(0)}\right\| \quad \leq \quad \sqrt{\eta_{\max}\left(\left(\mathbf{A}^k\right)^\intercal \mathbf{A}^k\right)}\left\|\gamma^{(0)}\right\|, \tag{2.25}$$

where $\eta_{\max}\left(\left(\mathbf{A}^k\right)^\intercal \mathbf{A}^k\right)$ is the maximum eigenvalue of $\left(\mathbf{A}^k\right)^\intercal \mathbf{A}^k$.

From (2.25) one can see that the convergence rate of the CMD algorithm for the least squares problem is determined by $\eta_{\max}\left(\left(\mathbf{A}^k\right)^\intercal \mathbf{A}^k\right)$, which is affected by both f and ρ. Although we do not find an explicit expression of $\eta_{\max}\left(\left(\mathbf{A}^k\right)^\intercal \mathbf{A}^k\right)$, we can compute it numerical values easily. We did the calculation for $p = 10$ and Figure 2.2 displays the calculated $\eta_{\max}\left(\left(\mathbf{A}^k\right)^\intercal \mathbf{A}^k\right)$ as a function of $\log(k)$ for different combinations of (f, ρ). It is not surprising to see that as k (the number of iterations) increases, $\eta_{\max}\left(\left(\mathbf{A}^k\right)^\intercal \mathbf{A}^k\right)$ goes to zero, for all factors considered there. As shown in Figure 2.2 panel (a), when the correlation is low, $f = 1$ has the fastest convergence. However, when $\rho = 0.5$ as shown in Figure 2.2 panel (b), $f = 2$ starts to outperform $f = 1$. When the correlation is even higher like in Figure 2.2 panels (c) and (d), $f = 2, 3, 4, 5$ clearly dominates $f = 1$.

The above theoretical results are derived for the least squares problem. The analysis is not directly generalized to the more general ℓ_1 penalized least squares or logistic regression. However, the analysis does show us that in using the coordinate decent scheme to solve a multivariate optimization problem, taking the steepest descent in each coordinate direction is not necessarily the best strategy for achieving convergence in the multi-dimension space.

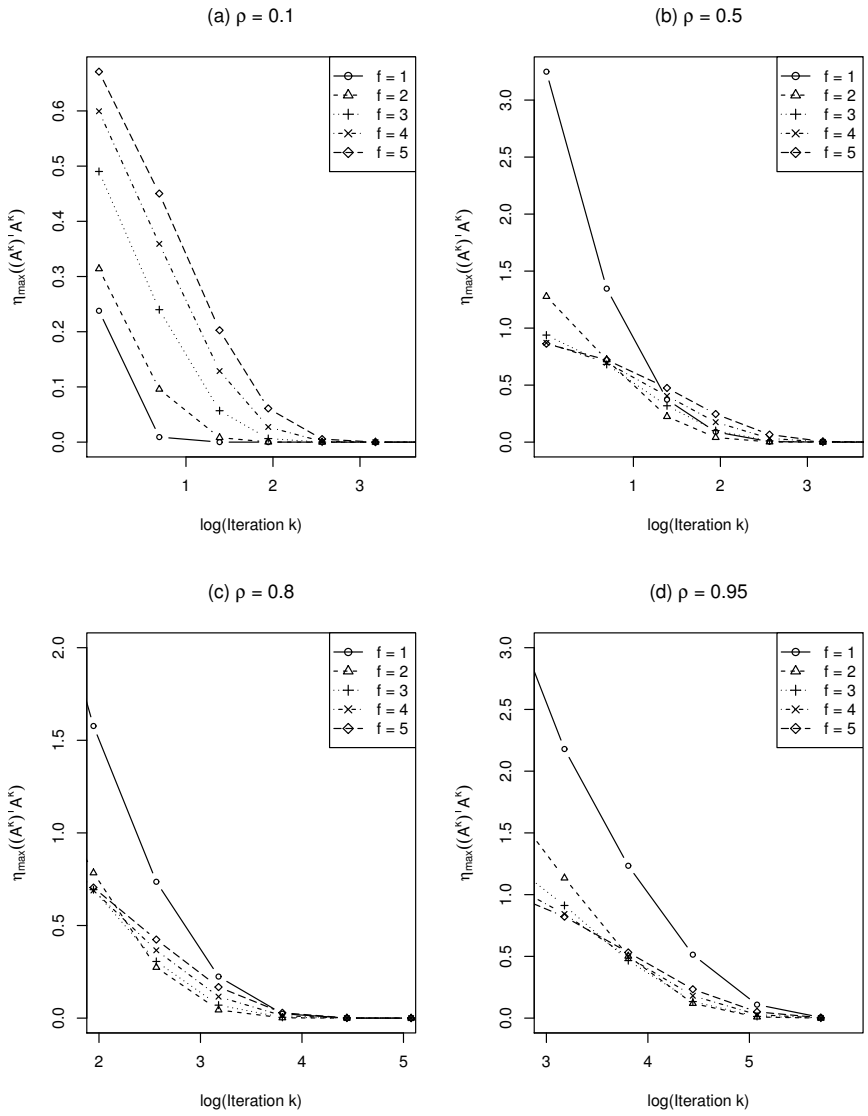

Figure 2.2: The CMD algorithm for the ordinary least squares problem with $p = 10$ predictors. $\eta_{\max}\left(\left(\mathbf{A}^k\right)^\intercal \mathbf{A}^k\right)$ (as defined in (2.25)) against the number of iterations (in logarithm scale) for 5 different factors ($f = 1, 2, 3, 4, 5$). Each panel corresponds to a different correlation level.

Chapter 3

A Fast Unified Algorithm for Group-Lasso Problems

3.1 Chapter Overview

This chapter concerns a class of group-lasso learning problems where the objective function is the sum of an empirical loss and the group-lasso penalty. For a class of loss function satisfying a quadratic majorization condition, we derive a unified algorithm called blockwise-majorization-decent (BMD) for efficiently computing the solution paths of the corresponding group-lasso penalized learning problem. BMD allows for general design matrices, without requiring the predictors to be group-wise orthonormal. As illustration examples, we develop concrete algorithms for solving the group-lasso penalized least squares and several group-lasso penalized large margin classifiers. These group-lasso models have been implemented in an R package `gglasso` publicly available from the Comprehensive R Archive Network (CRAN) at `http://cran.r-project.org/web/packages/gglasso`. On simulated and real data, `gglasso` consistently outperforms the existing software for computing the group-lasso that implements either the classical blockwise descent algorithm or Nestrov's method.

3.2 Introduction

The lasso [20] is a very popular technique for variable selection for high-dimensional data. Consider the classical linear regression problem where we have a continuous response $\mathbf{y} \in \mathbb{R}^n$ and an $n \times p$ design matrix $\mathbf{X}$. The lasso linear regression solves the following ℓ_1 penalized least squares:

$$\underset{\beta_0,\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{y} - \beta_0 - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=1}^{p} |\beta_j|. \tag{3.1}$$

The group-lasso [23] is a generalization of the lasso for doing group-wise variable selection. [23] motivated the group-wise variable selection problem by two important examples. The first example concerns the multi-factor ANOVA problem where each factor is expressed through a set of dummy variables. In the ANOVA model, deleting an irrelevant factor is equivalent to deleting a group of dummy variables. The second example is the commonly used additive model in which each nonparametric component may be expressed as a linear combination of basis functions of the original variables. Removing a component in the additive model amounts to removing a group of coefficients of the basis functions. In general, suppose that the predictors are put into K non-overlapping groups such that $(1, 2, \ldots, p) = \bigcup_{k=1}^{K} I_k$ where the cardinality of index set I_k is p_k and $I_k \bigcap I_{k'} = \emptyset$ for $k \neq k'$. Consider the linear regression model again and the group-lasso linear regression model solves the following penalized least squares:

$$\underset{\beta_0,\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{y} - \beta_0 - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{k=1}^{K} \sqrt{p_k}\|\boldsymbol{\beta}^{(k)}\|_2, \tag{3.2}$$

where $\|\boldsymbol{\beta}^{(k)}\|_2 = \sqrt{\sum_{j \in I_k} \beta_j^2}$. The group-lasso idea has been used in penalized logistic regression [31].

The group-lasso is computationally more challenging than the lasso. The entire solution paths of the lasso penalized least squares can be efficiently computed by the *least angle regression* (LARS) algorithm [1]. See also the homotopy algorithm of [32]. However, the LARS-type algorithm is not applicable to the group-lasso penalized least squares, because its solution paths are not piecewise linear. Another efficient algorithm for solving the lasso problem is the *coordinate descent* algorithm [18, 8, 33, 34, 35, 36]. [23] implemented a block-wise descent algorithm for the group-lasso penalized

least squares by following the idea of [8]. However, their algorithm requires the group-wise orthonormal condition, i.e., $\mathbf{X}_{(k)}^{\mathsf{T}}\mathbf{X}_{(k)} = \mathbf{I}_{p_k}$ where $\mathbf{X}_{(k)} = [\cdots X_j \cdots]$, $j \in I_k$. [31] also developed a block coordinate gradient descent algorithm BCGD for solving the group-lasso penalized logistic regression. Meier's algorithm is implemented in an R package `grplasso` available from the Comprehensive R Archive Network (CRAN) at `http://cran.r-project.org/web/packages/grplasso`. Later [37] proposed ISTA-BC algorithm, which is an extension of the ISTA/FISTA [38] based on variable step-lengths. The author also pointed out that ISTA-BC can be viewed as a simplified version of the Block Coordinate Descent algorithm.

From an optimization viewpoint, it is more interesting to solve the group-lasso with a general design matrix. From a statistical perspective, the group-wise orthonormal condition should not be the basis of a good algorithm for solving the group-lasso problem, even though we can transform the predictors within each group to meet the group-wise orthonormal condition. The reason is that even when the group-wise orthonormal condition holds for the observed data, it can be easily violated when removing a fraction of the data or perturbing the dataset as in bootstrap or sub-sampling. In other words, we cannot perform cross-validation, bootstrap or sub-sampling analysis of the group-lasso, if the algorithm's validity depends on the group-wise orthonormal condition. In a popular MATLAB package `SLEP`, [39] implemented Nesterov's method [40, 41] for a variety of sparse learning problems. For the group-lasso case, `SLEP` provides functions for solving the group-lasso penalized least squares and logistic regression. Nesterov's method can handle general design matrices. The `SLEP` package is available at `http://www.public.asu.edu/~jye02/Software/SLEP`.

In this chapter we consider a general formulation of the group-lasso penalized learning where the learning procedure is defined by minimizing the sum of an empirical loss and the group-lasso penalty. The aforementioned group-lasso penalized least squares and logistic regression are two examples of the general formulation. We propose a simple unified algorithm, *blockwise-majorization-decent* (BMD), for solving the general group-lasso learning problems under the condition that the loss function satisfies a quadratic majorization (QM) condition. BMD is remarkably simple and has provable numerical convergence properties. We show that the QM condition indeed holds for many popular loss functions used in regression and classification, including the squared error loss, the

Huberized hinge loss, the squared hinge loss and the logistic regression loss. It is also important to point out that BMD works for general design matrices, without requiring the group-wise orthogonal assumption. We have implemented the proposed algorithm in an R package `gglasso` which contains the functions for fitting the group-lasso penalized least squares, logistic regression, Huberized SVM using the Huberized hinge loss and squared SVM using the squared hinge loss. The Huberized hinge loss and squared hinge loss are interesting losses functions for classification from machine learning viewpoint. In fact, there has been both theoretical and empirical evidence showing that the Huberized hinge loss is better than the hinge loss [42, 19]. The group-lasso penalized Huberized SVM and squared SVM are not implemented in `grplasso` and `SLEP`. To our best knowledge, `gglasso` is the first publicly available software for computing the group-lasso penalized Huberized SVM and squared SVM for high-dimensional data.

Here we use breast cancer data [43] to demonstrate the speed advantage of `gglasso` over `grplasso` and `SLEP`. This is a binary classification problem where $n = 42$ and $p = 22,283$. We fit a sparse additive logistic regression model using the group-lasso. Each variable contributes an additive component which is expressed by five B-spline basis functions. The group-lasso penalty is imposed on the coefficients of five B-spline basis functions for each variable. Therefore, the corresponding group-lasso logistic regression model has $22,283$ groups and each group has 5 coefficients to be estimated. Displayed in Figure 3.1 are three solution path plots produced by `grplasso`, `SLEP` and `gglasso`. We computed the group-lasso solutions at 100 λ values on an Intel Xeon X5560 (Quad-core 2.8 GHz) processor. It took `SLEP` about 450 and `grplasso` about 360 seconds to compute the logistic regression paths, while `gglasso` used only about 10 seconds.

3.3 Group-Lasso Models and The QM Condition

3.3.1 Group-lasso penalized empirical loss

To define a general group-lasso model, we need to use abstract notation. Throughout this chapter we use $\mathbf{x}$ to denote the generic predictors which are used to fit the group-lasso model. Note that $\mathbf{x}$ may not be the original variables in the raw data. For example,

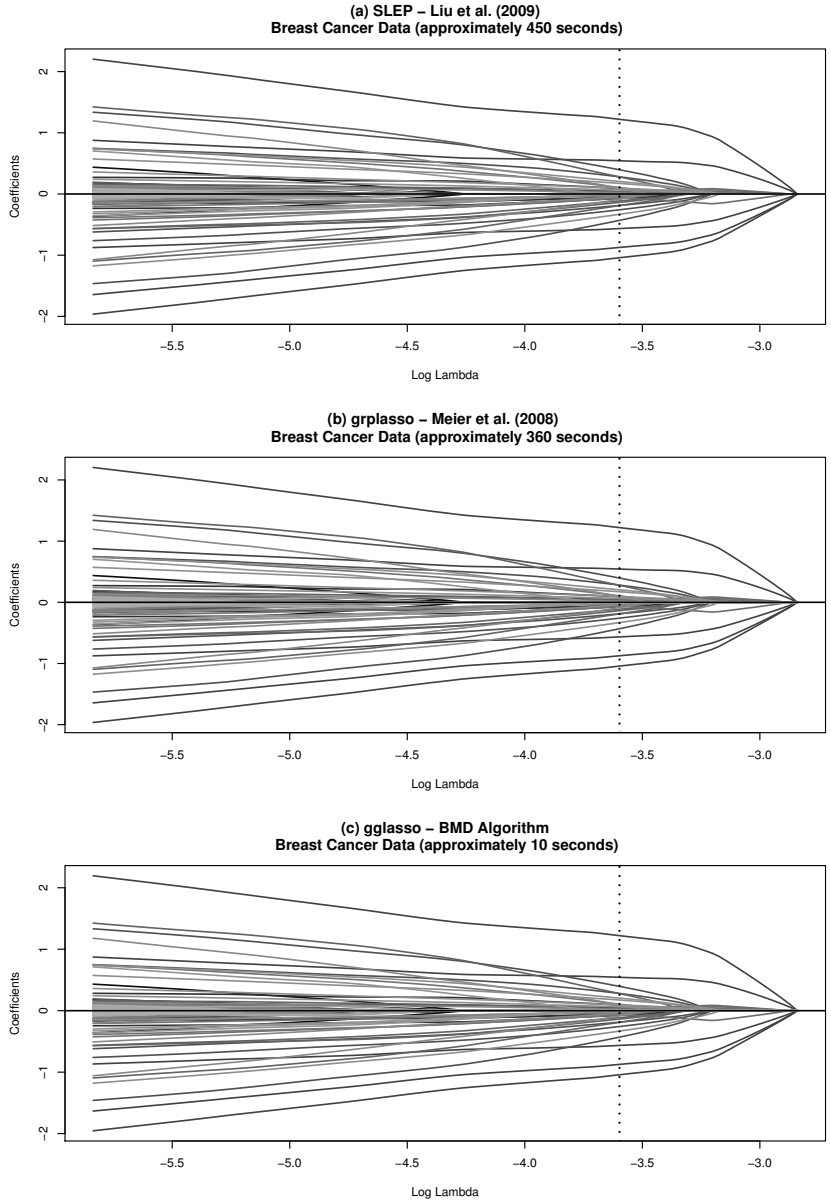

Figure 3.1: Fit a sparse additive logistic regression model using the group-lasso on the breast cancer data [43] with $n = 42$ patients and $22,283$ genes (groups). Each gene's contribution is modeled by 5 B-Spline basis functions. The solution paths are computed at 100 λ values. The vertical dotted lines indicate the selected λ ($\log \lambda = -3.73$) which selects 8 genes.

if we use the group-lasso to fit an additive regression model. The original predictors are $z_1, \ldots, z_q$ but we generate $\mathbf{x}$ variables by using basis functions of $z_1, \ldots, z_q$. For instance, $x_1 = z_1, x_2 = z_1^2, x_3 = z_1^3$, $x_4 = z_2, x_5 = z_2^2$, etc. We assume that the user has defined the $\mathbf{x}$ variables and we only focus on how to compute the group-lasso model defined in terms of the $\mathbf{x}$ variables.

Let $\mathbf{X}$ be the design matrix with n rows and p columns where n is the sample size of the raw data. If an intercept is used in the model, we let the first column of $\mathbf{X}$ be a vector of 1. Assume that the group membership is already defined such that $(1, 2, \ldots, p) = \bigcup_{k=1}^{K} I_k$ and the cardinality of index set I_k is p_k, $I_k \bigcap I_{k'} = \emptyset$ for $k \neq k', 1 \leq k, k' \leq K$. Group k contains $x_j, j \in I_k$, for $1 \leq k \leq K$. If an intercept is included, then $I_1 = \{1\}$. Given the group partition, we use $\boldsymbol{\beta}^{(k)}$ to denote the segment of $\boldsymbol{\beta}$ corresponding to group k. This notation is used for any p-dimensional vector.

Suppose that the statistical model links the predictors to the response variable y via a linear function $f = \boldsymbol{\beta}^{\mathsf{T}}\mathbf{x}$. Let $\Phi(y, f)$ be the loss function used to fit the model. In this work we primarily focus on statistical methods for regression and binary classification, although our algorithms are developed for a general loss function. For regression, the loss function $\Phi(y, f)$ is often defined as $\Phi(y - f)$. For binary classification, we use $\{+1, -1\}$ to code the class label y and consider the large margin classifiers where the loss function $\Phi(y, f)$ is defined as $\Phi(yf)$. We obtained an estimate of $\boldsymbol{\beta}$ via the group-lasso penalized empirical loss formulation defined as follows:

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \tau_i \Phi(y_i, \boldsymbol{\beta}^{\mathsf{T}}\mathbf{x}_i) + \lambda \sum_{k=1}^{K} w_k \|\boldsymbol{\beta}^{(k)}\|_2, \tag{3.3}$$

where $\tau_i \geq 0$ and $w_k \geq 0$ for all i, k.

Note that we have included two kinds of weights in the general group-lasso formulation. The observation weights τ_is are introduced in order to cover methods such as weighted regression and weighted large margin classification. The default choice for τ_i is 1 for all i. We have also included penalty weights w_ks in order to make a more flexible group-lasso model. The default choice for w_k is $\sqrt{p_k}$. If we do not want to penalize a group of predictors, simply let the corresponding weight be zero. For example, the intercept is typically not penalized so that $w_1 = 0$. Following the adaptive lasso idea [44], one could define the adaptively weighted group-lasso which often has better estimation and variable selection performance than the un-weighted group-lasso [45]. Our

algorithms can easily accommodate both observation and penalty weights.

3.3.2 The QM condition

For notation convenience, we use $\mathbf{D}$ to denote the working data $\{\mathbf{y}, \mathbf{X}\}$ and let $L(\boldsymbol{\beta}|\mathbf{D})$ be the empirical loss, i.e.,

$$L(\boldsymbol{\beta} \mid \mathbf{D}) = \frac{1}{n}\sum_{i=1}^{n} \tau_i \Phi(y_i, \boldsymbol{\beta}^\intercal \mathbf{x}_i).$$

Definition 1. *The loss function Φ is said to satisfy the quadratic majorization (QM) condition, if and only if the following two assumptions hold:*

(i). $L(\boldsymbol{\beta} \mid \mathbf{D})$ is differentiable as a function of $\boldsymbol{\beta}$, i.e., $\nabla L(\boldsymbol{\beta}|\mathbf{D})$ exists everywhere.

(ii). There exists a $p \times p$ matrix $\mathbf{H}$, which may only depend on the data $\mathbf{D}$, such that for all $\boldsymbol{\beta}, \boldsymbol{\beta}^*$,

$$L(\boldsymbol{\beta} \mid \mathbf{D}) \leq L(\boldsymbol{\beta}^* \mid \mathbf{D}) + (\boldsymbol{\beta} - \boldsymbol{\beta}^*)^\intercal \nabla L(\boldsymbol{\beta}^*|\mathbf{D}) + \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta}^*)^\intercal \mathbf{H} (\boldsymbol{\beta} - \boldsymbol{\beta}^*). \quad (3.4)$$

The following lemma characterizes a class of loss functions that satisfty the QM condition.

Lemma 1. *Let τ_i, $1 \leq i \leq n$ be the observation weights. Let $\boldsymbol{\Gamma}$ be a diagonal matrix with $\boldsymbol{\Gamma}_{ii} = \tau_i$. Assume $\Phi(y, f)$ is differentiable with respect to f and write $\Phi'_f = \frac{\partial \Phi(y,f)}{\partial f}$. Then*

$$\nabla L(\boldsymbol{\beta}|\mathbf{D}) = \frac{1}{n}\sum_{i=1}^{n} \tau_i \Phi'_f(y_i, \mathbf{x}_i^\intercal \boldsymbol{\beta}) \mathbf{x}_i.$$

(1). If Φ'_f is Lipschitz continuous with constant C such that

$$|\Phi'_f(y, f_1) - \Phi'_f(y, f_2)| \leq C|f_1 - f_2| \quad \forall\ y, f_1, f_2,$$

then the QM condition holds for Φ and $\mathbf{H} = \frac{2C}{n}\mathbf{X}^\intercal \boldsymbol{\Gamma} \mathbf{X}$.

(2). If $\Phi''_f = \frac{\partial \Phi^2(y,f)}{\partial f^2}$ exits and

$$\Phi''_f \leq C_2 \quad \forall\ y, f,$$

then the QM condition holds for Φ and $\mathbf{H} = \frac{C_2}{n}\mathbf{X}^\intercal \boldsymbol{\Gamma} \mathbf{X}$.

In what follows we use Lemma 1 to verify that many popular loss functions indeed satisfy the QM condition. The results are summarized in Table 1.

We begin with the classical squared error loss for regression: $\Phi(y, f) = \frac{1}{2}(y - f)^2$. Then we have

$$\nabla L(\boldsymbol{\beta}|\mathbf{D}) = -\frac{1}{n}\sum_{i=1}^{n} \tau_i(y_i - \mathbf{x}_i^\intercal \boldsymbol{\beta})\mathbf{x}_i. \tag{3.5}$$

Because $\Phi''_f = 1$, Lemma 1 part (2) tell us that the QM condition holds with

$$\mathbf{H} = \mathbf{X}^\intercal \boldsymbol{\Gamma} \mathbf{X}/n \equiv \mathbf{H}^{\text{ls}}. \tag{3.6}$$

We now discuss several margin-based loss functions for binary classification. We code y by $\{+1, -1\}$. The logistic regression loss is defined as $\Phi(y, f) = \text{Logit}(yf) = \log(1 + \exp(-yf))$. We have $\Phi'_f = -y\frac{1}{1+\exp(yf)}$ and $\Phi''_f = y^2\frac{\exp(yf)}{(1+\exp(yf))^2} = \frac{\exp(yf)}{(1+\exp(yf))^2}$. Then we write

$$\nabla L(\boldsymbol{\beta} \mid \mathbf{D}) = -\frac{1}{n}\sum_{i=1}^{n} \tau_i y_i \mathbf{x}_i \frac{1}{1 + \exp(y_i \mathbf{x}_i^\intercal \boldsymbol{\beta})}. \tag{3.7}$$

Because $\Phi''_f \leq 1/4$, by Lemma 1 part (2) the QM condition holds for the logistic regression loss and

$$\mathbf{H} = \frac{1}{4}\mathbf{X}^\intercal \boldsymbol{\Gamma} \mathbf{X}/n \equiv \mathbf{H}^{\text{logit}}. \tag{3.8}$$

The squared hinge loss has the expression $\Phi(y, f) = \text{sqsvm}(yf) = [(1 - yf)_+]^2$ where

$$(1 - t)_+ = \begin{cases} 0, & t > 1 \\ 1 - t, & t \leq 1. \end{cases}$$

By direct calculation we have

$$\Phi'_f = \begin{cases} 0, & yf > 1 \\ -2y(1 - yf), & yf \leq 1, \end{cases}$$

$$\nabla L(\boldsymbol{\beta} \mid \mathbf{D}) = -\frac{1}{n}\sum_{i=1}^{n} 2\tau_i y_i \mathbf{x}_i (1 - y_i \mathbf{x}_i^\intercal \boldsymbol{\beta})_+. \tag{3.9}$$

We can also verify that $|\Phi'_f(y, f_1) - \Phi'_f(y, f_2)| \leq 2|f_1 - f_2|$. By Lemma 1 part (1) the QM condition holds for the squared hinge loss and

$$\mathbf{H} = 4\mathbf{X}^\intercal \boldsymbol{\Gamma} \mathbf{X}/n \equiv \mathbf{H}^{\text{sqsvm}}. \tag{3.10}$$

Loss	$-\nabla L(\boldsymbol{\beta} \mid \mathbf{D})$	$\mathbf{H}$
Least squares	$\frac{1}{n}\sum_{i=1}^n \tau_i(y_i - \mathbf{x}_i^\mathsf{T}\boldsymbol{\beta})\mathbf{x}_i$	$\mathbf{X}^\mathsf{T}\boldsymbol{\Gamma}\mathbf{X}/n$
Logistic regression	$\frac{1}{n}\sum_{i=1}^n \tau_i y_i \mathbf{x}_i \frac{1}{1+\exp(y_i\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta})}$	$\frac{1}{4}\mathbf{X}^\mathsf{T}\boldsymbol{\Gamma}\mathbf{X}/n$
Squared hinge loss	$\frac{1}{n}\sum_{i=1}^n 2\tau_i y_i \mathbf{x}_i(1 - y_i\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta})_+$	$4\mathbf{X}^\mathsf{T}\boldsymbol{\Gamma}\mathbf{X}/n$
Huberized hinge loss	$\frac{1}{n}\sum_{i=1}^n \tau_i y_i \mathbf{x}_i \mathrm{hsvm}'(y_i\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta})$	$\frac{2}{\delta}\mathbf{X}^\mathsf{T}\boldsymbol{\Gamma}\mathbf{X}/n$

Table 3.1: The QM condition is verified for the least squares, logistic regression, squared hinge loss and Huberized hinge loss.

The Huberized hinge loss is defined as $\Phi(y, f) = \mathrm{hsvm}(yf)$ where

$$\mathrm{hsvm}(t) = \begin{cases} 0, & t > 1 \\ (1-t)^2/2\delta, & 1-\delta < t \le 1 \\ 1 - t - \delta/2, & t \le 1-\delta. \end{cases}$$

By direct calculation we have $\Phi'_f = y\mathrm{hsvm}'(yf)$ where

$$\mathrm{hsvm}'(t) = \begin{cases} 0, & t > 1 \\ (1-t)/\delta, & 1-\delta < t \le 1 \\ 1, & t \le 1-\delta, \end{cases}$$

$$\nabla L(\boldsymbol{\beta} \mid \mathbf{D}) = -\frac{1}{n}\sum_{i=1}^n \tau_i y_i \mathbf{x}_i \mathrm{hsvm}'(y_i\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta}). \tag{3.11}$$

We can also verify that $|\Phi'_f(y, f_1) - \Phi'_f(y, f_2)| \le \frac{1}{\delta}|f_1 - f_2|$. By Lemma 1 part (1) the QM condition holds for the Huberized hinge loss and

$$\mathbf{H} = \frac{2}{\delta}\mathbf{X}^\mathsf{T}\boldsymbol{\Gamma}\mathbf{X}/n \equiv \mathbf{H}^{\mathrm{hsvm}}. \tag{3.12}$$

3.4 BMD Algorithm

3.4.1 Derivation

In this section we derive the *blockwise-majorization-descent* (BMD) algorithm for computing the solution of (3.3) when the loss function satisfies the QM condition. The

objective function is

$$L(\boldsymbol{\beta} \mid \mathbf{D}) + \lambda \sum_{k=1}^{K} w_k \|\boldsymbol{\beta}^{(k)}\|_2. \tag{3.13}$$

Let $\widetilde{\boldsymbol{\beta}}$ denote the current solution of $\boldsymbol{\beta}$. Without loss of generality, let us derive the BMD update of $\widetilde{\boldsymbol{\beta}}^{(k)}$, the coefficients of group k. Define $\mathbf{H}^{(k)}$ as the sub-matrix of $\mathbf{H}$ corresponding to group k. For example, if group 2 is $\{2, 4\}$ then $\mathbf{H}_2$ is a 2×2 matrix with $\mathbf{H}_{11}^{(2)} = \mathbf{H}_{2,2}, \mathbf{H}_{12}^{(2)} = \mathbf{H}_{2,4}, \mathbf{H}_{21}^{(2)} = \mathbf{H}_{4,2}, \mathbf{H}_{22}^{(2)} = \mathbf{H}_{4,4}$.

Write $\boldsymbol{\beta}$ such that $\boldsymbol{\beta}^{(k')} = \widetilde{\boldsymbol{\beta}}^{(k')}$ for $k' \neq k$. Given $\boldsymbol{\beta}^{(k')} = \widetilde{\boldsymbol{\beta}}^{(k')}$ for $k' \neq k$, the optimal $\boldsymbol{\beta}^{(k)}$ is defined as

$$\underset{\boldsymbol{\beta}^{(k)}}{\operatorname{argmin}} L(\boldsymbol{\beta} \mid \mathbf{D}) + \lambda w_k \|\boldsymbol{\beta}^{(k)}\|_2. \tag{3.14}$$

Unfortunately, there is no closed form solution to (3.14) for a general loss function with general design matrix. We overcome the computational obstacle by taking advantage of the QM condition. From (3.4) we have

$$L(\boldsymbol{\beta} \mid \mathbf{D}) \leq L(\widetilde{\boldsymbol{\beta}} \mid \mathbf{D}) + (\boldsymbol{\beta} - \widetilde{\boldsymbol{\beta}})^\intercal \nabla L(\widetilde{\boldsymbol{\beta}}|\mathbf{D}) + \frac{1}{2}(\boldsymbol{\beta} - \widetilde{\boldsymbol{\beta}})^\intercal \mathbf{H} (\boldsymbol{\beta} - \widetilde{\boldsymbol{\beta}}).$$

Write $U(\widetilde{\boldsymbol{\beta}}) = -\nabla L(\widetilde{\boldsymbol{\beta}}|\mathbf{D})$. Using

$$\boldsymbol{\beta} - \widetilde{\boldsymbol{\beta}} = (\underbrace{0, \ldots, 0}_{k-1}, \boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)}, \underbrace{0, \ldots, 0}_{K-k}),$$

we can write

$$L(\boldsymbol{\beta} \mid \mathbf{D}) \leq L(\widetilde{\boldsymbol{\beta}} \mid \mathbf{D}) - (\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)})^\intercal U^{(k)} + \frac{1}{2}(\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)})^\intercal \mathbf{H}^{(k)} (\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)}). \tag{3.15}$$

Next, let γ_k be the largest eigenvalue of $\mathbf{H}^{(k)}$. Thus we can further relax the upper bound in (3.15) as

$$L(\boldsymbol{\beta} \mid \mathbf{D}) \leq L(\widetilde{\boldsymbol{\beta}} \mid \mathbf{D}) - (\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)})^\intercal U^{(k)} + \frac{1}{2}\gamma_k (\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)})^\intercal (\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)}). \tag{3.16}$$

Instead of minimizing (3.14) we solve

$$\underset{\boldsymbol{\beta}^{(k)}}{\operatorname{argmin}} L(\widetilde{\boldsymbol{\beta}} \mid \mathbf{D}) - (\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)})^\intercal U^{(k)} + \frac{1}{2}\gamma_k (\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)})^\intercal (\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)}) + \lambda w_k \|\boldsymbol{\beta}^{(k)}\|_2. \tag{3.17}$$

Algorithm 6 The BMD algorithm for general group-lasso learning.

- For $k = 1, \ldots, K$, compute γ_k, the largest eigenvalue of $\mathbf{H}^{(k)}$.
- Initialize $\widetilde{\boldsymbol{\beta}}$.
- Repeat the following cyclic blockwise updates until convergence:
  - for $k = 1, \ldots, K$, do (1)–(3)
    - (1) Compute $U(\widetilde{\boldsymbol{\beta}}) = -\nabla L(\widetilde{\boldsymbol{\beta}}|\mathbf{D})$.
    - (2) Compute $\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new}) = \frac{1}{\gamma_k}\left(U^{(k)} + \gamma_k \widetilde{\boldsymbol{\beta}}^{(k)}\right)\left(1 - \frac{\lambda w_k}{\|U^{(k)} + \gamma_k \widetilde{\boldsymbol{\beta}}^{(k)}\|_2}\right)_+$.
    - (3) Set $\widetilde{\boldsymbol{\beta}}^{(k)} = \widetilde{\boldsymbol{\beta}}^{(k)}(\text{new})$.

Denote by $\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new})$ the solution to (3.17). It is straightforward to see that $\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new})$ has a simple closed-from expression

$$\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new}) = \frac{1}{\gamma_k}\left(U^{(k)} + \gamma_k \widetilde{\boldsymbol{\beta}}^{(k)}\right)\left(1 - \frac{\lambda w_k}{\|U^{(k)} + \gamma_k \widetilde{\boldsymbol{\beta}}^{(k)}\|_2}\right)_+. \tag{3.18}$$

Algorithm 6 summarizes the details of BMD.

We can prove the decent property of BMD by using the MM principle [46, 16, 47]. Define

$$Q(\boldsymbol{\beta} \mid \mathbf{D}) = L(\widetilde{\boldsymbol{\beta}} \mid \mathbf{D}) - (\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)})^{\mathsf{T}} U^{(k)} + \frac{1}{2}\gamma_k (\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)})^{\mathsf{T}} (\boldsymbol{\beta}^{(k)} - \widetilde{\boldsymbol{\beta}}^{(k)}) + \lambda w_k \|\boldsymbol{\beta}^{(k)}\|_2. \tag{3.19}$$

Obviously, $Q(\boldsymbol{\beta} \mid \mathbf{D}) = L(\boldsymbol{\beta} \mid \mathbf{D}) + \lambda w_k \|\boldsymbol{\beta}^{(k)}\|_2$ when $\boldsymbol{\beta}^{(k)} = \widetilde{\boldsymbol{\beta}}^{(k)}$ and (3.16) shows that for $\boldsymbol{\beta}^{(k)} \neq \widetilde{\boldsymbol{\beta}}^{(k)}$,

$$Q(\boldsymbol{\beta} \mid \mathbf{D}) > L(\boldsymbol{\beta} \mid \mathbf{D}) + \lambda w_k \|\boldsymbol{\beta}^{(k)}\|_2.$$

By the definition of $\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new})$, we have

$$\begin{aligned} L(\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new}) \mid \mathbf{D}) + \lambda w_k \|\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new})\|_2 &\leq Q(\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new}) \mid \mathbf{D}) \\ &\leq Q(\widetilde{\boldsymbol{\beta}} \mid \mathbf{D}) \\ &= L(\widetilde{\boldsymbol{\beta}} \mid \mathbf{D}) + \lambda w_k \|\widetilde{\boldsymbol{\beta}}^{(k)}\|_2, \end{aligned}$$

which shows that after each update the objective function in (3.13) is driven downhill.

3.4.2 ISTA-BC with variable stepsizes

After the completion of our work we noticed a unpublished manuscript by [37] where a fast algorithm named ISTA-BC was proposed for solving the group-lasso penalized least squares. It is a block-wise extension of the Iterative Shrinkage Thresholding Algorithm (ISTA) [38]. The critical component of ISTA-BC is that it uses a variable stepsize for each block in the descent operation. The stepsizes are computed using a separate backtracking line-search algorithm [37]. To compare ISTA-BC with BMD, it is necessary to mention details of ISTA-BC first. Recall that the optimization problem of the group-lasso penalized least squares is

$$\underset{\beta_0,\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{2n}\|\mathbf{y}-\beta_0-\mathbf{X}\boldsymbol{\beta}\|_2^2+\lambda\sum_{k=1}^{K}\omega_k\|\boldsymbol{\beta}^{(k)}\|_2,$$

During the sub-iteration for updating the k-th block, for a chosen stepsize $T_k>0$, the algorithm considers the following quadratic approximation at the current value of $\widetilde{\boldsymbol{\beta}}^{(k)}$:

$$Q_{T_k}(\boldsymbol{\eta}^{(k)},\widetilde{\boldsymbol{\beta}}^{(k)})=g(\widetilde{\boldsymbol{\beta}})+(\boldsymbol{\eta}^{(k)}-\widetilde{\boldsymbol{\beta}}^{(k)})^{\intercal}\nabla g(\widetilde{\boldsymbol{\beta}})+\frac{T_k}{2}\left\|\boldsymbol{\eta}^{(k)}-\widetilde{\boldsymbol{\beta}}^{(k)}\right\|_2^2+\lambda w_k\|\boldsymbol{\eta}^{(k)}\|_2, \quad (3.20)$$

where $g(\widetilde{\boldsymbol{\beta}})=\frac{1}{2n}\|\mathbf{y}-\tilde{\beta}_0-\mathbf{X}\widetilde{\boldsymbol{\beta}}\|_2^2$. We know (3.20) has a closed form minimizer

$$q_{T_k}(\widetilde{\boldsymbol{\beta}}^{(k)})=\underset{\boldsymbol{\eta}^{(k)}}{\operatorname{argmin}}\, Q_{T_k}(\boldsymbol{\eta}^{(k)},\widetilde{\boldsymbol{\beta}}^{(k)})=\frac{1}{T_k}\left([-\nabla g(\widetilde{\boldsymbol{\beta}})]^{(k)}+T_k\widetilde{\boldsymbol{\beta}}^{(k)}\right)\left(1-\frac{\lambda w_k}{\|[-\nabla g(\widetilde{\boldsymbol{\beta}})]^{(k)}+T_k\widetilde{\boldsymbol{\beta}}^{(k)}\|_2}\right)_+.$$

The stepsize is computed using backtracking line-search. It is the smallest T_k such that the following descent condition is satisfied

$$g(\boldsymbol{\beta}=[\cdots q_{T_k}(\widetilde{\boldsymbol{\beta}}^{(k)})\cdots])+\lambda\omega_k\|q_{T_k}(\widetilde{\boldsymbol{\beta}}^{(k)})\|_2\le Q_{T_k}(q_{T_k}(\widetilde{\boldsymbol{\beta}}^{(k)}),\widetilde{\boldsymbol{\beta}}^{(k)}), \quad (3.21)$$

where in (3.21) $\boldsymbol{\beta}=[\cdots q_{T_k}(\widetilde{\boldsymbol{\beta}}^{(k)})\cdots]$ has $\boldsymbol{\beta}^{(k')}=\widetilde{\boldsymbol{\beta}}^{(k')}$ for $k'\neq k$. Then ISTA-BC updates the estimates of the k-th block using $\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new})=q_{T_k}(\widetilde{\boldsymbol{\beta}}^{(k)})$. The complete algorithm of ISTA-BC with backtracking line-search is presented in Algorithm 7.

Conceptually, ISTA-BC is a block-wise extension of the ISTA, while BMD is a combination of MM principle and block-wise descent. Computationally, ISTA-BC uses backtracking line search for computing update step-lengths, while BMD's update is much simpler. Numerically, as will be demonstrated in the next section, BMD is generally faster and more accurate than ISTA-BC.

Algorithm 7 ISTA-BC with backtracking line-search

- **Step 0.** During the sub-iteration for updating block k, let $T_k[0] > 0$ and $a = 2$.
- **Step m.** Repeatedly compute $T_k[m] \equiv aT_k[m-1]$ until T_k satisfies the following condition (3.21).
- Update $\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new}) = q_{T_k}(\widetilde{\boldsymbol{\beta}}^{(k)})$.

3.4.3 Implementation

We have implemented Algorithm 6 for solving the group-lasso penalized least squares, logistic regression, Huberized SVM and squared SVM. These functions are contained in an R package `gglasso` publicly available from the Comprehensive R Archive Network (CRAN) at `http://cran.r-project.org/web/packages/gglasso`. We always include the intercept term in the model. Without loss of generality we always center the design matrix beforehand.

We solve each group-lasso model for a sequence of λ values from large to small. The default number of points is 100. Let $\lambda[l]$ denote these grid points. We use the warm-start trick to implement the solution path, that is, the computed solution at $\lambda = \lambda[l]$ is used as the initial value for using Algorithm 6 to compute the solution at $\lambda = \lambda[l+1]$. We define $\lambda[1]$ as the smallest λ value such that all predictors have zero coefficients, except the intercept. In such a case let $\hat{\beta}_1$ be the optimal solution of the intercept. Then the solution at $\lambda[1]$ is $\widehat{\boldsymbol{\beta}}[1] = (\hat{\beta}_1, 0, \ldots, 0)$ as the null model estimates. By the Karush-Kuhn-Tucker conditions we can find that

$$\lambda[1] = \max_{k=1,\ldots,K} \left\| [\nabla L(\widehat{\boldsymbol{\beta}}[1]|\mathbf{D})]^{(k)} \right\|_2 / w_k, \quad w_k \neq 0.$$

For least squares and logistic regression models, $\hat{\beta}_1$ has a simple expression:

$$\hat{\beta}_1(\text{LS}) = \frac{\sum_{i=1}^n \tau_i y_i}{\sum_{i=1}^n \tau_i} \qquad \text{group-lasso penalized least squares} \tag{3.22}$$

$$\hat{\beta}_1(\text{Logit}) = \log\left(\frac{\sum_{y_i=1} \tau_i}{\sum_{y_i=-1} \tau_i}\right) \qquad \text{group-lasso penalized logistic regression} \tag{3.23}$$

For the other two models, we use the following iterative procedure to solve for $\hat{\beta}_1$:

1. Initialize $\hat{\beta}_1 = \hat{\beta}_1(\text{Logit})$ in large margin classifiers.
2. Compute $\hat{\beta}_1(\text{new}) = \hat{\beta}_1 - \frac{1}{\gamma_1}\nabla L((\hat{\beta}_1, 0, \ldots, 0)|\mathbf{D})_1$ where $\gamma_1 = \frac{1}{n}\sum_{i=1}^{n}\tau_i$.
3. Let $\hat{\beta}_1 = \hat{\beta}_1(\text{new})$.
4. Repeat 2-3 until convergence.

For computing the solution at each λ we also utilize the strong rule introduced in [48]. Suppose that we have computed $\widehat{\boldsymbol{\beta}}(\lambda[l])$, the solution at $\lambda[l]$. To compute the solution at $\lambda[l+1]$, before using Algorithm 6 we first check if group k satisfies the following inequality:

$$\left\| [\nabla L(\widehat{\boldsymbol{\beta}}(\lambda[l])|\mathbf{D})]^{(k)} \right\|_2 \geq w_k(2\lambda[l+1] - \lambda[l]). \tag{3.24}$$

Let $S = \{j : j \in I_k, \text{ group k passes the check in (3.24)}\}$. We use Algorithm 6 to solve the group-lasso model with a reduced data set $\{\mathbf{y}, \mathbf{X}_S\}$ where $\mathbf{X}_S = [\cdots X_j \cdots]$, $j \in S$ corresponds to the design matrix with only the groups of variables in S. Suppose the solution is $\widehat{\boldsymbol{\beta}}_S$. If $\widehat{\boldsymbol{\beta}} = (\widehat{\boldsymbol{\beta}}_S, \mathbf{0})$ satisfies the KKT condition, then we have found the desired solution. If not, any group in which there is at least one variable that violates the KKT condition should be added to S, and the procedure continues until S does not change.

In Algorithm 6 we use a simple updating formula to compute $\nabla L(\widetilde{\boldsymbol{\beta}}|\mathbf{D})$, because it only depends on $R = \mathbf{y} - \mathbf{X}\widetilde{\boldsymbol{\beta}}$ for regression and $R = \mathbf{y} \cdot \mathbf{X}\widetilde{\boldsymbol{\beta}}$ for classification. After updating $\widetilde{\boldsymbol{\beta}}^{(k)}$, for regression we can update R by $R - \mathbf{X}^{(k)}(\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new}) - \widetilde{\boldsymbol{\beta}}^{(k)})$, for classification update R by $R + \mathbf{y} \cdot \mathbf{X}^{(k)}(\widetilde{\boldsymbol{\beta}}^{(k)}(\text{new}) - \widetilde{\boldsymbol{\beta}}^{(k)})$. In cyclic blockwise descent updating, we do that on the active-set first which contains those groups whose current coefficients are nonzero. After BMD is converged on the active-set, we then run a complete cycle to see if any group is to be included into the active-set. If not, the algorithm is stopped. Otherwise, BMD is repeated on the updated active-set.

In order to make a fair comparison to `ISTA-BC`, `grplasso` and `SLEP`, we tested three different convergence criteria in `gglasso`:

1. $\max_j \frac{|\tilde{\beta}_j(\text{current}) - \tilde{\beta}_j(\text{new})|}{1 + |\tilde{\beta}_j(\text{current})|} < \epsilon$, for $j = 1, 2 \ldots, p$.
2. $\left\| \widetilde{\boldsymbol{\beta}}(\text{current}) - \widetilde{\boldsymbol{\beta}}(\text{new}) \right\|_2 < \epsilon$.

3. $\max_k(\gamma_k) \cdot \max_j \frac{|\tilde{\beta}_j(\text{current}) - \tilde{\beta}_j(\text{new})|}{1+|\tilde{\beta}_j(\text{current})|} < \epsilon$, for $j = 1, 2 \ldots, p$ and $k = 1, 2 \ldots, K$.

Convergence criterion 1 is used in `grplasso` and convergence criterion 2 is used in `SLEP`. We also implemented `ISTA-BC` algorithm [37] using both criterion 1 (`ISTA-BC` (LS1)) and 2 (`ISTA-BC` (LS2)). For the group-lasso penalized least squares and logistic regression, we used both convergence criteria 1 and 2 in `gglasso`. For the group-lasso penalized Huberized SVM and squared SVM, we used convergence criterion 3 in `gglasso`. Compared to criterion 1, criterion 3 uses an extra factor $\max_k(\gamma_k)$ in order to take into account the observation that $\widetilde{\boldsymbol{\beta}}^{(k)}(\text{current}) - \widetilde{\boldsymbol{\beta}}^{(k)}(\text{new})$ depends on $\frac{1}{\gamma_k}$. The default value for ϵ is 10^{-4}.

3.5 Numerical Examples

In this section, we use simulation and real data to demonstrate the efficiency of the BMD algorithm in terms of timing performance and solution accuracy. All numerical experiments were carried out on an Intel Xeon X5560 (Quad-core 2.8 GHz) processor. In this section, let `gglasso` (LS1) and `gglasso` (LS2) denote the group-lasso penalized least squares solutions computed by `gglasso` where the convergence criterion is criterion 1 and criterion 2, respectively. Likewise, we define `gglasso` (Logit1) and `gglasso` (Logit2) for the group-lasso penalized logistic regression. Similar notation is applied to `SLEP`, `grplasso` and `ISTA-BC`.

3.5.1 Timing comparison

We design a simulation model by combining the *FHT* model introduced in [36] and the simulation model 3 in [23]. We generate original predictors X_j, $j = 1, 2 \ldots, q$ from a multivariate normal distribution with a compound symmetry correlation matrix such that the correlation between X_j and $X_{j'}$ is ρ for $j \neq j'$. Let

$$Y^* = \sum_{j=1}^{q} (\frac{2}{3} X_j - X_j^2 + \frac{1}{3} X_j^3) \beta_j,$$

where $\beta_j = (-1)^j \exp(-(2j-1)/20)$. When fitting a group-lasso model, we treat $\{X_j, X_j^2, X_j^3\}$ as a group, so the final predictor matrix has the number of variables $p = 3q$.

For regression data we generate a response $Y = Y^* + k \cdot e$ where the error term e is generated from $N(0, 1)$. k is chosen such that the signal-to-noise ratio is 3.0. For classification data we generate the binary response Y according to

$$\Pr(\mathrm{Y} = -1) = 1/(1 + \exp(-Y^*)), \quad \Pr(\mathrm{Y} = +1) = 1/(1 + \exp(Y^*)).$$

We considered the following combinations of (n, p):

Scenario 1. $(n, p) = (100, 3000)$ and $(n, p) = (300, 9000)$.

Scenario 2. $n = 200$ and $p = 600, 1200, 3000, 6000, 9000$, shown in Figure 3.2.

For each (n, p, ρ) combination we recorded the timing (in seconds) of computing the solution paths at 100 λ values of each group-lasso penalized model by `gglasso`, `ISTA-BC`, `SLEP` and `grplasso`. The results was averaged over 10 independent runs.

Table 3.2 shows results from Scenario 1. We see that `gglasso` has the best timing performance. In the group-lasso penalized least squares case, `gglasso` (LS2) is about 12 times faster than `SLEP` (LS2) and is about 2 times faster than `ISTA-BC` (LS2). In the group-lasso penalized logistic regression case, `gglasso` (Logit2) is about 2-6 times faster than `SLEP` (Logit2) and `gglasso` (Logit1) is about 5-10 times faster than `grplasso` (Logit1).

Figure 3.2 shows results from Scenario 2 in which we examine the impact of dimension on the timing of `gglasso`. We fixed n at 200 and plotted the run time (in log scale) against p for three correlation levels 0.2, 0.5 and 0.8. We see that higher ρ increases the timing of `gglasso` in general. For each fixed correlation level, the timing increases linearly with the dimension.

3.5.2 Quality comparison

In this section we show that `gglasso` is also more accurate than `ISTA-BC`, `grplasso` and `SLEP` under the same convergence criterion. We test the accuracy of solutions by checking their KKT conditions. Theoretically, $\boldsymbol{\beta}$ is the solution of (3.3) if and only if the following KKT conditions hold:

$$\begin{aligned} \left[\nabla L(\boldsymbol{\beta}|\mathbf{D})\right]^{(k)} + \lambda w_k \cdot \frac{\boldsymbol{\beta}^{(k)}}{\|\boldsymbol{\beta}^{(k)}\|_2} = \mathbf{0} \qquad & \text{if } \boldsymbol{\beta}^{(k)} \neq \mathbf{0}, \\ \left\| \left[\nabla L(\boldsymbol{\beta}|\mathbf{D})\right]^{(k)} \right\|_2 \leq \lambda w_k \qquad & \text{if } \boldsymbol{\beta}^{(k)} = \mathbf{0}, \end{aligned}$$

where $k = 1, 2, \ldots, K$. The theoretical solution for the convex optimization problem (3.3) should be unique and always passes the KKT condition check. However, a numerical solution could only approach this analytical value within certain precision therefore may fail the KKT check. Numerically, we declare $\boldsymbol{\beta}^{(k)}$ passes the KKT condition check if

$$\begin{aligned} \left\| [\nabla L(\boldsymbol{\beta}|\mathbf{D})]^{(k)} + \lambda w_k \cdot \frac{\boldsymbol{\beta}^{(k)}}{\|\boldsymbol{\beta}^{(k)}\|_2} \right\|_2 \leq \varepsilon \qquad & \text{if } \boldsymbol{\beta}^{(k)} \neq \mathbf{0}, \\ \left\| [\nabla L(\boldsymbol{\beta}|\mathbf{D})]^{(k)} \right\|_2 \leq \lambda w_k + \varepsilon \qquad & \text{if } \boldsymbol{\beta}^{(k)} = \mathbf{0}, \end{aligned}$$

for a small $\varepsilon > 0$. In this chapter we set $\varepsilon = 10^{-4}$.

For the solutions of the FHT model scenario 1 computed in section 3.5.1, we also calculated the number of coefficients that violated the KKT condition check at each λ value. Then this number was averaged over the 100 values of λs. This process was then repeated 10 times on 10 independent datasets. As shown in Table 3.3, in the group-lasso penalized least squares case, `gglasso` (LS1) has zero violation count compared with non-zero violation count of `ISTA-BC` (LS1); `gglasso` (LS2) also has smaller violation counts compared with `ISTA-BC` (LS2) and `SLEP` (LS2). In the group-lasso penalized classification cases, `gglasso` (Logit1) has less KKT violation counts than `grplasso` (Logit1) does when both use convergence criterion 1, and `gglasso` (Logit2) has less KKT violation counts than `SLEP` (Logit2) when both use convergence criterion 2. Overall, it is clear that `gglasso` is numerically more accurate than `ISTA-BC`, `grplasso` and `SLEP`. In addition, gglasso (HSVM) and `gglasso` (SqSVM) both pass KKT checks without any violation.

3.5.3 Real data analysis

In this section we compare `gglasso`, `ISTA-BC`, `grplasso` and `SLEP` on several real data examples. Table 5.5 summarizes the datasets used in this section. We fit a sparse

Timing Comparison						
Data	$n = 100$ $p = 3000$			$n = 300$ $p = 9000$		
ρ	0.2	0.5	0.8	0.2	0.5	0.8
Regression						
ISTA-BC (LS1)	3.00	3.59	7.15	11.25	32.14	64.57
gglasso (LS1)	1.05	1.33	3.57	5.53	18.62	38.84
SLEP (LS2)	7.31	8.90	10.76	31.39	40.53	36.56
ISTA-BC (LS2)	1.14	1.14	1.54	5.47	5.77	4.95
gglasso (LS2)	0.60	0.60	0.63	2.93	2.98	2.80
Classification						
grplasso (Logit1)	31.78	38.19	58.45	111.70	158.44	239.67
gglasso (Logit1)	3.16	5.65	10.39	19.42	22.98	37.39
SLEP (Logit2)	5.50	5.86	2.39	24.68	22.14	6.80
gglasso (Logit2)	0.95	0.95	0.76	6.11	5.35	3.68
gglasso (HSVM)	4.36	8.60	14.63	24.46	33.77	65.29
gglasso (SqSVM)	5.35	10.21	15.32	30.80	41.00	73.68

Table 3.2: The *FHT* model scenario 1. Reported numbers are timings (in seconds) of **gglasso**, **ISTA-BC**, **grplasso** and **SLEP** for computing solution paths at 100 λ values using the group-lasso penalized least squares, logistics regression, Huberized SVM and squared SVM models. Results are averaged over 10 independent runs.

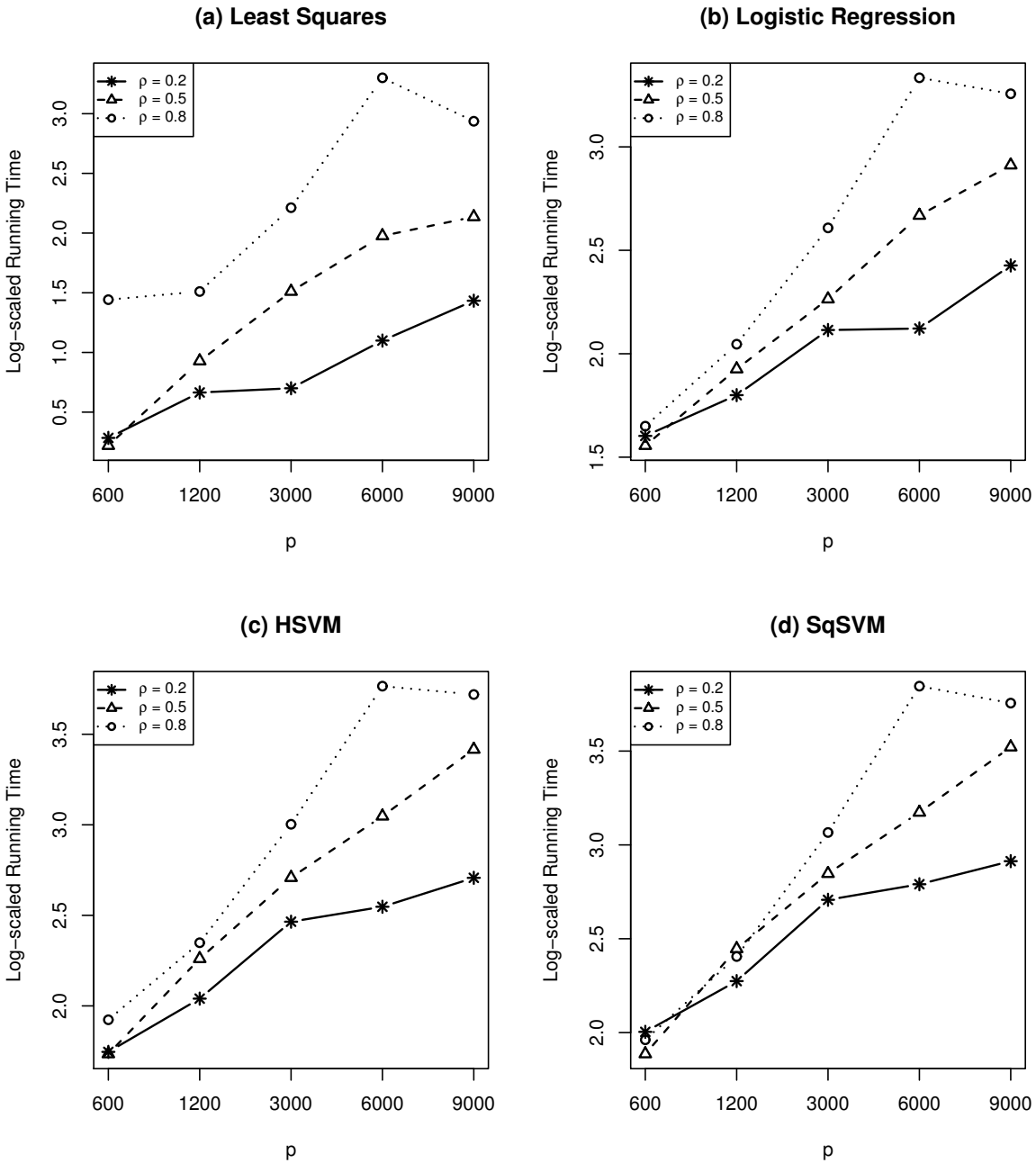

Figure 3.2: The *FHT* model scenario 2. The average running time of 10 independent runs (in the natural logarithm scale) of **gglasso** for computing solution paths of (a) least squares; (b) logistic regression; (c) Huberized SVM; (d) squared SVM. In all cases $n = 200$.

Quality Comparison: KKT Condition Check						
Data	$n = 100$ $p = 3000$			$n = 300$ $p = 9000$		
ρ	0.2	0.5	0.8	0.2	0.5	0.8
Regression						
ISTA-BC (LS1)	2	2	0	17	4	0
gglasso (LS1)	0	0	0	0	0	0
SLEP (LS2)	23	21	17	53	48	30
ISTA-BC (LS2)	26	22	16	60	48	27
gglasso (LS2)	23	21	16	43	46	27
Classification						
grplasso (Logit1)	0	5	17	0	28	37
gglasso (Logit1)	0	0	0	0	0	0
SLEP (Logit2)	7	23	29	15	70	143
gglasso (Logit2)	9	24	24	4	50	48
gglasso (HSVM)	0	0	0	0	0	0
gglasso (SqSVM)	0	0	0	0	0	0

Table 3.3: The *FHT* model scenario 1. Reported numbers are the average number of coefficients among p coefficients that violated the KKT condition check (rounded down to the next smaller integer) using **gglasso**, **ISTA-BC**, **grplasso** and **SLEP**. Results are averaged over the λ sequence of 100 values and averaged over 10 independent runs.

Dataset	Type	n	q	p	Data Source
Autompg	R	392	7	31	[49]
Bardet	R	120	200	1000	[50]
Cardiomypathy	R	30	6319	31595	[51]
Spectroscopy	R	103	100	500	[52]
Breast	C	42	22283	111415	[43]
Colon	C	62	2000	10000	[28]
Prostate	C	102	6033	30165	[29]
Sonar	C	208	60	300	[53]

Table 3.4: Real Datasets. n is the number of instances. q is the number of original variables. p is the number of predictors after expansion. "R" means regression and "C" means classification.

additive regression model for the regression-type data and fit a sparse additive logistic regression model for the classification-type data. The group-lasso penalty is used to select important additive components. All data were standardized in advance such that each original variable has zero mean and unit sample variance. Some datasets contain only numeric variables but some datasets have both numeric and categorical variables. For any categorical variable with M levels of measurement, we recoded it by $M-1$ dummy variables and treated these dummy variables as a group. For each continuous variable, we used five B-Spline basis functions to represent its effect in the additive model. Those five basis functions are considered as a group. For example, in Colon data the original data have 2000 numeric variables. After basis function expansion there are 10000 predictors in 2000 groups.

For each dataset, we report the average timings of 10 independent runs for computing the solution paths at 100 λ values. We also report the average number of the KKT check violations. The results are summarized in Table 4.5. It is clear that `gglasso` outperforms `ISTA-BC`, `grplasso` and `SLEP`.

Before ending this section we would like to use a real data example to demonstrate why the group-lasso could be advantageous over the lasso. On sonar data we compared the lasso penalized logistic regression and the group-lasso penalized logistic regression.

We randomly split the sonar data into training and test sets according to 4:1 ratio, and found the optimal λ for each method using five-fold cross-validation on the training data. Then we calculated the misclassification error rate on the test set. We used `glmnet` [36] to compute the lasso penalized logistic regression. The process was repeated 100 times. In the group-lasso model, we define the group-wise L_2 coefficient norm $\theta_j(\lambda)$ for the jth variable by $\theta_j(\lambda) = \sqrt{\sum_{i=1}^{5} \hat{\beta}_{ji}^2(\lambda)}$. Then the jth variable enters the final model if and only if $\theta_j(\lambda) \neq 0$. Figure 3.3 shows the solution paths of the tuned lasso and group-lasso logistic model from one run, where in the group-lasso plot we plot $\theta_j(\lambda)$ against $\log \lambda$. To make a more direct comparison, we also plot the absolute value of each coefficient in the lasso plot. The fitted lasso logistic regression model selected 40 original variables while the group-lasso logistic regression model selected 26 original variables. When looking at the average misclassification error of 100 runs, we see that the group-lasso logistic regression model is significantly more accurate than the lasso logistic regression model. Note that the sample size is 208 in the Sonar data, thus the misclassification error calculation is meaningful.

Supplementary materials

Our methods have been implemented in an R package gglasso publicly available from the Comprehensive R Archive Network (CRAN) at http://cran.r-project.org/web/packages/gglasso.

Group-lasso Regression on Real Data				
Dataset	*Autompg*	*Bardet*	*Cardiomypathy*	*Spectroscopy*
	Sec. — KKT	Sec. — KKT	Sec. — KKT	Sec. — KKT
ISTA-BC (LS1)	4.46 — 0	0.76 — 4	2.67 — 0	0.96 — 0
gglasso (LS1)	1.79 — 1	0.43 — 2	2.68 — 0	0.50 — 0
SLEP (LS2)	3.14 — 0	9.96 — 0	78.23 — 0	9.37 — 0
ISTA-BC (LS2)	5.66 — 0	1.55 — 0	2.43 — 0	1.31 — 0
gglasso (LS2)	2.51 — 0	0.77 — 0	2.48 — 0	0.76 — 0

Group-lasso Classification on Real Data				
Dataset	*Colon*	*Prostate*	*Sonar*	*Breast*
	Sec. — KKT	Sec. — KKT	Sec. — KKT	Sec. — KKT
grplasso (Logit1)	60.42 — 0	111.75 — 0	24.55 — 0	439.76 — 0
gglasso (Logit1)	1.13 — 0	3.877 — 0	1.54 — 0	9.62 — 0
SLEP (Logit2)	75.31 — 0	166.91 — 0	5.49 — 0	358.75 — 0
gglasso (Logit2)	2.23 — 0	4.36 — 0	2.88 — 0	10.24 — 0
gglasso (HSVM)	1.15 — 0	3.53 — 0	0.66 — 0	9.15 — 0
gglasso (SqSVM)	1.45 — 0	3.79 — 0	1.27 — 1	9.58 — 0

Table 3.5: Group-lasso penalized regression and classification on real datasets. Reported numbers are: (a) timings (in seconds), total time for 100 λ values; (b) the average number of coefficients among p coefficients that violated the KKT condition check. Results are averaged over 10 independent runs.

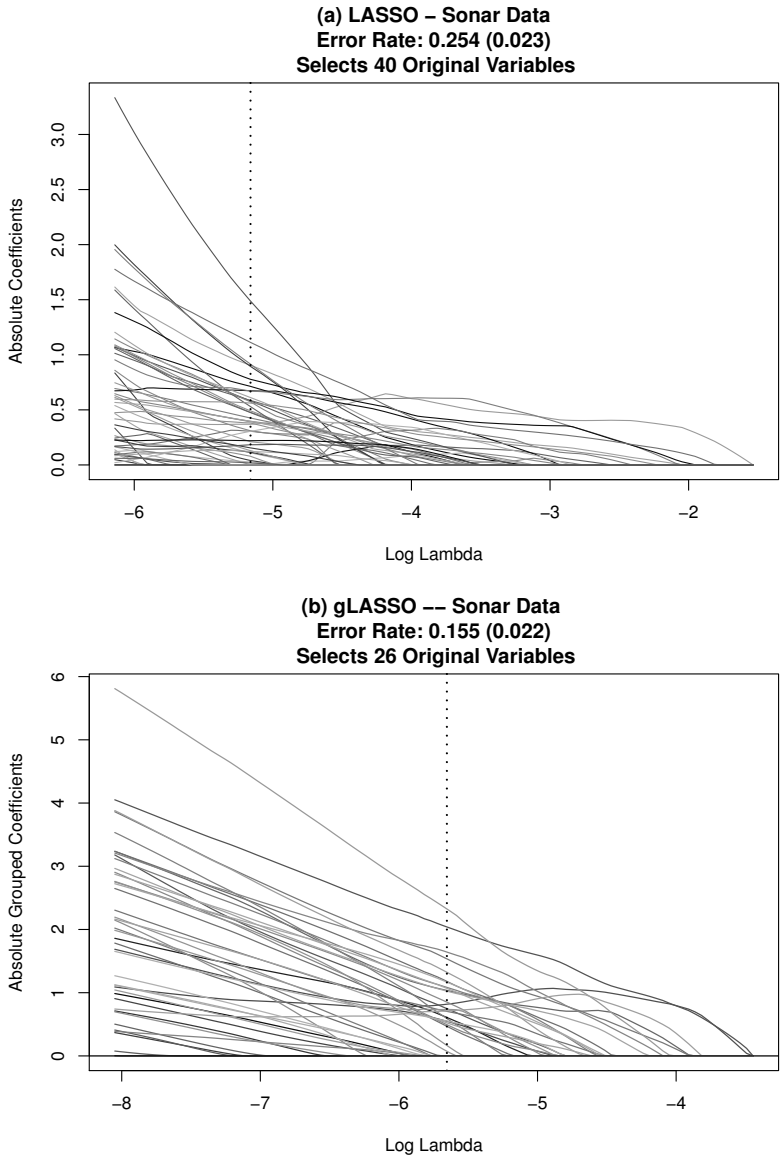

Figure 3.3: Compare the lasso penalized logistic regression and the group-lasso penalized logistic regression on Sonar data with $n = 208$ and $p = 60$. (a) solution paths of lasso penalized logistic regression model, in which each absolute coefficient $|\beta_j|$ is plotted against $\log \lambda$ for $j = 1, \ldots, p$. (b) solution paths of group-lasso penalized logistic regression model with $p^* = 300$ expanded variables grouped into $K = 60$ groups, in which each group norm $\|\boldsymbol{\beta}^{(k)}\|_2$ is plotted against $\log \lambda$ for $k = 1, \ldots, K$. The vertical dotted lines indicate the best models chosen by cross-validation.

Chapter 4

An Efficient Algorithm for The HHSVM and Its Generalizations

4.1 Chapter Overview

The hybrid Huberized support vector machine (HHSVM) has proved its advantages over the ℓ_1 support vector machine (SVM) in terms of classification and variable selection. Similar to the ℓ_1 SVM, the HHSVM enjoys a piecewise linear path property and can be computed by a LARS-type piecewise linear solution path algorithm. In this chapter we propose a generalized coordinate descent (GCD) algorithm for computing the solution path of the HHSVM. The GCD algorithm takes advantage of a majorization-minimization trick to make each coordinate-wise update simple and efficient. Extensive numerical experiments show that the GCD algorithm is much faster than the LARS-type path algorithm. We further extend the GCD algorithm to solve a class of elastic net penalized large margin classifiers, demonstrating the generality of the GCD algorithm. We have implemented the GCD algorithm in a publicly available R package `gcdnet`.

4.2 Introduction

The support vector machine [54] is a very popular large margin classifier. Despite its competitive performance in terms of classification accuracy, a major limitation of the support vector machine is that it cannot automatically select relevant variables for

classification, which is very important in high-dimensional classification problems such as tumor classification with microarrays. The SVM has an equivalent formulation as the ℓ_2 penalized hinge loss [55]. [56] proposed the ℓ_1 SVM by replacing the ℓ_2 penalty in the SVM with the ℓ_1 penalty. The ℓ_1 penalization (a.k.a. lasso) [20] is a very popular technique for achieving sparsity with high dimensional data. There has been a large body of theoretical work to support the ℓ_1 regularization. A comprehensive reference is [57]. [19] later proposed a hybrid Huberized support vector machine (HHSVM) which uses the elastic net penalty [2] for regularization and variable selection and uses the Huberized squared hinge loss for efficient computation. [19] showed that the HHSVM outperforms the standard SVM and the ℓ_1 SVM for high-dimensional classification. [19] extended the LARS algorithm for the lasso regression model [1, 32, 3] to compute the solution paths of the HHSVM.

The main purpose of this chapter is to introduce a new efficient algorithm for computing the HHSVM. Our study was motivated by the recent success of using coordinate descent algorithms for computing the elastic net penalized regression and logistic regression [12]. See also [58]. For the lasso regression, the coordinate descent algorithm amounts to an iterative cyclic soft-thresholding operation. Despite its simplicity, the coordinate descent algorithm can even outperform the LARS algorithm, especially when the dimension is much larger than the sample size. See Table 1 of [12]. Other papers on coordinate descent algorithms for the lasso include [8], [25], [34], [11], [10] and among others. The HHSVM poses a major challenge for applying the coordinate descent algorithm, because the Huberized hinge loss function does not have a smooth first derivative everywhere. As a result, the coordinate descent algorithm for the elastic net penalized logistic regression [12] cannot be used for solving the HHSVM.

To overcome the computational difficulty, we propose a new generalized coordinate descent (GCD) algorithm for solving the solution paths of the HHSVM. We also give a further extension of the GCD algorithm to general problems. Our algorithm adopts the Majorization-Minimization (MM) principle into the coordinate decent loop. We use a majorization-minimization trick to make each coordinate-wise update simple and efficient. In addition, the MM principle ensures the descent property of the GCD algorithm which is crucial for all coordinate decent algorithms. Extensive numerical examples show that the GCD can be much faster than the LARS-type path algorithm in [19]. Here we

use the prostate cancer data [29] to briefly illustrate the speed advantage of our algorithm. See Figure 5.1. The prostate data have 102 observations and each has 6033 gene expression values. It took the LARS-type path algorithm about 5 minutes to compute the HHSVM paths, while the GCD used only 3.5 seconds to get the identical solution paths.

A closer examination of the GCD algorithm reveals that it can be used for solving other large margin classifiers. We have derived a generic GCD algorithm for solving a class of elastic net penalized large margin classifiers. We further demonstrate the generic GCD algorithm by considering the squared SVM loss and the logistic regression loss. We study through numeric examples how the shape and smoothness of the loss function affect the computational speed of the generic GCD algorithm.

4.3 The HHSVM and GCD Algorithm

4.3.1 The HHSVM

In a standard binary classification problem we are given n pairs of training data $\{\mathbf{x}_i, y_i\}$ for $i = 1, \ldots, n$ where $\mathbf{x}_i \in \mathbb{R}^p$ are predictors and $y_i \in \{-1, 1\}$ denotes class labels. The linear support vector machine (SVM) [54, 59, 60] looks for the hyperplane with the largest margin that separates the input data for class 1 and -1

$$\begin{aligned} &\min_{(\beta_0,\boldsymbol{\beta})} \tfrac{1}{2}||\boldsymbol{\beta}||^2 + \gamma \textstyle\sum_{i=1}^n \xi_i \\ &\text{subject to } \xi_i \geq 0,\ y_i(\beta_0 + \mathbf{x}_i^\intercal \boldsymbol{\beta}) \geq 1 - \xi_i\ \forall i, \end{aligned} \tag{4.1}$$

where $\xi = (\xi_1, \xi_2, \ldots, \xi_n)$ are the slack variables and $\gamma > 0$ is a constant. Let $\lambda = 1/(2\gamma)$. Then (4.1) has an equivalent ℓ_2 penalized hinge loss formulation [55]

$$\min_{(\beta_0,\boldsymbol{\beta})} \frac{1}{n} \sum_{i=1}^n [1 - y_i(\beta_0 + \mathbf{x}_i^\intercal \boldsymbol{\beta})]_+ + \frac{\lambda}{2}||\boldsymbol{\beta}||_2^2. \tag{4.2}$$

The loss function $L(t) = [1 - t]_+$ has the expression

$$[1 - t]_+ = \begin{cases} 0, & t > 1 \\ 1 - t, & t \leq 1, \end{cases}$$

which is called the hinge loss in the literature. The SVM has very competitive performance in terms of classification. However, because of the ℓ_2 penalty the SVM uses

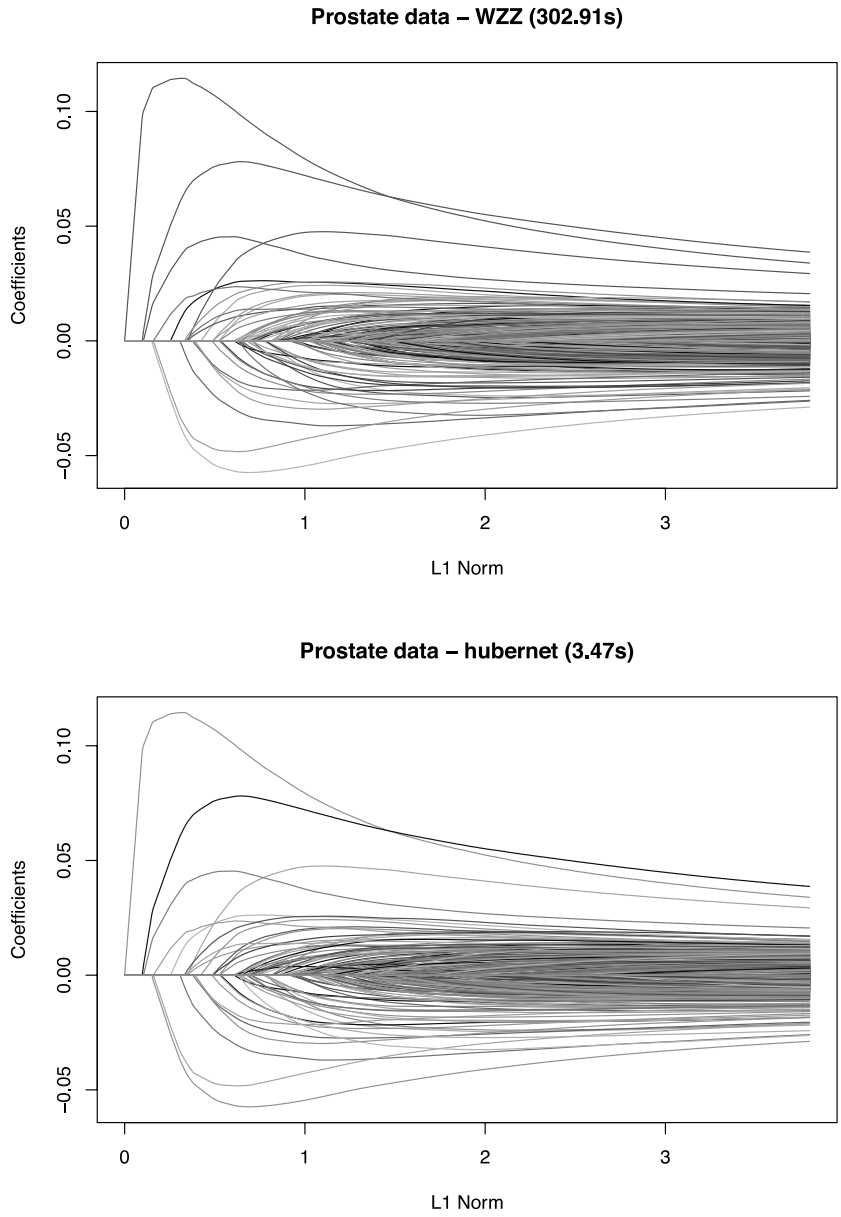

Figure 4.1: Solution paths and timings of the HHSVM on the prostate cancer data with 102 observations and 6033 predictors. The top panel shows the solution paths computed by the LARS-type algorithm in [19]; the bottom panel shows the solution paths computed by GCD. GCD is 87 times faster.

all variables in classification, which could be a great disadvantage in high-dimensional classification [61]. The ℓ_1 SVM proposed by [56] is defined by

$$\min_{(\beta_0,\boldsymbol{\beta})} \frac{1}{n}\sum_{i=1}^{n}[1-y_i(\beta_0+\mathbf{x}_i^{\intercal}\boldsymbol{\beta})]_+ + \frac{\lambda}{2}||\boldsymbol{\beta}||_1. \tag{4.3}$$

Just like in the lasso regression model, the ℓ_1 penalty produces a sparse $\boldsymbol{\beta}$ in (4.3). Thus the ℓ_1 SVM is able to automatically discard irrelevant features. In the presence of many noise variables, the ℓ_1 SVM has significant advantages over the standard SVM [61].

The elastic net penalty [2] is an important generalization of the lasso penalty. The elastic net penalty is defined as

$$P_{\lambda_1,\lambda_2}(\boldsymbol{\beta}) = \sum_{j=1}^{p} p_{\lambda_1,\lambda_2}(\beta_j) = \sum_{j=1}^{p}\left(\lambda_1|\beta_j| + \frac{\lambda_2}{2}\beta_j^2\right), \tag{4.4}$$

where $\lambda_1, \lambda_2 \geq 0$ are regularization parameters. The ℓ_1 part of the elastic net is responsible for variable selection. The ℓ_2 part of the elastic net helps handle strong correlated variables, which is common in high-dimensional data, and improves prediction.

[19] used the elastic net in the SVM classification:

$$\min_{(\beta_0,\boldsymbol{\beta})} \frac{1}{n}\sum_{i=1}^{n}\phi_c(y_i(\beta_0+\mathbf{x}_i^{\intercal}\boldsymbol{\beta})) + P_{\lambda_1,\lambda_2}(\boldsymbol{\beta}). \tag{4.5}$$

Note that $\phi_c(\cdot)$ in (4.5) is the Huberized hinge loss

$$\phi_c(t) = \begin{cases} 0, & t > 1 \\ (1-t)^2/2\delta, & 1-\delta < t \leq 1 \\ 1-t-\delta/2, & t \leq 1-\delta, \end{cases}$$

where $\delta > 0$ is a pre-specific constant. The default choice for δ is 2 unless specified otherwise. Displayed in Figure 4.2 panel (a) is the Huberized hinge loss with $\delta = 2$. The Huberized hinge loss is very similar to the hinge loss in shape. In fact, when δ is small, the two loss functions are almost identical. See Figure 4.2 panel (b) for a graphical illustration. Unlike the hinge loss, the Huberized hinge loss function is differentiable everywhere and has continuous first derivative. [19] derived a LARS-type path algorithm for computing the solution paths of the HHSVM. Compared with the LARS-type algorithm for the ℓ_1 SVM [61], the LARS-type algorithm for the HHSVM has significantly lower computational cost, thanks to the differentiability property of the Huberized hinge loss.

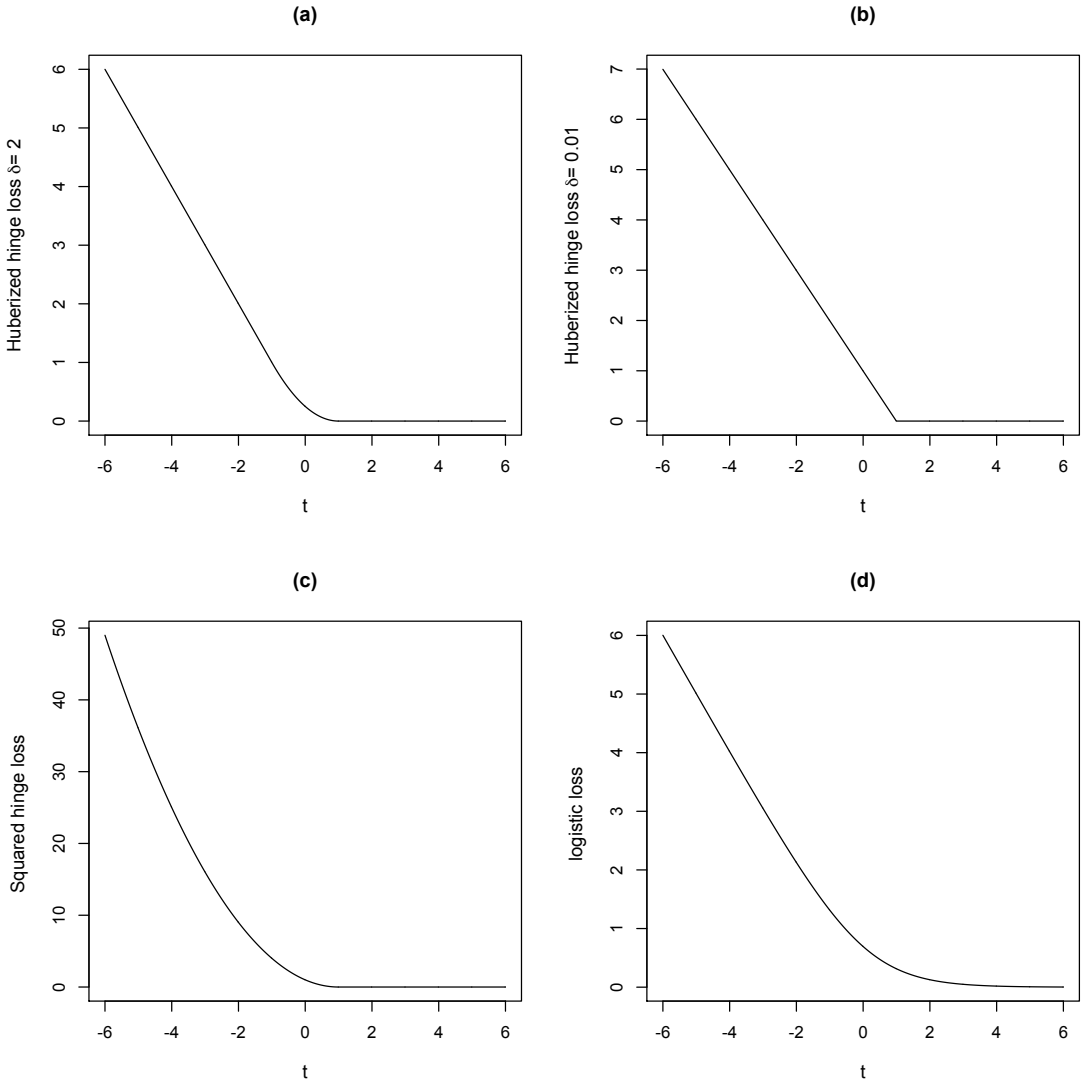

Figure 4.2: (a) The Huberized hinge loss function (with $\delta = 2$); (b) The Huberized hinge loss function (with $\delta = 0.01$); (c) The squared hinge loss function; (d) The logistic loss function.

4.3.2 A generalized coordinate descent algorithm

Besides the LARS-type algorithm, coordinate descent algorithms have been successfully used to compute the elastic net penalized linear model and generalized linear models. See the R package `glmnet` by [12]. Despite its simplicity, the coordinate descent can even outperform the more sophisticated LARS algorithm [12]. Motivated by the great success of `glmnet`, we consider developing a fast coordinate descent algorithm for computing the HHSVM with the goal to outperform the LARS-type algorithm derived in [19].

Without loss of generality assume the input data are standardized: $\frac{1}{n}\sum_{i=1}^{n} x_{ij} = 0$, $\frac{1}{n}\sum_{i=1}^{n} x_{ij}^2 = 1$, for $j = 1, \ldots, p$. For the HHSVM we can write down the standard coordinate decent algorithm as follows. Define the current margin $r_i = y_i(\tilde{\beta}_0 + \mathbf{x}_i^\intercal \tilde{\boldsymbol{\beta}})$ and

$$F(\beta_j|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}}) = \frac{1}{n}\sum_{i=1}^{n} \phi_c(r_i + y_i x_{ij}(\beta_j - \tilde{\beta}_j)) + p_{\lambda_1,\lambda_2}(\beta_j). \tag{4.6}$$

For fixed λ_1 and λ_2, the standard coordinate descent algorithm [18] proceeds as follows:

1. Initialization: $(\tilde{\beta}_0, \tilde{\boldsymbol{\beta}})$
2. Cyclic coordinate descent: for $j = 0, 1, 2, \ldots, p$: update $\tilde{\beta}_j$ by minimizing the objective function
$$\tilde{\beta}_j = \underset{\beta_j}{\operatorname{argmin}} F(\beta_j|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}}). \tag{4.7}$$
3. Repeat Step 2 till convergence.

The major difficulty in using the above coordinate descent procedure is that the univariate minimization problem in (4.7) does not have a closed form solution, unlike the penalized least squares. The univariate ℓ_1 penalized least squares has a neat solution by soft-thresholding. However, solving (4.7) requires an iterative algorithm. The same problem occurs when computing the elastic net penalized logistic regression. In `glmnet` [12] handled this difficulty by using the Newton-Raphson idea on top of the coordinate descent. After computing the partial derivatives and the Hessian matrix of the logistic regression loss, we face an elastic net penalized weighted least squares which can be easily solved by invoking an iterative coordinate-wise soft-thresholding procedure [12]. However, the Huberized hinge loss does not even have the second derivative, so the idea in `glmnet` is not directly applicable in the HHSVM.

We show that the computational obstacle can be resolved by a neat trick. We approximate the F function in (4.6) by a penalized quadratic function defined as

$$Q(\beta_j|\tilde{\beta}_0,\tilde{\boldsymbol{\beta}}) = \frac{\sum_{i=1}^n \phi_c(r_i)}{n} + \frac{\sum_{i=1}^n \phi_c'(r_i)y_i x_{ij}}{n}(\beta_j-\tilde{\beta}_j) + \frac{1}{\delta}(\beta_j-\tilde{\beta}_j)^2 + p_{\lambda_1,\lambda_2}(\beta_j), \quad (4.8)$$

where $\phi_c^{'}(t)$ is the first derivative of $\phi_c(t)$. We can easily solve the minimizer of (4.8) by a simple soft-thresholding rule [2]:

$$\begin{aligned}\hat{\beta}_j^{\mathrm{C}} &= \underset{\beta_j}{\operatorname{argmin}} Q(\beta_j|\tilde{\beta}_0,\tilde{\boldsymbol{\beta}}) \\ &= \frac{S\left(\frac{2}{\delta}\tilde{\beta}_j - \frac{\sum_{i=1}^n \phi_c'(r_i)y_i x_{ij}}{n}, \lambda_1\right)}{\frac{2}{\delta}+\lambda_2}, \end{aligned} \quad (4.9)$$

where $S(z,t) = (|z|-t)_+\mathrm{sgn}(z)$. We then set $\tilde{\beta}_j = \hat{\beta}_j^{\mathrm{C}}$ as the new estimate.

We use the same trick to update intercept β_0. Similarly to (4.8), we consider minimizing a quadratic approximation

$$Q(\beta_0|\tilde{\beta}_0,\tilde{\boldsymbol{\beta}}) = \frac{\sum_{i=1}^n \phi_c(r_i)}{n} + \frac{\sum_{i=1}^n \phi_c'(r_i)y_i}{n}(\beta_0-\tilde{\beta}_0) + \frac{1}{\delta}(\beta_0-\tilde{\beta}_0)^2, \quad (4.10)$$

which has a minimizer

$$\hat{\beta}_0^{\mathrm{C}} = \tilde{\beta}_0 - \frac{\delta}{2}\frac{\sum_{i=1}^n \phi_c'(r_i)y_i}{n}. \quad (4.11)$$

We set $\tilde{\beta}_0 = \hat{\beta}_0^{\mathrm{C}}$ as the new estimate.

To sum up, we have a generalized coordinate descent algorithm for solving the HHSVM; see Algorithm 8. The beauty of Algorithm 8 is that it is remarkably simple and almost identical to the coordinate descent algorithm for computing the elastic net penalized regression. In the next section we provide rigorous justification of the use of these Q functions and prove that each univariate update decreases the objective function of the HHSVM.

4.3.3 Implementation

We have implemented Algorithm 8 in a publicly available R package `gcdnet`. As demonstrated in `glmnet` some implementation tricks can boost the speed of a coordinate descent algorithm. We use these tricks in our implementation of Algorithm 8.

Algorithm 8 The generalized coordinate descent algorithm for the HHSVM.

- Initialize $(\tilde{\beta}_0, \tilde{\boldsymbol{\beta}})$.
- Iterate 2(a)-2(b) until convergence:
  - 2(a). Cyclic coordinate descent: for $j = 1, 2, \ldots, p$,
    - (2.a.1) Compute $r_i = y_i(\tilde{\beta}_0 + \mathbf{x}_i^{\intercal}\tilde{\boldsymbol{\beta}})$.
    - (2.a.2) Compute
    $$\hat{\beta}_j^{\mathrm{C}} = \frac{S\left(\frac{2}{\delta}\tilde{\beta}_j - \frac{\sum_{i=1}^n \phi_c'(r_i)y_i x_{ij}}{n}, \lambda_1\right)}{\frac{2}{\delta} + \lambda_2}.$$
    - (2.a.3) Set $\tilde{\beta}_j = \hat{\beta}_j^{\mathrm{C}}$.
  - 2(b). Update the intercept term
    - (2.b.1) Re-compute $r_i = y_i(\tilde{\beta}_0 + \mathbf{x}_i^{\intercal}\tilde{\boldsymbol{\beta}})$.
    - (2.b.2) Compute
    $$\hat{\beta}_0^{\mathrm{C}} = \tilde{\beta}_0 - \frac{\delta}{2}\frac{\sum_{i=1}^n \phi_c'(r_i)y_i}{n}.$$
    - (2.b.3) Set $\tilde{\beta}_0 = \hat{\beta}_0^{\mathrm{C}}$.

For each fixed λ_2, we compute the solutions for a fine grid of λ_1s. We start with $\lambda_{\max}$ which is the smallest λ_1 to set all $\beta_j, 1 \le j \le p$ to be zero. To compute $\lambda_{\max}$, we first obtain estimates $\hat{y}$ for the null model without any predictor:

$$\hat{y} = \underset{y}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \phi_c(y_i y) = \underset{y}{\operatorname{argmin}} [\frac{n_+}{n} \phi_c(y) + \frac{n_-}{n} \phi_c(-y)].$$

Then by the KKT conditions $\frac{1}{n} |\sum_{i=1}^{n} \phi_c'(\hat{y}) y_i x_{ij}| \le \lambda_1$ for all $j = 1, \ldots, p$, we have

$$\lambda_{\max} = \frac{1}{n} \max_j |\phi_c'(\hat{y}) y_i x_{ij}|.$$

We set $\lambda_{\min} = \tau \lambda_{\max}$ and the default value of τ is $\tau = 10^{-2}$ for $n < p$ data and $\tau = 10^{-4}$ for $n \ge p$ data. Between $\lambda_{\max}$ and $\lambda_{\min}$ we place K points uniformly in the log-scale. The default value for K is 98 such that there are 100 λ_1 values.

We use the warm-start trick to implement the solution paths. Without loss of generality let $\lambda_1[1] = \lambda_{\max}$ and $\lambda_1[100] = \lambda_{\min}$. We already have $\hat{\boldsymbol{\beta}} = 0$ for $\lambda_1[1]$. For $\lambda_1[k+1]$, the solution at $\lambda_1[k]$ is used as the initial value in Algorithm 8.

For computing the solution at each λ_1 we also utilize the active-set cycling idea. The active-set contains those variables whose current coefficients are nonzero. After a complete cycle through all the variables, we only apply coordinate-wise operations on the active set till convergence. Then we run another complete cycle to see if the active set changes. If the active set is not changed, then the algorithm is stopped. Otherwise, the active-set cycling process is repeated.

We need to repeatedly compute r_i in steps 2(a) and 2(b) of Algorithm 8. For that we use an updating formula. For $j = 1, \ldots, p, 0$, suppose β_j is updated, let $\delta_j = \hat{\beta}_j^{\mathrm{C}} - \tilde{\beta}_j$. Then we update r_i by $r_i = r_i + y_i x_{ij} \delta_j$.

We mention the convergence criterion used in Algorithm 8. After each cyclic update we declare convergence if $\frac{2}{\delta} \max_j \left(\hat{\beta}_j^{\text{current}} - \hat{\beta}_j^{\text{new}} \right)^2 < \epsilon$, as done in `glmnet` [12]. In `glmnet` the default value for ϵ is 10^{-6}. In `gcdnet` we use 10^{-8} as the default value of ϵ.

4.3.4 Approximating the SVM

Although the default value of δ is 2 unless specified otherwise, Algorithm 8 works for any positive δ. This flexibility allows us to explore the possibility of using Algorithm 8 to obtain an approximate solution path of the elastic net penalized support vector

machine. The motivation for doing so comes from the fact that $\lim_{\delta\to 0}\phi_c(t) = [1-t]_+$. In Figure 4.2 panel (b) we show the Huberized hinge functions with $\delta = 0.01$ which is nearly identical to the hinge loss.

Define

$$R(\boldsymbol{\beta},\beta_0) = \frac{1}{n}\sum_{i=1}^{n}[1-y_i(\beta_0+\mathbf{x}_i^\intercal\boldsymbol{\beta})]_+ + P_{\lambda_1,\lambda_2}(\boldsymbol{\beta}),$$

and

$$R(\boldsymbol{\beta},\beta_0|\delta) = \frac{1}{n}\sum_{i=1}^{n}\phi_c\left(y_i(\beta_0+\mathbf{x}_i^\intercal\boldsymbol{\beta})\right) + P_{\lambda_1,\lambda_2}(\boldsymbol{\beta}).$$

Let $(\widehat{\boldsymbol{\beta}}^{\text{svm}},\hat{\beta}_0^{\text{svm}})$ be the minimizer of $R(\boldsymbol{\beta},\beta_0)$. Algorithm 8 gives the unique minimizer of $R(\boldsymbol{\beta},\beta_0|\delta)$ for each given δ. Denote the solution as $(\widehat{\boldsymbol{\beta}}(\delta),\hat{\beta}_0(\delta))$. We notice that

$$\phi_c(t) \le (1-t)_+ \le \phi_c(t)+\delta/2 \quad \forall t,$$

which yields the following inequalities

$$R(\boldsymbol{\beta},\beta_0|\delta) \le R(\boldsymbol{\beta},\beta_0) \le R(\boldsymbol{\beta},\beta_0|\delta)+\delta/2. \tag{4.12}$$

From (4.12) we conclude that

$$\inf_{\boldsymbol{\beta},\beta_0} R(\boldsymbol{\beta},\beta_0) \le R(\widehat{\boldsymbol{\beta}}(\delta),\hat{\beta}_0(\delta)) \le \inf_{\boldsymbol{\beta},\beta_0} R(\boldsymbol{\beta},\beta_0)+\delta/2. \tag{4.13}$$

Here is a quick proof of (4.13):

$$\begin{aligned}
\inf_{\boldsymbol{\beta},\beta_0} R(\boldsymbol{\beta},\beta_0) &\le R(\widehat{\boldsymbol{\beta}}(\delta),\hat{\beta}_0(\delta)) \\
&\le R(\widehat{\boldsymbol{\beta}}(\delta),\hat{\beta}_0(\delta)|\delta)+\delta/2 \\
&\le R(\widehat{\boldsymbol{\beta}}^{\text{svm}},\hat{\beta}_0^{\text{svm}}|\delta)+\delta/2 \\
&\le R(\widehat{\boldsymbol{\beta}}^{\text{svm}},\hat{\beta}_0^{\text{svm}})+\delta/2 \\
&= \inf_{\boldsymbol{\beta},\beta_0} R(\boldsymbol{\beta},\beta_0)+\delta/2.
\end{aligned}$$

The two inequalities in (4.13) are independent of λ_1,λ_2. They suggest that we can use Algorithm 8 to compute the solution of a HHSVM with a tiny δ (say, $\delta = 0.01$) and then treat the outcome of Algorithm 8 as a good approximation to the solution of the elastic net penalized SVM. We have observed that a smaller δ results in a longer computing time. Our experience suggests that $\delta = 0.01$ delivers a good tradeoff between approximation accuracy and computing time.

4.4 GCD and The MM Principle

In the section we show that Algorithm 8 is a genuine coordinate descent algorithm. These Q functions used in each univariate update are closely related to the Majorization-Minimization (MM) principle [15, 17, 16]. The MM principle is a more liberal form of the famous Expectation-Maximization (EM) algorithm [14] in that the former often does not work with "missing data". See [26] and references therein.

We show that the Q function in (4.8) majorizes the F function in (4.6):

$$\begin{aligned} F(\beta_j|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}}) &\leq Q(\beta_j|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}}), & (4.14)\\ F(\tilde{\beta}_j|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}}) &= Q(\tilde{\beta}_j|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}}). & (4.15) \end{aligned}$$

With (4.14) and (4.15) we can easily verify the descent property of majorization update given in (4.9):

$$\begin{aligned} F(\hat{\beta}_j^{\mathrm{C}}|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}}) &= Q(\hat{\beta}_j^{\mathrm{C}}|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}}) + F(\hat{\beta}_j^{\mathrm{C}}|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}}) - Q(\hat{\beta}_j^{\mathrm{C}}|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}})\\ &\leq Q(\hat{\beta}_j^{\mathrm{C}}|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}})\\ &\leq Q(\tilde{\beta}_j|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}})\\ &= F(\tilde{\beta}_j|\tilde{\beta}_0, \tilde{\boldsymbol{\beta}}). \end{aligned}$$

We now prove (4.14) (note that (4.15) is trivial). By the mean value theorem

$$\phi_c(r_i + y_i x_{ij}(\beta_j - \tilde{\beta}_j)) = \phi_c(r_i) + \phi_c^{'}(r_i + t^* y_i x_{ij}(\beta_j - \tilde{\beta}_j)) y_i x_{ij}(\beta_j - \tilde{\beta}_j) \qquad (4.16)$$

for some $t^* \in (0, 1)$. Moreover, we observe that the difference of the first derivatives for the function ϕ satisfies

$$|\phi_c'(a) - \phi_c'(b)| = \begin{cases} 0 & \text{if } (a > 1, b > 1) \text{ or } (a < 1 - \delta, b < 1 - \delta),\\ |a - b|/\delta & \text{if } (1 - \delta < a \leq 1, 1 - \delta < b \leq 1),\\ |a - (1 - \delta)|/\delta & \text{if } (1 - \delta < a \leq 1, b < 1 - \delta),\\ |b - (1 - \delta)|/\delta & \text{if } (1 - \delta < b \leq 1, a < 1 - \delta),\\ |a - 1|/\delta & \text{if } (1 - \delta < a \leq 1, b > 1),\\ |b - 1|/\delta & \text{if } (1 - \delta < b \leq 1, a > 1). \end{cases}$$

Therefore we have

$$|\phi_c^{'}(a) - \phi_c^{'}(b)| \leq |a-b|/\delta \quad \forall a, b, \tag{4.17}$$

and hence

$$\begin{aligned} |\phi_c^{'}(r_i + t^* y_i x_{ij}(\beta_j - \tilde{\beta}_j)) - \phi_c^{'}(r_i)| &\leq 1/\delta |t^* y_i x_{ij}(\beta_j - \tilde{\beta}_j)| & (4.18) \\ &\leq 1/\delta |y_i x_{ij}(\beta_j - \tilde{\beta}_j)|. & (4.19) \end{aligned}$$

Combining (4.16) and (4.19) we have

$$\begin{aligned} \phi_c(r_i + y_i x_{ij}(\beta_j - \tilde{\beta}_j)) &= \phi_c(r_i) + \phi_c^{'}(r_i) y_i x_{ij}(\beta_j - \tilde{\beta}_j) \\ &\quad + \left(\phi_c^{'}(r_i + t^* y_i x_{ij}(\beta_j - \tilde{\beta}_j)) - \phi_c^{'}(r_i)\right) y_i x_{ij}(\beta_j - \tilde{\beta}_j) \\ &\leq \phi_c(r_i) + \phi_c^{'}(r_i) y_i x_{ij}(\beta_j - \tilde{\beta}_j) + 1/\delta \left(y_i x_{ij}(\beta_j - \tilde{\beta}_j)\right)^2. \end{aligned}$$

Summing over i at the both sides of the above inequality and using $y_i^2 = 1$, $\frac{1}{n}\sum_{i=1}^n x_{ij}^2 = 1$, we get (4.14).

4.5 A Further extension of the GCD Algorithm

In this section we further develop a generic GCD algorithm for solving a class of large margin classifiers. Define an elastic net penalized large margin classifier as follows

$$\min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{n} \sum_{i=1}^n L(y_i(\beta_0 + \mathbf{x}_i^\intercal \boldsymbol{\beta})) + P_{\lambda_1, \lambda_2}(\boldsymbol{\beta}), \tag{4.20}$$

where $L(\cdot)$ is a convex loss function. The coordinate descent algorithm cyclically minimizes

$$F(\beta_j | \tilde{\beta}_0, \tilde{\boldsymbol{\beta}}) = \frac{1}{n} \sum_{i=1}^n L(r_i + y_i x_{ij}(\beta_j - \tilde{\beta}_j)) + p_{\lambda_1, \lambda_2}(\beta_j) \tag{4.21}$$

with respect to β_j, where $r_i = y_i(\tilde{\beta}_0 + \mathbf{x}_i^\intercal \tilde{\boldsymbol{\beta}})$ is the current margin. The analysis in Section 3 shows that the foundation of the GCD algorithm for the HHSVM lies in the fact that for the Huberized hinge loss the F function has a simple quadratic majorization function. In order to generalize the GCD algorithm, we assume that the loss function L satisfies the following *quadratic majorization condition*

$$L(t + a) \leq L(t) + L'(t)a + \frac{M}{2}a^2 \quad \forall t, a. \tag{4.22}$$

Under (4.22) we have

$$\begin{aligned} & F(\beta_j|\tilde{\beta}_0,\tilde{\boldsymbol{\beta}}) \\ \leq & \frac{1}{n}\sum_{i=1}^{n}[L(r_i)+L'(r_i)y_ix_{ij}(\beta_j-\tilde{\beta}_j)+\frac{M}{2}x_{ij}^2(\beta_j-\tilde{\beta}_j)]+p_{\lambda_1,\lambda_2}(\beta_j) \\ = & \frac{\sum_{i=1}^{n}L(r_i)}{n}+\frac{\sum_{i=1}^{n}L'(r_i)y_ix_{ij}}{n}(\beta_j-\tilde{\beta}_j)+\frac{M}{2}(\beta_j-\tilde{\beta}_j)^2+p_{\lambda_1,\lambda_2}(\beta_j) \\ = & Q(\beta_j|\tilde{\beta}_0,\tilde{\boldsymbol{\beta}}), \end{aligned}$$

which means that Q is a quadratic majorization function of F. Therefore, like in Algorithm 1, we set the minimizer of Q as the new update

$$\hat{\beta}_j^{\mathrm{C}} = \underset{\beta_j}{\operatorname{argmin}} Q(\beta_j|\tilde{\beta}_0,\tilde{\boldsymbol{\beta}})$$

and the solution is given by

$$\hat{\beta}_j^{\mathrm{C}} = \frac{S(M\tilde{\beta}_j-\frac{\sum_{i=1}^{n}L'(r_i)y_ix_{ij}}{n},\lambda_1)}{M+\lambda_2}.$$

Likewise, the intercept is updated by

$$\hat{\beta}_0^{\mathrm{C}} = \tilde{\beta}_0-\frac{\sum_{i=1}^{n}L'(r_i)y_i}{Mn}.$$

To sum up, we now have a generic GCD algorithm for a class of large margin classifiers whose loss function satisfies the quadratic majorization condition; see Algorithm 9.

We now show that the quadratic majorization condition is actually satisfied by popular margin-based loss functions.

Lemma 1. *(a). If L is differentiable and has Lipschitz continuous first derivative, i.e.*

$$|L'(a)-L'(b)| \leq M_1|a-b| \quad \forall a,b. \tag{4.23}$$

then L satisfies the quadratic majorization condition with $M=2M_1$.

(b). If L is twice differentiable and had bounded second derivative, i.e.,

$$L''(t) \leq M_2 \quad \forall t,$$

then L satisfies the quadratic majorization condition with $M=M_2$.

Proof. For part (a) of Lemma 1, we notice that

$$L(t+a) = L(t) + L'(t_1)a = L(t) + L'(t)a + (L'(t_1) - L'(t))a$$

for some t_1 between t and $t+a$. Using (4.23) we have

$$|(L'(t_1) - L'(t))a| \leq M_1|(t_1 - t)a| \leq M_1 a^2,$$

which implies

$$L(t+a) \leq L(t) + L'(t)a + \frac{2M_1}{2}a^2.$$

For part (b) we simply use Taylor' expansion to the second order

$$L(t+a) = L(t) + L'(t)a + \frac{L''(t_2)}{2}a^2 \leq L(t) + L'(t)a + \frac{M_2}{2}a^2.$$

□

Figure 4.2 and Figure 4.3 show several popular loss functions and their derivatives. We use Lemma 1 to show that those loss functions satisfy the quadratic majorization condition.

In Section 3 we have shown that the Huberized hinge loss has Lipschitz continuous first derivative in (4.17) where $M_1 = 1/\delta$. By Lemma 1 it satisfies the quadratic majorization condition with $M = 2/\delta$. In this case Algorithm 9 reduces to Algorithm 8.

The squared hinge loss function has the expression $L(t) = [(1-t)_+]^2$ and its derivative is $L'(t) = -2(1-t)_+$. Direct calculation shows that (4.23) holds for the squared hinge loss with $M_1 = 2$. By Lemma 1 it satisfies the quadratic majorization condition with $M = 4$.

The logistic regression loss has the expression $L(t) = \log\left(1 + e^{-t}\right)$ and its derivative is $L'(t) = -(1+e^t)^{-1}$. The logistic regression loss is actually twice differentiable and its second derivative is bounded by 1/4: $L''(t) = \sum_{i=1}^{n} \frac{e^t}{(1+e^t)^2} \leq \frac{1}{4}$. By Lemma 1 it satisfies the quadratic majorization condition with $M = 1/4$.

We have implemented both Algorithm 8 and Algorithm 9 in an R-package `gcdnet`. In this chapter we use `hubernet`, `sqsvmnet` and `logitnet` to denote the funcion in `gcdnet` for computing the solution paths of the elastic net penalized Huberized SVM, squared SVM and logistic regression.

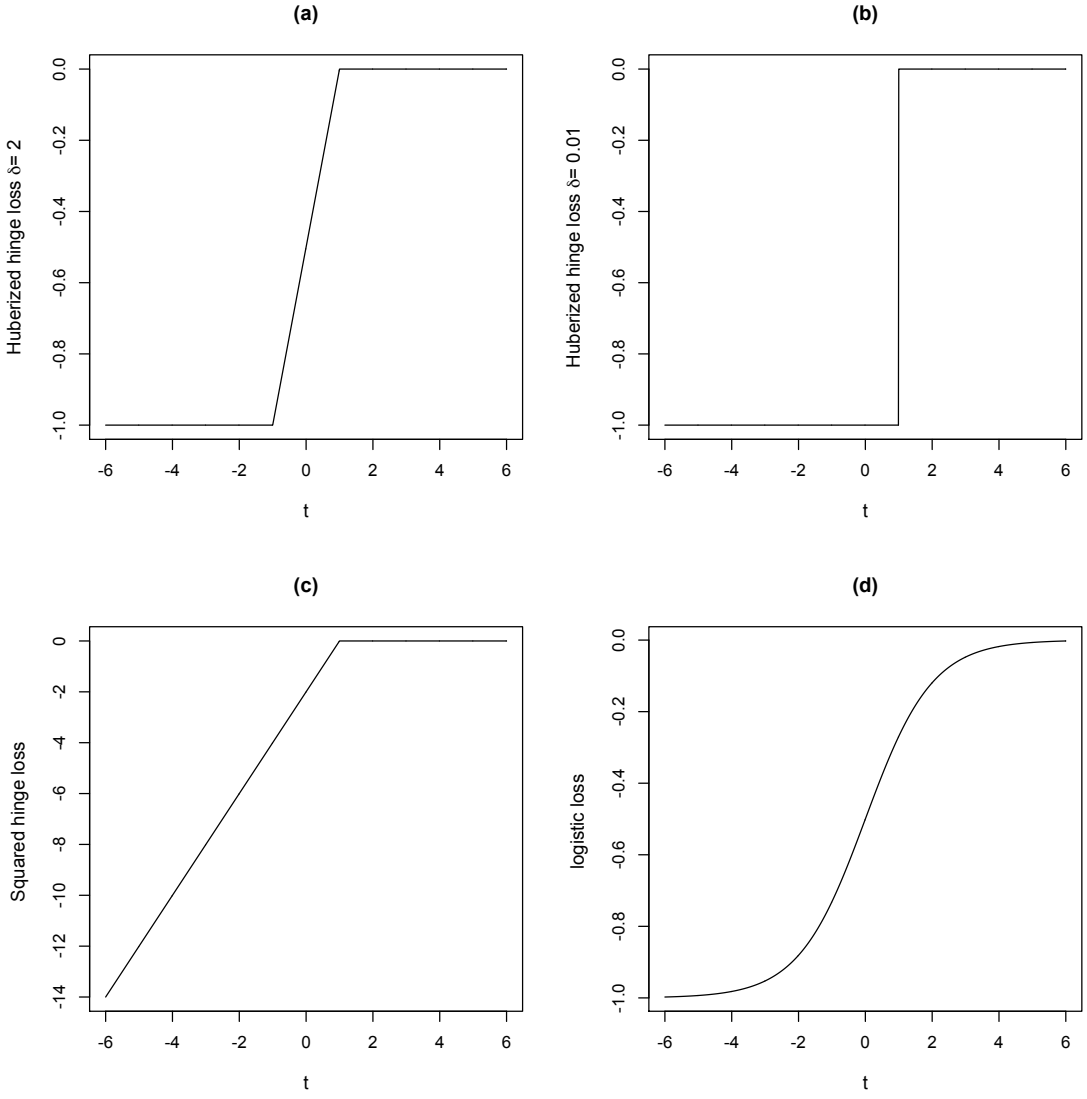

Figure 4.3: (a) The first derivative of the Huberized hinge loss function (with $\delta = 2$); (b) The first derivative of the Huberized hinge loss function (with $\delta = 0.01$); (c) The first derivative of the squared hinge loss function; (d) The first derivative of the logistic loss function.

Algorithm 9 The generic GCD algorithm for a class of large margin classifiers.

- Initialize $(\tilde{\beta}_0, \tilde{\boldsymbol{\beta}})$.
- Iterate 2(a)-2(b) until convergence:
  - 2(a). Cyclic coordinate descent: for $j = 1, 2, \ldots, p$,
    - (2.a.1) Compute $r_i = y_i(\tilde{\beta}_0 + \mathbf{x}_i^\intercal \tilde{\boldsymbol{\beta}})$.
    - (2.a.2) Compute
$$\hat{\beta}_j^{\mathrm{C}} = \frac{S(M\tilde{\beta}_j - \frac{\sum_{i=1}^n L'(r_i) y_i x_{ij}}{n}, \lambda_1)}{M + \lambda_2}.$$
    - (2.a.3) Set $\tilde{\beta}_j = \hat{\beta}_j^{\mathrm{C}}$.
  - 2(b). Update the intercept term
    - (2.b.1) Re-compute $r_i = y_i(\tilde{\beta}_0 + \mathbf{x}_i^\intercal \tilde{\boldsymbol{\beta}})$.
    - (2.b.2) Compute
$$\hat{\beta}_0^{\mathrm{C}} = \tilde{\beta}_0 - \frac{\sum_{i=1}^n L'(r_i) y_i}{Mn}.$$
    - (2.b.3) Set $\tilde{\beta}_0 = \hat{\beta}_0^{\mathrm{C}}$.

4.6 Numerical Experiments

4.6.1 Comparing two algorithms for the HHSVM

We compare the run-times of `hubernet` and a competing algorithm by [19] for the HHSVM. The latter method is a LARS-type path algorithm that exploits the piecewise linearity of the HHSVM solution paths. The source code is available at

http://www.stat.lsa.umich.edu/∼jizhu/code/hhsvm/.

For notation convenience, the LARS-type algorithm is denoted by `WZZ`. For each given λ_2, `WZZ` finds the piecewise linear solution paths of the HHSVM. `WZZ` automatically generates a sequence of λ_1 values. To make a fair comparison, the same λ_1 sequence is used in `hubernet`. All numerical experiments were carried out on an Intel Xeon X5560 (Quad-core 2.8 GHz) processor.

We first use the *FHT model* introduced in [12] to generate simulated data with n observations and p predictors from the following model

$$Z = \sum_{j=1}^{p} X_j \beta_j + k \cdot N(0,1)$$

where the covariance between predictors X_j and $X_{j'}$ has the same correlation ρ, with ρ ranging from zero to 0.95 and $\beta_j = (-1)^j \exp(-(2j-1)/20)$. We choose k such that the signal-to-noise ratio is 3.0. Generate a binary response Y according to $\Pr(\mathrm{Y} = -1) = 1/(1+\exp(-Z))$ and $\Pr(\mathrm{Y} = +1) = 1/(1+\exp(Z))$.

For each λ_2 in $(0, 10^{-4}, 10^{-2}, 1)$, we used `WZZ` and `hubernet` to compute the solution paths of the HHSVM over the same grid of λ_1 values. The process was repeated 10 times over 10 independent data sets. Tables 4.1 and 4.2 compare the run-times of `hubernet` and `WZZ`, where $\frac{t_{\mathbf{WZZ}}}{t_{\mathbf{hubernet}}}$ is the ratio of their run-times.

In Table 4.1 $n = 5000$ and $p = 100$, which corresponds to the traditional moderate-dimension-larger-sample-size case. We see that `hubernet` is about 25 to 30 times faster than `WZZ`. It is also interesting to note that the relative improvement in speed is nearly independent of the correlation level and λ_2, although the two factors have noticeable impact on the actual computing times.

In Table 4.2 $n = 100$ and $p = 5000$, which corresponds to the high-dimension-lower-sample-size case. When $\lambda_2 = 0$ the HHSVM uses the lasso penalty. In this case,

the correlation has little impact on the speed of `WZZ` while higher correlation slightly increases the timing of `hubernet`. Nevertheless, `hubernet` is about 4 to 7 times faster than `WZZ`. When $\lambda_2 > 0$ the HHSVM uses a true elastic net penalty. We see that the advantage of `hubernet` over `WZZ` increases as λ_2 becomes larger. A closer examination shows that `WZZ` is significantly lowered down by a larger λ_2, while λ_2 has a much weaker impact on `hubernet`'s timings.

We now use some popular benchmark data sets to compare `WZZ` and `hubernet`. Five data sets were considered in this study. See Table 4.3 for details. The first data set Arcene is obtained from UCI Machine Learning Repository [30]. We also considered the breast cancer data in [43], the colon cancer data in [28], the leukemia data in [62] and the prostate cancer data in [29]. For each data set we randomly split the data into a training set and a test set with ratio 4:1. The HHSVM was trained and tuned by 5-fold cross-validation on the training set. Note that we did a 2-dimension cross-validation to find the best pair of (λ_2, λ_1) that incurs minimal misclassification error. Let λ_2^{CV} denote the chosen λ_2 by cross-validation. We reported the timing of `WZZ` and `hubernet` for computing solution paths of the HHSVM with $\lambda_2 = \lambda_2^{\mathrm{CV}}$. The whole process was repeated 10 times. As can be seen from Table 4.3 `hubernet` is much faster than `WZZ` in all examples. It is also interesting to see that `hubernet` is very fast on all five data sets but `WZZ` can be very slow on prostate data.

4.6.2 Comparing hubernet, sqsvmnet and logitnet

As shown in Section 4 the GCD algorithm provides a unified solution to three elastic net penalized large margin classifiers using Huberized hinge loss, squared hinge loss and logistic regression loss. Here we compare the run times of `hubernet` for the HHSVM, `sqsvmnet` for the elastic net penalized squared hinge loss and `logitnet` for the elastic net penalized logistic regression. We want to see how the loss function affects the computing time of GCD.

For the elastic net penalized logistic regression, [12] have developed a very efficient algorithm by combining Newton-Raphson and penalized weighted least squares. Their software is available in the R-package `glmnet`. However `glmnet` uses a different form of the elastic net penalty: $P_{\lambda,\alpha}(\beta) = \lambda \sum_{j=1}^{p} \left[\frac{1}{2}(1-\alpha)\beta_j^2 + \alpha|\beta_j|\right]$. Fortunately, the `glmnet` code can be easily modified for the elastic net penalty $P_{\lambda_1,\lambda_2}(\beta)$ in (4.4). We

$n = 5000,\ p = 100$						
ρ	0	0.1	0.2	0.5	0.8	0.95
$\lambda_2 = 0$						
WZZ	88.42	85.51	86.97	86.56	96.24	113.83
hubernet	2.97	2.87	2.93	2.84	3.21	4.03
$\frac{t_{\mathbf{WZZ}}}{t_{\mathbf{hubernet}}}$	29.79	29.68	30.48	29.98	28.25	28.25
$\lambda_2 = 10^{-4}$						
WZZ	87.87	84.93	86.43	86.08	95.78	113.24
hubernet	2.99	2.87	2.93	2.86	3.24	4.05
$\frac{t_{\mathbf{WZZ}}}{t_{\mathbf{hubernet}}}$	29.39	29.59	29.50	30.10	29.56	27.96
$\lambda_2 = 10^{-2}$						
WZZ	80.53	76.65	79.38	78.73	87.45	107.24
hubernet	2.68	2.59	2.65	2.57	2.96	3.81
$\frac{t_{\mathbf{WZZ}}}{t_{\mathbf{hubernet}}}$	30.05	29.59	29.95	30.63	29.54	28.15
$\lambda_2 = 1$						
WZZ	12.43	12.37	12.37	12.37	12.54	27.54
hubernet	0.45	0.48	0.49	0.50	0.53	1.03
$\frac{t_{\mathbf{WZZ}}}{t_{\mathbf{hubernet}}}$	27.62	25.77	25.24	24.74	23.66	26.74

Table 4.1: Timings (in seconds) for **WZZ** and **hubernet** for $n = 5000, p = 100$ data. Total time over the same grid of λ_1 values chosen by **WZZ**, averaged over 10 independent runs.

$n = 100$, $p = 5000$						
ρ	0	0.1	0.2	0.5	0.8	0.95
$\lambda_2 = 0$						
WZZ	5.86	5.67	5.93	5.56	5.73	5.54
hubernet	0.87	0.82	0.86	0.86	1.01	1.16
$\frac{t_{\mathbf{WZZ}}}{t_{\mathbf{hubernet}}}$	6.74	6.91	6.90	6.47	5.67	4.78
$\lambda_2 = 10^{-4}$						
WZZ	208.39	208.13	203.15	208.46	203.87	188.75
hubernet	2.71	2.69	2.70	2.76	2.94	3.01
$\frac{t_{\mathbf{WZZ}}}{t_{\mathbf{hubernet}}}$	76.90	77.37	75.24	75.53	69.34	62.71
$\lambda_2 = 10^{-2}$						
WZZ	256.96	256.76	255.94	257.79	252.89	234.48
hubernet	2.97	2.96	3.03	2.95	2.95	2.96
$\frac{t_{\mathbf{WZZ}}}{t_{\mathbf{hubernet}}}$	86.52	86.74	84.47	87.39	85.73	79.22
$\lambda_2 = 1$						
WZZ	292.75	292.87	292.91	292.84	291.45	282.51
hubernet	2.63	2.64	2.61	2.56	2.46	2.41
$\frac{t_{\mathbf{WZZ}}}{t_{\mathbf{hubernet}}}$	111.31	110.94	112.23	114.39	118.48	117.22

Table 4.2: Timings (in seconds) for **WZZ** and **hubernet** for $n = 100, p = 5000$ data. Total time over the same grid of λ_1 values chosen by **WZZ**, averaged over 10 independent runs.

HHSVM on Benchmark Data Sets					
Data	n	p	**WZZ**	**hubernet**	$\frac{t_{\mathbf{WZZ}}}{t_{\mathbf{hubernet}}}$
Arcene	100	10000	37.17	2.46	15.09
Breast cancer	42	22283	3.16	0.46	6.85
Colon	62	2000	14.17	0.42	33.96
Leukemia	72	7128	52.20	1.42	36.69
Prostate	102	6033	302.91	3.47	87.2

Table 4.3: Timings (in seconds) of **WZZ** and **hubernet** for some benchmark real data, averaged over 10 runs.

have re-implemented `glmnet` and denote it by `logitnet-FHT` for notation convenience.

We first use the *FHT model* to generate simulation data with $n = 100, p = 5000$. Table 4.4 shows the run times (in seconds) of `hubernet` ($\delta = 2$), `hubernet` ($\delta = 0.01$), `sqsvmnet`, `logitnet` and `logitnet-FHT`. Each method solves the solution paths for 100 λ_1 values for each (λ_2, ρ) combination. First of all, Table 4.4 shows that all these methods are computationally efficient. Relatively speaking, the Huberized hinge loss ($\delta = 2$) and squared hinge loss lead to the fastest classifiers computed by GCD. As argued in Section 2.4, `hubernet` ($\delta = 0.01$) can be used to approximate the elastic net penalized SVM. Compared with the default `hubernet` (with $\delta = 2$), `hubernet` ($\delta = 0.01$) is about 10 times slower.

We also observe that `logitnet-FHT` is about 8 to 10 times faster than `logitnet`. A possible explanation is that `logitnet-FHT` takes advantages of the Hessian matrix of the logistic regression model in its Newton-Raphson step, while `logitnet` simply uses 1/4 to replace the second derivatives, which can be very conservative. On the other hand, `hubernet` ($\delta = 2$) and `sqsvmnet` are at least comparable to `logitnet-FHT` and the former two even have noticeable advantages when the correlation is high.

We now compare timings and classification accuracy of `hubernet` ($\delta = 2$), `hubernet` ($\delta = 0.01$), `sqsvmnet`, `logitnet` and `logitnet-FHT` on the five benchmark datasets. See details in Table 4.5. Each model was trained and tuned in the same way as described in Section 4.6.1. Average misclassification error on the test set from 10 independent splits is reported. Also reported is the run time (in seconds) for computing the solution

ρ	0	0.1	0.2	0.5	0.8	0.95
$\lambda_2 = 0$						
hubernet($\delta = 2$)	0.64	0.63	0.65	0.65	0.81	0.89
hubernet($\delta = 0.01$)	6.25	5.46	5.87	7.21	9.11	8.92
sqsvmnet	0.49	0.47	0.48	0.50	0.59	0.65
logitnet	3.85	3.85	3.82	4.77	6.06	8.02
logitnet-FHT	0.48	0.50	0.54	0.62	0.81	1.24
$\lambda_2 = 10^{-4}$						
hubernet($\delta = 2$)	0.64	0.62	0.64	0.67	0.81	0.88
hubernet($\delta = 0.01$)	6.20	5.62	5.71	6.76	8.60	8.94
sqsvmnet	0.50	0.50	0.50	0.53	0.62	0.68
logitnet	3.78	3.81	3.79	4.69	6.07	8.05
logitnet-FHT	0.48	0.50	0.54	0.62	0.81	1.23
$\lambda_2 = 10^{-2}$						
hubernet($\delta = 2$)	0.83	0.72	0.80	0.78	0.79	0.87
hubernet($\delta = 0.01$)	6.07	5.31	5.38	7.05	9.47	9.39
sqsvmnet	0.51	0.47	0.47	0.50	0.58	0.63
logitnet	4.24	4.18	4.49	5.11	7.29	8.14
logitnet-FHT	0.51	0.52	0.56	0.63	0.80	1.15
$\lambda_2 = 1$						
hubernet($\delta = 2$)	0.76	0.75	0.72	0.72	0.76	0.76
hubernet($\delta = 0.01$)	12.92	13.29	13.69	15.69	20.99	11.45
sqsvmnet	0.64	0.61	0.60	0.62	0.74	0.73
logitnet	4.89	4.63	4.53	4.69	4.98	5.17
logitnet-FHT	0.78	0.74	0.76	0.75	0.71	0.77

Table 4.4: Timings (in seconds) for **hubernet** ($\delta = 2$), **hubernet** ($\delta = 0.01$), **sqsvmnet**, **logitnet** and **logitnet-FHT** in the elastic net penalized classification methods for $n = 100, p = 5000$ data. Total time for 100 values of λ_1, averaged over 10 independent runs.

Classification Methods Comparison on Real Data										
	Arcene		Breast cancer		Colon		Leukemia		Prostate	
Method	Err. %	Sec.	Err. %	Sec.	Err. %	Sec.	Err. %	Sec.	Err. %	Sec.
hubernet ($\delta = 2$)	21.00	1.35	19.71	1.12	17.90	0.23	3.16	0.73	9.35	0.77
hubernet ($\delta = 0.01$)	23.58	23.92	19.71	15.36	15.10	7.35	2.41	5.55	8.47	6.12
sqsvmnet	23.00	1.13	19.14	1.15	19.30	0.18	3.25	0.54	8.88	0.66
logitnet	25.00	9.71	19.57	6.53	21.20	1.13	3.91	6.23	9.05	4.77
logitnet-FHT	25.00	1.85	19.57	1.08	21.20	0.14	3.91	0.65	9.05	0.67

Table 4.5: Testing error (%) and timings (in seconds) for some benchmark real data. The timings are for computing solution paths for **hubernet** ($\delta = 2$), **hubernet** ($\delta = 0.01$), **sqsvmnet**, **logitnet** and **logitnet-FHT** with λ_2 chosen by cross-validation and over the grid of 100 λ_1 values, averaged over 10 runs.

paths with λ_2 chosen by 5-fold cross-validation. We can see that the `hubernet` with $\delta = 2$ or $\delta = 0.01$ has better classification accuracy than other classifiers on Arcene, Colon, Leukemia and Prostate, while `sqsvmnet` has the smallest error on Breast cancer. In terms of timings, there are 3 datasets for which `sqsvmnet` is the winner and `logitnet-FHT` wins on the other two. The timing results for `hubernet` ($\delta = 2$) are very close to those of `sqsvmnet` and `logitnet-FHT`. Overall, `hubernet` ($\delta = 2$) delivers very competitive performance on these benchmark datasets.

Chapter 5

A Cocktail Algorithm for Penalized Cox's Regression

5.1 Chapter Overview

We introduce a cocktail algorithm, a good mixture of coordinate decent, the majorization-minimization principle and the strong rule, for computing the solution paths of the elastic net penalized Cox's proportional hazards model. The cocktail algorithm enjoys a proven convergence property. We have implemented the cocktail algorithm in an R package `fastcox`. Numerical examples show that `cocktail` is comparable to `coxnet` [63] in speed and often delivers better quality solutions.

5.2 Introduction

Cox's proportional hazards model [64] is a standard statistical model for studying the relation between survival time and a set of covariates. Consider the standard survival data of the form $(y_i, \mathbf{x}_i, d_i)_{i=1}^n$, where y_i is the survival time, $1 - d_i$ is the censoring indicator ($d_i = 1$ indicates no censoring and $d_i = 0$ indicates right censoring) and $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^\intercal$ represents the p-dimension covariates. For the sake of simplicity, assume there are no tied failure times and the censoring is non-informative. Denote by $t_1 < t_2 < \cdots < t_S$ the distinct failure times and let R_s be the risk set at time $t_s - 0$. Cox's proportional hazards model assumes that the hazard function at time t

given predictor values $\mathbf{x}$ is $h(t|\mathbf{x}) = h_0(t)\exp(\mathbf{x}^\top\boldsymbol{\beta}^*)$ where $h_0(t)$ represents the baseline hazard function. Statistical inference of Cox's model is through the partial likelihood function:

$$L(\boldsymbol{\beta}) = \prod_{s=1}^{S} \frac{\exp(\mathbf{x}_{i_s}^\top\boldsymbol{\beta})}{\sum_{i\in R_s}\exp(\mathbf{x}_i^\top\boldsymbol{\beta})},$$

where i_s is the index of the failure at time t_s. The usual Cox's estimator of $\boldsymbol{\beta}$ is obtained by maximizing the partial likelihood.

With the advances in modern technology, high-dimensional data frequently appear in fields such as medical and biological sciences, finances and economics, etc. The maximum partial likelihood estimator does not work well in the presence of high-dimensional covariates. To combat the high-dimensionality, sparse penalized Cox's models have been considered in the literature. Let $P_\lambda(\boldsymbol{\beta})$ be a penalty function that is non-differentiable at zero. Consider the penalized partial likelihood estimator

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{n}\left[\sum_{s=1}^{S} -\mathbf{x}_{i_s}^\top\boldsymbol{\beta} + \log(\sum_{i\in R_s}\exp(\mathbf{x}_i^\top\boldsymbol{\beta}))\right] + P_\lambda(\boldsymbol{\beta}).$$

For example, [65] used the lasso penalty [66], $P_\lambda(\boldsymbol{\beta}) = \lambda\sum_{i=j}^{p}|\beta_j|$, to fit a lasso penalized Cox's regression model.

[24] proposed the elastic net penalty as an improved variant of the lasso for high-dimensional data. The elastic net penalty is defined as

$$P_{\lambda,\alpha}(\boldsymbol{\beta}) = \sum_{j=1}^{p}\lambda\left(\alpha|\beta_j| + \frac{1}{2}(1-\alpha)\beta_j^2\right),$$

with $\lambda > 0$ and $0 < \alpha \leq 1$. The ℓ_1 part of the elastic net is responsible for achieving sparsity. The lasso can be viewed as a special case of the elastic net with $\alpha = 1$. By using its quadratic part, the elastic net improves upon the lasso in two aspects. First, it can better handle the correlated covariates which are very common in high-dimensional data. Second, the solution paths are more stable due to the quadratic regularization and hence it leads to improved prediction. [6] developed a predictor-corrector algorithm for computing the elastic net penalized Cox's model, see the `glmpath` package available from the Comprehensive R Archive Network (CRAN) at `http://cran.r-project.org/package=glmpath`.

[67] discussed a gradient descent algorithm for solving the ℓ_1 penalized Cox's model and implemented his algorithm in an R package `penalized`, available from CRAN

at `http://cran.r-project.org/web/packages/penalized`. [63] recently developed `coxnet` for computing the elastic net penalized Cox model. Their function is included in the `glmnet` package, available from CRAN at `http://cran.r-project.org/package=glmnet`. By numerical examples, [63] showed that `coxnet` is much faster than `coxpath` and `penalized`.

It is important to note that `coxnet` uses some heuristic arguments to approximate the Hessian matrix of the log-partial likelihood in order to boost computation speed. As a result, it is unclear whether `coxnet` always converges to the right solution, although [63] reported that they did not encounter any convergence problem. In this chapter, we introduce a new principled fast algorithm for computing the elastic net penalized Cox model. Our algorithm combines the strengths of three optimization ideas: coordinate descent, the majorization-minimization principle and the strong rule. It is thus named a cocktail algorithm. We show that the cocktail algorithm always converges to the right solution. We build an R package `fastcox` to implement the cocktail algorithm and we show that `cocktail` is comparable to `coxnet` in speed and often gives higher quality solutions.

5.3 Coordinate Majorization Descent

5.3.1 Derivation

In this section we derive the coordinate-majorization-descent (CMD) algorithm to minimize the following objective function

$$G(\boldsymbol{\beta}) = \ell(\boldsymbol{\beta}) + \sum_{j=1}^{p} \lambda \left[\alpha w_j |\beta_j| + \frac{1}{2}(1-\alpha)\beta_j^2 \right], \tag{5.1}$$

where

$$\ell(\boldsymbol{\beta}) = n^{-1} \sum_{s=1}^{S} -\mathbf{x}_{i_s}^{\mathsf{T}} \boldsymbol{\beta} + \log\Big(\sum_{i \in R_s} \exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta}) \Big).$$

Note that including the non-negative weights w_js allows for more flexible estimation. If we want to always include x_j in the final model then we typically do not impose a sparse penalty on β_j, which can be easily done by setting $w_j = 0$. Often, the adaptively weighted lasso [68] is preferred over the lasso for variable selection.

We begin with the observation that $\ell(\boldsymbol{\beta})$ is a smooth convex function of $\boldsymbol{\beta}$ and the penalty function is convex and separable. This observation suggests that we can try the coordinate descent algorithm for minimizing the objective function in (5.1) [18]. Recently, coordinate descent has been successfully used to solve the lasso-type penalized models [69, 35, 36]. Define the objective function for fixed λ and α and β_k where $k \neq j$ to be

$$\begin{aligned} g(\beta_j) &= \ell(\beta_j | \beta_k = \tilde{\beta}_k, k \neq j) \\ &\quad + \lambda \left[\alpha w_j |\beta_j| + \frac{1}{2}(1-\alpha)\beta_j^2 \right], \end{aligned} \tag{5.2}$$

the coordinate descent algorithm proceeds as follows:

- Initialize $\tilde{\boldsymbol{\beta}}$.
- Cyclic coordinate descent: for $j = 1, \ldots, p$, update the estimator by

$$\tilde{\beta}_j \leftarrow \underset{\beta_j}{\operatorname{argmin}}\, g(\beta_j).$$

- Repeat the coordinate descent cycle till convergence.

Coordinate descent is efficient for the ℓ_1-penalized least squares because each coordinate descent update can be computed by a simple soft-thresholding rule. However, the univariate minimization problem in (5.2) does not have a simple closed-form solution. The same computational difficulty appears in many applications of the Expectation-Maximization algorithm where the maximization step does not have a simple computational form. To alleviate such difficulty, [70] proposed to increase the objective function rather than maximize it at each maximization step, which results in the generalized Expectation-Maximization algorithm. We borrow the same idea to overcome the computational difficulty in (5.2). Our solution makes use of the majorization-minimization/maximization (MM) principle. For some good review papers on the MM principle, the readers are referred to [46], [16] and [71].

Instead of minimizing (5.2), we propose to find an update of $\tilde{\beta}_j$ such that the univariate function in (5.2) is decreased. It turns out that such an update can be computed by a soft-thresholding rule. To write the updating formula we need some additional

Algorithm 10 CMD for the elastic net penalized Cox's regression

1. Initialize $\tilde{\boldsymbol{\beta}}$.
2. For $j = 1, \ldots, p$,
$$\tilde{\beta}_j^{\text{new}} = \frac{S(D_j\tilde{\beta}_j - \ell_j'(\tilde{\boldsymbol{\beta}}), \lambda\alpha w_j)}{D_j + \lambda(1-\alpha)}.$$
3. Update $\tilde{\boldsymbol{\beta}} = \tilde{\boldsymbol{\beta}}^{\text{new}}$.
4. Repeat steps 2–3 till convergence of $\tilde{\boldsymbol{\beta}}$.

notation. Define
$$D_j = \sum_{s=1}^{S} \frac{1}{4n} \left(\max_{i \in R_s}(x_{ij}) - \min_{i \in R_s}(x_{ij}) \right)^2,$$
for $j = 1, 2, \ldots, p$, and let $\ell_j'(\boldsymbol{\beta})$ denote the partial derivative of the negative log partial likelihood with respect to β_j. We have
$$\ell_j'(\boldsymbol{\beta}) = n^{-1} \sum_{s=1}^{S} [-x_{i_s,j} + \frac{\sum_{i \in R_s} x_{i,j} \exp(\mathbf{x}_i^{\mathsf{T}}\boldsymbol{\beta})}{\sum_{i \in R_s} \exp(\mathbf{x}_i^{\mathsf{T}}\boldsymbol{\beta})}],$$
for $j = 1, 2, \ldots, p$. We approximate (5.2) by a penalized quadratic function defined as
$$\begin{aligned} q(\beta_j|\tilde{\boldsymbol{\beta}}) &= \ell(\tilde{\boldsymbol{\beta}}) + \ell_j'(\tilde{\boldsymbol{\beta}})(\beta_j - \tilde{\beta}_j) + \frac{D_j}{2}(\beta_j - \tilde{\beta}_j)^2 \\ &\quad + \lambda \left[\alpha w_j |\beta_j| + \frac{1}{2}(1-\alpha)\beta_j^2 \right]. \end{aligned} \tag{5.3}$$
The proposed update is the minimizer of (5.3)
$$\tilde{\beta}_j^{\text{new}} = \frac{S(D_j\tilde{\beta}_j - \ell_j'(\tilde{\boldsymbol{\beta}}), \lambda\alpha w_j)}{D_j + \lambda(1-\alpha)},$$
where $S(z,t) = (|z| - t)_+ \text{sign}(z)$ is the soft-thresholding operator.

With the help of the MM principle, we can use an iterative soft-thresholding procedure, see Algorithm 1, for solving the elastic net penalized Cox's regression, which is much like the iterative soft-thresholding procedure for the elastic net penalized least squares problem.

5.3.2 The descent property of CMD

We now prove the descent property of Algorithm 1 which can be seen from the following two lemmas.

Lemma 2. $\tilde{\beta}_j^{\text{new}} = \frac{S(D_j\tilde{\beta}_j - \ell_j'(\tilde{\boldsymbol{\beta}}), \lambda\alpha w_j)}{D_j + \lambda(1-\alpha)}$ *minimizes* $q(\beta_j|\tilde{\boldsymbol{\beta}})$ *defined in* (5.3).

Lemma 3. $\ell_j'' \leq D_j$ *for all* $\boldsymbol{\beta} \in \mathbb{R}^p$.

Lemma 1 basically shows the soft-thresholding solution to the univariate lasso regression. We only prove Lemma 2.

Proof of lemma 2. We first compute

$$\ell_j'' = \frac{1}{n}\sum_{s=1}^{S}\left[\frac{\sum_{i\in R_s} x_{i,j}^2 \exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})}{\sum_{i\in R_s}\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})} - \left(\frac{\sum_{i\in R_s} x_{i,j}\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})}{\sum_{i\in R_s}\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})}\right)^2\right].$$

Inside $[\cdots]$ can be regarded as the variance of a discrete random variable Z whose distribution is $P(Z = x_{i,j}) = \frac{\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})}{\sum_{i\in R_s}\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})}$. The variance is maximized when Z has a two-point distribution on $\max_{i\in R_s}(x_{ij})$, $\min_{i\in R_s}(x_{ij})$ with equal probability. □

From Lemmas 1 and 2 we have the following inequalities

$$q(\tilde{\beta}_j^{\text{new}}|\tilde{\boldsymbol{\beta}}) \leq q(\tilde{\beta}_j|\tilde{\boldsymbol{\beta}}),$$

$$g(\beta_j) \leq q(\beta_j|\tilde{\boldsymbol{\beta}}) \quad \forall\beta_j \in \mathbb{R}.$$

Therefore, we can conclude

$$g(\tilde{\beta}_j^{\text{new}}) \leq q(\tilde{\beta}_j^{\text{new}}|\tilde{\boldsymbol{\beta}}) \leq q(\beta_j|\tilde{\boldsymbol{\beta}}) = g(\beta_j),$$

which justifies the descent property of the CMD algorithm.

5.3.3 Solution path implementation

For each given α, we use the CMD algorithm to compute the solutions of the elastic net penalized Cox's regression model at a grid of decreasing λ values. The default number is 100. We use the commonly used warm-start and active set tricks, as done in `glmnet`.

To use warm-start we first need to find the largest λ value, denoted by $\lambda_{\max}$, that is defined as the smallest λ that shrinks all β_js to zero. By the KKT conditions it is easy to show for $w_j \neq 0$

$$\lambda_{\max} = n^{-1}\alpha^{-1} \max_j \left\{ \frac{1}{|w_j|} | \sum_{s=1}^{S} (-x_{i_s,j} + \frac{\sum_{i \in R_s} x_{i,j}}{|R_s|}) | \right\}.$$

For the ordinary $n \geq p$ data we let $\lambda_{\min} = 0.0001\lambda_{\max}$. When $p > n$ we let $\lambda_{\min} = 0.01\lambda_{\max}$. We choose a grid of 100 points uniformly in the log scale on $(\lambda_{\min}, \lambda_{\max})$. Let $\lambda[1] = \lambda_{\max}$ and $\lambda[100] = \lambda_{\min}$. Warm-start takes the solution at the k-th grid point $\lambda[k]$ as the initial value for computing the solution at $\lambda[k+1]$.

The active set is defined as the collection of variables whose current coefficient estimates are nonzeros. After a complete cycle through all p coefficients, we only repeat the coordinate descent on the active set till convergence. Then we run another complete cycle to check whether the active set is changed. If the active set remains unchanged, the algorithm is done, otherwise the process is repeated.

5.4 CMD with the strong rule

[27] recently introduced the strong rule for improving the computational speed of `glmnet`. To implement CMD in the `fastcox` package, we have also used the strong rule on top of the CMD algorithm. Suppose that we have already computed the solution $\hat{\boldsymbol{\beta}}[\lambda_k]$ at $\lambda = \lambda_k$, before computing the solution at λ_{k+1}, we first compute $\ell_j'(\hat{\boldsymbol{\beta}}[\lambda_k])$ for $j = 1, 2, \ldots, p$. The strong rule claims that the variables that satisfy the condition

$$\left| \ell_j'(\hat{\boldsymbol{\beta}}[\lambda_k]) \right| \geq \alpha(2\lambda_{k+1} - \lambda_k) w_j$$

are very likely to have nonzero coefficients at λ_{k+1}. Let V be the collection of such variables and write V^c as its complement. If the strong rule guesses correctly, we only need to focus on solving $\hat{\boldsymbol{\beta}}_V$ by calling Algorithm 1 on a reduced dataset $(y_i, \mathbf{x}_{iV}, d_i)_{i=1}^n$, where $\mathbf{x}_{iV} = (\cdots \mathbf{x}_{i,j} \cdots)$ and j is an index in the set V. Suppose that $\hat{\boldsymbol{\beta}}_V$ is computed and the next step is to check whether the strong rule indeed guesses correctly. For that we just need to check whether $\boldsymbol{\beta} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0})$ satisfies the KKT condition at $\lambda = \lambda_{k+1}$. In other words, if for each $j \in V^c$ the following KKT condition holds:

$$\left| \ell_j' \left(\boldsymbol{\beta} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0}) \right) \right| \leq \alpha \lambda_{k+1} w_j.$$

Algorithm 11 The CMD algorithm with the strong rule for the solution at λ_{k+1}.

1. Initialize $\tilde{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}[\lambda_k]$.

2. Screen p variables using the strong rule, create an initial survival set V such that for $j \in V$

$$\left|\ell_j'(\hat{\boldsymbol{\beta}}[\lambda_k])\right| \geq \alpha(2\lambda_{k+1} - \lambda_k)w_j.$$

3. Call Algorithm 1 on a reduced dataset $(y_i, \mathbf{x}_{iV}, d_i)_{i=1}^n$ to solve $\hat{\beta}_V$.

4. Compute a set U as the part of V^c that failed KKT check:

$$U = \{j : j \in V^c \text{ and } \left|\ell_j'\left(\boldsymbol{\beta} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0})\right)\right| > \alpha\lambda_{k+1}w_j\}.$$

5. If $U = \varnothing$ then stop the loop and return $\hat{\boldsymbol{\beta}} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0})$. Otherwise update $V = V \bigcup U$ and go to step 3.

Then $\boldsymbol{\beta} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0})$ is the solution at $\lambda = \lambda_{k+1}$. Otherwise, we update V by $V = V \bigcup U$ where

$$U = \{j : j \in V^c \text{ and } \left|\ell_j'\left(\boldsymbol{\beta} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0})\right)\right| > \alpha\lambda_{k+1}w_j\}.$$

Note that the V set can only grow larger after each update and hence the strong rule iteration will always stop after a finite number of updates, which means the strong rule will eventually guess correctly.

For penalized least squares and logistic regression, the strong rule has a magic property in that it almost always guesses correctly and we rarely need to actually update V by adding U. See [27] for more detailed discussion. We have observed that this incredible phenomenon continues to hold for the penalized Cox regression model. Algorithm 2 shows how we use the strong rule on top of the CMD algorithm. Algorithm 2 is indeed the pseudocode of `cocktail`.

5.5 Numerical Studies

We have implemented the CMD algorithm with the strong rule in a publicly available R package `fastcox`. [63] have showed that `coxnet` is much faster than `coxpath` and `penalized`. Hence we only compare `cocktail` with `coxnet` which is a part of R package `glmnet` 1.7.1. All computations were carried out on an 2.4GHz Intel Core i5 processor. In `coxnet`, the convergence criterion is $\max_j D_j \left(\tilde{\beta}_j^{\text{old}} - \tilde{\beta}_j^{\text{new}}\right)^2 < \epsilon^2$. We used the same convergence criterion in `cocktail` and $\epsilon = 10^{-5}$ in all examples presented in this section.

5.5.1 Timing comparison

We considered the simulation model by [63]. We generated data with n observations and p predictors from the following model

$$Y^{true} = \exp\left(\sum_{j=1}^{p} X_j \beta_j + k \cdot N(0,1)\right),$$

where Y^{true} is the "true" survival time and the correlation between predictors X_j and $X_{j'}$ is ρ, with ρ ranges from zero to 0.95, and $\beta_j = (-1)^j \exp(-(2j-1)/20)$, k is chosen such that the signal-to-noise ratio is 3.0. Likewise, censoring times are generated by $C = \exp(k \cdot N(0,1))$. The recorded survival time is $Y = \min(Y^{true}, C)$. The observation is censored if $C < Y^{true}$.

In Table 5.1, $n = 100, p = 5,000$, for each α in $(0.1, 0.2, 0.5, 0.8, 1)$, we computed the solution paths of the penalized Cox's model for the same sequence of λ values using `coxnet` and `cocktail`. We repeated the process 20 times on 3 independent data sets and reported the average running time. We also performed a similar timing comparison for $n = 200,\ p = 10,000$ in Table 5.2. We see that `cocktail` has better speed performance with small α and low correlation data while `coxnet` is faster for large α and high correlation data. When `cocktail` is the winner, it can be 2 times faster than `coxnet` and vice versa. Thus, it is fair to say that both packages are comparable in speed.

5.5.2 Quality comparison

We show that with the same convergence criterion, `cocktail` can provide a more accurate solution than `coxnet` does. We test the accuracy of solutions by checking their

$n = 100, p = 5,000$						
ρ	0	0.1	0.2	0.5	0.8	0.95
$\alpha = 0.1$						
coxnet	23.64	21.74	23.80	25.93	16.89	15.17
cocktail	11.35	9.91	12.78	18.89	26.97	30.44
$\alpha = 0.2$						
coxnet	21.29	22.83	21.05	24.42	14.67	13.07
cocktail	9.41	12.15	11.05	19.52	20.73	27.77
$\alpha = 0.5$						
coxnet	18.11	17.12	17.24	16.94	13.19	10.15
cocktail	11.98	13.52	13.61	17.66	23.08	18.42
$\alpha = 0.8$						
coxnet	10.42	10.86	10.59	13.26	11.16	16.61
cocktail	13.14	15.48	15.88	22.27	25.96	24.55
$\alpha = 1$						
coxnet	9.55	9.27	9.56	10.73	11.06	17.95
cocktail	35.06	31.19	28.75	31.77	35.19	44.40

Table 5.1: Timings (in seconds) for **coxnet** and **cocktail**. Total time for the same λ sequence of 100 values, averaged over 20 independent runs.

$n = 200, p = 10,000$						
Correlation	0	0.1	0.2	0.5	0.8	0.95
$\alpha = 0.1$						
coxnet	96.5	70.9	79.7	113.6	77.9	42.7
cocktail	45.6	45.0	51.1	71.7	90.9	138.0
$\alpha = 0.2$						
coxnet	94.1	89.0	114.5	77.4	44.6	49.2
cocktail	35.7	43.5	64.8	62.6	69.3	124.3
$\alpha = 0.5$						
coxnet	52.6	70.4	65.3	62.5	49.3	52.1
cocktail	45.2	59.9	51.9	78.0	102.4	133.6
$\alpha = 0.8$						
coxnet	38.9	41.3	35.8	39.8	32.1	57.8
cocktail	54.1	66.6	74.4	69.2	76.0	76.0
$\alpha = 1$						
coxnet	39.5	40.4	42.8	52.9	30.3	48.5
cocktail	150.2	135.2	127.8	224.6	145.9	164.6

Table 5.2: Timings (in seconds) for **coxnet** and **cocktail**. Total time for the same λ sequence of 100 values, averaged over 20 independent runs.

KKT conditions. Theoretically, $\boldsymbol{\beta}$ is the solution to (5.1) if and only if the following conditions hold

$$\begin{aligned} \ell_j'(\boldsymbol{\beta}) + \lambda(1-\alpha)\beta_j + \alpha\lambda w_j \cdot \operatorname{sgn}(\beta_j) = 0 \qquad & \text{if } \beta_j \neq 0, \\ |\ell_j'(\boldsymbol{\beta})| \leq \alpha\lambda w_j \qquad & \text{if } \beta_j = 0, \end{aligned}$$

where $j = 1, 2, \ldots, p$. Numerically, we declare β_j passes the KKT condition check if

$$\begin{aligned} |\ell_j'(\boldsymbol{\beta}) + \lambda(1-\alpha)\beta_j + \alpha\lambda w_j \cdot \operatorname{sgn}(\beta_j)| \leq \epsilon \qquad & \text{if } \beta_j \neq 0, \\ |\ell_j'(\boldsymbol{\beta})| \leq \alpha\lambda w_j + \epsilon \qquad & \text{if } \beta_j = 0, \end{aligned}$$

for a small $\epsilon > 0$. For the solutions computed in section 4.1, we calculated the average number of coefficients that violated the KKT condition check at each λ value. Then this number was averaged over the 100 values of λs. This process was repeated 3 times on 3 independent datasets. As shown in Table 5.3 and Table 5.4, `cocktail` has much smaller violation counts than `coxnet` in most cases. Only for $\alpha = 1$, which corresponds to the lasso case, `coxnet` is slightly better than `cocktail`, but the KKT violation counts are very small in this case. Overall, it is clear that `cocktail` is numerically more accurate than `coxnet`.

5.5.3 Real data analysis

In this section we use the lung cancer data from [72] to examine timings and accuracies of `coxnet` and `cocktail`. The data is from a microarray experiment investigating survival of cancer patients with lung adenocarcinomas. The data set contains expression data for 86 patients with 7,129 probe sets. We chose α from $(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)$ and computed the solution paths of the penalized Cox's model for the same λ sequence using `coxnet` and `cocktail`. The process was repeated 20 times. For space consideration we only report the timing and accuracy results of $\alpha = 0.1, 0.2, 0.3, 0.5, 0.8, 1$ in Table 5.5. One can see that the two algorithms are comparable in terms of the running time: `cocktail` has better speed performance with small α while `coxnet` is faster with large α. In terms of solution quality which is measured by the average number of coefficients that violated the KKT condition check, `cocktail` is always the winner. This is consistent with the simulation results.

$n = 100, p = 5,000$						
ρ	0	0.1	0.2	0.5	0.8	0.95
$\alpha = 0.1$						
coxnet	556	497	517	434	285	153
cocktail	6	11	19	59	136	116
$\alpha = 0.2$						
coxnet	313	325	292	233	185	98
cocktail	6	5	12	72	90	76
$\alpha = 0.5$						
coxnet	146	141	124	102	88	54
cocktail	5	6	9	25	51	40
$\alpha = 0.8$						
coxnet	50	50	36	23	41	32
cocktail	5	6	7	13	28	28
$\alpha = 1$						
coxnet	5	2	4	7	20	24
cocktail	7	6	6	12	20	24

Table 5.3: Reported numbers are the average number of coefficients among 5,000 coefficients that violated the KKT condition check (rounded down to the next smaller integer) using **coxnet** and **cocktail**. Results are averaged over the λ sequence of 100 values and averaged over 20 independent runs.

$n = 200, p = 10,000$						
ρ	0	0.1	0.2	0.5	0.8	0.95
$\alpha = 0.1$						
coxnet	921	729	731	669	573	197
cocktail	12	14	107	136	227	140
$\alpha = 0.2$						
coxnet	471	527	545	404	247	129
cocktail	7	12	61	95	93	95
$\alpha = 0.5$						
coxnet	203	251	184	123	170	80
cocktail	3	11	20	66	88	62
$\alpha = 0.8$						
coxnet	90	69	42	60	82	47
cocktail	10	14	13	29	48	38
$\alpha = 1$						
coxnet	1	2	3	24	38	37
cocktail	9	12	13	30	44	42

Table 5.4: Reported numbers are the average number of coefficients among 10000 coefficients that violated the KKT condition check (rounded down to the next smaller integer) using **coxnet** and **cocktail**. Results are averaged over the λ sequence of 100 values and averaged over 20 independent runs.

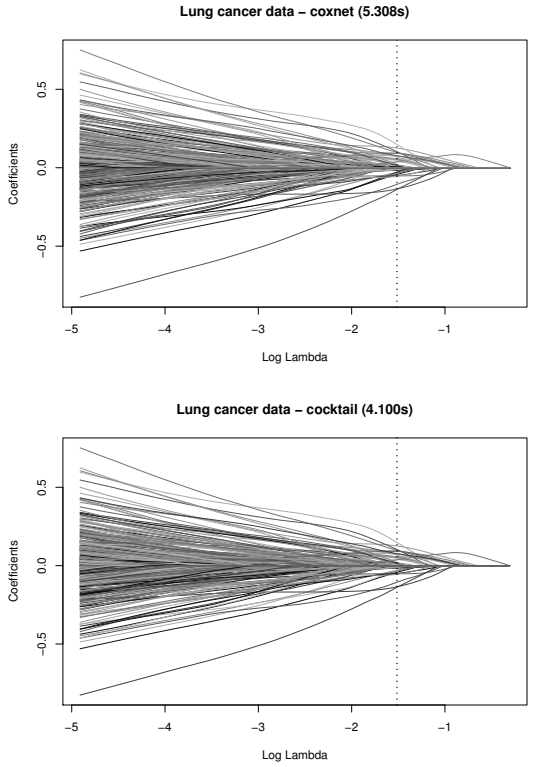

Figure 5.1: Solution paths and timings of the elastic net penalized Cox's model on the lung cancer data from [72] with 86 observations and 7,129 predictors. The top panel shows the solution paths computed by **coxnet** in 5.308 seconds; the bottom panel shows the solution paths computed by **cocktail** in 4.100 seconds. The optimal $(\alpha, \log(\lambda))$ pair is $(0.3, -1.51)$. The elastic net penalized Cox's model, which is indicated by the vertical dotted line, selects 39 genes.

Lung cancer ($n = 86, p = 7,129$)						
α	0.1	0.2	0.3	0.5	0.8	1
Timings						
coxnet	6.30	5.33	5.31	4.57	4.42	5.49
cocktail	4.14	3.60	4.10	4.30	5.58	7.77
KKT Check						
coxnet	42	20	10	1	0	0
cocktail	0	1	1	0	0	0

Table 5.5: Timings (in seconds) and KKT check for **coxnet** and **cocktail** for the lung cancer data from [72]. Reported values for KKT check are the average number of coefficients among 7,129 coefficients that violated the KKT conditions (rounded down to the next smaller integer) using **coxnet** and **cocktail**. Total time for the same λ sequence, averaged over 20 runs.

In Figure 5.1 we also plot the solution paths of the tuned elastic net penalized Cox's model for the lung cancer data. One can see that the two solution path plots are virtually identical. We did a 2-dimension search using 5-fold cross-validation to find the best pair of (α, λ) that incurs maximal log partial likelihood. The fitted penalized Cox model selected 39 genes. It took `cocktail` 4.1 seconds to complete the solution path calculation, while it took `coxnet` 5.3 seconds to get the results.

Chapter 6

Conclusion

6.1 Discussion

In this thesis we have derived a unified algorithm for computing the solution paths of a class of sparse penalized models.

By using the MM principle, we have developed a generalized coordinate descent algorithm called CMD for solving the ℓ_1 penalized regression and logistic regression. The empirical examples suggest that the CMD algorithm can be substantially faster than the original CD algorithm used for the R package glmnet, as long as the correlations among predictors are not weak. The gain in speed is solely due to the use of MM principle within the coordinate decent loop. The R package glmnet2 and the functions used for the simulation in this paper are available at the following publicly accessible webpage `http://code.google.com/p/glmnet2`.

We have also derived a unified blockwise-majorizatoin-descent algorithm for computing the solution paths of a class of group-lasso penalized models. We have demonstrated the efficiency of the algorithm on four group-lasso models: the group-lasso penalized least squares, the group-lasso penalized logistic regression, the group-lasso penalized HSVM and the group-lasso penalized SqSVM. Our algorithm can be readily applied to other interesting group-lasso penalized models. All we need to do is to check the QM condition for the given loss function.

`grplasso` is a popular R package for the group-lasso penalized logistic regression, but the underlying algorithm is limited to twice differentiable loss functions. `SLEP`

implements Nesterov's method for the group-lasso penalized least squares and logistic regression. In principle, Nesterov's method can be used to solve other group-lasso penalized models. For the group-lasso penalized least squares and logistic regression cases, our package `gglasso` is faster than `SLEP` and `grplasso`. Although we do not claim that blockwise-majorizatoin-descent is superior than Nesterov's method, the numerical evidence clearly shows the practical usefulness of blockwise-majorizatoin-descent.

We should point out that Nesterov's method is a more general optimization algorithm than blockwise-descent or blockwise-majorizatoin-descent. Note that blockwise-descent or blockwise-majorizatoin-descent can only work for blockwise separable penalty functions in general. What we have shown in this thesis is that a more general algorithm like Nesterov's method can be slower than a specific algorithm like BMD for a given set of problems. The same message was reported in a comparison done by Tibshirani in which the coordinate descent algorithm was shown to outperform Nesterov's method for the lasso regression. See the details at `http://www-stat.stanford.edu/~tibs/comparison.txt`.

We also presented a generalized coordinate descent algorithm for efficiently computing the solution paths of the HHSVM. We also generalized the GCD to other large margin classifiers and demonstrated their performances. The GCD algorithm has been implemented in an R package `gcdnet` which is publicly available from `http://code.google.com/p/gcdnet/`.

In the optimization literature there are other efficient algorithms for solving the HHSVM and the elastic net penalized squared SVM, logistic regression. [73] proposed a coordinate gradient descent method for solving ℓ_1 penalized smooth optimization problems. [41] proposed the composite gradient mapping idea for minimizing the sum of a smooth convex function and a non-smooth convex function such as the ℓ_1 penalty. These algorithms can be applied to the large margin classifiers considered in this thesis. We do not argue here that GCD is superior than these alternatives. The message we wish to convey is that the marriage between coordinate decent and MM principle could yield an elegant, stable and yet very efficient algorithm for high-dimensional classification.

By combining the strengths of the MM principle, cyclic coordinate descent and the strong rule, we have derived a fast cocktail algorithm for solving the elastic net penalized Cox's model. Our algorithm is comparable in speed to the fastest software for solving

the elastic net penalized Cox's model in the literature.

It is attempting to directly apply the Newton-Raphson algorithm to solve the penalized Cox's model, as done for the penalized logistic regression model in [36]. The difficulty is, as pointed out in [63], the computation of the Hessian matrix of the log partial likelihood is very expensive in the Newton-Raphson loop. To avoid the difficulty [63] only computed the diagonals of the Hessian matrix with respect to $\mathbf{x}^\top\boldsymbol{\beta}$ and pretended the Hessian matrix is a diagonal matrix. However, it is unclear whether their treatment always guarantees `coxnet` to converge to the right solution. It is important to point out that even when we compute the exact Hessian matrix in each Newton-Raphson iteration, the Newton-Raphson update does not always guarantee convergence. A more careful Newton-Raphson algorithm involves step-size adjustment by techniques such as trust-region methods [34]. Such issues do not exist in the cocktail algorithm. Thanks to the MM principle, the cocktail algorithm has a proven convergence property. The R package `fastcox` that implements our cocktail algorithm is available on CRAN and `http://code.google.com/p/fastcox/`.

6.2 Discussion and outlook

We can extend our algorithm to other non-convex penalties. [74] provides the insight into the choice of penalties such that the penalized estimates possesses the properties of sparsity, continuity and unbiasedness. They showed that, to guarantee those conditions, the penalty function must be singular at the origin to and non-convex over $(0, \infty)$. Penalties such as SCAD [74], MCP [75] and SICA [76] satisfy both requirements. They all depend on λ, so denote

$$\lambda P(\boldsymbol{\beta}) = \sum_{j=1}^{p} p_\lambda(|\beta_j|)$$

in (1.1). Their derivative are defined on $[0, \infty)$ by

$$\begin{aligned}
\text{SCAD}: \quad & p'_\lambda(|\beta_j|) = \lambda I(|\beta_j \le \lambda|) + \frac{(a\lambda - |\beta_j|)_+}{(a-1)\lambda} I(|\beta_j| > \lambda) \\
\text{MCP}: \quad & p'_\lambda(|\beta_j|) = \frac{(a\lambda - |\beta_j|)_+}{a} \\
\text{SICA}: \quad & p'_\lambda(|\beta_j|) = \lambda \frac{a(a+1)}{(a+|\beta_j|)^2}
\end{aligned}$$

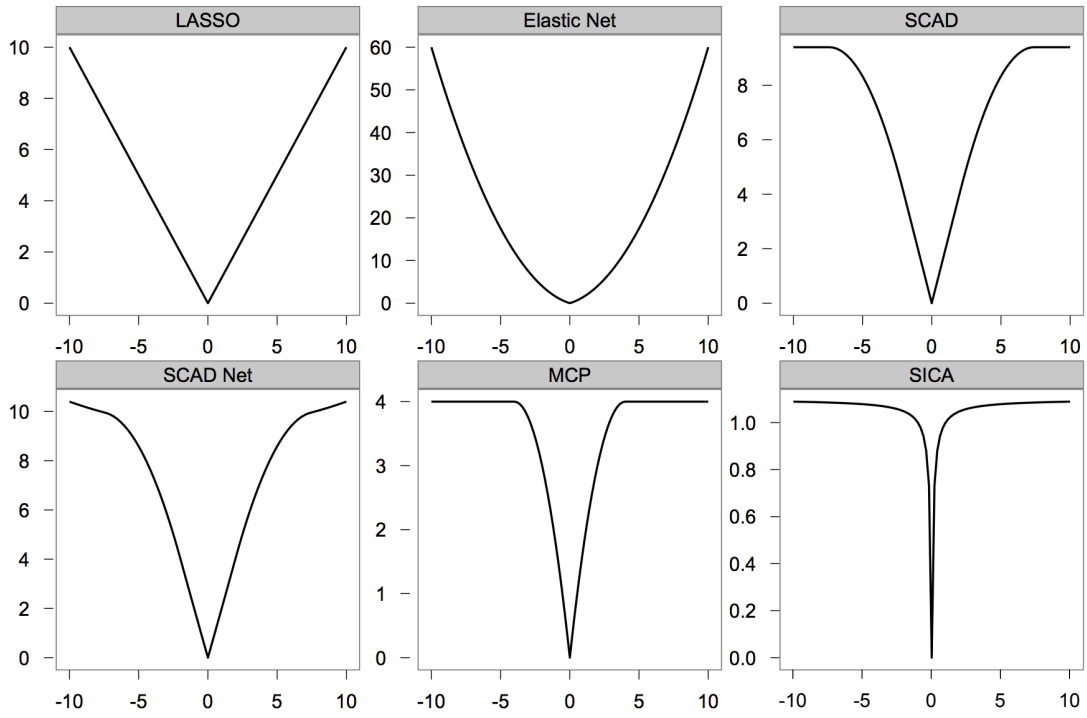

Figure 6.1: Penalty functions. LASSO; Elastic Net: $\lambda = 0.5$; SCAD: $a = 3.7, \lambda = 2$; SCAD Net: $a = 3.7, \lambda = 2, \lambda_2 = 0.01$; MCP : $a = 2, \lambda = 2$; SICA: $a = 0.1$.

where a is a tuning parameter that affects the range over which the penalty is applied. As shown in Figure 6.1, these non-convex penalties share some common features: the similar or same rate of penalization as LASSO is applied when estimate sizes $|\beta_j|$ are small, but that penalization is continuously relaxed as size increases. This assures the elimination of the unimportant variables from the model while leaving the important ones unpenalized.

Despite of all the descent theoretical properties of the non-convex penalties, however, the singularity and non-convexity prevent one from getting a global minimum in optimization problem. To obtain a local minimum, [74] proposed a local quadratic approximation (QMA). While [77] describe a more advantageous local linear approximation (LLA). In LLA they showed for a differentiable concave penalty function $p_\lambda(\cdot)$

on $[0, \infty)$, the local linear approximation to the penalty function

$$l_j(|\beta_j|) \equiv p_\lambda(|\tilde{\beta}_j|) + p'_\lambda(|\tilde{\beta}_j|)(|\beta_j| - |\tilde{\beta}_j|) \geq p_\lambda(|\beta_j|)$$

is a majorization function of $p_\lambda(|\beta_j|)$. So considering the minimization problem with non-convex penalty

$$G(\beta_j) \equiv f(\beta_j | \beta_i = \tilde{\beta}_i, i \neq j) + p_\lambda(|\beta_j|).$$

apply QMC and LLA together we have

$$\beta^* = \underset{\beta_j}{\operatorname{argmin}} H(\beta_j) \equiv q_j(\beta_j) + l_j(|\beta_j|)$$

where q_j is the quadratic majorization and l_j is the local linear approximate. Now we minimize $H(\cdot)$ w.r.t β_j

$$\min_{\beta_j} \left[f'_j(\tilde{\beta}_j)(\beta_j - \tilde{\beta}_j) + \frac{M_j}{2}(\beta_j - \tilde{\beta}_j)^2 + p'_\lambda(|\tilde{\beta}_j|)|\beta_j| \right]$$

by soft-thresholding rule the minimizer β_j^* is,

$$\beta_j^* = \frac{S\left(M_j \tilde{\beta}_j - f'_j(\tilde{\beta}_j), p'_\lambda(|\tilde{\beta}_j|)\right)}{M_j + \lambda_2}$$

where $S(z,t) = (|z| - t)_+ \text{sgn}(z)$. It is not hard for one to show that the local descent property holds. This QMC+LLA idea can be applied during the univariate optimization step of our algorithm. In addition to QMC+LLA, another idea for solving non-convex penalty problem is to apply QMC only and use the concave thresholding rule. Consider minimizing majorization function with SCAD or MCP penalty.

$$\beta_j^* = \underset{\beta_j}{\operatorname{argmin}} \left[f'_j(\tilde{\beta}_j)(\beta_j - \tilde{\beta}_j) + \frac{M_j}{2}(\beta_j - \tilde{\beta}_j)^2 + p_\lambda(|\beta_j|) \right]$$

One can show that by some simple operation it is equivalent to minimize,

$$\beta_j^* = \underset{\beta_j}{\operatorname{argmin}} \left\{ \frac{1}{2}(\beta_j - z)^2 + \Lambda p_\lambda |\beta_j| \right\}, \tag{6.1}$$

where $z = M_j \tilde{\beta}_j - f'_j(\tilde{\beta}_j)$, $\Lambda = M_j^{-1}$. Let $z_0 = \text{sgn}(z)(|z| - \Lambda\lambda)_+$. For SCAD penalty, the univariate penalized least squares problem has an analytical solution [78]. We denote by $q(\beta)$ the objective function and β_j^* the minimizer of (6.1).

1. If $|z| \leq \lambda$, $\beta_j^* = z_0$.
2. If $\lambda < |z| \leq a\lambda$,
   (a) If $|z| \leq (\Lambda + 1)\lambda$, $\beta_j^* = z_0$
   (b) If $|z| > (\Lambda + 1)\lambda$,
   $$\beta_j^* = \text{sgn}(z)\frac{|z| - \Lambda\lambda(a-1)^{-1}a}{1 - (a-1)^{-1}\Lambda}$$
3. If $|z| \leq a\lambda$.
   (a) If $|z| \leq (\Lambda + 1)\lambda$,
      i. If $q(z_0) \leq q(z)$, $\beta_j^* = z_0$
      ii. if $q(z_0) > q(z)$,$\beta_j^* = z$
   (b) If $|z| > (\Lambda + 1)\lambda$, $\beta_j^* = z$

Similarly if we consider MCP penalty in (6.1). It is easy to verify that the minimizer β_j^* is:

1. If $|z| > a\lambda$,
   (a) If $\Lambda a\lambda < z^2$, $\beta_j^* = z$.
   (b) If $\Lambda a\lambda \geq z^2$, $\beta_j^* = 0$.
2. If $|z| \leq a\lambda$,
   (a) If $\Lambda \leq a$, $\beta_j^* = 0$
   (b) If $|z| > (\Lambda + 1)\lambda$,
   $$\beta_j^* = \text{sgn}(z)\left(\frac{|z| - \Lambda\lambda}{1 - a^{-1}\Lambda}\right)_+$$

References

[1] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of statistics*, 32(2):407–451, 2004.

[2] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B*, 67(2):301–320, 2005.

[3] S. Rosset and J. Zhu. Piecewise linear regularized solution paths. *The Annals of Statistics*, 35(3):1012–1030, 2007.

[4] S. Rosset, J. Zhu, and T. Hastie. Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research*, 5:941–973, 2004.

[5] P. Zhao and B. Yu. Stagewise lasso. *The Journal of Machine Learning Research*, 8:2701–2726, 2007.

[6] M.Y. Park and T. Hastie. L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):659–677, 2007.

[7] M. Yuan and H. Zou. Efficient Global Approximation of Generalized Nonlinear l1-Regularized Solution Paths and Its Applications. *Journal of the American Statistical Association*, 104(488):1562–1574, 2009.

[8] W.J. Fu. Penalized regressions: the bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416, 1998.

[9] A. Genkin, D.D. Lewis, and D. Madigan. Large-scale Bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304, 2007.

[10] T.T. Wu and K. Lange. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, 2(1):224–244, 2008.

[11] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, 2007.

[12] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[13] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent. *Journal of Statistical Software*, 39:1–13, 2011.

[14] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B.*, 39(1):1–38, 1977.

[15] J. De Leeuw and W.J. Heiser. Convergence of correction matrix algorithms for multidimensional scaling. *Geometric representations of relational data*, pages 735–752, 1977.

[16] D.R. Hunter and K. Lange. A tutorial on MM algorithms. *The American Statistician*, 58(1):30–37, 2004.

[17] K. Lange, D.R. Hunter, and I. Yang. Optimization transfer using surrogate objective functions. *Journal of Computational and Graphical Statistics*, 9(1):1–20, 2000.

[18] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494, 2001.

[19] L. Wang, J. Zhu, and H. Zou. Hybrid huberized support vector machines for microarray classification and gene selection. *Bioinformatics*, 24(3):412, 2008.

[20] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B.*, 58:267–288, 1996.

[21] H. Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101(476):1418–1429, 2006.

[22] C. Li and H. Li. Network-constrained regularization and variable selection for analysis of genomic data. *Bioinformatics*, 24(9):1175, 2008.

[23] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society, Series B*, 68:49–67, 2006.

[24] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67:301–320, 2005.

[25] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57(11):1413–1457, 2004.

[26] T.T. Wu and K. Lange. The mm alternative to em. *Statistical Science*, 25:492–505, 2010.

[27] R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R.J. Tibshirani. Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2010.

[28] U. Alon, N. Barkai, D.A. Notterman, K. Gish, S. Ybarra, D. Mack, and A.J. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745, 1999.

[29] D. Singh, P.G. Febbo, K. Ross, D.G. Jackson, J. Manola, C. Ladd, P. Tamayo, A.A. Renshaw, A.V. D'Amico, J.P. Richie, et al. Gene expression correlates of clinical prostate cancer behavior. *Cancer cell*, 1(2):203–209, 2002.

[30] A. Frank and A. Asuncion. UCI machine learning repository,university of california, irvine, school of information and computer sciences, http://archive.ics.uci.edu/ml, 2010.

[31] L. Meier, S. van de Geer, and P. Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society, Series B*, 70:53–71, 2008.

[32] M.R. Osborne, B. Presnell, and B.A. Turlach. A new approach to variable selection in least squares problems. *IMA Journal of Numerical Analysis*, 20(3):389–403, 2000.

[33] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communica- tions on Pure and Applied Mathematics,*, 57:1413–1457, 2004.

[34] A. Genkin, D.D. Lewis, and D. Madigan. Large-scale Bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304, 2007.

[35] T.T. Wu and K. Lange. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, 2:224–244, 2008.

[36] J. Friedman, T. Hastie, and R. Tibshirani. Regularized paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33:1–22, 2010.

[37] Z Qin, Katya Scheinberg, and Donald Goldfarb. Efficient block-coordinate descent algorithms for the group lasso. *Optimization Online*, 2010.

[38] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[39] J. Liu, S. Ji, and J. Ye. *SLEP: Sparse Learning with Efficient Projections*. Arizona State University, 2009.

[40] Y. Nesterov. Introductory lectures on convex optimization: A basic course. *Operations Research*, 2004.

[41] Y. Nesterov. Gradient methods for minimizing composite objective function. Technical report, Technical Report, Center for Operations Research and Econometrics (CORE), Catholic University of Louvain (UCL), 2007.

[42] T. Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *Annals of Statistics*, 32:56–85, 2004.

[43] K. Graham, A. de Las Morenas, A. Tripathi, C. King, M. Kavanah, J. Mendez, M. Stone, J. Slama, M. Miller, G. Antoine, et al. Gene expression in histologically normal epithelium from breast cancer patients and from cancer-free prophylactic

mastectomy patients shares a similar profile. *British journal of cancer*, 102(8):1284–1293, 2010.

[44] Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101:1418–1429, 2006.

[45] H. Wang and C. Leng. A note on adaptive group lasso. *Computational Statistics and Data Analysis*, 52:5277–5286, 2008.

[46] K. Lange, D. Hunter, and I. Yang. Optimization transfer using sur- rogate objective functions (with discussion). *Journal of Computational and Graphical Statistics*, 9:1–20, 2000.

[47] T. Wu and K. Lange. The MM alternative to EM. *Statistical Science*, 4:492–505, 2010.

[48] R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R.J. Tibshirani. Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society: Series B*, 74:245–266, 2012.

[49] J.R. Quinlan. Combining instance-based and model-based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 236–243, 1993.

[50] T.E. Scheetz, K.Y.A. Kim, R.E. Swiderski, A.R. Philp, T.A. Braun, K.L. Knudtson, A.M. Dorrance, G.F. DiBona, J. Huang, T.L. Casavant, et al. Regulation of gene expression in the mammalian eye and its relevance to eye disease. *Proceedings of the National Academy of Sciences*, 103(39):14429–14434, 2006.

[51] M.R. Segal, K.D. Dahlquist, and B.R. Conklin. Regression approaches for microarray data analysis. *Journal of Computational Biology*, 10(6):961–980, 2003.

[52] S. Sæbø, T. Almøy, J. Aarøe, and A.H. Aastveit. St-pls: a multi-directional nearest shrunken centroid type classifier via pls. *Journal of Chemometrics*, 22(1):54–62, 2008.

[53] R.P. Gorman and T.J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural networks*, 1(1):75–89, 1988.

[54] Vladimir Vapnik. *The nature of statistical learning theory*. springer, 2000.

[55] T. Hastie, R. Tibshirani, and J.H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Verlag, 2001.

[56] Paul S Bradley and Olvi L Mangasarian. Feature selection via concave minimization and support vector machines. In *ICML*, volume 98, pages 82–90, 1998.

[57] P. Bühlmann and S. van de Geer. *Statistics for High Dimensional Data*. Spinger, 2011.

[58] A.J. van der Kooij. *Prediction accuracy and stability of regression with optimal scaling transformations*. PhD thesis, Child & Family Studies and Data Theory (AGP-D), Department of Education and Child Studies, Faculty of Social and Behavioural Sciences, Leiden University, 2007.

[59] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[60] Theodoros Evgeniou, Massimiliano Pontil, and Tomaso Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

[61] Ji Zhu, Saharon Rosset, Trevor Hastie, and Rob Tibshirani. 1-norm support vector machines. *Advances in neural information processing systems*, 16(1):49–56, 2004.

[62] 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, 1999.

[63] Noah Simon, Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for cox's proportional hazards model via coordinate descent. *Journal of Statistical Software*, 39(5):1–13, 2011.

[64] D. Cox. Regression models and life tables (with discussion). *Journal of the Royal Statistical Society, Series B*, 74:187–220, 1972.

[65] R. Tibshirani. The lasso method for variable selection in the Cox model. *Statistics in Medicine*, 16:385–395, 1997.

[66] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1996.

[67] J.J. Goeman. L1 penalized estimation in the cox proportional hazards model. *Biometrical Journal*, 52(1):70–84, 2010.

[68] Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101:1418–1429, 2006.

[69] J. Friedman, T. Hastie, H. Hoefling, and R. Tibshirani. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 2(1):302–322, 2007.

[70] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood form incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

[71] T.T. Wu and K. Lange. The mm alternative to em. *Statistical Science*, 25(4):492–505, 2010.

[72] D.G. Beer, S.L.R. Kardia, C.C. Huang, T.J. Giordano, A.M. Levin, D.E. Misek, L. Lin, G. Chen, T.G. Gharib, D.G. Thomas, et al. Gene-expression profiles predict survival of patients with lung adenocarcinoma. *Nature medicine*, 8(8):816–824, 2002.

[73] Paul Tseng and Sangwoon Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1-2):387–423, 2009.

[74] J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001.

[75] C.H. Zhang. Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics*, 38(2):894–942, 2010.

[76] J. Lv and Y. Fan. A unified approach to model selection and sparse recovery using regularized least squares. *Annals of Statistics*, 37(6A):3498–3528, 2009.

[77] H. Zou and R. Li. One-step sparse estimates in nonconcave penalized likelihood models. *Annals of Statistics*, 36(4):1509, 2008.

[78] J. Fan and J. Lv. Non-concave penalized likelihood with NP-dimensionality. *Arxiv preprint arXiv:0910.1119*, 2009.

Appendix A

Technical Details in Chapter 3

Appendix: proofs

Proof of Lemma 1. Part (1). For any $\boldsymbol{\beta}$ and $\boldsymbol{\beta}^*$, write $\boldsymbol{\beta} - \boldsymbol{\beta}^* = V$ and define $g(t) = L(\boldsymbol{\beta}^* + tV \mid \mathbf{D})$ so that

$$g(0) = L(\boldsymbol{\beta}^* \mid \mathbf{D}), \quad g(1) = L(\boldsymbol{\beta} \mid \mathbf{D}).$$

By the mean value theorem, $\exists\ a \in (0,1)$ such that

$$g(1) = g(0) + g'(a) = g(0) + g'(0) + [g'(a) - g'(0)]. \tag{A.1}$$

Write $\Phi'_f = \frac{\partial \Phi(y,f)}{\partial f}$. Note that

$$g'(t) = \frac{1}{n} \sum_{i=1}^{n} \tau_i \Phi'_f(y_i, \mathbf{x}_i^\top(\boldsymbol{\beta}^* + tV))(\mathbf{x}_i^\top V). \tag{A.2}$$

Thus $g'(0) = (\boldsymbol{\beta} - \boldsymbol{\beta}^*)^\intercal \nabla L(\boldsymbol{\beta}^* | \mathbf{D})$. Moreover, from (A.2) we have

$$\begin{aligned}
|g'(a) - g'(0)| &= |\frac{1}{n}\sum_{i=1}^{n} \tau_i [\Phi'_f(y_i, \mathbf{x}_i^\intercal(\boldsymbol{\beta}^* + aV)) - \Phi'_f(y_i, \mathbf{x}_i^\intercal \boldsymbol{\beta}^*)](\mathbf{x}_i^\intercal V)| \\
&\leq \frac{1}{n}\sum_{i=1}^{n} \tau_i |\Phi'_f(y_i, \mathbf{x}_i^\intercal(\boldsymbol{\beta}^* + aV)) - \Phi'_f(y_i, \mathbf{x}_i^\intercal \boldsymbol{\beta}^*)||\mathbf{x}_i^\intercal V| \\
&\leq \frac{1}{n}\sum_{i=1}^{n} C\tau_i |\mathbf{x}_i^\intercal aV||\mathbf{x}_i^\intercal V| \qquad \text{(A.3)} \\
&\leq \frac{1}{n}\sum_{i=1}^{n} C\tau_i \|\mathbf{x}_i^\intercal V\|_2^2 \\
&= \frac{C}{n} V^\intercal [\mathbf{X}^\intercal \boldsymbol{\Gamma} \mathbf{X}] V, \qquad \text{(A.4)}
\end{aligned}$$

where in (A.3) we have used the inequality $|\Phi'(y, f_1) - \Phi'(y, f_2)| \leq C|f_1 - f_2|$. Plugging (A.4) into (A.1) we have

$$L(\boldsymbol{\beta} \mid \mathbf{D}) \leq L(\boldsymbol{\beta}^* \mid \mathbf{D}) + (\boldsymbol{\beta} - \boldsymbol{\beta}^*)^\intercal \nabla L(\boldsymbol{\beta}^* | \mathbf{D}) + \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta}^*)^\intercal \mathbf{H} (\boldsymbol{\beta} - \boldsymbol{\beta}^*),$$

with $\mathbf{H} = \frac{2C}{n} \mathbf{X}^\intercal \boldsymbol{\Gamma} \mathbf{X}$.

Part (2). Write $\Phi''_f = \frac{\partial \Phi^2(y,f)}{\partial f^2}$. By Taylor's expansion, $\exists\ b \in (0, 1)$ such that

$$g(1) = g(0) + g'(0) + g''(b). \qquad \text{(A.5)}$$

Note that

$$g''(b) = \frac{1}{n}\sum_{i=1}^{n} \tau_i \Phi''_f(y_i, \mathbf{x}_i^\intercal(\boldsymbol{\beta}^* + bV))(\mathbf{x}_i^\intercal V)^2 \leq \frac{1}{n}\sum_{i=1}^{n} C_2 \tau_i (\mathbf{x}_i^\intercal V)^2, \qquad \text{(A.6)}$$

where we have used the inequality $\Phi''_f \leq C_2$. Plugging (A.6) into (A.5) we have

$$L(\boldsymbol{\beta} \mid \mathbf{D}) \leq L(\boldsymbol{\beta}^* \mid \mathbf{D}) + (\boldsymbol{\beta} - \boldsymbol{\beta}^*)^\intercal \nabla L(\boldsymbol{\beta}^* | \mathbf{D}) + \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta}^*)^\intercal \mathbf{H} (\boldsymbol{\beta} - \boldsymbol{\beta}^*),$$

with $\mathbf{H} = \frac{C_2}{n} \mathbf{X}^\intercal \boldsymbol{\Gamma} \mathbf{X}$.

□