Efficient Regularized Solution Path Algorithms with Applications in Machine Learning and Data Mining

by

Li Wang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Business Administration)
in The University of Michigan
2008

Doctoral Committee:

Professor Michael D. Gordon, Co-Chair
Associate Professor Ji Zhu, Co-Chair
Associate Professor Scott A. Moore
Associate Professor Yaoyun Shi
Assistant Professor Kathy Z. Yuan

TABLE OF CONTENTS

LIST OF FIGURES . . . . . **iv**

LIST OF TABLES . . . . . **vi**

CHAPTER

I. Introduction 1

1.1 Support Vector Machines 1
1.2 The L_1-norm Support Vector Machine 2
1.3 Solution Path Algorithms 4
1.4 Outline of the Dissertation 4

II. Doubly Regularized Support Vector Machines 6

2.1 Background 6
2.2 Grouping Effect of the DrSVM 9
2.3 The DrSVM Algorithms 13
2.3.1 Proof of Theorem II.2 14
2.3.2 Initial Solution 18
2.3.3 Main Program 20
2.3.4 Solution Path for Fixed λ_1 22
2.3.5 Computational Complexity 24
2.4 Numerical Results 24
2.4.1 Simulation 25
2.4.2 Microarray Analysis 29
2.4.3 Handwritten Digit Recognition 32
2.5 Discussion 34

III. Hybrid Huberized Support Vector Machines for Microarray Classification and Gene Selection 37

3.1 Background 37
3.2 The Hybrid Huberized Support Vector Machine 38
3.3 The HHSVM Algorithm 43
3.3.1 Problem Setup 43
3.3.2 Initial Solution 45
3.3.3 Solution Path 46
3.3.4 Computational Complexity 49
3.4 Numerical Results 50
3.4.1 Simulation 50
3.4.2 Real Data Analysis 52
3.5 Conclusion 57

IV. Image Denoising via Solution Paths 58

4.1 Background 58
4.1.1 PCA-based Methods 59
4.2 Models 61
4.3 Solution Path Algorithms 63
4.3.1 Problem Setup 63
4.3.2 Initial Solution 65
4.3.3 Solution Path 66
4.3.4 Computational Complexity 69
4.4 Real Data 70
4.5 Conclusion 75

V. Financial Market Forecasting Using a Two-Step Kernel Learning Method for the Support Vector Regression 76

5.1 Background 76
5.2 Two-Step Kernel Learning for the Support Vector Regression 78
5.2.1 Support Vector Regression 78
5.2.2 Model for a Two-Step Kernel Learning Method 81
5.3 The Solution Path Algorithm 84
5.3.1 Problem Setup 84
5.3.2 Initial Solution 87
5.3.3 Solution Path 88
5.3.4 Computational Complexity 91
5.4 Financial Market Forecasting 92
5.5 Discussion 97

BIBLIOGRAPHY **101**

LIST OF FIGURES

Figure

1.1 The hinge loss of the SVM. *Elbow* indicates the point $1 - yf = 0$, *Left* indicates the region to the left of the elbow, and *Right* indicates the region to the right of the elbow. 3

2.1 2-dimensional contour plots. The L_2-norm $\|\boldsymbol{\beta}\|_2^2 = 1$, the L_1-norm $\|\boldsymbol{\beta}\|_1 = 1$, and the elastic-net $0.5\|\boldsymbol{\beta}\|_2^2 + 0.5\|\boldsymbol{\beta}\|_1 = 1$. 8

2.2 Comparison of different SVMs on a simple simulation data. The solid curves correspond to relevant variables, and the dashed curves correspond to irrelevant variables. The relevant variables are highly correlated. The upper left panel is for the L_2-norm SVM, the upper right panel is for the L_1-norm SVM, the bottom panels are for the DrSVM. The bottom left panel fixes $\lambda_1 = 15$, and changes λ_2; the bottom right panel fixed $\lambda_2 = 160$, and changes λ_1. We can see the DrSVM identified all (correlated) relevant variables, and shrunk their coefficients close to each other. 10

2.3 The simulation setup is the same as in section 5.1, except the size of the training data is $n = 8 + 8$, the number of input variables is $p = 5$, and only the first variable x_1 is relevant to the optimal classification boundary. The solid line corresponds to $\hat{\beta}_1$, the dashed lines correspond to $\hat{\beta}_2, \ldots, \hat{\beta}_5$. The left panel is for $\hat{\boldsymbol{\beta}}_{\lambda_2}(\lambda_1)$ (with $\lambda_2 = 30$), and the right panel is for $\hat{\boldsymbol{\beta}}_{\lambda_1}(\lambda_2)$ (with $\lambda_1 = 6$). 14

2.4 Heatmap of the selected 78 genes. We have ordered the genes by hierarchical clustering, and similarly for all $38 + 34$ samples. 31

2.5 Some examples of handwritten digit 6 and digit 9. Despite the variation in writing style, one can still see the distinctive parts of the shape which best tell apart one class against the other class. 33

2.6 Selected features: the center of each circle indicates the position of the selected feature, and the size of each circle is proportional to the corresponding feature's *importance*. . . . 34

2.7 The hinge loss and the Huberized hinge loss (with $\delta = -1$). The Huberized hinge loss is differentiable everywhere, and has a similar shape as the hinge loss. 36

3.1 The hinge and the huberized hinge loss functions (with $\delta = 2$). Note that the *Elbow* corresponds to the region $(1 - \delta, 1]$; the *Left* and the *Right* correspond to the regions $(-\infty, 1 - \delta]$ and $(1, \infty)$, respectively. 41

3.2 The upper part shows the number of selected genes vs the selection frequency for the L_1-norm SVM (left) and the HHSVM (right) on 100 random splits of the colon cancer dataset. The lower part "zooms in" to the region where the "selection frequency" is bigger than 50 (The left panel is for the L_1-norm SVM and the right panel is for the HHSVM). 56

3.3 Number of samples vs the frequency that a sample (as a test observation) was correctly classified on 100 random splits of the colon cancer dataset. The left panel is for the L_1-norm SVM, and the right panel is for the HHSVM. 56

4.1 The original face image (left panel) and the corresponding image corrupted by impulse noises (right panel). 59

4.2 The first 10 PC-images. 70

4.3 "Complete corruption" scenario. The denoised images using the OLS method (left panel) and the "idealistic" method (right panel). 73

4.4 "Complete corruption" scenario. The PC-projected denoised images, i.e., $\boldsymbol{U}\boldsymbol{\beta}$. Different images correspond to different values of s (from small to large). 73

4.5 "Complete corruption" scenario. The pixel-by-pixel denoised images, i.e., $\boldsymbol{x} + \boldsymbol{\alpha}$, using LAD-LASSO (first row), LS-LASSO (second row), LAD-Adaptive-LASSO (third row) and LS-Adaptive-LASSO (fourth row). Different images correspond to different values of s (from small to large). 74

4.6 Left panel: The corrupted pixels take values between 0 and 255 ("incomplete corruption"). Right panel: The corrupted pixels form a block ("block corruption"). . 74

4.7 The pixel-by-pixel denoised images, i.e., $\boldsymbol{x} + \boldsymbol{\alpha}$, using LAD-Adaptive-LASSO. The first row is for "incompletely corrupted" pixels (left panel in Figure 4.6), and the second row is for "blockly corrupted" pixels (right panel in Figure 4.6). Different images correspond to different values of s (from small to large). 75

5.1 Commonly used loss functions in the SVR 79

5.2 Piecewise linear solution path for a randomly generated data set 91

5.3 Training, validation and test sets 96

5.4 Cumulative log-returns of the four methods on the S&P500 index 98

5.5 Cumulative log-returns of the four methods on the NASDAQ index 99

LIST OF TABLES

Table

2.1 Comparison of the prediction performance when all input variables are independent. p_0 is the number of relevant variables. 26

2.2 Comparison of variable selection when all input variables are independent. p_0 is the number of relevant variables. q_{signal} is the number of selected relevant variables. q_{noise} is the number of selected noise variables. 26

2.3 Comparison of the prediction performance when the relevant variables are highly correlated. p_0 is the number of relevant variables. 28

2.4 Comparison of variable selection when the relevant variables are highly correlated. p_0 is the number of relevant variables. q_{signal} is the number of selected relevant variables. q_{noise} is the number of selected noise variables. 28

2.5 Comparison of the prediction performance when the relevant variables have different class means and the pairwise correlations are not all equal. p_0 is the number of relevant variables. 29

2.6 Comparison of variable selection when the relevant variables have different class means and the pairwise correlations are not all equal. p_0 is the number of relevant variables. q_{signal} is the number of selected relevant variables. q_{noise} is the number of selected noise variables. 29

2.7 Results on the Leukemia Dataset 30

2.8 Result for 6 vs 9 from the MINIST database. Training size is $250 + 250$, and test size is $750 + 750$. Each digit consists of 804 shape features. 34

3.1 Outline of the HHSVM Algorithm 45

3.2 Comparison of test errors. The number of training observations is 50, the number of total input variables is 300, and the number of relevant variables is 10. The results are averages of test errors over 100 repetitions on a 10,000 test set, and the numbers in parentheses are the corresponding standard errors. In "Scenario I", the input variables are independent, and in "Scenario II", the relevant variables are highly correlated. 52

3.3 Comparison of variable selection. The setup are the same as those described in Table 3.4.1. q_{signal} is the number of selected relevant variables, and q_{noise} is the number of selected noise variables. 52

3.4 Results on 100 random splits of the original datasets: the upper part is for the colon cancer dataset, and the lower part is for the lung cancer dataset. The numbers in the parentheses are the corresponding standard errors. 53

3.5 The 32 most frequently selected genes by the L_1-SVM from the colon cancer dataset. "Selection Frequency" is the number of times that the corresponding gene was selected out of 100 random splits of the training and test data. 54

3.6 The 100 most frequently selected genes by the HHSVM from the colon cancer dataset. "Selection Frequency" is the number of times that the corresponding gene was selected out of 100 random splits of the training and test data. 55

4.1 The results are averages over the 20 testing images. The numbers in the parentheses are the corresponding standard errors. "PC-Projection" is for the denoised image $\boldsymbol{U\beta}$, and "Pixel-by-Pixel" is for the denoised image $\boldsymbol{x} + \boldsymbol{\alpha}$. "LAD" is for the LAD-LASSO method (4.3), and "LS" is for the LS-LASSO method (4.4). "LAD-A" and "LS-A" are the corresponding adaptive version (4.5) and (4.6). "OLS" is the ordinary least squares method, and "Ideal" is the "idealistic" method. 72

5.1 Outline of the Algorithm 92

5.2 Performance on the S&P500 and NASDAQ indices. "Average biweekly return" is the average of log-returns over 250 different test sets, and the numbers in the parentheses are the corresponding standard errors. "p-values" were computed from the paired t-test against the buy-and-hold strategy. 97

5.3 Kernel selection on the S&P500 index. Each row corresponds to a candidate kernel; the subscript "1" corresponds to $\sigma^2 = m_j^2$ and the subscript "2" corresponds to $\sigma^2 = 10m_j^2$; the superscript indicates the part of the input features that the kernel is based on. The second column contains the average estimated combining coefficient s_j over the 250 different trials, and the numbers in the parentheses are the corresponding standard errors. The third column records the selection frequency for each kernel out of 250 trials. 100

CHAPTER I

Introduction

Regularization is an essential component in machine learning and data mining. It prevents over-fitting data of a model. In particular, when there are a large number of predictors, possibly more than the number of samples, data fitting without regularization is likely to generate useless models, which can not predict well on unobserved data. In this chapter, we briefly introduce two regularization methods in classification, solution path algorithms and give the outline of this dissertation.

1.1 Support Vector Machines

The support vector machine (SVM) is a widely used tool for classification [71]. It was first motivated by the geometric consideration of maximizing the *margin* [5, 14]. If given a set of training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, where the input $\boldsymbol{x}_i \in \mathbb{R}^p$ is a vector with p predictor variables, and the output $y_i \in \{1, -1\}$ denotes the class label, the SVM finds a hyperplane that separates the two classes of data points by the largest distance:

$$\begin{aligned} \max_{\beta_0, \boldsymbol{\beta}} \quad & \frac{1}{\|\boldsymbol{\beta}\|_2^2} \\ \text{subject to} \quad & y_i(\beta_0 + \boldsymbol{x}_i^{\intercal}\boldsymbol{\beta}) \geq 1 - \xi_i, i = 1, \ldots, n \\ & \xi_i \geq 0, \sum_{i=1}^{n} \xi_i \leq B \end{aligned}$$

where ξ_i are slack variables that describe the overlap between the two classes, and B is a tuning parameter that controls the overlap between the two classes.

Many researchers noted the relationship between the SVM and the regularized function estimation, i.e. the above optimization is equivalent to

$$\min_{\beta_0,\boldsymbol{\beta}} \sum_{i=1}^{n} [1 - y_i(\beta_0 + \boldsymbol{x}_i^{\intercal}\boldsymbol{\beta})]_+ + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2 \tag{1.1}$$

where λ is the tuning parameter, playing the same role as B. The classification rule for a new input $\boldsymbol{x}$ is then given by $\text{sign}(\beta_0 + \boldsymbol{x}^{\intercal}\boldsymbol{\beta})$. Notice (1.1) has the form *loss* + *penalty*; hence λ controls the balance between the loss and the penalty. The function $(1-\cdot)_+$ is called the *hinge loss*, and is plotted in Figure 1.1. Notice that it has a non-differentiable point at 1. The penalty is the L_2-norm of the coefficient vector, the same as that used in the ridge regression [36]. The idea of penalizing by the sum of squares of the parameters is also used in neural networks, where it is known as *weight decay*. The ridge penalty shrinks the fitted coefficients toward zero. It is well known that this shrinkage has the effect of controlling the variance of fitted coefficients, hence possibly improving the fitted model's prediction accuracy via the bias-variance trade-off, especially when there are many highly correlated variables. An overview of the SVM as regularized function estimation can be found in [9, 18, 72, 34].

1.2 The L_1-norm Support Vector Machine

Instead of using the L_2-norm penalty, several researchers have considered replacing it in (1.1) with the L_1-norm penalty, and fitting an L_1-norm SVM model [6, 65, 79]:

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

The L_1-norm penalty was first used for signal processing and regression problems by [47, 68, 13].

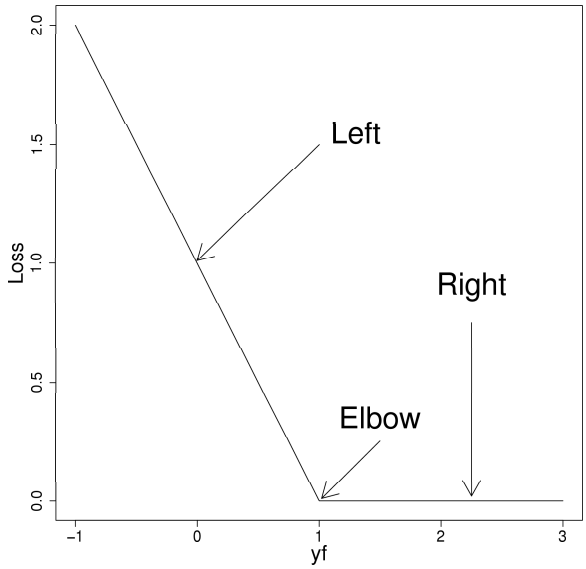

Figure 1.1: The hinge loss of the SVM. *Elbow* indicates the point $1 - yf = 0$, *Left* indicates the region to the left of the elbow, and *Right* indicates the region to the right of the elbow.

Similar to the L_2-norm penalty, the L_1-norm penalty also shrinks the fitted coefficients toward zero, which also benefits from the reduction in the fitted coefficients' variance. Another important property of the L_1-norm penalty is that because of its L_1 nature, making λ sufficiently large will cause some of the fitted coefficients be *exactly* zero. Thus, as λ varies, the L_1-norm penalty performs a kind of continuous variable selection, while this is not the case for the L_2-norm penalty. It is interesting to note that the L_2-norm penalty corresponds to a Gaussian prior for the β_j's, while the L_1-norm penalty corresponds to a double-exponential prior. The double-exponential density has heavier tails than the Gaussian density. This reflects the greater tendency of the L_1-norm penalty to produce some large fitted coefficients and leave others at 0, especially in high dimensional problems. See Figure 2.1 for contours of the L_2-norm penalty and the L_1-norm penalty.

1.3 Solution Path Algorithms

Let's consider a generic regularized optimization problem $\hat{\boldsymbol{\beta}} = arg\min_{\boldsymbol{\beta}} L(y, \boldsymbol{X}\boldsymbol{\beta}) + \lambda J(\boldsymbol{\beta})$, where $L(\cdot)$ and $J(\cdot)$ are the loss and penalty functions, respectively. [59] studies the general characterization of (L, J) pairs, whose corresponding optimal solution $\hat{\boldsymbol{\beta}}(\lambda)$ has the piecewise linear property, i.e. $\partial\hat{\boldsymbol{\beta}}(\lambda)/\partial\lambda$ is piecewise constant with respect to λ. For (L, J) pairs with piecewise linear property, efficient solution path algorithms can be derived and they calculate the optimal solutions $\hat{\boldsymbol{\beta}}(\lambda)$ for every possible value of λ. The LASSO [68] is a canonical example. [17] proves the piecewise linear property for the LASSO and the LAR-LASSO algorithm is developed, which calculates the entire solution path for the LASSO.

1.4 Outline of the Dissertation

In this dissertation, we propose four machine learning and data mining methods based on the regularization idea and we develop their solution path algorithms. Chapter 2 proposes the doubly regularized support vector machine (DrSVM) for classification. The DrSVM uses the *elastic-net* penalty, a mixture of the L_2-norm and the L_1-norm penalties. By doing so, the DrSVM performs automatic variable selection in a way similar to the L_1-norm SVM. In addition, the DrSVM encourages highly correlated variables to be selected (or removed) together, which is called the *grouping effect*. We also develop solution path algorithms for the DrSVM. Based on the DrSVM, chapter 3 proposes the hybrid huberized support vector machine (HHSVM). The HHSVM uses the *elastic-net* penalty and the huberized hinge loss function. Similar to the DrSVM, the HHSVM performs automatic variable selection and has the *grouping effect*. Furthermore, since the huberized hinge loss function is differentiable everywhere, the computational cost is significantly reduced for calculating its solu-

tion path. Chapter 4 proposes two models for image denoising, where the L_1-norm of the pixel updates is used as the penalty. The L_1-norm penalty has the advantage of changing only the noisy pixels, while leaving the non-noisy pixels untouched. Efficient algorithms are designed to compute entire solution paths of these L_1-norm penalized models, which facilitate the selection of a balance between the "loss" and the "penalty." Chapter 5 proposes a two-step kernel learning method based on the support vector regression (SVR) for financial time series forecasting. Given a number of candidate kernels, our method learns a sparse linear combination of these kernels so that the resulting kernel can be used to predict well on future data. The L_1-norm regularization approach is used to achieve kernel learning. Since the regularization parameter must be carefully selected, to facilitate parameter tuning, we develop an efficient solution path algorithm that solves the optimal solutions for all possible values of the regularization parameter.

CHAPTER II

Doubly Regularized Support Vector Machines

2.1 Background

It has been argued that the L_1-norm penalty has advantages over the L_2-norm penalty under certain scenarios [16, 23, 51], such as when there are redundant noise variables. However, the L_1-norm penalty also suffers from two serious limitations [81]:

1. When there are several highly correlated input variables in the data set, and they are all relevant to the output variable, the L_1-norm penalty tends to pick only one or few of them and shrinks the rest to 0. For example, in microarray analysis, expression levels for genes that share one biological pathway are usually highly correlated, and these genes all contribute to the biological process, but the L_1-norm penalty usually selects only one gene from the group, and does not care which one is selected. The ideal method should be able to eliminate trivial genes, and automatically include the whole group of relevant genes.

2. In the $p > n$ case, as shown in [60], the L_1-norm penalty can keep at most n input variables. Again, we use microarray as an example: the sample size n is usually on the order of 10 or 100, while the dimension of the input p is typically on the order of $1,000$ or even $10,000$. Using the L_1-norm penalty can, at most, identify n non-zero fitted coefficients, but it is unlikely that only 10 genes are

involved in a complicated biological process.

Zou and Hastie [81] proposed the *elastic-net* penalty to fix these two limitations. The elastic-net penalty is a mixture of the L_1-norm penalty and the L_2-norm penalty, combining good features of the two. Similar to the L_1-norm penalty, the elastic-net penalty simultaneously performs automatic variable selection and continuous shrinkage; the new advantages are that groups of correlated variables now can be selected together, and the number of selected variables is no longer limited by n.

In this paper, we apply the elastic-net penalty to the support vector machine. Specifically, we consider the following doubly regularized support vector machine, which we call the DrSVM:

$$\min_{\beta_0,\boldsymbol{\beta}} \sum_{i=1}^{n} [1 - y_i(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta})]_+ + \frac{\lambda_2}{2}\|\boldsymbol{\beta}\|_2^2 + \lambda_1\|\boldsymbol{\beta}\|_1 \tag{2.1}$$

where both λ_1 and λ_2 are tuning parameters. The role of the L_1-norm penalty is to allow variable selection, and the role of the L_2-norm penalty is to help groups of correlated variables get selected together. We show that for classification problems, the L_2-norm penalty tends to make highly correlated input variables have similar fitted coefficients, which is the *grouping effect*. We also see the number of selected input variables is not limited by n anymore. Figure 2.1 compares contours of the L_2-norm, the L_1-norm, and the elastic-net penalty.

To get a good classification rule that performs well on future data, it is important to select appropriate tuning parameters λ_1 and λ_2. In practice, people can prespecify a finite grid of values for λ_1 and λ_2 that covers a wide range, then use either a separate validation data set or cross-validation to do a grid search, and find values for the $(\lambda_1,\ \lambda_2)$ pair that give the best performance among the given grid. In this paper, we illustrate that the solution path for a fixed value of λ_2, denoted as $\boldsymbol{\beta}_{\lambda_2}(\lambda_1)$,

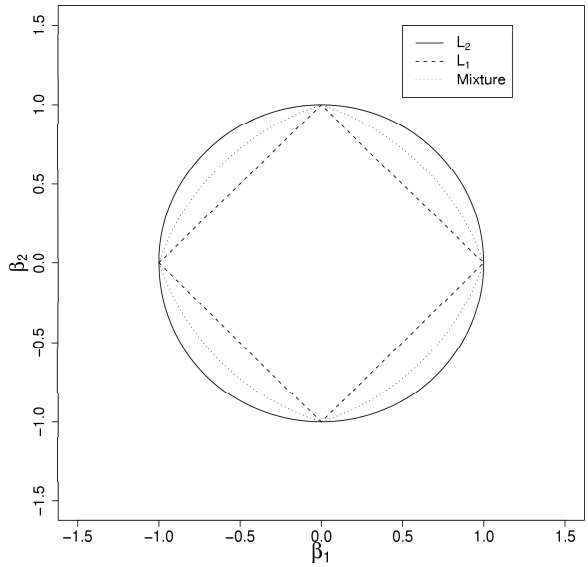

Figure 2.1: 2-dimensional contour plots. The L_2-norm $\|\boldsymbol{\beta}\|_2^2 = 1$, the L_1-norm $\|\boldsymbol{\beta}\|_1 = 1$, and the elastic-net $0.5\|\boldsymbol{\beta}\|_2^2 + 0.5\|\boldsymbol{\beta}\|_1 = 1$.

is *piece-wise linear* as a function of λ_1 (in the $\mathbb{R}^p$ space); and for a fixed value of λ_1, the solution path, denoted as $\boldsymbol{\beta}_{\lambda_1}(\lambda_2)$, is piece-wise linear as a function of $1/\lambda_2$. We further propose efficient algorithms to compute the exact whole solution paths, that help us understand how the solution changes with λ_1 and λ_2 and facilitate the adaptive selection of the tuning parameters.

Before delving into the technical details, we illustrate the concept of grouping effect and piece-wise linearity of the solution paths $\boldsymbol{\beta}_{\lambda_2}(\lambda_1)$ and $\boldsymbol{\beta}_{\lambda_1}(\lambda_2)$ with a simple example. We generate 30 training data in each of two classes. Each input $\boldsymbol{x}_i$ is a $p = 30$ dimensional vector. For the "+" class, $\boldsymbol{x}_i$ has a normal distribution with mean and covariance matrix

$$\begin{aligned} \boldsymbol{\mu}_+ &= (\underbrace{1,\ldots,1}_{5},\underbrace{0,\ldots,0}_{25})^{\mathsf{T}} \\ \boldsymbol{\Sigma} &= \begin{pmatrix} \boldsymbol{\Sigma}^*_{5\times 5} & \mathbf{0}_{5\times 25} \\ \mathbf{0}_{25\times 5} & \boldsymbol{I}_{25\times 25} \end{pmatrix} \end{aligned}$$

where the diagonal elements of $\mathbf{\Sigma}^*$ are 1 and the off-diagonal elements are all equal to $\rho = 0.8$. The "$-$" class has a similar distribution, except that

$$\boldsymbol{\mu}_- = (\underbrace{-1, \ldots, -1}_{5}, \underbrace{0, \ldots, 0}_{25})^{\mathsf{T}}$$

So $x_1, \ldots, x_5$ are highly correlated, the Bayes optimal classification boundary is given by

$$x_1 + \cdots + x_5 = 0$$

and the Bayes error is 0.138. Figure 2.2 compares the result from the standard L_2-norm SVM, the L_1-norm SVM, and the DrSVM. The solid paths are for $x_1, \ldots, x_5$, which are the relevant variables; the dashed paths are for $x_6, \ldots, x_{30}$, which are the irrelevant variables. As we can see, the L_2-norm SVM kept all variables in the fitted model, the L_1-norm SVM did variable selection, but failed to identify the group of correlated variables; the DrSVM successfully selected all five relevant variables, and shrunk their coefficients close to each other.

In section 2.2, we show the grouping effect of the DrSVM. In section 4.3, we describe algorithms that compute the whole solution paths of the DrSVM. In section 4.4, we present numerical results on both simulation data and real world data. We conclude the paper with a discussion section.

2.2 Grouping Effect of the DrSVM

In this section, we illustrate that the DrSVM has the grouping effect for correlated variables. The result holds not only for the hinge loss function of the SVM, but also for general Lipschitz continuous loss functions.

Consider the following more general optimization problem:

$$\min_{\beta_0, \boldsymbol{\beta}} \sum_{i=1}^{n} \phi(y_i, f(\boldsymbol{x}_i)) + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2^2 + \lambda_1 \|\boldsymbol{\beta}\|_1 \tag{2.2}$$

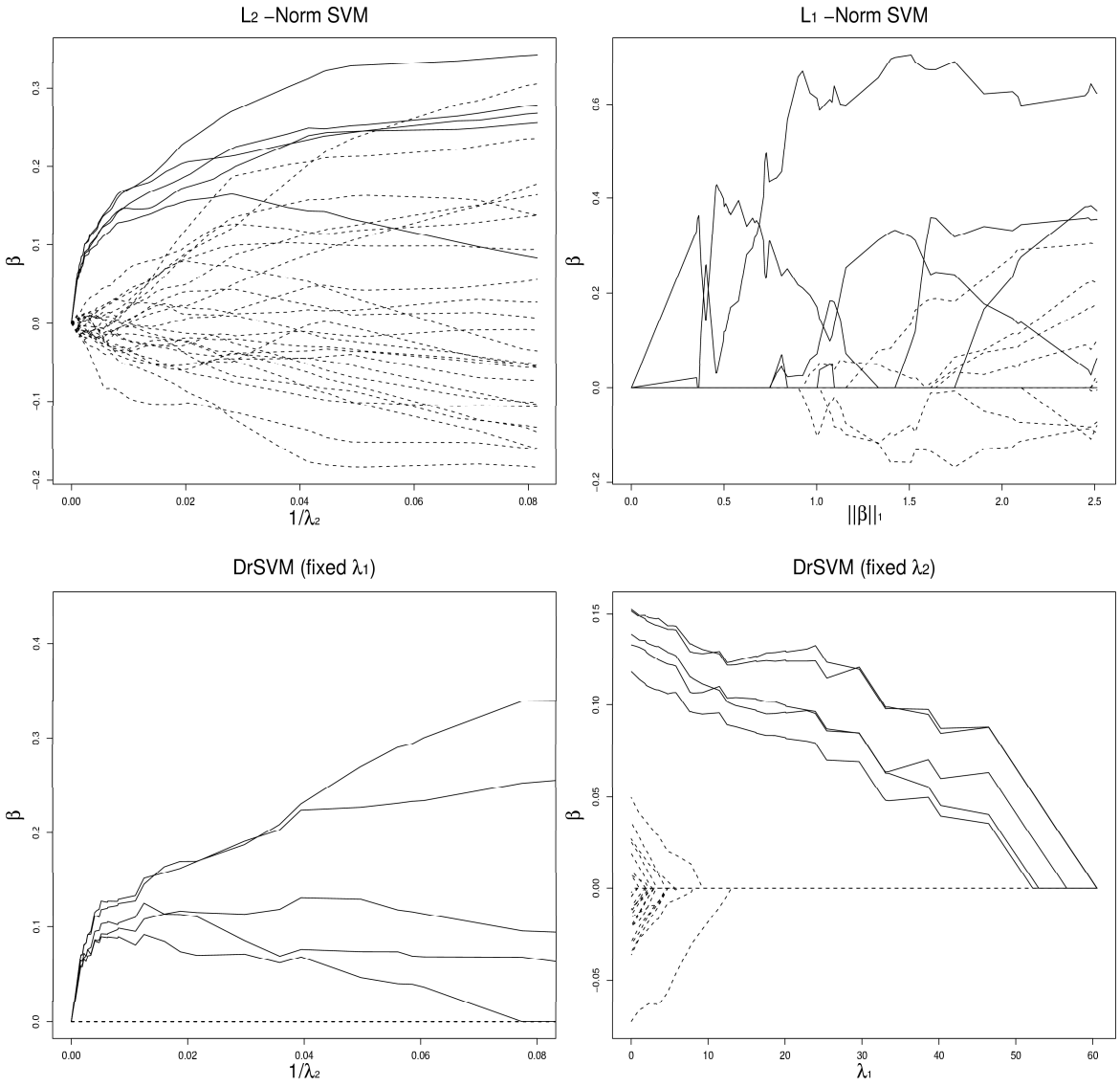

Figure 2.2: Comparison of different SVMs on a simple simulation data. The solid curves correspond to relevant variables, and the dashed curves correspond to irrelevant variables. The relevant variables are highly correlated. The upper left panel is for the L_2-norm SVM, the upper right panel is for the L_1-norm SVM, the bottom panels are for the DrSVM. The bottom left panel fixes $\lambda_1 = 15$, and changes λ_2; the bottom right panel fixed $\lambda_2 = 160$, and changes λ_1. We can see the DrSVM identified all (correlated) relevant variables, and shrunk their coefficients close to each other.

where $f(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}_i^\intercal \boldsymbol{\beta}$, $\phi(y, f) = \phi(yf)$ is a function of the *margin*. We further assume that $\phi(t)$ is Lipschitz continuous, i.e.

$$|\phi(t_1) - \phi(t_2)| \leq M|t_1 - t_2| \text{ for some positive finite } M.$$

It is simple to verify that this condition holds for many commonly used loss functions for classification, for example, the hinge loss function (SVM) and the binomial deviance (logistic regression). Then we have the following theorem:

Theorem II.1. *Denote the solution to (2.2) as $\hat{\beta}_0$ and $\hat{\boldsymbol{\beta}}$. Assume the loss function ϕ is Lipschitz continuous, then for any pair (j, l), we have*

$$\left|\hat{\beta}_j - \hat{\beta}_l\right| \leq \frac{M}{\lambda_2}\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_1 = \frac{M}{\lambda_2}\sum_{i=1}^{n}|x_{ij} - x_{il}| \tag{2.3}$$

Furthermore, if the input variable $\boldsymbol{x}_j, \boldsymbol{x}_l$ are centered and normalized, then

$$\left|\hat{\beta}_j - \hat{\beta}_l\right| \leq \frac{\sqrt{n}M}{\lambda_2}\sqrt{2(1-\rho)} \tag{2.4}$$

where $\rho = cor(\boldsymbol{x}_j, \boldsymbol{x}_l)$ is the sample correlation between $\boldsymbol{x}_j$ and $\boldsymbol{x}_l$.

Proof

Consider another set of coefficients

$$\hat{\beta}_0^* = \hat{\beta}_0,\ \hat{\beta}_{j'}^* = \begin{cases} \frac{1}{2}(\hat{\beta}_j + \hat{\beta}_l) & \text{if } j' = j \text{ or } j' = l \\ \hat{\beta}_{j'} & \text{otherwise} \end{cases}$$

By the definition of $\hat{\beta}_0$ and $\hat{\boldsymbol{\beta}}$, we have

$$\sum_{i=1}^{n}\phi(y_i, \hat{\beta}_0^* + \boldsymbol{x}_i^\intercal\hat{\boldsymbol{\beta}}^*) + \frac{\lambda_2}{2}\|\hat{\boldsymbol{\beta}}^*\|_2^2 + \lambda_1\|\hat{\boldsymbol{\beta}}^*\|_1 - \sum_{i=1}^{n}\phi(y_i, \hat{\beta}_0 + \boldsymbol{x}_i^\intercal\hat{\boldsymbol{\beta}}) - \frac{\lambda_2}{2}\|\hat{\boldsymbol{\beta}}\|_2^2 - \lambda_1\|\hat{\boldsymbol{\beta}}\|_1 \geq 0 \tag{2.5}$$

where

$$
\begin{aligned}
& \sum_{i=1}^{n} \left[\phi(y_i, \hat{\beta}_0^* + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}}^*) - \phi(y_i, \hat{\beta}_0 + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}}) \right] \\
\leq & \sum_{i=1}^{n} \left| \phi(y_i, \hat{\beta}_0^* + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}}^*) - \phi(y_i, \hat{\beta}_0 + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}}) \right| \\
\leq & \sum_{i=1}^{n} M \left| y_i(\hat{\beta}_0^* + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}}^*) - y_i(\hat{\beta}_0 + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}}) \right| \qquad (2.6) \\
= & \sum_{i=1}^{n} M \left| \boldsymbol{x}_i^\intercal (\hat{\boldsymbol{\beta}}^* - \hat{\boldsymbol{\beta}}) \right| \\
= & \sum_{i=1}^{n} M \left| \frac{1}{2}(x_{ij} - x_{il})(\hat{\beta}_j - \hat{\beta}_l) \right| \\
= & \frac{M}{2} \left| \hat{\beta}_j - \hat{\beta}_l \right| \sum_{i=1}^{n} |x_{ij} - x_{il}| \\
= & \frac{M}{2} \left| \hat{\beta}_j - \hat{\beta}_l \right| \cdot \|\boldsymbol{x}_j - \boldsymbol{x}_l\|_1. \qquad (2.7)
\end{aligned}
$$

We also have

$$
\begin{aligned}
\|\hat{\boldsymbol{\beta}}^*\|_1 - \|\hat{\boldsymbol{\beta}}\|_1 &= |\hat{\beta}_j^*| + |\hat{\beta}_l^*| - |\hat{\beta}_j| - |\hat{\beta}_l| \\
&= |\hat{\beta}_j + \hat{\beta}_l| - |\hat{\beta}_j| - |\hat{\beta}_l| \leq 0 \qquad (2.8)
\end{aligned}
$$

and

$$
\begin{aligned}
\|\hat{\boldsymbol{\beta}}^*\|_2^2 - \|\hat{\boldsymbol{\beta}}\|_2^2 &= |\hat{\beta}_j^*|^2 + |\hat{\beta}_l^*|^2 - |\hat{\beta}_j|^2 - |\hat{\beta}_l|^2 \\
&= -\frac{1}{2}|\hat{\beta}_j - \hat{\beta}_l|^2 \qquad (2.9)
\end{aligned}
$$

Now combining (3.6), (3.7) and (3.8), (3.4) implies that

$$
\frac{M}{2} \left| \hat{\beta}_j - \hat{\beta}_l \right| \cdot \|\boldsymbol{x}_j - \boldsymbol{x}_l\|_1 - \frac{\lambda_2}{2}|\hat{\beta}_j - \hat{\beta}_l|^2 \geq 0. \qquad (2.10)
$$

Hence, (2.3) is obtained.

For (2.4), we simply use the inequality

$$
\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_1 \leq \sqrt{n}\sqrt{\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2^2} = \sqrt{n}\sqrt{2(1-\rho)}. \qquad (2.11)
$$

□

We used the Lipschitz continuity in (3.5), where it was applied to loss functions for classification, i.e., functions of the margin. For the hinge loss, it is easy to see that the Lipschitz constant $M = 1$, hence, Theorem II.1 holds for the DrSVM. It is also worth noting that the theorem holds for all $\lambda_1 \geq 0$, so the grouping effect is from the L_2-norm penalty.

2.3 The DrSVM Algorithms

In this section, we propose algorithms that can solve the DrSVM efficiently. Specifically, we propose algorithms that can solve the whole solution path $\boldsymbol{\beta}_{\lambda_2}(\lambda_1)$ (when λ_2 is fixed) and $\boldsymbol{\beta}_{\lambda_1}(\lambda_2)$ (when λ_1 is fixed). Our algorithms hinge on the following two results:

Theorem II.2. *When λ_2 is fixed, the solution $\boldsymbol{\beta}_{\lambda_2}(\lambda_1)$ is a piecewise linear function of λ_1.*

Theorem II.3. *When λ_1 is fixed, the solution $\boldsymbol{\beta}_{\lambda_1}(\lambda_2)$ is a piecewise linear function of $1/\lambda_2$.*

Figure 2.3 illustrates the point. Any segment between two adjacent vertical lines is linear. When λ_2 is fixed, the basic idea of our algorithm is to start with λ_1 equal to ∞, find the direction of the linear path, move the solution in that direction until it hits a *joint* (the asterisk points in Figure 2.3), then adjust the direction of the path, and move on. The algorithm when λ_1 is fixed functions in a similar manner (the right panel of Figure 2.3).

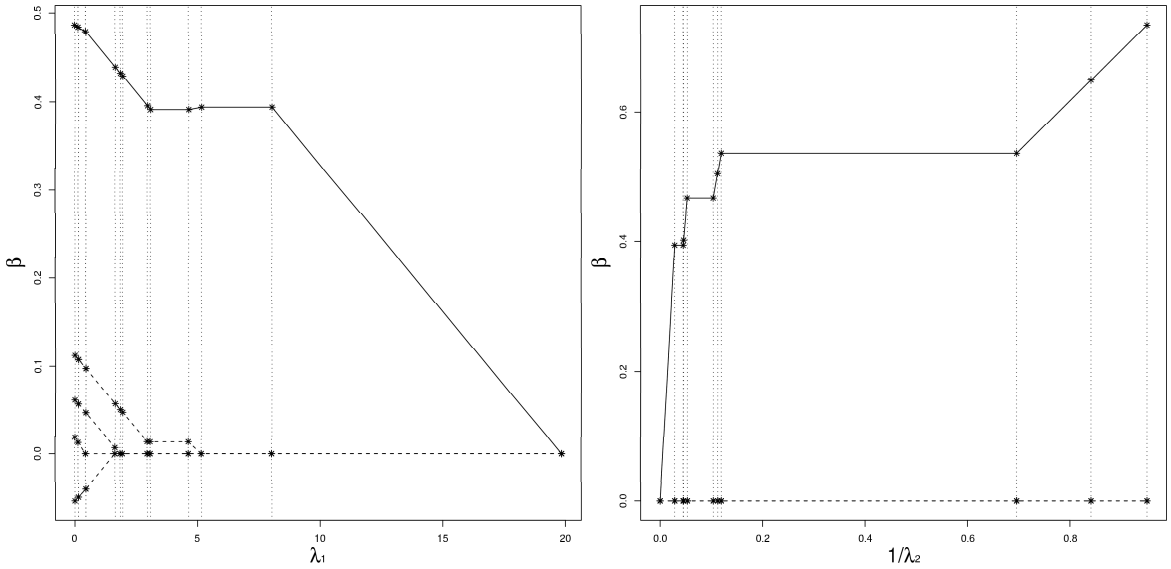

Figure 2.3: The simulation setup is the same as in section 5.1, except the size of the training data is $n = 8 + 8$, the number of input variables is $p = 5$, and only the first variable x_1 is relevant to the optimal classification boundary. The solid line corresponds to $\hat{\beta}_1$, the dashed lines correspond to $\hat{\beta}_2, \ldots, \hat{\beta}_5$. The left panel is for $\hat{\boldsymbol{\beta}}_{\lambda_2}(\lambda_1)$ (with $\lambda_2 = 30$), and the right panel is for $\hat{\boldsymbol{\beta}}_{\lambda_1}(\lambda_2)$ (with $\lambda_1 = 6$).

2.3.1 Proof of Theorem II.2

The optimization problem (2.1) for the DrSVM is equivalent to a quadratic programming problem:

$$\min_{\beta_0, \boldsymbol{\beta}} \quad \sum_{i=1}^{n} \epsilon_i + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2^2 \tag{2.12}$$

$$\text{subject to} \quad 1 - y_i f_i \leq \epsilon_i \tag{2.13}$$

$$\epsilon_i \geq 0, \ i = 1, \ldots, n \tag{2.14}$$

$$\|\boldsymbol{\beta}\|_1 = |\beta_1| + \cdots + |\beta_p| \leq s \tag{2.15}$$

where $f_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$. Notice the hinge loss is replaced by a linear constraint, the L_1-norm penalty is replaced by an L_1-norm constraint, and the tuning parameter λ_1 is replaced by s. The optimization problem (2.1) and the quadratic programming problem are equivalent in the sense that for any value of λ_1, there exists a value of s, such that the solution to (2.1) and the solution to the quadratic programming problem are identical. To solve for the quadratic programming problem, we write the

Lagrange:

$$\sum_{i=1}^{n}\epsilon_i+\frac{\lambda_2}{2}\|\boldsymbol{\beta}\|_2^2+\sum_{i=1}^{n}\alpha_i(1-y_if_i-\epsilon_i)-\sum_{i=1}^{n}\gamma_i\epsilon_i+\eta(\sum_{j=1}^{p}|\beta_j|-s)$$

where $\alpha_i \geq 0$, $\gamma_i \geq 0$ and $\eta \geq 0$ are Lagrange multipliers. Taking derivative of the Lagrange with respect to $\beta_0, \boldsymbol{\beta}$ and ϵ_i, we have

- $\sum_{i=1}^{n}\alpha_i y_i = 0$
- $\lambda_2\beta_j - \sum_{i=1}^{n}\alpha_i y_i x_{ij} + \eta\text{sign}(\beta_j) = 0$, for $j \in \mathcal{V}$
- $1-\alpha_i-\gamma_i = 0,\ i = 1,\ldots,n$

where $\mathcal{V}$ contains the indices of non-zero coefficients, i.e. $\mathcal{V} = \{j : \beta_j \neq 0\}$. Notice the value of β_j is fully determined by the values of α_i and η. We also have the Karush-Kuhn-Tucker (KKT) conditions from the quadratic programming:

- $\alpha_i(1-y_if_i-\epsilon_i) = 0,\ i = 1,\ldots,n$
- $\gamma_i\epsilon_i = 0,\ i = 1,\ldots,n$
- $\eta(\sum_{i=1}^{p}|\beta_j|-s) = 0$

We use $\mathcal{L}$ (Left) to denote the set of data points for which $1-y_if_i > 0$, $\mathcal{R}$ (Right) for $1-y_if_i < 0$, and $\mathcal{E}$ (Elbow) for $1-y_if_i = 0$ (See Figure 1.1). Inspecting the KKT conditions, we find

- $i \in \mathcal{L} \Longrightarrow \gamma_i = 0, \alpha_i = 1$
- $i \in \mathcal{R} \Longrightarrow \gamma_i = 1, \alpha_i = 0$
- $i \in \mathcal{E} \Longrightarrow 0 \leq \gamma_i, \alpha_i \leq 1$ and $\gamma_i + \alpha_i = 1$

So, for data points in $\mathcal{L}$ and $\mathcal{R}$, their α_i are determined. To solve for β_j, we also need α_i values for data points in $\mathcal{E}$, especially how these values change (between 0 and 1) when s increases.

When s is small enough, the constraint (2.15) is active, i.e. $\|\boldsymbol{\beta}\|_1 = s$. When s increases to a certain value, say s^*, this constraint will become inactive, and the solution will not change beyond the value of s^*. This corresponds to $\lambda_1 = 0$ in (2.1). Suppose for a value $s < s^*$, the solution is $(\beta_0, \boldsymbol{\beta})$, hence $\mathcal{V}, \mathcal{L}, \mathcal{R}$ and $\mathcal{E}$ are also known. Then $(\beta_0, \boldsymbol{\beta})$ have to satisfy the following equations derived from the Lagrange and KKT conditions:

$$\lambda_2\beta_j - \sum_{i=1}^{n} \alpha_i y_i x_{ij} + \eta \text{sign}(\beta_j) = 0 \; j \in \mathcal{V} \tag{2.16}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.17}$$

$$y_i(\beta_0 + \sum_{j\in\mathcal{V}} \beta_j x_{ij}) = 1 \; i \in \mathcal{E} \tag{2.18}$$

$$\|\boldsymbol{\beta}\|_1 = \sum_{j\in\mathcal{V}} \text{sign}(\beta_j)\beta_j = s \tag{2.19}$$

This linear system consists of $|\mathcal{E}| + |\mathcal{V}| + 2$ equations and $|\mathcal{E}| + |\mathcal{V}| + 2$ unknowns: α_i's, β_j's, β_0 and η. They can be further reduced to $|\mathcal{E}| + 2$ equations and $|\mathcal{E}| + 2$ unknowns by plugging (2.16) into (2.18) and (2.19). If the system is nonsingular, the solution is unique. In the case of singularity, the optimal solution is not unique, but the optimal region can still be determined.

When s increases by a small enough amount, by continuity, the sets $\mathcal{V}, \mathcal{L}, \mathcal{R}$ and $\mathcal{E}$ will not change, such that the structure of the above linear system will not change. Taking right derivatives with respect to s, we have

$$\lambda_2\frac{\Delta\beta_j}{\Delta s} - \sum_{i\in\mathcal{E}} \frac{\Delta\alpha_i}{\Delta s} y_i x_{ij} + \text{sign}(\beta_j)\frac{\Delta\eta}{\Delta s} = 0 \; j \in \mathcal{V} \tag{2.20}$$

$$\sum_{i\in\mathcal{E}} \frac{\Delta\alpha_i}{\Delta s} y_i = 0 \tag{2.21}$$

$$\frac{\Delta\beta_0}{\Delta s} + \sum_{j\in\mathcal{V}} \frac{\Delta\beta_j}{\Delta s} x_{ij} = 0, \; i \in \mathcal{E} \tag{2.22}$$

$$\sum_{j\in\mathcal{V}} \text{sign}(\beta_j)\frac{\Delta\beta_j}{\Delta s} = 1 \tag{2.23}$$

which does not depend on the value of s. This implies that the solution, α_i's, β_j's, β_0 and η, will change linearly in s. When the increase in s is big enough, one of the $\mathcal{V}, \mathcal{L}, \mathcal{R}$ and $\mathcal{E}$ sets will change, so the structure of the linear system will change, which corresponds to a different linear piece on the solution path. Hence, the solution path is piecewise linear in s. Notice that η is equivalent to λ_1; therefore $\beta_0, \boldsymbol{\beta}$ and α_i are also piecewise linear in λ_1, and Theorem II.2 holds. □

To identify changes in the structure of the linear system (or the asterisk points in Figure 2.3), we define four types of *events*, corresponding to the changes in $\mathcal{V}, \mathcal{L}, \mathcal{R}$ and $\mathcal{E}$:

1. A data point leaves $\mathcal{E}$ to $\mathcal{L}$ or $\mathcal{R}$. This happens when an α_i changes from within the region $(0, 1)$ to the boundary 1 or 0.

2. A data point reaches $\mathcal{E}$ from $\mathcal{L}$ or $\mathcal{R}$. This happens when a residual $(1 - y_i f_i)$ reaches 0.

3. An active variable in $\mathcal{V}$ becomes inactive. This happens when a non-zero coefficient $\beta_j \neq 0$ becomes 0.

4. An inactive variable joins the active variable set $\mathcal{V}$. To identify this event, we define the *generalized correlation* for variable j as:

$$c_j = \lambda_2 \beta_j - \sum_{i=1}^{n} \alpha_i y_i x_{ij} \tag{2.24}$$

   From (2.16), we can see that all active variables in $\mathcal{V}$ have the same absolute generalized correlation value, which is η. Therefore, an inactive variable will join the active variable set when its absolute generalized correlation reaches η.

In the next two sections, we describe the algorithm that computes the whole solution path $\beta_{\lambda_2}(\lambda_1)$ in detail. The basic idea is to start at $s = 0$ (or equivalently

$\lambda_1 = \infty$), find the right derivatives of β_0 and β_j with respect to s, increase s until an event happens, then adjust the linear system (2.20) – (2.23), and find out the new right derivatives. The algorithm stops when no further event will happen.

2.3.2 Initial Solution

In this section, we compute the initial right derivatives of β_0 and β_j. Let n_+ be the number of training data points in the "+" class, and n_- be the number of training data points in the "−" class. We distinguish between two cases: the balanced case ($n_+ = n_-$) and the unbalanced case ($n_+ \neq n_-$).

The Balanced Case

When $s = 0$, therefore $\boldsymbol{\beta} = 0$, the objective function (4.7) becomes

$$\min_{\beta_0} \sum_{i=1}^{n} (1 - y_i \beta_0)_+$$

Since $n_+ = n_-$, there is no unique solution for β_0, and any value of $\beta_0 \in [-1, 1]$ will give the same minimum.

Although β_0 is not unique, all the α_i are equal to 1. Using (2.24), the generalized correlation of variable x_j is $-\sum_{i=1}^{n} y_i x_{ij}$. When s increases by an infinitesimal amount, some variable(s) will join the active variable set $\mathcal{V}$, and $\mathcal{V}$ can be identified as

$$\mathcal{V} = \left\{ j : |\sum_{i=1}^{n} y_i x_{ij}| = \max_j |\sum_{i=1}^{n} y_i x_{ij}| \right\}$$

The signs of the corresponding coefficients are simply $\text{sign}(\sum_{i=1}^{n} y_i x_{ij})$. Then using (2.20) and (2.23), one can solve for the right derivatives of $\beta_j, j \in \mathcal{V}$ and η with respect to s.

When s increases, by continuity and the balance between n_+ and n_-, all α_i will

stay at 1 before an event happens. Therefore

$$\begin{aligned} 1-(\beta_0+\sum_{j\in\mathcal{V}}\beta_j x_{ij}) &\geq 0,\ i\in I_+ \\ 1+(\beta_0+\sum_{j\in\mathcal{V}}\beta_j x_{ij}) &\geq 0,\ i\in I_- \end{aligned}$$

where I_+ and I_- contain indices of the "+" class points and the "−" class points, respectively. The above inequalities imply that the solution for β_0 is not unique, and β_0 can be any value in the interval

$$\left[\max_{i\in I_-}(-1-\sum_{j\in\mathcal{V}}\beta_j x_{ij}),\ \min_{i\in I_+}(1-\sum_{j\in\mathcal{V}}\beta_j x_{ij})\right]$$

When s increases, β_j changes, and the length of this interval will shrink toward zero, which corresponds to two data points (from different classes) hitting the elbow simultaneously.

The Unbalanced Case

Without loss of generality, we assume $n_+ > n_-$. When $s = 0$, the solution is $\beta_0 = 1$ and $\boldsymbol{\beta} = 0$, which implies all the I_- points are in $\mathcal{L}$ and all the I_+ points are in $\mathcal{E}$. When s increases by an infinitesimal amount, some variable(s) will join the active variable set $\mathcal{V}$. By continuity, all the I_- points will still stay in $\mathcal{L}$, but the I_+ points will split: some will join $\mathcal{L}$, some will join $\mathcal{R}$, and the rest will stay at $\mathcal{E}$. From (2.20) – (2.23), we can see in order to determine the right derivatives of β_0, β_j, α_i and η with respect to s, it is crucial to identify the active variable set $\mathcal{V}$ and the elbow set $\mathcal{E}$. For the initial solution, it turns out that $\mathcal{V}$ and $\mathcal{E}$ can be identified via

the following linear programming problem:

$$\begin{aligned} \min_{\beta_0,\boldsymbol{\beta}} \quad & \sum_{i\in I_+}\epsilon_i + \sum_{i\in I_-}(1-y_i f_i) \\ \text{subject to} \quad & 1-y_i f_i \le \epsilon_i,\ i\in I_+ \\ & \epsilon_i \ge 0,\ i\in I_+ \\ & \|\boldsymbol{\beta}\|_1 = |\beta_1|+\cdots+|\beta_p| \le s \end{aligned}$$

Notice the loss for "−" class points has been changed from $(1-yf)_+$ to $(1-yf)$. This allows us to use *any* value of s, and always get the same $\mathcal{V}$ and $\mathcal{E}$ via the linear programming. Once the $\mathcal{V}$ and the $\mathcal{E}$ are identified, the initial right derivatives of β_0, β_j, α_i and η with respect to s can be solved from (2.20) – (2.23).

2.3.3 Main Program

After the initial right derivatives of β_0, β_j, α_i and η are identified, the main algorithm proceeds as the following:

1. Compute

   - the derivative of the residual for every non-elbow point

     $$\frac{\Delta r_i}{\Delta s} = -y_i\left(\frac{\Delta\beta_0}{\Delta s}+\sum_{j\in\mathcal{V}}x_{ij}\frac{\Delta\beta_j}{\Delta s}\right),\ i\notin\mathcal{E}$$

     where r_i is the current residual $(1-y_i f_i)$ for point i.

   - the derivative of the generalized correlation for every inactive variable

     $$\frac{\Delta c_j}{\Delta s} = -\sum_{i\in\mathcal{E}}\frac{\Delta\alpha_i}{\Delta s}y_i x_{ij},\ j\notin\mathcal{V}$$

     where c_j is the current generalized correlation value for variable x_j, given by (2.24).

2. Compute how much increase of s is needed to get to each type of event:

- an elbow point leaves the elbow

$$\delta_s^1 = \min_{i \in \mathcal{E}} \max \left(\frac{0 - \alpha_i}{\Delta\alpha_i / \Delta s}, \frac{1 - \alpha_i}{\Delta\alpha_i / \Delta s} \right)$$

- a non-elbow point hits the elbow

$$\delta_s^2 = \min_{i \in \mathcal{E}_+^c} \left(\frac{0 - r_i}{\Delta r_i / \Delta s} \right)$$

  where $\mathcal{E}_+^c = \{i : \frac{0 - r_i}{\Delta r_i / \Delta s} > 0, i \notin \mathcal{E}\}$.

- an active variable becomes inactive

$$\delta_s^3 = \min_{j \in \mathcal{V}_+} \left(\frac{0 - \beta_j}{\Delta\beta_j / \Delta s} \right)$$

  where $\mathcal{V}_+ = \{j : \frac{0 - \beta_j}{\Delta\beta_j / \Delta s} > 0, j \in \mathcal{V}\}$.

- an inactive variable joins the active set

$$\delta_s^4 = \min_{j \notin \mathcal{V}} \max \left(\frac{-\eta - c_j}{\Delta c_j / \Delta s + \Delta\eta / \Delta s}, \frac{\eta - c_j}{\Delta c_j / \Delta s - \Delta\eta / \Delta s} \right)$$

- the generalized correlation of active variables reduces to zero

$$\delta_s^5 = \frac{0 - \eta}{\Delta\eta / \Delta s}$$

The first four items correspond to the four types of events introduced in section 2.3.1. The last item corresponds to one termination criterion of the algorithm.

3. Find which event happens first

$$\delta_s = \min(\delta_s^1, \delta_s^2, \delta_s^3, \delta_s^4, \delta_s^5)$$

and update

$$\begin{aligned}
\alpha_i &\longleftarrow \alpha_i + \delta_s \frac{\Delta\alpha_i}{\Delta s}, \ i \in \mathcal{E} \\
\beta_0 &\longleftarrow \beta_0 + \delta_s \frac{\Delta\beta_0}{\Delta s} \\
\beta_j &\longleftarrow \beta_j + \delta_s \frac{\Delta\beta_j}{\Delta s}, \ j \in \mathcal{V} \\
\eta &\longleftarrow \eta + \delta_s \frac{\Delta\eta}{\Delta s}
\end{aligned}$$

4. Update $\mathcal{L}, \mathcal{R}, \mathcal{E}$ and $\mathcal{V}$.

5. If any one of the following termination criterion is satisfied, stop the algorithm

   - The generalized correlation reduces to zero.
   - Two classes have been perfectly separated.
   - A pre-specified maximum iteration number is reached.

   Otherwise, use (2.20) – (2.23) to compute the new derivatives, and go back to step 1.

2.3.4 Solution Path for Fixed λ_1

We have described an algorithm for solving the solution path of the DrSVM when λ_2 is fixed. In this section, we briefly describe a similar algorithm that solves the solution path of the DrSVM when λ_1 is fixed. We first prove Theorem II.3.

Proof of Theorem II.3

When λ_1 is fixed and λ_2 changes, the solution has to satisfy (2.16) – (2.18) in section 2.3.1, which are derived from the Lagrange and KKT conditions. Let $D = 1/\lambda_2$ and $\alpha_i^* = D\alpha_i$, (2.16) – (2.18) become:

$$\begin{aligned} \beta_j - \sum_{i=1}^{n} \alpha_i^* y_i x_{ij} &= -\lambda_1 D \text{sign}(\beta_j)\ j \in \mathcal{V} \\ \sum_{i=1}^{n} \alpha_i^* y_i &= 0 \\ y_i \left(\beta_0 + \sum_{j \in \mathcal{V}} x_{ij} \beta_j \right) &= 1\ i \in \mathcal{E} \end{aligned}$$

This system consists of $|\mathcal{E}| + |\mathcal{V}| + 1$ equations and $|\mathcal{E}| + |\mathcal{V}| + 1$ unknowns: β_0, β_j $(j \in \mathcal{V})$, α_i^* $(i \in \mathcal{E})$. Therefore, using the same argument as in section 2.3.1, one can show the solution $(\beta_0, \boldsymbol{\beta})$ is piecewise linear in D (or $1/\lambda_2$). □

Similarly, one can show that α_i are also piecewise linear, but piecewise linear in λ_2, rather than $1/\lambda_2$. These facts facilitate us in designing an efficient algorithm to compute the solution path of the DrSVM when λ_1 is fixed. The idea is similar to the algorithm in section 2.3.3, with minor modifications in computing how much increase of D (or decrease of λ_2) is needed to get to the next event:

- an elbow point leaves the elbow

$$\delta^1_{\lambda_2} = \max_{i\in\mathcal{E}} \min\left(\frac{0-\alpha_i}{\Delta\alpha_i/\Delta\lambda_2}, \frac{1-\alpha_i}{\Delta\alpha_i/\Delta\lambda_2}\right)$$

- a non-elbow point hits the elbow

$$\delta^2_D = \min_{i\in\mathcal{E}^c_+}\left(\frac{0-r_i}{\Delta r_i/\Delta D}\right)$$

  where $\mathcal{E}^c_+ = \{i : \frac{0-r_i}{\Delta r_i/\Delta s} > 0, i \notin \mathcal{E}\}$.

- an active variable becomes inactive

$$\delta^3_D = \min_{j\in\mathcal{V}_+}\left(\frac{0-\beta_j}{\Delta\beta_j/\Delta D}\right)$$

  where $\mathcal{V}_+ = \{j : \frac{0-\beta_j}{\Delta\beta_j/\Delta s} > 0, j \in \mathcal{V}\}$.

- an inactive variable joins the active set

$$\delta^4_{\lambda_2} = \max_{j\notin\mathcal{V}} \min\left(\frac{-\lambda_1-c_j}{\Delta c_j/\Delta\lambda_2}, \frac{\lambda_1-c_j}{\Delta c_j/\Delta\lambda_2}\right)$$

- the tuning parameter λ_2 reduces to zero

$$\delta^5_{\lambda_2} = -\lambda_2$$

Again, the first four items correspond to the four types of events in section 2.3.1. The last item corresponds to one termination criterion of the algorithm. Which event happens first can then be determined by

$$\delta_{\lambda_2} = \max\left(\frac{-\lambda_2^2\delta^1_D}{1+\lambda_2\delta^1_D}, \delta^2_{\lambda_2}, \frac{-\lambda_2^2\delta^3_D}{1+\lambda_2\delta^3_D}, \delta^4_{\lambda_2}, \delta^5_{\lambda_2}\right)$$

The rest of the algorithm is the same as in section 2.3.3.

2.3.5 Computational Complexity

The major computational cost is associated with solving the linear system (2.20) – (2.23) at each step, which involves $|\mathcal{E}| + 2$ equations and unknowns (after plugging (2.20) in (2.22) and (2.23)). Solving such a system involves $O(|\mathcal{E}|^3)$ computations. However, for any two consecutive steps, the two linear systems usually differ by only one row or one column (corresponding to one of the four types of events); therefore, the computational cost can be reduced to $O(|\mathcal{E}|^2)$ via the inverse updating/downdating. The computation of $\Delta\beta_j/\Delta s$ in (2.20) requires $O(|\mathcal{E}| \cdot |\mathcal{V}|)$ computations after getting $\Delta\alpha_i/\Delta s$. Notice due to the nature of (2.20) – (2.23), $|\mathcal{E}|$ is always less than or equal to $\min(n, p)$, and since $|\mathcal{V}| \leq p$, the computational cost at each step can be estimated (bounded) as $O(\min^2(n, p) + p \min(n, p))$.

It is difficult to predict the number of steps on the solution path for any arbitrary data. Our experience so far suggests that the total number of steps is $O(\min(n, p))$. This can be heuristically understood in the following way: if $n < p$, the training data are perfectly separable by a linear model, then it takes $O(n)$ steps for every data point to pass through the elbow to achieve the zero loss; if $n > p$, then it takes $O(p)$ steps to include every variable into the fitted model. Overall, this suggests the total computational cost is $O(p \min^2(n, p) + \min^3(n, p))$.

2.4 Numerical Results

In this section, we use both simulation data and real world data to illustrate the DrSVM. In particular, we want to show that with high dimensional data, the DrSVM is able to remove irrelevant variables, and identify relevant (sometimes correlated) variables.

2.4.1 Simulation

We first consider the scenario when all input variables are independent. The "+" class has a normal distribution with mean and covariance

$$\begin{aligned} \boldsymbol{\mu}_+ &= (\underbrace{0.5, \ldots, 0.5}_{5}, \underbrace{0, \ldots, 0}_{p-5})^{\mathsf{T}} \\ \boldsymbol{\Sigma} &= \boldsymbol{I}_{p\times p} \end{aligned}$$

The "−" class has a similar distribution except that

$$\boldsymbol{\mu}_- = (\underbrace{-0.5, \ldots, -0.5}_{5}, \underbrace{0, \ldots, 0}_{p-5})^{\mathsf{T}}$$

So the Bayes optimal classification rule only depends on $x_1, \ldots, x_5$, and the Bayes error is 0.132, independent of the dimension p.

We consider both the $n > p$ case and the $n \ll p$ case. In the $n > p$ case, we generate $100 = 50 + 50$ training data, each input $\boldsymbol{x}_i$ is a $p = 10$ dimensional vector; in the $n \ll p$ case, we generate $50 = 25 + 25$ training data, each input $\boldsymbol{x}_i$ is a $p = 300$ dimensional vector. We compare the L_1-norm SVM, the L_2-norm SVM, and the DrSVM. For fairness, we use $20,000$ validation data to select the tuning parameters for each method, then apply the selected models to a separate $20,000$ testing data set.[1] Each experiment is repeated for 30 times. The means of the prediction errors and the corresponding standard errors (in parentheses) are summarized in Table 3.4.1. As we can see, the prediction errors of the L_1-norm SVM and the DrSVM are similar: both are close to the optimal Bayes error when $n > p$, and degrade a little bit when $n \ll p$. This is not the case for the L_2-norm SVM: in the $n > p$ case, the prediction error is only slightly worse than that of the L_1-norm SVM and the DrSVM, but it

[1] The main purpose of the simulation is to demonstrate that the DrSVM can be useful when the variables are highly correlated, especially when $n \ll p$. We are also interested to see how much improvement (in terms of prediction accuracy) the DrSVM can gain over the L_2-norm SVM and the L_1-norm SVM. Using cross-validation can add an extra level of complexity, and it will be unclear whether the difference in the result is due to the cross-validation or due to the elastic-net penalty. Therefore we chose to use a large validation dataset, rather than cross-validation, to isolate the effect of the elastic-net penalty.

degrades dramatically in the $n \ll p$ case. This is due to the fact that the L_2-norm SVM uses all input variables, and its prediction accuracy is polluted by the noise variables.

	n	p	p_0	Test Error
L_2 SVM	100	10	5	0.145 (0.007)
L_1 SVM				0.142 (0.008)
DrSVM				0.139 (0.005)
L_2 SVM	50	300	5	0.323 (0.018)
L_1 SVM				0.199 (0.031)
DrSVM				0.178 (0.021)

Table 2.1: Comparison of the prediction performance when all input variables are independent. p_0 is the number of relevant variables.

Besides the prediction error, we also compare the selected variables of the L_1-norm SVM and the DrSVM (The L_2-norm SVM keeps all input variables). In particular, we consider

- q_{signal} = number of selected relevant variables
- q_{noise} = number of selected noise variables

The results are in Table 3.4.1. Again, we see that the L_1-norm SVM and the DrSVM perform similarly; both are able to identify the relevant variables (the L_1-norm SVM missed 1 on average) and remove most of the irrelevant variables.

	n	p	p_0	q_{signal}	q_{noise}
L_1 SVM	100	10	5	5.00 (0.00)	2.43 (1.52)
DrSVM				5.00 (0.00)	1.80 (1.30)
L_1 SVM	50	300	5	3.87 (0.82)	4.33 (4.86)
DrSVM				4.53 (0.57)	6.37 (4.35)

Table 2.2: Comparison of variable selection when all input variables are independent. p_0 is the number of relevant variables. q_{signal} is the number of selected relevant variables. q_{noise} is the number of selected noise variables.

Now we consider the scenario when the relevant variables are correlated. Similar as the independent scenario, the "+" class has a normal distribution, with mean and

covariance

$$\begin{aligned} \boldsymbol{\mu}_+ &= (\underbrace{1,\ldots,1}_{5},\underbrace{0,\ldots,0}_{p-5})^{\mathsf{T}} \\ \boldsymbol{\Sigma} &= \begin{pmatrix} \boldsymbol{\Sigma}^*_{5\times 5} & \mathbf{0}_{5\times(p-5)} \\ \mathbf{0}_{(p-5)\times 5} & \boldsymbol{I}_{(p-5)\times(p-5)} \end{pmatrix} \end{aligned}$$

where the diagonal elements of $\boldsymbol{\Sigma}^*$ are 1 and the off-diagonal elements are all equal to $\rho = 0.8$. The "$-$" class has a similar distribution except that

$$\boldsymbol{\mu}_- = (\underbrace{-1,\ldots,-1}_{5},\underbrace{0,\ldots,0}_{p-5})^{\mathsf{T}}$$

So the Bayes optimal classification rule depends on $x_1,\ldots,x_5$, which are highly correlated. The Bayes error is 0.138, independent of the dimension p.

Again, we consider both the $n > p$ case and the $n \ll p$ case. In the $n > p$ case, $n = 50+50$ and $p = 10$. In the $n \ll p$ case, $n = 25+25$ and $p = 300$. Each experiment is repeated for 30 times. The result for the prediction errors are shown in Table 2.4.1. Now when changing from the $n > p$ case to the $n \ll p$ case, the performance of the L_1-norm SVM, as well as the L_2-norm SVM, degrades, but the DrSVM performs about the same. Table 2.4.1 compares the variables selected by the L_1-norm SVM and the DrSVM, which sheds some light on what happened. Both the L_1-norm SVM and the DrSVM are able to identify relevant variables. However, when the relevant variables are highly correlated, the L_1-norm SVM tends to keep only a small subset of the relevant variables, and overlook the others, while the DrSVM tends to identify all of them, due to the grouping effect. Both methods seem to work well in removing irrelevant variables.

In the last, we consider a scenario where the relevant variables have different contributions to the classification, and the pairwise correlations are not all equal.

	n	p	p_0	Test Error
L_2 SVM				0.142 (0.003)
L_1 SVM	100	10	5	0.144 (0.003)
DrSVM				0.140 (0.001)
L_2 SVM				0.186 (0.012)
L_1 SVM	50	300	5	0.151 (0.007)
DrSVM				0.139 (0.004)

Table 2.3: Comparison of the prediction performance when the relevant variables are highly correlated. p_0 is the number of relevant variables.

	n	p	p_0	q_{signal}	q_{noise}
L_1 SVM	100	10	5	3.73 (0.69)	0.30 (0.53)
DrSVM				5.00 (0.00)	0.10 (0.31)
L_1 SVM	50	300	5	2.17 (0.83)	0.30 (0.60)
DrSVM				4.90 (0.40)	0.97 (2.03)

Table 2.4: Comparison of variable selection when the relevant variables are highly correlated. p_0 is the number of relevant variables. q_{signal} is the number of selected relevant variables. q_{noise} is the number of selected noise variables.

The basic setup is similar to the above two scenarios, except that

$$\begin{aligned}
\boldsymbol{\mu}_+ &= (\underbrace{1,\ldots,1}_{5},\underbrace{0,\ldots,0}_{p-5})^{\mathsf{T}} \\
\boldsymbol{\mu}_- &= (\underbrace{-1,\ldots,-1}_{5},\underbrace{0,\ldots,0}_{p-5})^{\mathsf{T}} \\
\boldsymbol{\Sigma}^* &= \begin{pmatrix}
1 & 0.8 & 0.8^2 & 0.8^3 & 0.8^4 \\
0.8 & 1 & 0.8 & 0.8^2 & 0.8^3 \\
0.8^2 & 0.8 & 1 & 0.8 & 0.8^2 \\
0.8^3 & 0.8^2 & 0.8 & 1 & 0.8 \\
0.8^4 & 0.8^3 & 0.8^2 & 0.8 & 1
\end{pmatrix}
\end{aligned}$$

The Bayes optimal classification boundary is given by

$$1.11x_1 + 0.22x_2 + 0.22x_3 + 0.22x_4 + 1.11x_5 = 0$$

and the Bayes error is 0.115. Notice that the true coefficients β_2, β_3 and β_4 are small compared with β_1 and β_5. To test our algorithm for the unbalanced case, we let $n = 60 + 40$ when $p = 10$, and $n = 30 + 20$ when $p = 300$. Each experiment is

repeated for 30 times. The results are summarized in Table 2.4.1 and 2.4.1. As we can see, the DrSVM still dominates the L_1-norm SVM in terms of identifying relevant variables.

	n	p	p_0	Test Error
L_2 SVM				0.128 (0.008)
L_1 SVM	100	10	5	0.117 (0.004)
DrSVM				0.115 (0.003)
L_2 SVM				0.212 (0.022)
L_1 SVM	50	300	5	0.125 (0.010)
DrSVM				0.120 (0.006)

Table 2.5: Comparison of the prediction performance when the relevant variables have different class means and the pairwise correlations are not all equal. p_0 is the number of relevant variables.

	n	p	p_0	q_{signal}	q_{noise}
L_1 SVM	100	10	5	3.70 (0.84)	1.48 (0.67)
DrSVM				4.53 (0.57)	0.53 (1.04)
L_1 SVM	50	300	5	3.03 (0.72)	1.23 (1.87)
DrSVM				4.23 (0.94)	2.93 (4.72)

Table 2.6: Comparison of variable selection when the relevant variables have different class means and the pairwise correlations are not all equal. p_0 is the number of relevant variables. q_{signal} is the number of selected relevant variables. q_{noise} is the number of selected noise variables.

2.4.2 Microarray Analysis

In this section, we apply the DrSVM to classification of gene microarrays. Classification of patient samples is an important aspect of cancer diagnosis and treatment. The L_2-norm SVM has been successfully applied to microarray cancer diagnosis problems [31, 49]. However, one weakness of the L_2-norm SVM is that it only predicts a cancer class label but does not automatically select relevant genes for the classification. Often a primary goal in microarray cancer diagnosis is to identify the genes responsible for the classification, rather than class prediction. The L_1-norm SVM has an inherent gene (variable) selection property due to the L_1-norm penalty, but the maximum number of genes that the L_1-norm SVM can select is upper bounded by n,

which is typically much smaller than p in microarray problems. Another drawback of the L_1-norm SVM, as seen in the simulation study, is that it usually fails to identify group of genes that share the same biological pathway, which have correlated expression levels. The DrSVM naturally overcomes these difficulties, and achieves the goals of classification of patients and (group) selection of genes simultaneously.

We use a leukemia dataset [29] to illustrate the point. This dataset consists of 38 training data and 34 test data of two types of acute leukemia, *acute myeloid leukemia* (AML) and *acute lymphoblastic leukemia* (ALL). Each datum is a vector of $p = 2,308$ genes. The tuning parameters are chosen according to 10-fold cross-validation, then the final model is fitted on all the training data and evaluated on the test data. The results are summarized in Table 2.7. As we can see, the DrSVM seems to have the best prediction performance. However, notice this is a very small (and "easy") dataset, so the difference may not be significant. It is also worth noting that the 22 genes selected by the L_1-norm SVM is a subset of the 78 genes selected by the DrSVM. Figure 2.4 shows the heatmap of the selected 78 genes. We have ordered the genes by hierarchical clustering, and similarly for all $38 + 34$ samples (based on the selected genes). Clear separation of the two classes is evident. Roughly speaking, the top set of genes overexpress for ALL and underexpress for AML; vice versa for the bottom set of genes.

	CV Error	Test Error	# of Genes
Golub	3/38	4/34	50
L_2-norm SVM	0/38	1/34	2,308
L_1-norm SVM	3/38	1/34	22
DrSVM	0/38	0/34	78

Table 2.7: Results on the Leukemia Dataset

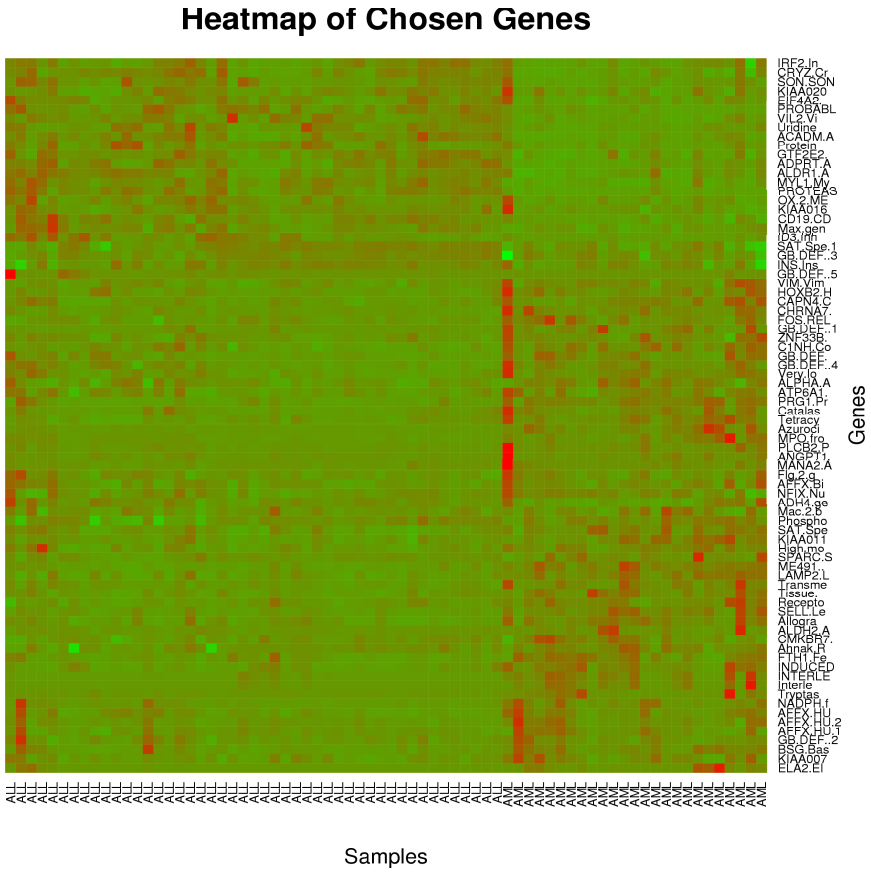

Figure 2.4: Heatmap of the selected 78 genes. We have ordered the genes by hierarchical clustering, and similarly for all $38 + 34$ samples.

2.4.3 Handwritten Digit Recognition

In this section, we consider a classical handwritten digit recognition example. Recognition of generic object categories is one of the major challenges in computer vision. Most current methods can roughly be divided into brightness-based and feature-based. Brightness-based methods use the raw pixels as the input vector (usually after certain normalization so that all images have approximately the same size and orientation). Feature-based methods first extract information from the image, such as shape, texture and color, then use these information as the input vector.

We will work with feature-based methods for they are closer to the biological vision system than brightness-based methods. In particular, we will concentrate on shape features, for in biological vision, shape cue is arguably the strongest among all types of cues (shape, texture, color, etc.) for visual object recognition [54].

There are two distinct challenges here:

- The size of the training dataset is small (a person does not need to see many images to generalize the notion of the new object to a novel image). On the other hand, due to the richness of information from a single image, shape features alone could be on the order of 1000 real numbers, such as edge locations and directions, corner locations, arrangement of features, etc. So the feature set is rich. This falls into the $p > n$ problem.

- Besides predicting the correct object category for a given image, another challenge is to identify the relevant features which contribute most to the classification. For example, when looking at a slate of handwritten digits (Figure 2.5), despite the variation in writing style one can still see the distinctive parts of the shape which best tell apart one class against the other class.

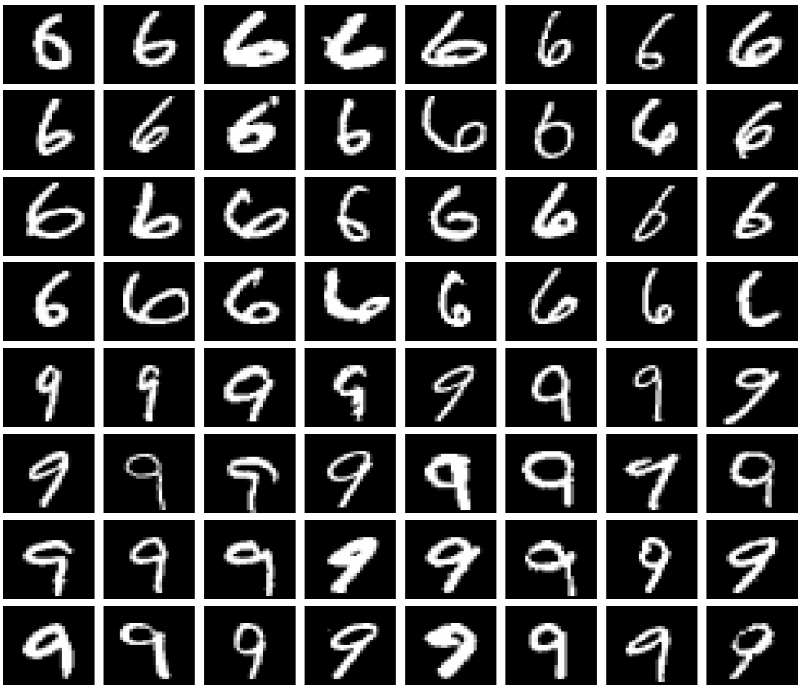

Figure 2.5: Some examples of handwritten digit 6 and digit 9. Despite the variation in writing style, one can still see the distinctive parts of the shape which best tell apart one class against the other class.

The proposed DrSVM method naturally applies here. We compare the L_2-norm SVM, the L_1-norm SVM and the DrSVM on a subset of the MINIST database [43]. The standard L_2-norm SVM has been applied to this database before and shown good performance [43], on par with human performance. However, once again, one weakness of the L_2-norm SVM is that it only predicts an object class but does not automatically select relevant features for the classification. The L_1-norm SVM has an inherent feature selection property, but tends to overlook correlated features. The DrSVM overcomes these two difficulties. We use digit 6 vs digit 9 as an example. We randomly sampled $250 + 250$ training data and $750 + 750$ test data from the MINIST database. Each digit (sample) consists of 804 shape features based on the so called *shape context distance* [3, 77]. Tuning parameters were selected using 10-fold cross-validation. Numerical results are summarized in Table 2.4.3. The DrSVM performs a little better than the standard L_2-norm SVM in terms of mis-classification error,

and it automatically selected 128 shape features, while the standard L_2-norm SVM used all 804 features. The corresponding figures are shown in Figure 2.6.

	CV Error	Test Error	# of Features
L_2 SVM	6/500	22/1500	804
L_1 SVM	6/500	17/1500	49
DrSVM	1/500	15/1500	128

Table 2.8: Result for 6 vs 9 from the MINIST database. Training size is $250 + 250$, and test size is $750 + 750$. Each digit consists of 804 shape features.

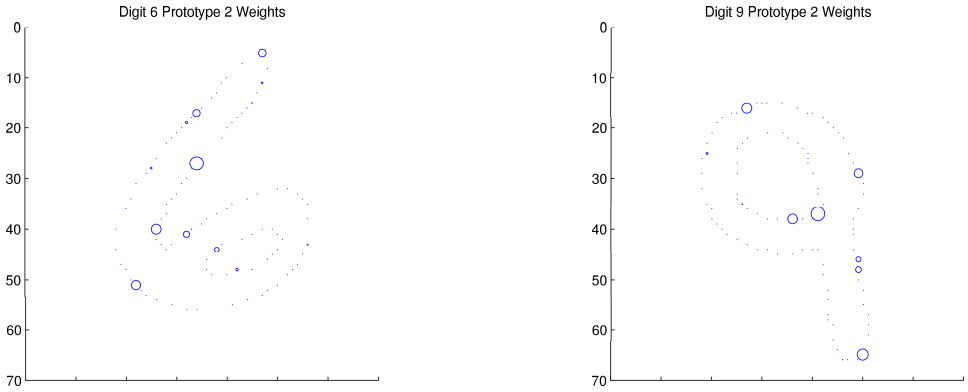

Figure 2.6: Selected features: the center of each circle indicates the position of the selected feature, and the size of each circle is proportional to the corresponding feature's *importance*.

2.5 Discussion

We have applied the elastic-net penalty to the hinge loss, and proposed the DrSVM method for classification problems. This method is especially useful with high dimensional data, with respect to effectively removing irrelevant variables and identifying relevant variables. Unlike previously developed support vector machines, e.g. the L_1-norm SVM, the DrSVM is able to select groups of variables that are correlated, and the number of selected variables is no longer bounded by the size of the training data, thus being able to deal with the $p \gg n$ problem. We also proposed efficient algorithms that can compute the whole solution paths of the DrSVM, which facilitate selection of the tuning parameters.

There are several interesting directions in which the DrSVM can be extended:

- How to efficiently utilize the solution paths? Computing one solution path is very efficient, but cross-validation can be computaionally expensive. The ideal situation is to have a model selection criterion that can be calculated along the solution path; then once the solution path is computed, the best tuning parameter can be identified. The GACV [73] is a computable proxy for the generalized Kullback-Liebler distance, and seems to be a good model selection criterion for the support vector machine. Since the DrSVM have two tuning parameters, λ_1 and λ_2, combining the GACV criterion and a two-step procedure seems to be promising: In the first step, one can fix a (relatively large) value for λ_2, compute the solution path $\beta_{\lambda_2}(\lambda_1)$, and use the GACV criterion to select a value for λ_1; this step corresponds to set up an appropriate threshold to remove noise variables. In the second step, one fixes λ_1 at the value selected in step one, compute the solution path $\beta_{\lambda_1}(\lambda_2)$, and again use the GACV criterion to identify a value for λ_2; this corresponds to get the correct grouping effect for correlated variables. The advantage of this strategy is that it avoids a two-dimensional grid search. Our preliminary results are encouraging, and we will explore this further.

- The algorithm proposed in section 4.3 is efficient. However, when both n and p are large, the initial solution (in the balanced case) may require substantial computational efforts. This is due the fact that the hinge loss function is not differentiable at the point $yf = 1$, and a linear programming is called upon to solve the initial solution. So the question is how one can modify the DrSVM to improve the computational efficiency? We consider to replace the hinge loss

with the *Huberized hinge loss* [58]. The Huberized hinge loss is defined as

$$\phi(yf) = \begin{cases} (1-\delta)/2 + (\delta - yf) & \text{if } yf \leq \delta \\ (1-yf)^2/(2(1-\delta)) & \text{if } \delta < yf \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

where $\delta < 1$. Figure 2.7 compares the Huberized hinge loss and the hinge loss. As we can see, the Huberized hinge loss is differentiable everywhere, and it is straightforward to get the initial solution. The Huberized hinge loss also has a similar shape as the hinge loss; therefore, one can expect the prediction performance of the Huberized hinge loss would be similar to the hinge loss.

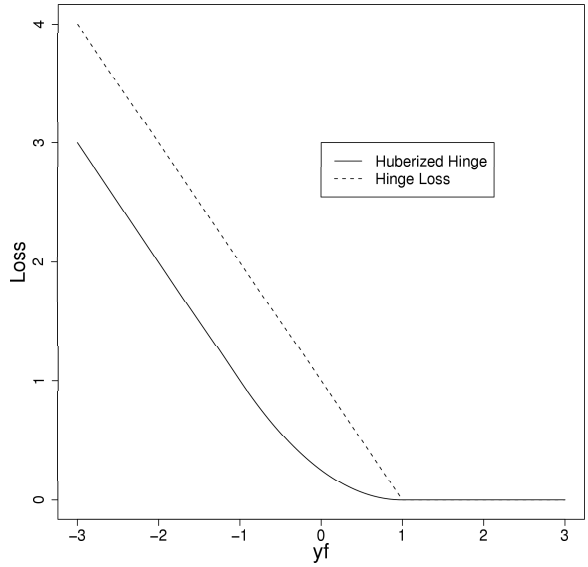

Figure 2.7: The hinge loss and the Huberized hinge loss (with $\delta = -1$). The Huberized hinge loss is differentiable everywhere, and has a similar shape as the hinge loss.

CHAPTER III

Hybrid Huberized Support Vector Machines for Microarray Classification and Gene Selection

3.1 Background

The DNA microarray technology is a powerful tool for biological and medical research. It can detect thousands of gene expression levels simultaneously, providing a wealth of information. On the other hand, however, microarray datasets usually contain only a small number of samples. These characteristics pose great challenges for sample classification and gene selection. The support vector machine (SVM) is one of the most effective methods for microarray classification [50, 57, 31]; however, a major limitation of the SVM is that it can not perform automatic gene selection. Since in microarray analysis, researchers are often interested in identifying informative genes, it is desirable to have a tool that can achieve both classification and gene selection simultaneously.

Guyon et al. [31] proposed the recursive feature elimination (RFE) method for the SVM. The method, called the SVM-RFE, recursively eliminates irrelevant genes. At each step, the SVM-RFE trains for an SVM classifier, ranks the genes according to some score function and eliminates one or more genes with the lowest ranking scores. This process is repeated until classification accuracy starts to degrade. The SVM-RFE method is computationally intensive, especially for microarray datasets,

which usually has a large number of genes. Bradley and Mangasarian [6] proposed the L_1-norm SVM, which can automatically select genes via the L_1-norm regularization. The L_1-norm SVM, however, has two limitations: (1) The number of selected genes is upper bounded by the sample size. Therefore, when the number of relevant genes exceeds the sample size, the L_1-norm SVM can only discover a portion of them. (2) For the highly correlated and relevant genes, the L_1-norm SVM tends to pick only one or a few of them.

In this paper, we propose a hybrid huberized support vector machine (HHSVM) for microarray classification. The HHSVM has the form of "loss" + "penalty," where it uses the huberized hinge function to measure the loss and the elastic-net penalty [81] to control the complexity of the model. The HHSVM has several major benefits:

- Similar to the L_1-norm SVM, it automatically selects genes;
- The number of selected genes is no longer upper bounded by the sample size;
- When genes are highly correlated, they tend to be selected or removed together, i.e., the *grouping effect*.

Furthermore, inspired by the LAR/LASSO method [17] and the general piecewise linear solution path strategy [59], we develop an efficient algorithm, which solves the entire solution path for every possible value of the regularization parameter.

The rest of the paper is organized as follows. In Section 2, we describe the HHSVM model. In Section 3, we develop the solution path algorithm. In Section 4, we apply the HHSVM method to simulation and real microarray datasets. We conclude the paper with Section 5.

3.2 The Hybrid Huberized Support Vector Machine

Zou and Hastie [81] argued that the L_1-norm penalty has two major limitations:

1. The number of variables selected by the L_1-norm penalty is upper bounded by the sample size n. In microarray analysis, we nearly always have the case $p \gg n$, but the L_1-norm SVM can identify at most n relevant genes;

2. For highly correlated and relevant variables, the L_1-norm penalty tends to select only one or a few of them. In microarray analysis, genes sharing the same biological pathway tend to have highly correlated expression levels. It is often desirable to identify all, rather than a few, of them if they are related to the underlying biological process.

To overcome these limitations, Zou and Hastie [81] proposed the elastic-net penalty:

$$\lambda_1 \|\boldsymbol{\beta}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2^2,$$

which is a hybrid of the L_1-norm and the L_2-norm penalties. The elastic-net penalty retains the variable selection feature of the L_1-norm penalty, and the number of selected variables is no longer bounded by n. Furthermore, the elastic-net penalty tends to provide similar estimated coefficients for highly correlated variables, i.e., the grouping effect; hence, highly correlated variables tend to be selected or removed together.

In this paper, we apply the elastic-net penalty to the SVM and propose the hybrid huberized support vector machine (HHSVM):

$$\min_{\beta_0, \boldsymbol{\beta}} \sum_{i=1}^{n} \phi\left(y_i(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta})\right) + \lambda_1 \|\boldsymbol{\beta}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2^2, \tag{3.1}$$

where $\lambda_1, \lambda_2 \geq 0$ are regularization parameters. Increasing λ_1 tends to eliminate more irrelevant variables; and increasing λ_2 makes the "grouping effect" more prominent, which we will illustrate by a theorem at the end of this section.

Notice that instead of using the standard hinge loss function of the SVM, we use the huberized hinge loss function [59] to measure "badness-of-fit":

$$\phi(yf) = \begin{cases} 0, & \text{for } yf > 1, \\ (1-yf)^2/2\delta, & \text{for } 1-\delta < yf \leq 1, \\ 1-yf-\delta/2, & \text{for } yf \leq 1-\delta, \end{cases}$$

where $\delta \geq 0$ is a pre-specified constant.

Figure 1 compares the standard hinge loss function and the huberized hinge loss function. Notice that they have similar shapes: when yf decreases (misclassification), they both increase linearly; when yf is bigger than 1, they are both equal to zero. Therefore, we expect the classification performance of these two functions should be similar to each other. Also notice that, unlike the hinge loss, the huberized hinge loss function is differentiable everywhere. As we will see in the next section, this differentiability can significantly reduce the computational cost for the HHSVM algorithm.

Before delving into details for the HHSVM algorithm, we illustrate the "grouping effect" with the following theorem:

Theorem III.1. *Let $\widehat{\beta}_0$ and $\widehat{\boldsymbol{\beta}}$ denote the solution for problem (3.1). For any pair (j, j'), we have*

$$|\widehat{\beta}_j - \widehat{\beta}_{j'}| \leq \frac{1}{\lambda_2}\|\boldsymbol{x}_j - \boldsymbol{x}_{j'}\|_1 = \frac{1}{\lambda_2}\sum_{i=1}^{n}|x_{ij} - x_{ij'}|. \tag{3.2}$$

If the input vector $\boldsymbol{x}_j$ and $\boldsymbol{x}_{j'}$ are centered and normalized, then

$$|\widehat{\beta}_j - \widehat{\beta}_{j'}| \leq \frac{\sqrt{n}}{\lambda_2}\sqrt{2(1-\rho)}, \tag{3.3}$$

where ρ is the sample correlation between $\boldsymbol{x}_j$ and $\boldsymbol{x}_{j'}$.

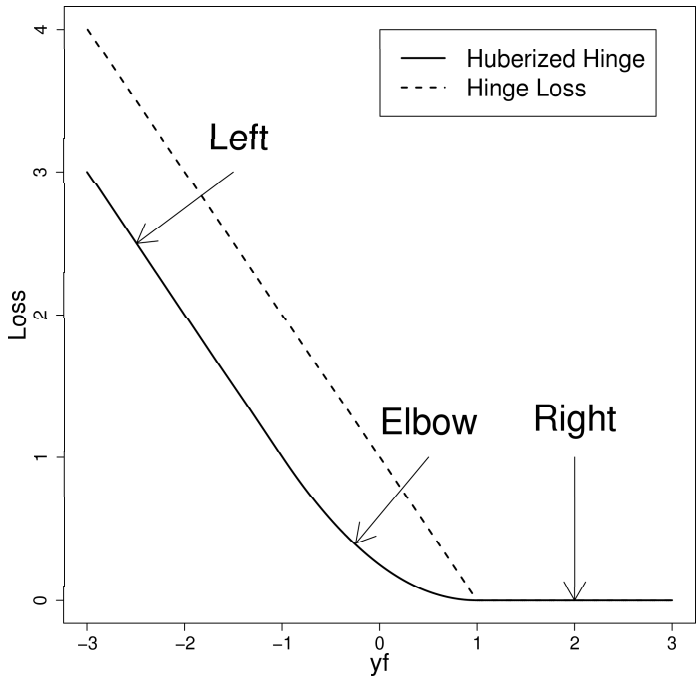

Figure 3.1: The hinge and the huberized hinge loss functions (with $\delta = 2$). Note that the *Elbow* corresponds to the region $(1-\delta, 1]$; the *Left* and the *Right* correspond to the regions $(-\infty, 1-\delta]$ and $(1, \infty)$, respectively.

Proof

Consider another set of coefficients

$$\hat{\beta}_0^* = \hat{\beta}_0,\ \hat{\beta}_\ell^* = \begin{cases} \frac{1}{2}(\hat{\beta}_j + \hat{\beta}_{j'}), & \text{if } \ell = j \text{ or } \ell = j', \\ \hat{\beta}_\ell, & \text{otherwise.} \end{cases}$$

By the definition of $\hat{\beta}_0$ and $\hat{\boldsymbol{\beta}}$, we have

$$\begin{aligned} &\sum_{i=1}^{n} \phi(y_i, \hat{\beta}_0^* + \boldsymbol{x}_i^\top \hat{\boldsymbol{\beta}}^*) + \frac{\lambda_2}{2}\|\hat{\boldsymbol{\beta}}^*\|_2^2 + \lambda_1 \|\hat{\boldsymbol{\beta}}^*\|_1 \\ &- \sum_{i=1}^{n} \phi(y_i, \hat{\beta}_0 + \boldsymbol{x}_i^\top \hat{\boldsymbol{\beta}}) - \frac{\lambda_2}{2}\|\hat{\boldsymbol{\beta}}\|_2^2 - \lambda_1 \|\hat{\boldsymbol{\beta}}\|_1 \quad \geq \quad 0, \end{aligned} \tag{3.4}$$

where

$$
\begin{aligned}
& \sum_{i=1}^{n} \Big[\phi(y_i, \hat{\beta}_0^* + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}}^*) - \phi(y_i, \hat{\beta}_0 + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}})\Big] \\
\leq & \sum_{i=1}^{n} \Big|\phi(y_i, \hat{\beta}_0^* + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}}^*) - \phi(y_i, \hat{\beta}_0 + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}})\Big| \\
\leq & \sum_{i=1}^{n} \Big|y_i(\hat{\beta}_0^* + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}}^*) - y_i(\hat{\beta}_0 + \boldsymbol{x}_i^\intercal \hat{\boldsymbol{\beta}})\Big| \qquad (3.5) \\
= & \sum_{i=1}^{n} \Big|\boldsymbol{x}_i^\intercal (\hat{\boldsymbol{\beta}}^* - \hat{\boldsymbol{\beta}})\Big| \\
= & \sum_{i=1}^{n} \Big|\frac{1}{2}(x_{ij} - x_{ij'})(\hat{\beta}_j - \hat{\beta}_{j'})\Big| \\
= & \frac{1}{2}\Big|\hat{\beta}_j - \hat{\beta}_{j'}\Big| \sum_{i=1}^{n} |x_{ij} - x_{ij'}| \\
= & \frac{1}{2}\Big|\hat{\beta}_j - \hat{\beta}_{j'}\Big| \cdot \|\boldsymbol{x}_j - \boldsymbol{x}_{j'}\|_1. \qquad (3.6)
\end{aligned}
$$

We also have

$$
\begin{aligned}
\|\hat{\boldsymbol{\beta}}^*\|_1 - \|\hat{\boldsymbol{\beta}}\|_1 &= |\hat{\beta}_j^*| + |\hat{\beta}_{j'}^*| - |\hat{\beta}_j| - |\hat{\beta}_{j'}| \\
&= |\hat{\beta}_j + \hat{\beta}_{j'}| - |\hat{\beta}_j| - |\hat{\beta}_{j'}| \leq 0, \qquad (3.7) \\
\|\hat{\boldsymbol{\beta}}^*\|_2^2 - \|\hat{\boldsymbol{\beta}}\|_2^2 &= |\hat{\beta}_j^*|^2 + |\hat{\beta}_{j'}^*|^2 - |\hat{\beta}_j|^2 - |\hat{\beta}_{j'}|^2 \\
&= -\frac{1}{2}|\hat{\beta}_j - \hat{\beta}_{j'}|^2. \qquad (3.8)
\end{aligned}
$$

Now combining (3.6), (3.7) and (3.8), (3.4) implies that

$$
\frac{1}{2}\Big|\hat{\beta}_j - \hat{\beta}_{j'}\Big| \cdot \|\boldsymbol{x}_j - \boldsymbol{x}_{j'}\|_1 - \frac{\lambda_2}{2}|\hat{\beta}_j - \hat{\beta}_{j'}|^2 \geq 0. \qquad (3.9)
$$

Hence, equation (8) in the theorem is obtained.

For equation (9) in the theorem, we simply use the inequality

$$
\|\boldsymbol{x}_j - \boldsymbol{x}_{j'}\|_1 \leq \sqrt{n}\sqrt{\|\boldsymbol{x}_j - \boldsymbol{x}_{j'}\|_2^2} = \sqrt{n}\sqrt{2(1-\rho)}. \qquad (3.10)
$$

□

Theorem III.1 suggests that highly correlated variables tend to have similar estimated coefficients; hence, they tend to be selected or removed together when λ_2 is sufficiently large.

3.3 The HHSVM Algorithm

The HHSVM involves two tuning parameters λ_1 and λ_2. According to our experience, the prediction performance is often more sensitive to λ_1, since it has more impact on selecting variables; therefore, the value of λ_1 must be chosen more carefully. For parameter tuning, one usually specifies a number of candidates, tests each of them and chooses the best one according to some criterion. However, this trial-and-error approach is computationally expensive and the optimal parameter setting can be easily missed. In this section, we propose an efficient algorithm, which solves the entire solution path for every possible value of λ_1 (when λ_2 is fixed). The algorithm is based on the fact that the solution $(\widehat{\beta}_0, \widehat{\boldsymbol{\beta}})$ is a piecewise linear function (in a multi-dimensional space) with respect to λ_1.

3.3.1 Problem Setup

Since the huberized hinge loss function has different definitions in different regions, for simplicity we define each region as:

- $\mathcal{R} = \{i : y_i f(\boldsymbol{x}_i) > 1\}$ (Right)
- $\mathcal{E} = \{i : 1 - \delta < y_i f(\boldsymbol{x}_i) \leq 1\}$ (Elbow)
- $\mathcal{L} = \{i : y_i f(\boldsymbol{x}_i) \leq 1 - \delta\}$ (Left)

where $f(\boldsymbol{x}_i) = \beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}$. We also define the indices for non-zero β_j as the active set $\mathcal{A}$:

- $\mathcal{A} = \{j : \beta_j \neq 0,\ j = 1, 2, ..., p\}$ (Active)

Since problem (3.1) is an unconstrained convex optimization problem, for any optimal solution the derivatives of its objective function must be zero. Setting the derivative with respect to β_0 to zero, we get:

$$\sum_{i\in\mathcal{E}} \frac{1}{\delta}(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta} - y_i) - \sum_{i\in\mathcal{L}} y_i = 0. \tag{3.11}$$

Setting the derivative with respect to β_j $(j \in \mathcal{A})$ to zero, we get:

$$\begin{aligned} \sum_{i\in\mathcal{E}} \frac{1}{\delta}(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta} - y_i)x_{ij} - \sum_{i\in\mathcal{L}} y_i x_{ij} + & \\ \lambda_2\beta_j + \lambda_1 sign(\beta_j) &= 0, \text{ for } j \in \mathcal{A}. \end{aligned} \tag{3.12}$$

In the linear system (3.11)-(3.12), there are $|\mathcal{A}|+1$ unknowns and $|\mathcal{A}|+1$ equations, where $|\mathcal{A}|$ represents the number of elements in set $\mathcal{A}$. Therefore, the solution β_0 and $\boldsymbol{\beta}_j$ $(j \in \mathcal{A})$ can be uniquely determined, given that the system is nonsingular.

When sets $\mathcal{L}, \mathcal{E}, \mathcal{R}$ and $\mathcal{A}$ are fixed, the structure of the linear system (3.11)-(3.12) is also fixed. Under this condition, β_0 and $\boldsymbol{\beta}_j$ $(j \in \mathcal{A})$ are linear functions of λ_1, which can be seen from (3.12). However, as λ_1 decreases, sooner or later some of the sets $\mathcal{L}, \mathcal{E}, \mathcal{R}$ and $\mathcal{A}$ will change. We call this an *event*. After an "event" occurs the new β_0 and $\boldsymbol{\beta}_j$ $(j \in \mathcal{A})$ are still linear functions of λ_1, but their derivatives with respect to λ_1 will change. Therefore, the entire solution path is piecewise linear in λ_1, and between any two consecutive events, β_0 and $\boldsymbol{\beta}_j$ $(j \in \mathcal{A})$ change linearly with λ_1. Each "event" corresponds to a kink on the piecewise linear solution path.

Our algorithm starts from $\lambda_1 = \infty$, continuously decreases λ_1, solves the solutions along this path, and terminates if λ_1 reaches 0. The algorithm provides the solutions on each kink. For any λ_1 between two consecutive kinks, the solution can be precisely obtained using linear interpolation. Table 1 shows the outline of the our algorithm.

Initialization: Calculate β_0^0, β_j^0 $(j = 1, ..., p)$, λ_1^0, $\mathcal{A}^0$, $\mathcal{L}^0, \mathcal{R}^0, \mathcal{E}^0$ according to Section 3.2 and set $k = 0$;
Step 1: Solve the linear system (3.14)-(3.15);
Step 2: Calculate $\Delta\lambda_1$ and determine the next event according to Section 3.3;
Step 3: If the stopping criterion is satisfied, terminate the algorithm;
Step 4: Otherwise, let $k = k + 1$ and update β_0^k, β_j^k $(j = 1, ..., p)$, λ_1^k, $\mathcal{A}^k$, $\mathcal{L}^k, \mathcal{R}^k$ and $\mathcal{E}^k$ according to Section 3.3;
Step 5: Goto Step 1.

Table 3.1: Outline of the HHSVM Algorithm

3.3.2 Initial Solution

The algorithm starts from $\lambda_1 = \infty$. From the objective function of problem (3.1), we can see that $\boldsymbol{\beta} = \mathbf{0}$ at this stage. Therefore, the problem reduces to:

$$\min_{\beta_0} \sum_{i=1}^{n} \phi\left(y_i \beta_0\right), \tag{3.13}$$

which involves only one parameter β_0. Since the huberized hinge loss function is convex and differentiable everywhere, this one-variable optimization problem can be easily solved. In fact, one can easily develop an analytical solution for the initial β_0, but due to the lack of space, we skip the details.

Let β_0^0 represent the optimal solution when $\lambda_1 = \infty$. Hence $\mathcal{A} = \emptyset$ $(\boldsymbol{\beta} = \mathbf{0})$ and sets $\mathcal{L}, \mathcal{E}$ and $\mathcal{R}$ can be determined using the values of $y_i \beta_0^0$ $(i = 1, ..., n)$. Reducing λ_1 tends to increase the magnitude of $\boldsymbol{\beta}$, and we can find a critical point, denoted as λ_1^0, where exactly one β_j $(j = 1, ..., p)$ joins $\mathcal{A}$, i.e., the estimate β_j will become non-zero if λ_1 is further reduced.

Since equation (3.12) must hold for any $j \in \mathcal{A}$, we can determine the critical point λ_1^0 by:

$$\lambda_1^0 = \max_{j \in \{1,..,p\}} \left| \sum_{i \in \mathcal{E}} \frac{1}{\delta} (\beta_0^0 - y_i) x_{ij} - \sum_{i \in \mathcal{L}} y_i x_{ij} \right|.$$

The corresponding variable that joins $\mathcal{A}$ first can be identified as:

$$j^* = \arg\max_{j\in\{1,..,p\}} \left| \sum_{i\in\mathcal{E}} \frac{1}{\delta}(\beta_0^0 - y_i)x_{ij} - \sum_{i\in\mathcal{L}} y_i x_{ij} \right|,$$

and the sign for β_{j^*} is:

$$sign(\beta_{j^*}) = sign\left(-\sum_{i\in\mathcal{E}} \frac{1}{\delta}(\beta_0^0 - y_i)x_{ij^*} + \sum_{i\in\mathcal{L}} y_i x_{ij^*} \right).$$

Let the superscript k indicate the iteration number and $k = 0$ for the initial stage. Now, we have $\mathcal{A}^k = \{j^*\}$, $\beta_0^k = \beta_0^0$, $\beta_j^k = 0$ $(j = 1, ..., p)$, $\lambda_1^k = \lambda_1^0$, and the $\mathcal{L}^k, \mathcal{R}^k, \mathcal{E}^k$ are determined by $y_i\beta_0^0$ $(i = 1, ..., n)$.

We emphasize again that the initial solution can be easily determined here because of the differentiability of the huberized hinge loss function; however, this is not the case for the hinge loss function. Since the hinge loss is not everywhere differentiable, it is difficult to determine the critical point λ_1^0 and the corresponding $\mathcal{A}^0, \mathcal{L}^0, \mathcal{E}^0$ and $\mathcal{R}^0$ at the initial stage. Zhu et al. [79] proposed an approach for solving the initial solution of the L_1-norm SVM. The approach was based on linear programming, but it can be computationally intensive, especially when p is large.

3.3.3 Solution Path

The algorithm continuously decreases λ_1 until it reaches 0. Let $\lambda_1 = \lambda_1^k + \Delta\lambda_1$, where $\Delta\lambda_1 < 0$. When λ_1 is reduced by a small enough amount, sets $\mathcal{L}, \mathcal{E}, \mathcal{R}$ and $\mathcal{A}$ do not change, because of the continuity of $f(\boldsymbol{x})$ with respect to λ_1. Therefore, based on the system (3.11)-(3.12), the derivatives of β_0 and β_j $(j \in \mathcal{A})$ with respect to λ_1 can be solved from the following equations:

$$\sum_{i\in\mathcal{E}} \left(\frac{\Delta\beta_0}{\Delta\lambda_1} + \sum_{k\in\mathcal{A}} x_{ik}\frac{\Delta\beta_k}{\Delta\lambda_1}\right) = 0, \tag{3.14}$$

$$\sum_{i\in\mathcal{E}} \frac{1}{\delta}\left(\frac{\Delta\beta_0}{\Delta\lambda_1} + \sum_{k\in\mathcal{A}} x_{ik}\frac{\Delta\beta_k}{\Delta\lambda_1}\right)x_{ij} + \lambda_2\frac{\Delta\beta_j}{\Delta\lambda_1} + sign(\beta_j) = 0, \text{ for } j \in \mathcal{A}. \tag{3.15}$$

Since there are $|\mathcal{A}|+1$ unknowns and $|\mathcal{A}|+1$ equations, $\frac{\Delta\beta_0}{\Delta\lambda_1}$ and $\frac{\Delta\beta_j}{\Delta\lambda_1}$ $(j \in \mathcal{A})$ are uniquely determined, given that the system is non-singular. Then when $|\Delta\lambda_1|$ is sufficiently small, the solution and fitted values are linear in λ_1:

$$\begin{aligned}
\beta_0 &= \beta_0^k + \frac{\Delta\beta_0}{\Delta\lambda_1}(\lambda_1 - \lambda_1^k), \\
\beta_j &= \beta_j^k + \frac{\Delta\beta_j}{\Delta\lambda_1}(\lambda_1 - \lambda_1^k), \quad \text{for } j \in \mathcal{A}, \\
f(\boldsymbol{x}_i) &= f^k(\boldsymbol{x}_i) + \left(\frac{\Delta\beta_0}{\Delta\lambda_1} + \sum_{j\in\mathcal{A}} \frac{\Delta\beta_j}{\Delta\lambda_1}\right)(\lambda_1 - \lambda_1^k).
\end{aligned}$$

If we keep reducing λ_1, some of the sets $\mathcal{L}, \mathcal{E}, \mathcal{R}$ and $\mathcal{A}$ will change. We call this an *event*, and four types of "events" may occur:

1. A point i reaches the boundary between $\mathcal{L}$ and $\mathcal{E}$;

2. A point i reaches the boundary between $\mathcal{R}$ and $\mathcal{E}$;

3. A parameter β_j becomes zero (j leaves $\mathcal{A}$);

4. A zero-valued parameter β_j becomes non-zero (j joins $\mathcal{A}$).

The boundary between $\mathcal{L}$ and $\mathcal{E}$ is $1-\delta$ and the boundary between $\mathcal{R}$ and $\mathcal{E}$ is 1. Therefore, when $y_i f(\boldsymbol{x}_i)$ for a point crosses $1-\delta$ or 1, one of the first two "events" occurs. To determine the step size $\Delta\lambda_1$ for the first "event," for each $i \in \mathcal{L}$ or $\mathcal{E}$ we calculate:

$$\Delta\lambda_1^i = \frac{1-\delta-y_i f^k(\boldsymbol{x}_i)}{y_i(\frac{\Delta\beta_0}{\Delta\lambda_1} + \sum_{j\in\mathcal{A}} \frac{\Delta\beta_j}{\Delta\lambda_1})}.$$

Let $\Delta\lambda_{1,1}$ represent the step size for the first "event." Since λ_1 only decreases, $\Delta\lambda_{1,1} \leq 0$ and its value should be determined by:

$$\Delta\lambda_{1,1} = \max\{\Delta\lambda_1^i : \ i \in \mathcal{L} \text{ or } \mathcal{E}, \ \Delta\lambda_1^i \leq 0\}.$$

Similarly, for the second "event" we calculate:

$$\Delta\lambda_1^i = \frac{1-y_i f^k(\boldsymbol{x}_i)}{y_i(\frac{\Delta\beta_0}{\Delta\lambda_1} + \sum_{j\in\mathcal{A}} \frac{\Delta\beta_j}{\Delta\lambda_1})},$$

for each $i \in \mathcal{R}$ or $\mathcal{E}$. And the step size for the second "event" is:

$$\Delta\lambda_{1,2} = \max\{\Delta\lambda_1^i : \ i \in \mathcal{R} \text{ or } \mathcal{E}, \ \Delta\lambda_1^i \leq 0\}.$$

When a non-zero β_j reduces to 0, the third "event" occurs. Therefore, we calculate for each $j \in \mathcal{A}$:

$$\Delta\lambda_1^j = -\beta_j^k / \frac{\Delta\beta_j}{\Delta\lambda_1}.$$

The step size for the third "event" is:

$$\Delta\lambda_{1,3} = \max\{\Delta\lambda_1^j : \ j \in \mathcal{A}, \ \Delta\lambda_1^j \leq 0\}.$$

The fourth "event" (a zero-valued β_j becomes non-zero) is a little complicated to determine. For $j = 1, \ldots, p$, we define:

$$C_j = \sum_{i \in \mathcal{E}} \frac{1}{\delta}(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta} - y_i)x_{ij} - \sum_{i \in \mathcal{L}} y_i x_{ij} + \lambda_2\beta_j,$$

which is part of the left hand side of equation (3.12). From (3.12) and the initial conditions, we know that:

- $|C_j| = \lambda_1, \ sign(C_j) = -sign(\beta_j)$, for $j \in \mathcal{A}$;
- $|C_j| < \lambda_1$, for $j \notin \mathcal{A}$.

Notice that when $|\Delta\lambda_1|$ is sufficiently small, C_j is also a linear function of λ_1:

$$C_j = C_j^k + \left[\sum_{i \in \mathcal{E}} \frac{1}{\delta}(\frac{\Delta\beta_0}{\Delta\lambda_1} + \sum_{k \in \mathcal{V}} \frac{\Delta\beta_k}{\Delta\lambda_1})x_{ij}\right](\lambda_1 - \lambda_1^k).$$

As λ_1 decreases, the value for a $|C_j|$ ($j \notin \mathcal{A}$) will first meet the decreasing λ_1, after which the corresponding β_j will become non-zero if we further reduce λ_1.

Therefore, to determine the step size for the fourth "event," for each $j \notin \mathcal{A}$ we

calculate:

$$\Delta\lambda_1^j = \begin{cases} \dfrac{C_j^k + \lambda_1^k}{-1 - \sum_{i\in\mathcal{E}} \frac{1}{\delta}(\frac{\Delta\beta_0}{\Delta\lambda_1} + \sum_{k\in\mathcal{V}} \frac{\Delta\beta_k}{\Delta\lambda_1})x_{ij}}, \\ \text{if } C_j \text{ decreases as } \lambda_1 \text{ is reduced;} \\ \\ \dfrac{C_j^k - \lambda_1^k}{1 - \sum_{i\in\mathcal{E}} \frac{1}{\delta}(\frac{\Delta\beta_0}{\Delta\lambda_1} + \sum_{k\in\mathcal{V}} \frac{\Delta\beta_k}{\Delta\lambda_1})x_{ij}}, \text{ otherwise.} \end{cases}$$

The step size $\Delta\lambda_{1,4}$ for the fourth "event" is:

$$\Delta\lambda_{1,4} = \max\{\Delta\lambda_1^j : \ j \notin \mathcal{A}\}.$$

After solving for $\Delta\lambda_{1,1}, \Delta\lambda_{1,2}, \Delta\lambda_{1,3}$ and $\Delta\lambda_{1,4}$, the overall step size $\Delta\lambda_1$ can be obtained:

$$\Delta\lambda_1 = \max\{\Delta\lambda_{1,1}, \Delta\lambda_{1,2}, \Delta\lambda_{1,3}, \Delta\lambda_{1,4}\}.$$

We then update $\mathcal{L}, \mathcal{E}, \mathcal{R}$ and $\mathcal{A}$ according to the corresponding "event," and also update $\lambda_1, \beta_0, \beta_j$, C_j and the fitted value $f(\boldsymbol{x}_i)$. Finally, let $k = k + 1$ and the algorithm goes to the next iteration: solving the linear system (3.14)-(3.15) and calculating the step size $\Delta\lambda_1$. This entire process is repeated, until λ_1 reaches 0.

Between any two consecutive "events", the solutions are linear in λ_1, and after an "event" occurs, the derivative of the solution with respect to λ_1 is changed. Therefore, the solution path is piecewise linear in λ_1, where each "event" corresponds to a kink on the path. The algorithm provides solutions at these kinks, and for any λ_1 between two consecutive kinks the solutions can be calculated precisely via linear interpolation.

3.3.4 Computational Complexity

The major computational cost in each iteration comes from solving the linear system (3.14)-(3.15). Since this system has $|\mathcal{A}| + 1$ unknowns, the computational cost seems to be $O(|\mathcal{A}|^3)$ for each iteration. However, between any two iterations, only one

element is changed for sets $\mathcal{L}, \mathcal{E}, \mathcal{R}$ and $\mathcal{A}$, so using inverse updating and downdating the computational cost can be reduced to $O(|\mathcal{A}|^2)$. It is difficult to predict the number of iterations. According to our experience, $O(\min(n, p))$ is a reasonable estimate. The heuristic is that the algorithm needs $O(n)$ to move all the points to $\mathcal{R}$ and $O(p)$ steps to add all the parameters into $\mathcal{A}$. Since p is the upper bound for $|\mathcal{A}|$, the overall computational cost is upper bounded by $O(\min(n, p)p^2)$.

3.4 Numerical Results

In this section, we use both simulation data and real microarray datasets to illustrate the HHSVM.

3.4.1 Simulation

The main purpose of the simulation is to demonstrate that when input variables are independent, the L_1-norm SVM and the HHSVM perform similarly, but the HHSVM can have an advantage when the variables are highly correlated, especially at identifying the relevant variables.

We first consider the scenario where all input variables are independent. The "+" class has a normal distribution with mean

$$\boldsymbol{\mu}_+ = (\underbrace{0.5, \ldots, 0.5}_{10}, \underbrace{0, \ldots, 0}_{p-10})^{\mathsf{T}},$$

and covariance $\boldsymbol{\Sigma} = \boldsymbol{I}_{p\times p}$. The "−" class has a similar distribution except that

$$\boldsymbol{\mu}_- = (\underbrace{-0.5, \ldots, -0.5}_{10}, \underbrace{0, \ldots, 0}_{p-10})^{\mathsf{T}}.$$

So the Bayes optimal classification rule only depends on $x_1, \ldots, x_{10}$, and the Bayes error is about 0.06, independent of the dimension p.

We generated 50=25+25 training data, each input $\boldsymbol{x}_i$ is a p=300-dimensional vector. We compared the standard L_2-norm SVM, the L_1-norm SVM, and the HHSVM.

We used 50 validation data to select the tuning parameters for each method, then applied the selected models to a separate 10,000 testing dataset. Each experiment was repeated 100 times. The means of the prediction errors and the corresponding standard errors (in parentheses) are summarized in Table 3.4.1. As we can see, the prediction errors of the L_1-norm SVM and the HHSVM are similar and both are better than that of the L_2-norm SVM. This is due to the fact that the L_2-norm SVM uses all input variables, and its prediction accuracy is "polluted" by the noise variables.

Besides the prediction error, we also compared the selected variables of the L_1-norm SVM and the HHSVM (The L_2-norm SVM keeps all input variables). In particular, we consider q_{signal} = number of selected relevant variables, and q_{noise} = number of selected noise variables. The results are in Table 3.4.1. Again, we see that the L_1-norm SVM and the HHSVM perform similarly; both are able to identify the relevant variables and remove most of the irrelevant variables.

Now we consider the scenario when the relevant variables are correlated. Similar as the independent scenario, the "+" class has a normal distribution, with mean

$$\boldsymbol{\mu}_+ = (\underbrace{1,\ldots,1}_{10},\underbrace{0,\ldots,0}_{p-10})^{\mathsf{T}}$$

and covariance

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}^*_{10\times 10} & \mathbf{0}_{10\times(p-10)} \\ \mathbf{0}_{(p-10)\times 10} & \boldsymbol{I}_{(p-10)\times(p-10)} \end{pmatrix},$$

where the diagonal elements of $\boldsymbol{\Sigma}^*$ are 1 and the off-diagonal elements are all equal to ρ=0.8. The "−" class has a similar distribution except that

$$\boldsymbol{\mu}_- = (\underbrace{-1,\ldots,-1}_{10},\underbrace{0,\ldots,0}_{p-10})^{\mathsf{T}}.$$

So the Bayes optimal classification rule depends on $x_1,\ldots,x_{10}$, which are highly correlated. The Bayes error is 0.13, independent of the dimension p.

Again, we considered n=25+25 and p=300. Each experiment was repeated 100 times. The results for the prediction errors are shown in Table 3.4.1. Now the performance of the HHSVM is slightly better than that of the L_1-norm SVM, after taking into account the standard error. The right part of Table 3.4.1 compares the variables selected by the L_1-norm SVM and the HHSVM, which sheds some light on what happened. Both the L_1-norm SVM and the HHSVM are able to identify relevant variables. However, when the relevant variables are highly correlated, the L_1-norm SVM tends to keep only a small subset of the relevant variables, and overlook the others, while the HHSVM tends to identify all of them, due to the grouping effect. Both methods seem to work well in removing irrelevant variables.

	Scenario I	Scenario II
L_2-norm SVM	0.214 (0.004)	0.160 (0.003)
L_1-norm SVM	0.143 (0.007)	0.160 (0.002)
HHSVM	0.133 (0.005)	0.143 (0.001)

Table 3.2: Comparison of test errors. The number of training observations is 50, the number of total input variables is 300, and the number of relevant variables is 10. The results are averages of test errors over 100 repetitions on a 10,000 test set, and the numbers in parentheses are the corresponding standard errors. In "Scenario I", the input variables are independent, and in "Scenario II", the relevant variables are highly correlated.

	Scenario I		Scenario II	
	q_{signal}	q_{noise}	q_{signal}	q_{noise}
L_1-norm SVM	7.2 (0.3)	6.5 (1.4)	2.5 (0.2)	2.9 (1.2)
HHSVM	7.6 (0.3)	7.1 (1.3)	7.9 (0.4)	3.3 (2.5)

Table 3.3: Comparison of variable selection. The setup are the same as those described in Table 3.4.1. q_{signal} is the number of selected relevant variables, and q_{noise} is the number of selected noise variables.

3.4.2 Real Data Analysis

The first dataset we considered is the colon cancer dataset in [1]. This dataset consists of 62 samples (40 colon cancer tumors and 22 normal tissues). Each sample consists of p=2,000 genes. We randomly split the samples into the training and

the test sets for 100 times; for each split, 42 samples (27 cancer samples and 15 normal tissues) were used for training and the rest 20 samples (13 cancer samples and 7 normal tissues) were for testing. For each split, we applied four methods, the standard L_2-norm SVM, the L_1-norm SVM, the SVM-RFE, and the HHSVM, to the training data. Tuning parameters were chosen using 10-fold cross-validation on the training data, and the chosen models were evaluated on the test data. For the standard L_2-norm SVM, we tried different kernels and found the linear kernel has the best performance. For the SVM-RFE, we used the same strategy as in [31], i.e., eliminating 10% of the remaining genes in each iteration. The results are summarized in the upper part of Table 3.4.2. We can see that the HHSVM seemed to have a slightly better classification accuracy than other methods. However, we note that this is not necessarily conclusive, for the sample size is small.

	Test Error	# of Genes
Colon cancer dataset		
SVM	14.65% (1.34%)	All
SVM-RFE	17.10% (0.87%)	64
L_1-norm SVM	15.84% (1.02%)	14.4 (3.6)
HHSVM	12.69% (0.93%)	94.5 (42.4)
Lung cancer dataset		
SVM	29.0% (0.6%)	All
SVM-RFE	28.4% (0.7%)	128
L_1-norm SVM	29.3% (0.4%)	7.2 (0.3)
HHSVM	27.8% (0.7%)	22.6 (1.9)

Table 3.4: Results on 100 random splits of the original datasets: the upper part is for the colon cancer dataset, and the lower part is for the lung cancer dataset. The numbers in the parentheses are the corresponding standard errors.

Tables 3.4.2-3.4.2 summarize the genes that were "frequently" selected by the L_1-norm SVM and the HHSVM. As we can see, only 5 genes were selected for more than 50 times by the L_1-norm SVM, while 40 genes were selected for more than 50 times by the HHSVM. The selection frequency implies that these genes may have certain power in differentiating the colon cancer samples from the normal tissues,

however, the L_1-norm SVM was not able to always identify them. Figure 3.2 shows the number of selected genes for each selection frequency out of the 100 random splits. Again, we can see that the HHSVM is more stable than the L_1-norm SVM in terms of selecting genes, i.e., over the 100 random splits, for important genes, the HHSVM selects them more frequently than the L_1-norm SVM (lower part of Figure 3.2), while for unimportant genes, the HHSVM selects them less frequently (upper part of Figure 3.2).

To assess the stability of prediction, we also recorded the frequency that each sample, as a test observation, was correctly classified. For example, if a sample appears in 70 test sets among the 100 random splits, and out of the 70 predictions, the sample was correctly classified for 60 times, then we recorded 60/70 for this sample. The results are shown in Figure 3.3. As we can see, for most samples, both the L_1-norm SVM and the HHSVM classified them correctly for most of the random splits, but there are also some samples where both methods tended to misclassify. Overall, the HHSVM seems to be slightly more stable than the L_1-norm SVM in terms of prediction.

Gene Number	Selection Frequency	Gene Number	Selection Frequency	Gene Number	Selection Frequency	Gene Number	Selection Frequency
#249	82	#1993	27	#515	16	#897	9
#377	65	#245	25	#281	16	#698	8
#765	64	#83	24	#1771	15	#1546	7
#1870	61	#365	22	#1892	13	#187	6
#1582	54	#802	21	#66	13	#266	6
#1772	33	#1325	20	#1042	12	#780	5
#1423	30	#75	19	#1047	10	#1058	5
#625	27	#493	17	#822	9	#1644	5

Table 3.5: The 32 most frequently selected genes by the L_1-SVM from the colon cancer dataset. "Selection Frequency" is the number of times that the corresponding gene was selected out of 100 random splits of the training and test data.

The second dataset we considered is a more recent lung cancer dataset [56], which consists of 198 samples (54 cancer tumors and 144 normal tissues). Each sample

Gene Number	Selection Frequency	Gene Number	Selection Frequency	Gene Number	Selection Frequency	Gene Number	Selection Frequency
#245	100	#1836	66	#1666	42	#617	35
#249	100	#1153	65	#75	41	#780	35
#377	100	#70	63	#164	41	#795	34
#493	100	#1582	62	#444	40	#1048	34
#765	100	#799	60	#1002	40	#1256	34
#1423	100	#802	59	#1440	39	#1411	34
#267	98	#897	58	#26	39	#1679	34
#286	97	#527	56	#49	39	#1843	34
#698	93	#625	55	#411	39	#15	33
#1772	92	#1110	54	#419	39	#237	33
#66	90	#350	53	#467	38	#260	33
#1870	87	#1671	52	#513	38	#261	33
#1325	84	#1042	51	#590	38	#262	33
#1993	83	#1473	50	#679	38	#263	33
#43	80	#1644	50	#812	38	#279	33
#1221	78	#83	49	#878	37	#306	33
#1058	77	#561	48	#1599	37	#365	33
#1873	76	#627	48	#1727	37	#621	32
#792	73	#1024	47	#1887	37	#661	32
#822	72	#1635	47	#807	36	#682	32
#14	71	#639	46	#1047	36	#1102	32
#281	70	#1567	45	#1196	36	#1258	32
#1346	70	#175	44	#341	35	#1370	32
#187	69	#391	43	#427	35	#1771	32
#1892	68	#1073	42	#581	35	#1798	32

Table 3.6: The 100 most frequently selected genes by the HHSVM from the colon cancer dataset. "Selection Frequency" is the number of times that the corresponding gene was selected out of 100 random splits of the training and test data.

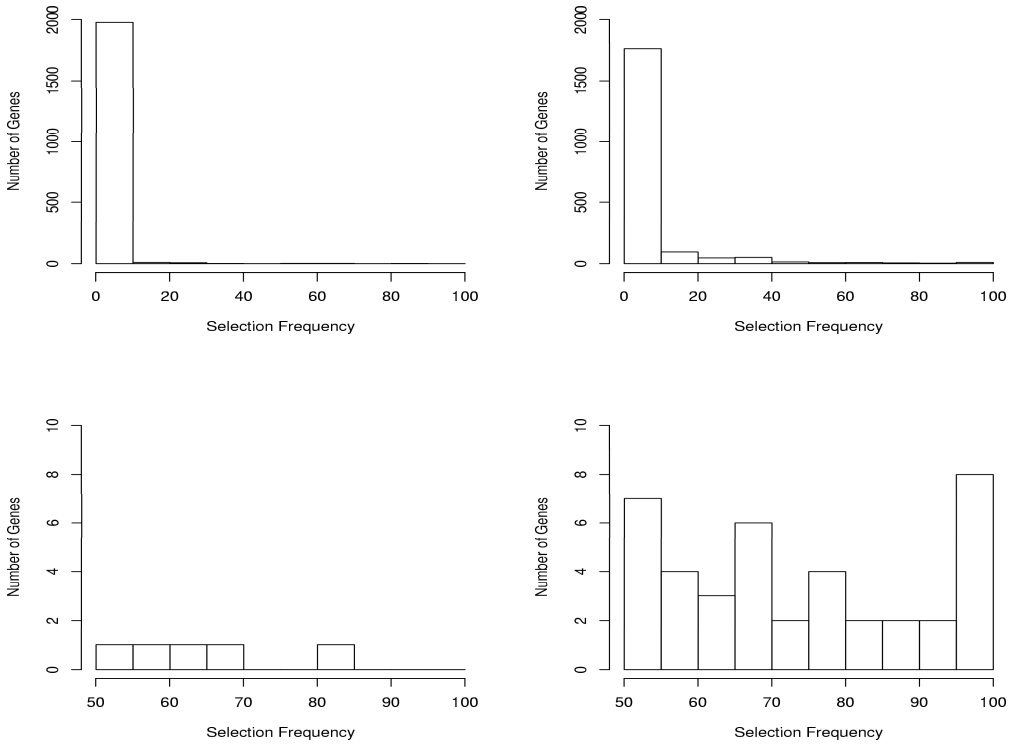

Figure 3.2: The upper part shows the number of selected genes vs the selection frequency for the L_1-norm SVM (left) and the HHSVM (right) on 100 random splits of the colon cancer dataset. The lower part "zooms in" to the region where the "selection frequency" is bigger than 50 (The left panel is for the L_1-norm SVM and the right panel is for the HHSVM).

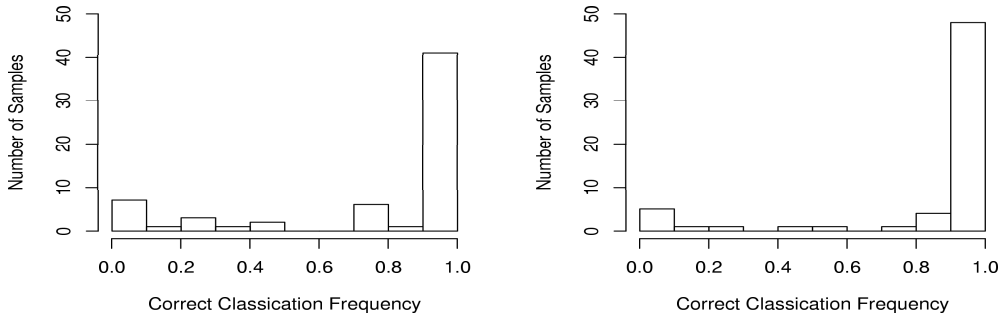

Figure 3.3: Number of samples vs the frequency that a sample (as a test observation) was correctly classified on 100 random splits of the colon cancer dataset. The left panel is for the L_1-norm SVM, and the right panel is for the HHSVM.

consists of expression measurements on 22,215 genes. We randomly splitted the data into training and test sets, with sample sizes 137 (37 cancer samples and 100 normal tissues) and 61, respectively. We repeated it 100 times. The lower part of Table 3.4.2 summarizes the results. The gene selection behavior of the L_1-norm SVM and the HHSVM for the lung cancer dataset are similar to those for the colon cancer dataset. Due to the lack of space, we skip the results.

3.5 Conclusion

In this article, we have proposed the hybrid huberized support vector machine (HHSVM) for microarray classification and gene selection. The HHSVM uses the huberized hinge loss function to measure the "badness-of-fit" and the elastic-net penalty to control the model complexity. The huberized hinge loss function allows efficient computation for calculating the entire solution path. The elastic-net penalty allows automatic flexible variable selection. We have presented some evidence that the new method tends to select more relevant variables (than the L_1-norm SVM method), especially when variables are highly correlated.

CHAPTER IV

Image Denoising via Solution Paths

4.1 Background

Image denoising problems arise in many engineering fields and applied physics, because in practice, images can often be contaminated when they are acquired by sensors or transferred in communication channels. This problem has been studied for decades, and researchers have proposed several categories of methods to tackle this problem, such as filtering methods [32, 61, 76], wavelet-based methods [8], principal component analysis (PCA)-based methods [48, 66] and sparse coding (SC) shrinkage methods [38, 62]. These methods apply in different contexts and use different strategies for denoising the contaminated image. When prior knowledge of the distribution of an image is not available, filtering methods are often used. Filtering methods tend to blur an image, however, and edges in the image are not well preserved. Wavelet-based methods are proposed to overcome this problem: they decompose a contaminated image using wavelet bases and reconstruct the image by shrinking some of the wavelet coefficients; in this fashion, edges are often better preserved. PCA-based methods and SC shrinkage methods are used when some prior knowledge of the statistical properties of the image are available. These statistical properties are often obtained from a collection of images, which share common characteristics of

the target image. The methods we propose belong to the category of PCA-based methods.

In most of the current literature, images are often assumed to be contaminated by an additive Gaussian noise. Additive Gaussian noise, which comes from the background, is often observed in analog systems. In that setting, *all* pixels are corrupted by normally distributed noise, and the magnitude of the noise is small. In this paper, we consider a different type of noise, the "impulse noise". In the impulse noise setting, only a portion of the pixels are (completely) corrupted while the values of other pixels remain accurate. Impulse noise is commonly found in digital systems, where digital images are corrupted because of, for example, malfunctioning pixels in the camera sensors, faulty memory locations in the hardware, transmission errors, analog-to-digital conversion errors, etc. Impulse noise poses new challenges to traditional image denoising methods, because the noise is no longer Gaussian; it is "sparse" and "spiky." Figure 4.1 shows an example of an original face image and a version corrupted by "impulse noise." Ideally, good denoising methods for impulse noise should be able to identify the corrupted pixels and correct only these pixels.

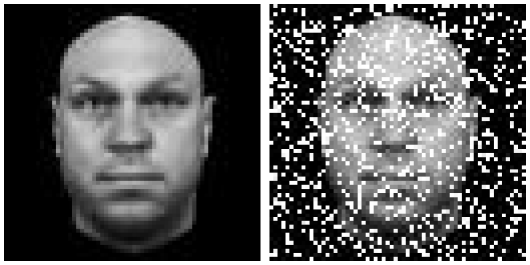

Figure 4.1: The original face image (left panel) and the corresponding image corrupted by impulse noises (right panel).

4.1.1 PCA-based Methods

Images can be represented by vectors. From a collection of images which share common characteristics, we can calculate their principal components (PC), which con-

tain important information of the images. PCA-based methods project the corrupted image to the space consisting of these PC vectors.

Let $\boldsymbol{x} \in \mathbb{R}^n$ denote the corrupted image and $\boldsymbol{U} \in \mathbb{R}^{n\times p}$ be the matrix with PCs as its columns. Without loss of generality, we assume both $\boldsymbol{x}$ and the column vectors of $\boldsymbol{U}$ are centered and standardized. Classical PCA-based methods project $\boldsymbol{x}$ to the column space of $\boldsymbol{U}$ via the least squares regression, i.e.,

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{x} - \boldsymbol{U}\boldsymbol{\beta}\|_2^2, \tag{4.1}$$

and the denoised image is constructed as $\boldsymbol{U}\boldsymbol{\beta}$. The least squares estimates often have low bias but high variance, which hurts the accuracy of the denoised image. In this paper, we propose regularization models that have the following form:

$$\min_{\boldsymbol{\alpha},\boldsymbol{\beta}} \ell(\boldsymbol{x} + \boldsymbol{\alpha}, \boldsymbol{U}\boldsymbol{\beta}) + \lambda(\|\boldsymbol{\alpha}\|_1 + \|\boldsymbol{\beta}\|_1), \tag{4.2}$$

where $\ell(\cdot,\cdot)$ is a loss function describing the discrepancy between $\boldsymbol{x}+\boldsymbol{\alpha}$ and $\boldsymbol{U}\boldsymbol{\beta}$, $\|\boldsymbol{\alpha}\|_1$ and $\|\boldsymbol{\beta}\|_1$ are L_1-norm penalties of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, equaling $|\alpha_1| + \cdots + |\alpha_n|$ and $|\beta_1| + \cdots + |\beta_p|$, and λ is a tuning parameter controlling the balance between the loss and the penalty.

There are two major differences between our models and previous models:

- The introduction of $\boldsymbol{\alpha}$: $\boldsymbol{\alpha}$ is a n-dimensional vector, and it is interpreted as a pixel-by-pixel updating vector for the "impulse noise" contaminated image. Hence our methods deliver two denoised images, $\boldsymbol{x}+\boldsymbol{\alpha}$ and $\boldsymbol{U}\boldsymbol{\beta}$, while previous models only deliver one denoised image $\boldsymbol{U}\boldsymbol{\beta}$, which is a linear combination of the PC images. As we will see later, $\boldsymbol{x}+\boldsymbol{\alpha}$ is often a better denoised image than $\boldsymbol{U}\boldsymbol{\beta}$ when the noise is the impulse type.

- The usage of the L_1-norm penalty: The L_1-norm (LASSO) penalty sets some of the elements in $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ exactly equal to zero, i.e., sparse property [47,

68, 13]. When constructing the denoised image, some of the PC images are important, and some of the PC images may not be important. The L_1-norm of $\boldsymbol{\beta}$ helps eliminate unimportant PC images. More importantly, since we consider the impulse type noises, i.e., some pixels of the original image are corrupted but other pixels are good, the L_1-norm of $\boldsymbol{\alpha}$ helps identify and repair only the corrupted pixels and leave the uncorrupted ones untouched. This is related to the variable selection problem in statistics, which has been studied extensively in the literature, for example, see [27], [7], [68], [26], and [21].

As we will see in numerical studies, these new models show promising results in denoising the impulse type noise.

The rest of the paper is organized as follows: In Section 4.2, we propose two models for image denoising that use the L_1-norm of the pixel updates as the penalty. In Section 4.3, we derive efficient algorithms that compute the entire solution paths of these two models. In Section 4.4, we present numerical results on a real image dataset. We conclude the paper with Section 4.5.

4.2 Models

Since the impulse type noise tends to have a heavy-tail distribution, and the squared error loss is not robust to outliers, we consider the least absolute deviation (LAD) loss, in addition to the least squares (LS) loss. Specifically, we consider the following regularization models for denoising impulsely contaminated images:

$$\text{LAD-LASSO} \quad : \quad \min_{\boldsymbol{\alpha},\boldsymbol{\beta}} \|\boldsymbol{x}+\boldsymbol{\alpha}-\boldsymbol{U}\boldsymbol{\beta}\|_1+\lambda(\|\boldsymbol{\alpha}\|_1+\|\boldsymbol{\beta}\|_1), \tag{4.3}$$

$$\text{LS-LASSO} \quad : \quad \min_{\boldsymbol{\alpha},\boldsymbol{\beta}} \|\boldsymbol{x}+\boldsymbol{\alpha}-\boldsymbol{U}\boldsymbol{\beta}\|_2^2+\lambda(\|\boldsymbol{\alpha}\|_1+\|\boldsymbol{\beta}\|_1). \tag{4.4}$$

To further improve models (4.3) and (4.4), we apply the adaptive idea which has been used in [7, 63, 78] and [80], i.e., to penalize different components in $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$

differently:

$$\text{(4.5) LAD-Adaptive-LASSO} \quad : \quad \min_{\boldsymbol{\alpha},\boldsymbol{\beta}} \|\boldsymbol{x}+\boldsymbol{\alpha}-\boldsymbol{U}\boldsymbol{\beta}\|_1 + \lambda\Big(\sum_{i=1}^{n}\frac{|\alpha_i|}{|\alpha_i^0|}+\sum_{j=1}^{p}\frac{|\beta_j|}{|\beta_j^0|}\Big),$$

$$\text{(4.6)} \quad \text{LS-Adaptive-LASSO} \quad : \quad \min_{\boldsymbol{\alpha},\boldsymbol{\beta}} \|\boldsymbol{x}+\boldsymbol{\alpha}-\boldsymbol{U}\boldsymbol{\beta}\|_2^2 + \lambda\Big(\sum_{i=1}^{n}\frac{|\alpha_i|}{|\alpha_i^0|}+\sum_{j=1}^{p}\frac{|\beta_j|}{|\beta_j^0|}\Big),$$

where $|\alpha_i^0|$ and $|\beta_j^0|$ can be regarded as pre-fixed adaptive weights. The intuition is that for unimportant PC images and uncorrupted pixels, we would like the corresponding weights $|\beta_j^0|$ and $|\alpha_i^0|$ to be small, hence β_j and α_i are heavily penalized, while for important PC images and corrupted pixels, we would like the corresponding weights $|\beta_j^0|$ and $|\alpha_i^0|$ to be big, hence β_j and α_i are lightly penalized. How to pre-specify the weights $|\beta_j^0|$ and $|\alpha_i^0|$ from the data is discussed in Section 4.4.

As in every regularization problem, choice of the regularization parameter λ is critical. In practice, people usually pre-specify a finite set of candidate values for λ that cover a wide range, then either use a separate validation dataset or certain model selection criterion to pick a value for λ that gives the best performance among the pre-specified set. For detailed description of different model selection criteria for image denoising, we refer the readers to [67]. There is no theoretical guidance on how to pick the candidate set of the regularization parameter; people have to rely on either their previous experience or the trial-and-error approach. Since the denoising performance is often sensitive to the value of the regularization parameter, parameter tuning can be very time-consuming and the best parameter can be easily missed.

In the next section, we derive very efficient algorithms that compute the *exact entire solution paths* of $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ as functions of λ, which facilitate selection of the regularization parameter.

4.3 Solution Path Algorithms

In this section, we derive efficient algorithms that compute the exact solution paths for LAD-Adaptive-LASSO (4.5) and LS-Adaptive-LASSO (4.6). Since the algorithm for LS-Adaptive-LASSO is a simple modification of the LAR/LASSO algorithm [17], we focus only on the algorithm for LAD-Adaptive-LASSO. We also note that the naive methods (4.3) and (4.4) are special cases of the adaptive methods, with $|\alpha_i^0| = 1$ and $|\beta_j^0| = 1$.

In the following sections, we use $\boldsymbol{u}_i$ to denote the ith row of the eigen-image matrix $\boldsymbol{U}$.

4.3.1 Problem Setup

Criterion (4.5) can be re-written in an equivalent way:

$$\min_{\boldsymbol{\alpha},\boldsymbol{\beta}} \quad \sum_{i=1}^{n} \epsilon_i \tag{4.7}$$

$$\text{subject to} \quad -\epsilon_i \leq x_i + \alpha_i - \boldsymbol{u}_i^{\mathrm{T}}\boldsymbol{\beta} \leq \epsilon_i, \tag{4.8}$$

$$\epsilon_i \geq 0,\ i = 1, \ldots, n, \tag{4.9}$$

$$\sum_{i=1}^{n} \frac{|\alpha_i|}{|\alpha_i^0|} + \sum_{j=1}^{p} \frac{|\beta_j|}{|\beta_j^0|} \leq s. \tag{4.10}$$

Notice the absolute value loss is replaced with two linear constraints in (4.8). Also notice that the tuning parameter λ is replaced by another equivalent tuning parameter s. We are going to show that the solutions $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are piecewise linear functions with respect to s, which allow us to develop an efficient algorithm to compute the

entire solution path. (4.7)–(4.10) give the Lagrangian primal function:

$$\begin{aligned} L_p \;:\; & \sum_{i=1}^{n}\epsilon_i + \lambda^*\Big(\sum_{i=1}^{n}\frac{|\alpha_i|}{|\alpha_i^0|} + \sum_{j=1}^{p}\frac{|\beta_j|}{|\beta_j^0|} - s\Big) \\ & + \sum_{i=1}^{n}\eta_i^+(x_i + \alpha_i - \boldsymbol{u}_i^\intercal\boldsymbol{\beta} - \epsilon_i) \\ & - \sum_{i=1}^{n}\eta_i^-(x_i + \alpha_i - \boldsymbol{u}_i^\intercal\boldsymbol{\beta} + \epsilon_i) \\ & - \sum_{i=1}^{n}\gamma_i\epsilon_i, \end{aligned}$$

where $\lambda^*, \eta_i^+, \eta_i^-$ and γ_i are non-negative Lagrangian multipliers. Setting the derivatives of L_p to zero, we arrive at:

$$\frac{\partial}{\partial\boldsymbol{\beta}} \;:\; -\sum_{i=1}^{n}(\eta_i^+ - \eta_i^-)u_{ij} + \lambda^*\frac{\text{sign}(\beta_j)}{|\beta_j^0|} = 0, \text{ for } j \text{ with } \beta_j \neq 0, \tag{4.11}$$

$$\frac{\partial}{\partial\boldsymbol{\alpha}} \;:\; (\eta_i^+ - \eta_i^-) + \lambda^*\frac{\text{sign}(\alpha_i)}{|\alpha_i^0|} = 0, \text{ for } i \text{ with } \alpha_i \neq 0, \tag{4.12}$$

$$\frac{\partial}{\partial\epsilon_i} \;:\; 1 - \eta_i^+ - \eta_i^- - \gamma_i = 0,\ i = 1, \ldots, n, \tag{4.13}$$

and the complementary slackness conditions are:

$$\eta_i^+(x_i + \alpha_i - \boldsymbol{u}_i^\intercal\boldsymbol{\beta} - \epsilon_i) = 0,\ i = 1, \ldots, n, \tag{4.14}$$

$$\eta_i^-(x_i + \alpha_i - \boldsymbol{u}_i^\intercal\boldsymbol{\beta} + \epsilon_i) = 0,\ i = 1, \ldots, n, \tag{4.15}$$

$$\gamma_i\epsilon_i = 0,\ i = 1, \ldots, n. \tag{4.16}$$

Since the Lagrange multipliers must be non-negative, we see from (4.13) that $0 \leq \eta_i^+, \eta_i^- \leq 1$. We can also see that (4.8) and (4.13)–(4.16) lead to the following:

$$\begin{aligned} x_i + \alpha_i - \boldsymbol{u}_i^\intercal\boldsymbol{\beta} > 0 \;&\Rightarrow\; \epsilon_i > 0,\ \gamma_i = 0,\ \eta_i^+ = 1,\ \eta_i^- = 0; \\ x_i + \alpha_i - \boldsymbol{u}_i^\intercal\boldsymbol{\beta} < 0 \;&\Rightarrow\; \epsilon_i > 0,\ \gamma_i = 0,\ \eta_i^+ = 0,\ \eta_i^- = 1; \\ x_i + \alpha_i - \boldsymbol{u}_i^\intercal\boldsymbol{\beta} = 0 \;&\Rightarrow\; \epsilon_i = 0,\ \gamma_i \in [0,1],\ \eta_i^+ \in [0,1],\ \eta_i^- \in [0,1]. \end{aligned}$$

We define $\eta_i = \eta_i^+ - \eta_i^-$. Hence, using these relationships, we can define the following five sets that will be used later when we calculate the solution path of LAD-Adaptive-LASSO:

- $\mathcal{E} = \{i : x_i + \alpha_i - \boldsymbol{u}_i^\intercal \boldsymbol{\beta} = 0, -1 \leq \eta_i \leq 1\}$ (Elbow)
- $\mathcal{R} = \{i : x_i + \alpha_i - \boldsymbol{u}_i^\intercal \boldsymbol{\beta} > 0, \eta_i = 1\}$ (Right of the elbow)
- $\mathcal{L} = \{i : x_i + \alpha_i - \boldsymbol{u}_i^\intercal \boldsymbol{\beta} < 0, \eta_i = -1\}$ (Left of the elbow)
- $\mathcal{V} = \{j : \beta_j \neq 0\}$ (Active set for $\boldsymbol{\beta}$)
- $\mathcal{W} = \{i : \alpha_i \neq 0\}$ (Active set for $\boldsymbol{\alpha}$)

For pixels in $\mathcal{R}$ and $\mathcal{L}$, their η_i are determined. Therefore, we will focus on pixels in $\mathcal{E}$.

The basic idea of our algorithm is as follows: We start with $s = 0$ and increase it, keeping track of the location of all pixels relative to the elbow and also of the magnitude of the fitted coefficients β_j and α_i along the way. As s increases, by continuity, points in the elbow $\mathcal{E}$ must linger on it. Since all points at the elbow have $x_i + \alpha_i - \boldsymbol{u}_i^\intercal \boldsymbol{\beta} = 0$, we can establish a path for $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$. The elbow set will stay stable until either a new pixel comes to the elbow or a non-zero fitted coefficient has dropped to zero.

4.3.2 Initial Solution

Initially, when $s = 0$, we can see from (4.8)–(4.10) that $\epsilon_i = |x_i|$, hence the initial dual variables η_i's and the initial sets $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{L}$ can be determined.

When $s \to 0^+$, one of the α_i's or β_j's will become non-zero. Let $\lambda_\beta^* = \max_j |\beta_j^0 \sum_{i=1}^n \eta_j u_{ij}|$ and $\lambda_\alpha^* = \max_i |\alpha_i^0 \eta_i|$, then from (4.11)–(4.12), we can see

- If $\lambda_\beta^* > \lambda_\alpha^*$, the initial $\mathcal{V} = \{j^* = \arg\max_j |\beta_j^0 \sum_{i=1}^n \eta_j u_{ij}|\}$, $\mathcal{W} = \emptyset$, $\text{sign}(\beta_{j^*}) = \text{sign}(\sum_{i=1}^n \eta_{j^*} u_{ij^*})$, and $\lambda^* = \lambda_\beta^*$.

- If $\lambda_\beta^* < \lambda_\alpha^*$, the initial $\mathcal{V} = \emptyset$, $\mathcal{W} = \{i^* = \arg\max_i |\alpha_i^0 \eta_i|\}$, $\text{sign}(\alpha_{i^*}) = -\text{sign}(\eta_{i^*})$, and $\lambda^* = \lambda_\alpha^*$.

In the following, we use ℓ to indicate the step number, with the initial $\ell = 0$.

4.3.3 Solution Path

When s is small, the constraint (4.10) is active, i.e., $\sum_{i=1}^n |\alpha_i|/|\alpha_i^0| + \sum_{j=1}^p |\beta_j|/|\beta_j^0| = s$. When s increases to a certain value, say s^*, this constraint will become inactive, and the solution will not change beyond the value of s^*. This corresponds to $\lambda = 0$ in (4.5). Suppose for a value $s < s^*$, we have the solution $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, hence $\mathcal{E}$, $\mathcal{R}$, $\mathcal{L}$, $\mathcal{V}$, and $\mathcal{W}$ are also known. Then $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ have to satisfy the following equations:

$$\begin{aligned} x_i + \alpha_i - \sum_{j \in \mathcal{V}} u_{ij}\beta_j &= 0, \text{ for } i \in \mathcal{E} \text{ and } i \in \mathcal{W}, \\ x_i - \sum_{j \in \mathcal{V}} u_{ij}\beta_j &= 0, \text{ for } i \in \mathcal{E} \text{ and } i \notin \mathcal{W}, \\ \sum_{i \in \mathcal{W}} \frac{|\alpha_i|}{|\alpha_i^0|} + \sum_{j \in \mathcal{V}} \frac{|\beta_j|}{|\beta_j^0|} &= s. \end{aligned}$$

This linear system consists of $|\mathcal{E}| + 1$ equations and $|\mathcal{V}| + |\mathcal{W}|$ unknowns. We also notice that the linear system (4.11)–(4.12) consists of $|\mathcal{V}| + |\mathcal{W}|$ equations and $|\mathcal{E}| + 1$ unknowns. Therefore, to satisfy both systems, we must have $|\mathcal{V}| + |\mathcal{W}| = |\mathcal{E}| + 1$.

When s increases by a small enough amount, the sets $\mathcal{E}$, $\mathcal{R}$, $\mathcal{L}$, $\mathcal{V}$ and $\mathcal{W}$ will not change due to their continuity with respect to s, and the structure of the above linear

system will not change. Taking right derivatives with respect to s, we have

$$\frac{\Delta\alpha_i}{\Delta s} - \sum_{j\in\mathcal{V}} u_{ij}\frac{\Delta\beta_j}{\Delta s} = 0, \text{ for } i\in\mathcal{E} \text{ and } i\in\mathcal{W}, \tag{4.17}$$

$$\sum_{j\in\mathcal{V}} u_{ij}\frac{\Delta\beta_j}{\Delta s} = 0, \text{ for } i\in\mathcal{E} \text{ and } i\notin\mathcal{W}, \tag{4.18}$$

$$\sum_{i\in\mathcal{W}} \frac{\text{sign}(\alpha_i^\ell)}{|\alpha_i^0|}\frac{\Delta\alpha_i}{\Delta s} + \sum_{j\in\mathcal{V}} \frac{\text{sign}(\beta_j^\ell)}{|\beta_j^0|}\frac{\Delta\beta_j}{\Delta s} = 1. \tag{4.19}$$

Since $|\mathcal{E}|+1 = |\mathcal{V}|+|\mathcal{W}|$, $\Delta\alpha_i/\Delta s$ and $\Delta\beta_j/\Delta s$ are constant and can be uniquely determined, assuming the linear system is non-singular. Therefore, if s increases by a small enough amount, α_i and β_j change linearly in s

$$\alpha_i = \alpha_i^\ell + (s-s^\ell)\frac{\Delta\alpha_i}{\Delta s}, \text{ for } i\in\mathcal{W}, \tag{4.20}$$

$$\beta_j = \beta_j^\ell + (s-s^\ell)\frac{\Delta\beta_j}{\Delta s}, \text{ for } j\in\mathcal{V}, \tag{4.21}$$

and the residual $r_i = x_i + \alpha_i - \boldsymbol{u}_i^\intercal\boldsymbol{\beta}$ changes as

$$r_i = r_i^\ell + (\alpha_i - \alpha_i^\ell) - \sum_{j\in\mathcal{V}}(\beta_j - \beta_j^\ell)u_{ij}. \tag{4.22}$$

When the increase in s is big enough, one of the $\mathcal{E}$, $\mathcal{V}$ and $\mathcal{W}$ sets will change, and the structure of the linear system will also change.

To identify changes in the structure of the linear system, we define three types of *events*, corresponding to the changes in $\mathcal{E}$, $\mathcal{V}$ and $\mathcal{W}$.

- A pixel hits the elbow $\mathcal{E}$ from $\mathcal{R}$ or $\mathcal{L}$, i.e., a residual $x_i+\alpha_i-\boldsymbol{u}_i^\intercal\boldsymbol{\beta}$ changes from non-zero to zero.
- A coefficient α_i changes from non-zero to zero.
- A coefficient β_j changes from non-zero to zero.

Given s^ℓ, equations (4.20)–(4.22) allow us to compute $s^{\ell+1}$, the value of s at which the next *event* will occur. This will be the smallest s larger than s^ℓ, such that either

ϵ_i for $i \notin \mathcal{E}$ reaches zero, or one of the coefficients α_i for $i \in \mathcal{W}$ or β_j for $j \in \mathcal{V}$ reaches zero.

When an *event* occurs, by the definition of an *event*, the condition $|\mathcal{E}|+1 = |\mathcal{V}|+|\mathcal{W}|$ no longer holds. Therefore, to make the KKT conditions satisfied, we need to take some *action*. We define three types of *actions*:

- remove a pixel from $\mathcal{E}$;
- add a new coefficient into $\mathcal{V}$;
- add a new coefficient into $\mathcal{W}$.

The choice can be determined by solving the dual variables using (4.11)–(4.12).

Let $\mathcal{E}^*$, $\mathcal{R}^*$, $\mathcal{L}^*$, $\mathcal{V}^*$ and $\mathcal{W}^*$ denote the sets immediately after the $(\ell+1)$th *event* has occurred. Notice that $|\mathcal{E}^*| = |\mathcal{V}^*| + |\mathcal{W}^*|$. For pixels in $\mathcal{R}^*$ and $\mathcal{L}^*$, the values of η_i are known and fixed, so we have

$$\begin{aligned} \sum_{i\in\mathcal{E}^*}(\eta_i - \eta_i^\ell)u_{ij} &= (\lambda^* - \lambda^{*\ell})\frac{\text{sign}(\beta_j^\ell)}{|\beta_j^0|},\ \forall j \in \mathcal{V}^*, \\ -(\eta_i - \eta_i^\ell) &= (\lambda^* - \lambda^{*\ell})\frac{\text{sign}(\alpha_i^\ell)}{|\alpha_i^0|},\ \forall i \in \mathcal{W}^*. \end{aligned}$$

To simplify, let $\frac{\Delta\eta_i}{\Delta\lambda^*} = (\eta_i - \eta_i^\ell)/(\lambda^* - \lambda^{*\ell})$. Then

$$\begin{aligned} \sum_{i\in\mathcal{E}^*}\frac{\Delta\eta_i}{\Delta\lambda^*}u_{ij} &= \frac{\text{sign}(\beta_j^\ell)}{|\beta_j^0|},\ \forall j \in \mathcal{V}^*, \\ -\frac{\Delta\eta_i}{\Delta\lambda^*} &= \frac{\text{sign}(\alpha_i)}{|\alpha_i^0|},\ \forall i \in \mathcal{W}^*. \end{aligned}$$

There are $|\mathcal{V}^*|+|\mathcal{W}^*|$ equations and $|\mathcal{E}^*|$ unknowns. Since $|\mathcal{E}^*| = |\mathcal{V}^*|+|\mathcal{W}^*|$, we can solve for $\Delta\eta_i/\Delta\lambda^*$, and we have

$$\eta_i = \eta_i^\ell + (\lambda^* - \lambda^{*\ell})\frac{\Delta\eta_i}{\Delta\lambda^*},\ \text{for } i \in \mathcal{E}^*. \tag{4.23}$$

Thus for $\lambda^{*\ell} > \lambda^* > \lambda^{*(\ell+1)}$, the η_i's proceed linearly in λ^*.

Regarding adding a coefficient into $\mathcal{V}$ or $\mathcal{W}$, from (4.11) and (4.12), we can see that when λ^* decreases, it corresponds to

(4.24)
$$\sum_{i\in\mathcal{E}^*}\left(\eta_i^\ell+(\lambda^*-\lambda^{*\ell})\frac{\Delta\eta_i}{\Delta\lambda^*}\right)u_{ij}+\sum_{i\in\mathcal{R}^*}u_{ij}-\sum_{i\in\mathcal{L}^*}u_{ij}=\lambda^*\frac{\operatorname{sign}(\beta_j)}{|\beta_j^0|},\ \text{for some } j\notin\mathcal{V}^*,$$

or

$$-(\eta_i^\ell+(\lambda^*-\lambda^{*\ell})\frac{\Delta\eta_i}{\Delta\lambda^*})=\lambda^*\frac{\operatorname{sign}(\alpha_i)}{|\alpha_i^0|},\ \text{for some } i\notin\mathcal{W}^*. \tag{4.25}$$

Equations (4.23)–(4.25) allow us to compute $\lambda^{*(\ell+1)}$, the largest λ^* smaller than $\lambda^{*\ell}$, such that either one of the η_i for $i\in\mathcal{E}$ reaches 1 or -1, or one of the coefficients α_i for $i\notin\mathcal{W}$ or β_j for $j\notin\mathcal{V}$ joins the active set.

Once an *action* is taken, we restore $|\mathcal{E}|+1=|\mathcal{V}|+|\mathcal{W}|$. The algorithm keeps increasing s and alternates between reaching the next *event* and taking an *action*, until λ^* reduces to 0.

4.3.4 Computational Complexity

The major computational cost for updating the solutions at any step ℓ involves two things: solving the primal system (4.17)–(4.19) and solving the dual system (4.23)–(4.23). Since there are $|\mathcal{E}|+1$ unknowns in each system, and $|\mathcal{E}|$ can be as large as n, it seems that the computational cost is expensive. However, by taking advantage of the special structure in these two systems, we can significantly reduce the computational cost. From (4.17), we can write

$$\frac{\Delta\alpha_i}{\Delta s}=\sum_{j\in\mathcal{V}}u_{ij}\frac{\Delta\beta_j}{\Delta s},\ \text{for } i\in\mathcal{W}.$$

Plug it into (4.19), we then get a system with only $|\mathcal{V}|$ unknowns. Similarly, we can simplify the dual system to be the same size. Since the sets differ by only one element between consecutive *events*, using inverse updating and downdating, the

computational complexity is only $O(|\mathcal{V}|^2)$. It is hard to predict the number of steps in the algorithm, but according to our experience, the total number of steps taken by the algorithm is on average $O(n)$. Since $|\mathcal{V}|$ is upper bounded by p, the overall computational cost is $O(np^2)$.

4.4 Real Data

In this section, we consider an application to a real world dataset. The dataset came from the Max-Planck Institute (MPI) face database [69], which contains face images of 100 males and 100 females. We transformed them into gray-scale images with resolution 64×64 (so n=4,096).

We randomly split the 200 images into 180 for training and 20 for testing. We used the training images to compute the principal components $\boldsymbol{U}$. For each of the testing images, we randomly selected 20% of the pixels and set them to the white color ("complete corruption"). Figure 4.1 in Section 5.1 illustrates an example. As we can see, with 20% corrupted pixels, the original face can hardly be recognized by visual inspection.

Figure 4.2 shows the first 10 principal components (or eigen-images). They are quite informative, representing different features of the faces. We used the first 100 eigen-images in our analysis.

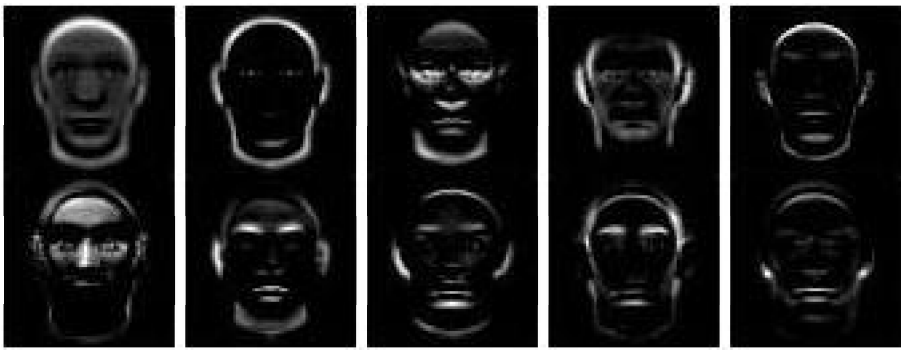

Figure 4.2: The first 10 PC-images.

Six different methods were compared: the ordinary least squares (OLS) (4.1), the "idealistic" method [70], the LAD-LASSO (4.3), the LS-LASSO (4.4), and their adaptive versions (4.5) and (4.6). The OLS and the "idealistic" methods were used as benchmarks. In the OLS method, all pixels in the corrupted image were projected onto the PC space as in (4.1). The "idealistic" method first removed the corrupted pixels from the noisy image and their corresponding components from the PC images, then projected only the "good" pixels onto the PC space (via OLS). Notice that the "idealistic" method usually cannot be implemented in practice, since we do not know in advance which pixels are corrupted and which are not.

In the adaptive methods, we used the OLS result $|\beta_j^{\mathrm{OLS}}|$ as the weight for β_j, and $|x_i - \boldsymbol{u}_i^{\mathrm{T}}\boldsymbol{\beta}^{\mathrm{OLS}}|$ as the weight for α_i.

Tuning parameters in different LASSO methods were chosen via five-fold cross-validation based on the training data (the eigen-images were also re-computed for each fold). The denoised image was compared with the original un-contaminated image using the mean squared error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - x_i^0)^2,$$

where $\hat{\boldsymbol{x}}$ is the denoised image, and $\boldsymbol{x}^0$ is the original un-contaminated image. Notice that each of the LASSO methods generated two denoised images: $\boldsymbol{x} + \boldsymbol{\alpha}$, the pixel by pixel denoised image, and $\boldsymbol{U}\boldsymbol{\beta}$, the PC-projected image. We compared both with the original un-contaminated image.

Numerical results are summarized in the first part of Table 4.4. As we can see, for this particular dataset, the OLS method performed much worse than other methods, which agrees with the general belief that some regularization is necessary. The PC-projected images $\boldsymbol{U}\boldsymbol{\beta}$ performed about the same across different methods, and they were all worse than the pixel-by-pixel denoised images. Among the pixel-by-pixel

denoised images, the adaptive methods tended to work better than the non-adaptive methods. The LAD-Adaptive method also performed slightly better than the LS-Adaptive method. Interestingly, the pixel-by-pixel denoised images performed better than the "idealistic" method, by a margin as large as 40%.

	PC-Projection				Pixel-by-Pixel					
	LS	LAD	LS-A	LAD-A	LS	LAD	LS-A	LAD-A	OLS	Ideal
Scenario I: "complete corruption"										
MSE	222.9	234.0	222.4	227.5	187.2	188.1	135.3	115.1	2085.3	203.2
	(13.9)	(14.8)	(12.7)	(14.4)	(12.1)	(12.2)	(9.8)	(12.9)	(18.2)	(12.2)
Scenario II: "incomplete corruption"										
MSE	221.5	225.1	216.9	218.3	163.8	161.5	149.2	147.3	624.9	203.3
	(14.2)	(14.2)	(13.1)	(13.5)	(10.7)	(12.3)	(9.3)	(11.7)	(14.5)	(12.2)
Scenario III: "block corruption"										
MSE	230.4	233.8	224.5	231.5	153.3	152.3	140.6	140.1	284.7	210.0
	(14.6)	(14.6)	(13.4)	(15.0)	(9.8)	(12.0)	(8.6)	(12.3)	(18.2)	(12.7)

Table 4.1: The results are averages over the 20 testing images. The numbers in the parentheses are the corresponding standard errors. "PC-Projection" is for the denoised image $\boldsymbol{U\beta}$, and "Pixel-by-Pixel" is for the denoised image $\boldsymbol{x} + \boldsymbol{\alpha}$. "LAD" is for the LAD-LASSO method (4.3), and "LS" is for the LS-LASSO method (4.4). "LAD-A" and "LS-A" are the corresponding adaptive version (4.5) and (4.6). "OLS" is the ordinary least squares method, and "Ideal" is the "idealistic" method.

Figure 4.3 shows examples of denoised images using the OLS method and the "idealistic" method. Figure 4.4 and Figure 4.5 show some of the representative images along the solution paths (for different values of s) from different LASSO methods. Since the PC-projected images, i.e., $\boldsymbol{U\beta}$, of different models look similar, we only show one of them in Figure 4.4. Figure 4.5 shows the pixel-by-pixel updated images, i.e., $\boldsymbol{x} + \boldsymbol{\alpha}$, corresponding to models (4.3)–(4.6). Different images correspond to different values of s (from small to large). There are several interesting patterns we can see from these figures: (1) The PC-projected images are always smooth (blurred), since they are constructed from a linear combination of PC images. (2) The LAD-based methods and the LS-based methods remove noise pixels in different ways. As s increases, the LAD-based methods keep working on a noisy pixel until it is completely recovered, before which the rest of the pixels are untouched. On the other hand the

LS-based methods tend to first repair pixels with the largest noise level, so they work on different pixels "simultaneously". (3) The LAD-Adaptive-LASSO method works differently from the LAD-LASSO method. As s increases, the LAD-Adaptive-LASSO method tends to select pixels with larger noise levels and fix them first, while the LAD-LASSO method tends to randomly select a noisy pixel and fix it.

Figure 4.3: "Complete corruption" scenario. The denoised images using the OLS method (left panel) and the "idealistic" method (right panel).

Figure 4.4: "Complete corruption" scenario. The PC-projected denoised images, i.e., $\boldsymbol{U}\boldsymbol{\beta}$. Different images correspond to different values of s (from small to large).

To further illustrate our methods, we also considered two other scenarios for the corrupted pixels. In the second scenario, instead of setting a corrupted pixel to the white color, i.e., "complete corruption", we set the pixel to a value uniform between 0 and 255 (left panel in Figure 4.6). In the third scenario, instead of randomly selecting pixels in an image as corrupted pixels, we restricted the corrupted pixels to form a "block" (right panel in Figure 4.6), and the corrupted pixels also took values uniform between 0 and 255. We note that when the corrupted pixels form a block, the L_1-norm penalty in (4.2) may not be the best choice, for it does not take into account the block structure. The results are summarized in the second and the third parts of Table 4.4

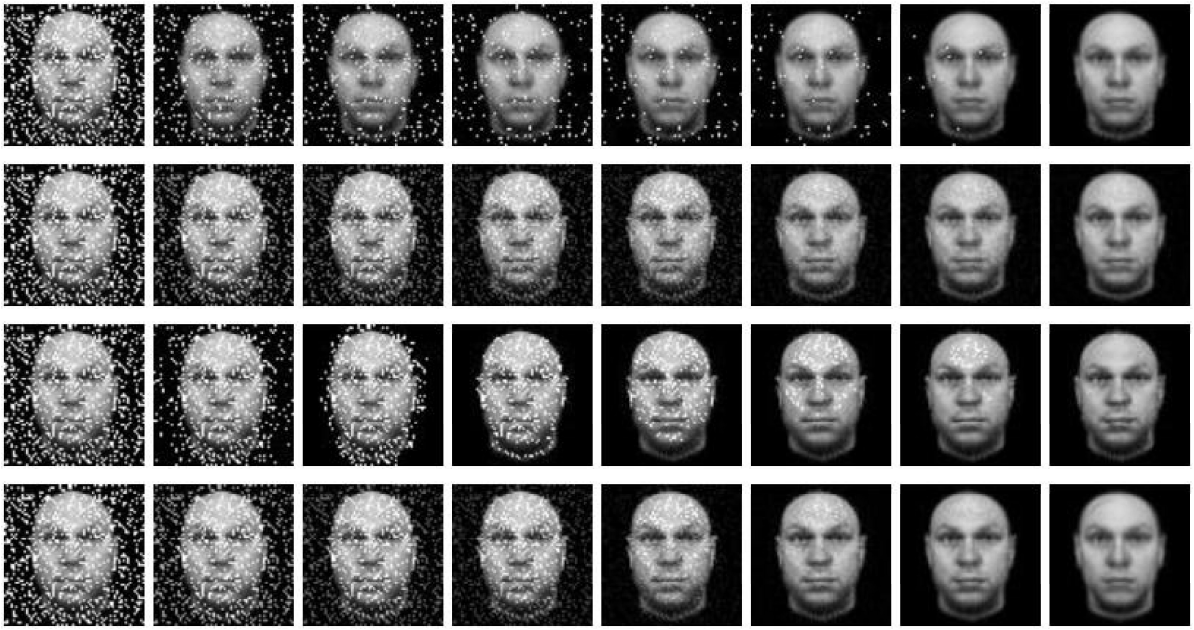

Figure 4.5: "Complete corruption" scenario. The pixel-by-pixel denoised images, i.e., $\boldsymbol{x} + \boldsymbol{\alpha}$, using LAD-LASSO (first row), LS-LASSO (second row), LAD-Adaptive-LASSO (third row) and LS-Adaptive-LASSO (fourth row). Different images correspond to different values of s (from small to large).

and Figure 4.7. Similar as the previous result, the OLS method performed the worst for this particular dataset, and the "pixel-by-pixel" based methods performed better than the "PC-projection" based methods. Among different "pixel-by-pixel" based methods, the adaptive methods were slightly better than the non-adaptive methods. Notice that since the corrupted pixels were less "spiky" than the previous example, the least squares loss (LS) and the least absolute deviation loss (LAD) performed similarly.

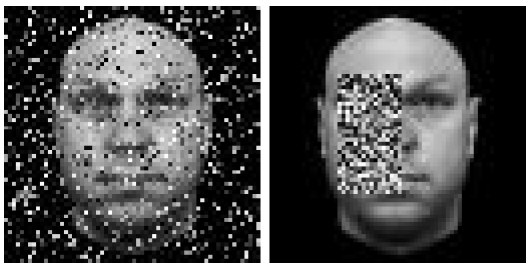

Figure 4.6: Left panel: The corrupted pixels take values between 0 and 255 ("incomplete corruption"). Right panel: The corrupted pixels form a block ("block corruption").

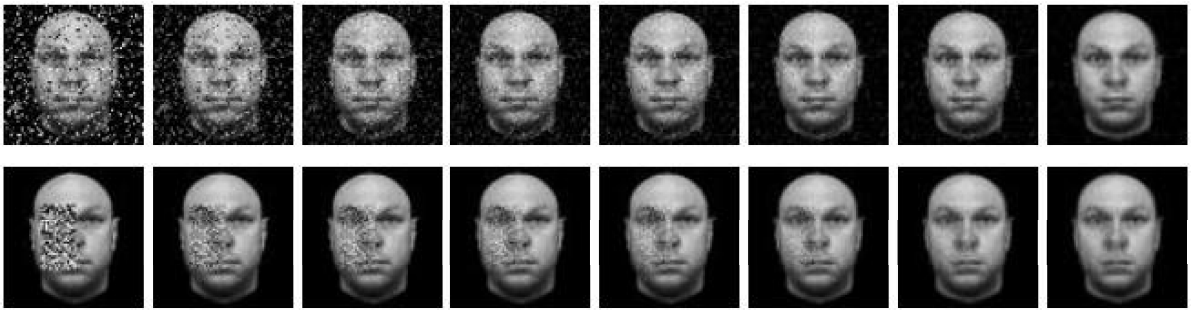

Figure 4.7: The pixel-by-pixel denoised images, i.e., $\boldsymbol{x} + \boldsymbol{\alpha}$, using LAD-Adaptive-LASSO. The first row is for "incompletely corrupted" pixels (left panel in Figure 4.6), and the second row is for "blockly corrupted" pixels (right panel in Figure 4.6). Different images correspond to different values of s (from small to large).

4.5 Conclusion

In this paper, we have proposed PCA-based models that use the LASSO penalty to remove the impulse type noise for digital images. We have developed efficient solution path algorithms for these models, which facilitate the selection of the tuning parameters. We have also presented some promising evidence for our methods on a real world dataset.

CHAPTER V

Financial Market Forecasting Using a Two-Step Kernel Learning Method for the Support Vector Regression

5.1 Background

Forecasting the financial market is a major challenge in both academia and business. Because of the noisy nature, financial time series are among the most difficult signals to forecast, which naturally leads to the debate on market predictability among the academics and market practitioners. The efficient market hypothesis [19, 20] and the random walk hypothesis [46] are major theories in economics and finance. The hypotheses state that the financial market evolves randomly and no excess returns can be made by predicting and timing the market. According to these hypotheses, it is impossible to consistently outperform the market, and the simple "buy-and-hold" is the best investment strategy. Many economists and professional investors, however, dispute these hypotheses, and they believe that the financial market is predictable to some degree. Lo et al. [45] showed the evidence of the predictability for the financial market using technical analysis and computational algorithms. Furthermore, Lo and MacKinlay [44] went through a series of tests and demonstrated the existence of trends and patterns in the financial market. Hirshleifer [35] provided a survey of empirical evidence for the capital market inefficiency and reviewed the explanation for these findings from the behavioral finance perspective.

Researchers in the machine learning and data mining community have also tried to forecast the financial market, using various learning algorithms. The support vector machine (SVM) is one of the tools that have shown promising results [10, 11, 28, 40, 37]. The SVM is a kernel based learning method. Kernel methods embed the original input space $\mathcal{X}$ into a Hilbert space $\mathcal{F}$ via kernel functions and look for linear relations in $\mathcal{F}$, which correspond to nonlinear relations in the original space $\mathcal{X}$.

Kernel functions usually contain tuning parameters. The values of the tuning parameters are crucial for the performance of the kernel methods. If a kernel function is given, the associated parameter values can be tuned by either trial-and-error or heuristic search, e.g., the gradient-based method [12, 39] and the evolutionary method [24, 52]. However, in financial market forecasting, in addition to tune the parameter for a given kernel function, another relevant issue is how to combine different kernel functions in order for the traders to make better decisions. For example, traders often observe information from various sources, including technical indicators, economic indices, political news, etc. A kernel function may be derived from each information source. Then a natural question is how to combine these information (kernels) to help the trader make better forecast decisions. This problem is referred to as kernel learning or kernel selection in the literature. Traditional kernel methods rely on a single kernel function. On the other hand, kernel learning often seeks for a linear combination of candidate kernels. The candidate kernels can be obtained from different kernel functions, parameter settings and/or data sources.

In this paper, we propose a two-step kernel learning method based on the support vector regression (SVR) for financial market forecasting. In the first step, we solve a standard SVR problem using all candidate kernels simultaneously; in the second step, we use scaling parameters, one for each candidate kernel, to achieve kernel learning.

The L_1-norm regularization is used to adjust the scaling parameters. Since the regularization parameter has to be selected carefully, to facilitate parameter tuning, we develop an efficient solution path algorithm, which solves the fitted model for every possible value of the regularization parameter. Then the best parameter value can be identified using a parameter selection criterion or a validation dataset.

Several other kernel learning algorithms have also been proposed recently based on different theories and frameworks. Lanckriet et al. [42], Bach et al. [2] and Ong et al. [53] formulated kernel learning as semidefinite programming (SDP) problems. Fung et al. [25] and Kim et al. [41] proposed methods implementing kernel learning by iteratively solving quadratic programming problems. Want et al. [75] used an orthogonal least square forward selection procedure to achieve a sparse representation of the kernel. Cristianini et al. [15] proposed a quantity measuring the "alignment" between a kernel and the data, and they developed algorithms to adapt the kernel matrix to the learning task. Our method, however, is based on a different framework, which is inspired by the non-negative garrote method [7]. The non-negative garrote uses scaling parameters for variable selection in linear models; we use the similar idea for selecting kernels in our two-step kernel learning method.

The rest of the paper is organized as follows: in Section 2, we describe our method and the solution path algorithm; in Section 3, we compare the performance of different methods based on historical data of the S&P500 and the NASDAQ market indices; and we conclude the paper in Section 4.

5.2 Two-Step Kernel Learning for the Support Vector Regression

5.2.1 Support Vector Regression

We first briefly review the SVR and refer interested readers to [71] and [64] for details. Let $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$ represent the n input vectors, where $\boldsymbol{x}_i \in \mathbb{R}^p$, and

$\{y_1, y_2, ..., y_n\}$ be the corresponding output, where $y_i \in \mathbb{R}$. Let $\mathbf{\Phi} : \mathcal{X} \to \mathcal{F}$ be a mapping function, which transfers a vector in the original space $\mathcal{X}$ into a Hilbert space $\mathcal{F}$. The SVR solves the following problem:

$$\min_{\beta_0, \boldsymbol{\beta}, \boldsymbol{\xi}} \quad \sum_{i=1}^{n} \ell(\xi_i) + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2 \tag{5.1}$$

$$\text{subject to} \quad \xi_i = y_i - \beta_0 - \boldsymbol{\beta}^{\intercal}\mathbf{\Phi}(\boldsymbol{x}_i),\ i = 1, \ldots, n. \tag{5.2}$$

In the above setup, $\ell(\cdot)$ is a loss function and $\lambda \geq 0$ is a regularization parameter, which controls the trade-off between the loss and the complexity of the fitted model. Figure 1 shows some of the popular loss functions that have been used for the SVR: the Laplace, the ε-insensitive, the quadratic and the Huber loss functions. The Laplace loss function is more robust to outliers than the quadratic loss function and it requires fewer parameters than the ε-insensitive and the Huber loss functions. Therefore, in this paper we focus on the Laplace loss function. However, we note that our method can be naturally applied to other loss functions as well after straightforward modifications.

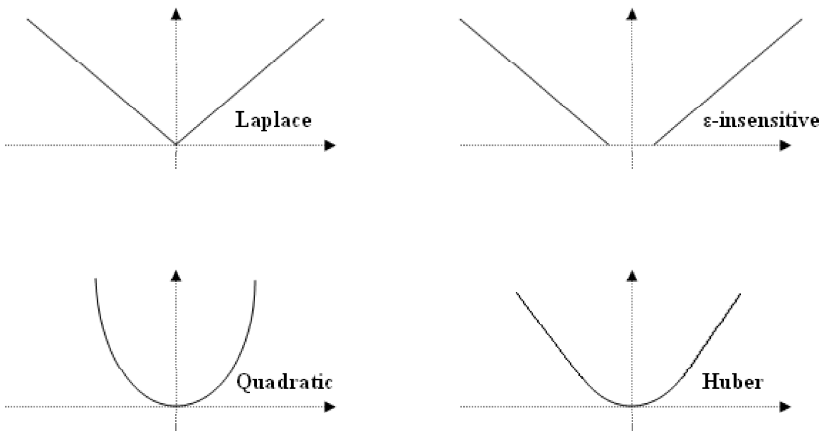

Figure 5.1: Commonly used loss functions in the SVR

With the Laplace loss function, (5.1)–(5.2) can be written as:

$$\begin{aligned} \min_{\beta_0,\boldsymbol{\beta},\boldsymbol{\xi}} \quad & \sum_{i=1}^{n}(\xi_i^+ + \xi_i^-) + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2 \\ \text{subject to} \quad & -\xi_i^- \le y_i - \beta_0 - \boldsymbol{\beta}^\intercal \boldsymbol{\Phi}(\boldsymbol{x}_i) \le \xi_i^+, \\ & \xi_i^+, \xi_i^- \ge 0,\ i = 1, \ldots, n. \end{aligned}$$

The corresponding Lagrangian function is:

$$\begin{aligned} L_P \ = \ & \sum_{i=1}^{n}(\xi_i^+ + \xi_i^-) + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2 + \\ & \sum_{i=1}^{n}\gamma_i^+(y_i - \beta_0 - \boldsymbol{\beta}^\intercal \boldsymbol{\Phi}(\boldsymbol{x}_i) - \xi_i^+) - \\ & \sum_{i=1}^{n}\gamma_i^-(y_i - \beta_0 - \boldsymbol{\beta}^\intercal \boldsymbol{\Phi}(\boldsymbol{x}_i) + \xi_i^-) - \\ & \sum_{i=1}^{n}\rho_i^+\xi_i^+ - \sum_{i=1}^{n}\rho_i^-\xi_i^-, \end{aligned}$$

where $\gamma_i^+, \gamma_i^-, \rho_i^+, \rho_i^-$ are non-negative Lagrangian multipliers. From the Lagrangian function, the original optimization problem can be transformed into its dual problem:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2\lambda}\boldsymbol{\alpha}^\intercal \boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{y}^\intercal \boldsymbol{\alpha} \\ \text{subject to} \quad & \sum_{i=1}^{n}\alpha_i = 0, \\ & -1 \le \alpha_i \le 1,\ i = 1, \ldots, n, \end{aligned} \tag{5.3}$$

where $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ is a symmetric positive definite matrix, with the (i, i') entry $K_{i,i'} = \boldsymbol{\Phi}(\boldsymbol{x}_i)^\intercal \boldsymbol{\Phi}(\boldsymbol{x}_{i'}) = K(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$.

Let $\widehat{\boldsymbol{\alpha}}$ represent its optimal solution, then the primal variables can be recovered as:

$$\widehat{\boldsymbol{\beta}} = \frac{1}{\lambda}\sum_{i=1}^{n}\widehat{\alpha}_i \boldsymbol{\Phi}(\boldsymbol{x}_i).$$

Notice that for any input vector $\boldsymbol{x}$, the fitted model is:

$$\widehat{f}(\boldsymbol{x}) = \widehat{\beta}_0 + \frac{1}{\lambda}\sum_{i=1}^{n}\widehat{\alpha}_i K(\boldsymbol{x}_i, \boldsymbol{x}). \tag{5.4}$$

Therefore, by using the kernel function $K(\cdot,\cdot)$, we can solve the SVR problem without explicitly defining the mapping function $\boldsymbol{\Phi}(\cdot)$. For example, radial basis and polynomial kernels are commonly used kernel functions:

$$\text{Radial} \quad : \quad K(\boldsymbol{x},\boldsymbol{x}') = \exp(-\|\boldsymbol{x}-\boldsymbol{x}'\|^2/\sigma^2) \tag{5.5}$$

$$\text{Polynomial} \quad : \quad K(\boldsymbol{x},\boldsymbol{x}') = (1+\langle \boldsymbol{x},\boldsymbol{x}'\rangle)^d \tag{5.6}$$

where σ and d are user-specified parameters.

5.2.2 Model for a Two-Step Kernel Learning Method

The standard SVR uses only a single mapping function $\boldsymbol{\Phi}(\cdot)$ or its corresponding kernel function $K(\cdot,\cdot)$. In our setting, we consider multiple mappings: $\boldsymbol{\Phi}_1(\cdot),\ldots,\boldsymbol{\Phi}_m(\cdot)$, where m is the number of candidate mappings available. Let $K_1(\cdot,\cdot),\ldots,K_m(\cdot,\cdot)$ be the corresponding kernel functions. We look for a sparse linear combination of these kernels that can be used to predict accurately on future data. For example, in the financial market forecasting setting, $K_1(\cdot,\cdot),\ldots,K_m(\cdot,\cdot)$ may correspond to different information sources. Our kernel learning method results in a new kernel function $K^*(\cdot,\cdot) = s_1K_1(\cdot,\cdot)+\cdots+s_mK_m(\cdot,\cdot)$, where $s_1,\ldots,s_m \geq 0$ are the associated combination coefficients. Our goal is to learn $s_1,\ldots,s_m$, and the method we propose consists of two steps.

In the first step, we solve an SVR problem that uses all candidate mappings (or kernels) simultaneously:

$$\min_{\beta_0,\boldsymbol{\beta}_j,\xi_i^+,\xi_i^-} \quad \sum_{i=1}^{n}(\xi_i^+ + \xi_i^-) + \frac{\lambda}{2}\sum_{j=1}^{m}\|\boldsymbol{\beta}_j\|_2^2 \tag{5.7}$$

$$\text{subject to} \quad -\xi_i^- \leq y_i - \beta_0 - \sum_{j=1}^{m}\boldsymbol{\beta}_j^{\intercal}\boldsymbol{\Phi}_j(\boldsymbol{x}_i) \leq \xi_i^+,$$

$$\xi_i^+,\xi_i^- \geq 0,\ i=1,\ldots,n. \tag{5.8}$$

Similar to the standard SVR, we can transform it into its dual format:

$$\begin{aligned} (5.9) \qquad \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2\lambda}\boldsymbol{\alpha}^{\intercal}\left(\sum_{j=1}^{m}\boldsymbol{K}_j\right)\boldsymbol{\alpha}+\boldsymbol{y}^{\intercal}\boldsymbol{\alpha} \\ \text{subject to} \quad & \sum_{i=1}^{n}\alpha_i=0, \\ & -1\le\alpha_i\le 1,\ i=1,\ldots,n, \end{aligned}$$

where the (i,i') entry of the kernel matrix $\boldsymbol{K}_j\in\mathbb{R}^{n\times n}$ is defined as $K_j(\boldsymbol{x}_i,\boldsymbol{x}_{i'})$. Notice that this problem is exactly the same as the standard SVR, except that the kernel matrix $\boldsymbol{K}$ in (5.3) is replaced by the summation of candidate kernel matrices ($\sum_{j=1}^{m}\boldsymbol{K}_j$). Notice that the above dual is a quadratic programming (QP) problem, and we can solve it efficiently using the sequential minimal optimization (SMO) method [55] or the solution path algorithm [30]. Let $\widehat{\boldsymbol{\alpha}}$ be its optimal solution, then each coefficient vector $\widehat{\boldsymbol{\beta}}_j$ can be written as:

$$(5.10) \qquad \widehat{\boldsymbol{\beta}}_j \;=\; \frac{1}{\lambda}\sum_{i=1}^{n}\widehat{\alpha}_i\boldsymbol{\Phi}_j(\boldsymbol{x}_i),\ j=1,\ldots,m.$$

In the second step, we introduce scaling parameters s_j $(j=1,\ldots,m)$, one for each of the candidate mappings (or kernels), and consider

$$(5.11) \qquad \boldsymbol{\beta}_j^{\text{new}}=s_j\widehat{\boldsymbol{\beta}}_j,\ j=1,\ldots,m.$$

To solve for s_j, we consider to regularize the L_1-norm of s_j [68, 6], i.e.:

$$\begin{aligned} (5.12) \qquad \min_{s_j,\beta_0,\xi_i^+,\xi_i^-} \quad & \sum_{i=1}^{n}\left(\xi_i^+ + \xi_i^-\right) \\ \text{subject to} \quad & -\xi_i^- \le y_i - \beta_0 - \sum_{j=1}^{m} s_j\widehat{\boldsymbol{\beta}}_j^{\intercal}\boldsymbol{\Phi}_j(\boldsymbol{x}_i) \le \xi_i^+, \end{aligned}$$

$$(5.13) \qquad \sum_{j=1}^{m}s_j\le C,$$

$$(5.14) \qquad s_j\ge 0,\ j=1,\ldots,m,$$

$$(5.15) \qquad \xi_i^+,\xi_i^-\ge 0,\ i=1,...,n,$$

where $C \geq 0$ is a regularization parameter. Notice that in the second step, $\widehat{\boldsymbol{\beta}}_j$ are known and fixed, and we are interested in solving for β_0 and s_j. Denote the solution as $\widehat{\beta}_0^*$ and $\widehat{s}_j$ $(j = 1, \ldots, m)$; using (5.10) and (5.11), we then have the final fitted model:

$$\begin{aligned} \widehat{f}(\boldsymbol{x}) &= \widehat{\beta}_0^* + \sum_{j=1}^{m} \widehat{s}_j \widehat{\boldsymbol{\beta}}_j^{\mathsf{T}} \boldsymbol{\Phi}_j(\boldsymbol{x}) \\ &= \widehat{\beta}_0^* + \sum_{j=1}^{m} \widehat{s}_j \left(\frac{1}{\lambda} \sum_{i=1}^{n} \widehat{\alpha}_i K_j(\boldsymbol{x}_i, \boldsymbol{x}) \right). \end{aligned}$$

The final fitted model can be further written as:

$$\widehat{f}(\boldsymbol{x}) = \widehat{\beta}_0^* + \frac{1}{\lambda} \sum_{i=1}^{n} \widehat{\alpha}_i K^*(\boldsymbol{x}_i, \boldsymbol{x}), \tag{5.16}$$

with the corresponding learned new kernel:

$$K^*(\cdot, \cdot) = \sum_{j=1}^{m} \widehat{s}_j K_j(\cdot, \cdot). \tag{5.17}$$

In (5.13), we apply the L_1-norm regularization idea [7, 68, 6], which has been used for variable selection in the setting of linear regression: the singularities of the constraints $\sum_{j=1}^{m} s_j \leq C$ and $s_j \geq 0$ tend to generate a sparse solution for $\boldsymbol{s}$, i.e., some of the s_j's are estimated to be exact zero. The idea is similar to the non-negative garrote method [7], which uses scaling parameters for variable selection in the setting of linear regression. In our method, we use scaling parameters to select kernels. If an $\widehat{s}_j$ is equal to zero, the corresponding kernel is removed from the estimated combined kernel (5.17), hence the method implements automatic kernel selection or kernel learning. The intuition is that if the jth mapping (or kernel) is not important for prediction, the corresponding $\widehat{\boldsymbol{\beta}}_j^{\mathsf{T}} \boldsymbol{\Phi}_j(\boldsymbol{x})$ tends to have a small "magnitude," hence the s_j gets heavily penalized and tends to be estimated as zero, while if the jth mapping (or kernel) is important for the prediction, $\widehat{\boldsymbol{\beta}}_j^{\mathsf{T}} \boldsymbol{\Phi}_j(\boldsymbol{x})$ tends to have a big "magnitude," hence s_j gets lightly penalized and tends to be estimated as non-zero.

5.3 The Solution Path Algorithm

5.3.1 Problem Setup

The regularization parameter C controls the bias-variance trade-off. Increasing the value of C tends to select more kernels and generates a more complicated model, which reduces the bias but increases the variance of the fitted model; and vice versa when decreasing the value of C. Hence the value of C is critical for the performance of the fitted model. In practice, people can pre-specify a number of values for C, solve the optimization problem for each of them, and use a validation set or some model selection criterion to choose the best model. Instead of using such a trial-and-error approach, we develop an efficient solution path algorithm that computes the solutions for all possible values of C, which facilitates the selection for an optimal value of C. Starting from $C = 0$, the algorithm continuously increases C and calculates $\widehat{s}_j$ and $\widehat{\beta}_0^*$ along the path, until the solution does not further change with C. The algorithm takes advantage of the piecewise linearity property of $\widehat{s}_j$ and $\widehat{\beta}_0^*$ with respect to C. We acknowledge that our algorithm is inspired by the LAR/LASSO method [17] and the general piecewise linear solution path strategy of [59]. Similar ideas have also been used in the SVM to generate the optimal solution path of the regularization parameter for over-fitting control [33, 30] or of "hyperparameters" for the kernel function [74]. However, we make a note that the algorithm we develop here is significantly different from the earlier algorithms, because we are now dealing with a non-differentiable loss function *and* a non-differentiable penalty, while all the earlier algorithms deal with cases where either the loss function is differentiable or the penalty is differentiable.

To simplify the notation, we define:

$$\widehat{f}_j(\boldsymbol{x}) = \widehat{\boldsymbol{\beta}}_j^{\mathsf{T}} \boldsymbol{\Phi}_j(\boldsymbol{x}) = \frac{1}{\lambda} \sum_{i=1}^{n} \widehat{\alpha}_i K_j(\boldsymbol{x}_i, \boldsymbol{x}). \tag{5.18}$$

Then the Lagrangian function of (5.12)–(5.15) is:

$$\begin{aligned} L_P &= \sum_{i=1}^{n}(\xi_i^+ + \xi_i^-) + \eta(\sum_{j=1}^{m} s_j - C) - \sum_{j=1}^{m} \varepsilon_j s_j + \\ &\sum_{i=1}^{n} \gamma_i^+ (y_i - \beta_0 - \sum_{j=1}^{m} s_j \widehat{f}_j(\boldsymbol{x_i}) - \xi_i^+) - \\ &\sum_{i=1}^{n} \gamma_i^- (y_i - \beta_0 - \sum_{j=1}^{m} s_j \widehat{f}_j(\boldsymbol{x_i}) + \xi_i^-) - \\ &\sum_{i=1}^{n} \rho_i^+ \xi_i^+ - \sum_{i=1}^{n} \rho_i^- \xi_i^-, \end{aligned}$$

where $\gamma_i^+, \gamma_i^-, \eta, \varepsilon_j, \rho_i^+, \rho_i^- \geq 0$ are Lagrangian multipliers. Let $\gamma_i = \gamma_i^+ - \gamma_i^-$ $(i = 1, ..., n)$, then the Karush-Kuhn-Tucker (KKT) conditions are:

$$\frac{\partial}{\partial \beta_0}: \quad \sum_{i=1}^{n} \gamma_i = 0, \tag{5.19}$$

$$\frac{\partial}{\partial s_j}: \quad -\sum_{i=1}^{n} \gamma_i \widehat{f}_j(\boldsymbol{x_i}) + \eta - \varepsilon_j = 0, \tag{5.20}$$

$$\frac{\partial}{\partial \xi_i^+}: \quad 1 - \gamma_i^+ - \rho_i^+ = 0, \tag{5.21}$$

$$\frac{\partial}{\partial \xi_i^-}: \quad 1 - \gamma_i^- - \rho_i^- = 0. \tag{5.22}$$

Let $f(\boldsymbol{x}_i) = \beta_0 + \sum_{j=1}^{m} s_j \widehat{f}_j(\boldsymbol{x}_i)$, then the complementary slackness conditions are:

$$\gamma_i^+ (y_i - f(\boldsymbol{x}_i) - \xi_i^+) = 0, \tag{5.23}$$

$$\gamma_i^- (y_i - f(\boldsymbol{x}_i) - \xi_i^-) = 0, \tag{5.24}$$

$$\eta(\sum_{j=1}^{m} s_j - C) = 0, \tag{5.25}$$

$$\rho_i^+ \xi_i^+ = 0, \tag{5.26}$$

$$\rho_i^- \xi_i^- = 0, \tag{5.27}$$

$$\varepsilon_j s_j = 0. \tag{5.28}$$

Note that all the Lagrangian multipliers are non-negative. Let $\xi_i = \xi_i^+ - \xi_i^-$ and

$\rho_i = \rho_i^+ - \rho_i^-$, conditions (5.21) to (5.28) lead to the following relationships:

$$\begin{aligned} y_i - f(\boldsymbol{x}_i) > 0: &\qquad \gamma_i = 1, &\qquad \xi_i > 0; \\ y_i - f(\boldsymbol{x}_i) < 0: &\qquad \gamma_i = -1, &\qquad \xi_i < 0; \\ y_i - f(\boldsymbol{x}_i) = 0: &\quad -1 \le \gamma_i \le 1, &\quad \xi_i = 0. \end{aligned}$$

Using these relationships, we can define the following sets:

- $\mathcal{R} = \{i : y_i - f(\boldsymbol{x}_i) > 0,\ \gamma_i = 1\}$ (the Right portion of the Laplace loss function)
- $\mathcal{E} = \{i : y_i - f(\boldsymbol{x}_i) = 0,\ 0 \le \gamma_i \le 1\}$ (the Elbow of the Laplace loss function)
- $\mathcal{L} = \{i : y_i - f(\boldsymbol{x}_i) < 0,\ \gamma_i = -1\}$ (the Left portion of the Laplace loss function)
- $\mathcal{A} = \{j : s_j \neq 0\}$ (the Active set of the kernels)

Note that if $i \in \mathcal{L}$ or $\mathcal{R}$, then γ_i is known; only when $i \in \mathcal{E}$, the γ_i becomes unknown. From the KKT conditions and the derived relationships, we can get two linear systems for the primal and dual variables. We call the following equations as the primal system:

$$y_i - \beta_0 - \sum_{j \in \mathcal{A}} s_j \widehat{f}_j(\boldsymbol{x_i}) = 0,\ i \in \mathcal{E}, \tag{5.29}$$

$$\sum_{j \in \mathcal{A}} s_j = C. \tag{5.30}$$

According to (5.28), if $j \in \mathcal{A}$ ($s_j \neq 0$), then $\varepsilon_j = 0$. So we also have the following dual system:

$$\sum_{i \in \mathcal{E}} \gamma_i + |\mathcal{R}| - |\mathcal{L}| = 0, \tag{5.31}$$

$$\sum_{i=1}^{n} \gamma_i \widehat{f}_j(\boldsymbol{x_i}) - \eta = 0,\ j \in \mathcal{A}. \tag{5.32}$$

For any optimal solution at any C, the primal and the dual systems must hold; and vice versa, if we solve these systems, we obtain the optimal solution. Note that in the

primal system there are $|\mathcal{A}|+1$ unknowns and $|\mathcal{E}|+1$ equations; in the dual system, there are $|\mathcal{E}|+1$ unknowns and $|\mathcal{A}|+1$ equations. Therefore, to satisfy both systems, we must have $|\mathcal{A}| = |\mathcal{E}|$.

5.3.2 Initial Solution

Initially, the algorithm starts from $C = 0$, where $s_j = 0$ and $f(\boldsymbol{x}_i) = \beta_0$. Therefore, (5.12)–(5.15) is reduced to an optimization problem with only one parameter β_0. Since $y_1, ..., y_n$ are real numbers, we assume that their values are distinct. Let $y_{(1)}, y_{(2)}, ..., y_{(n)}$ be the output values in ascending order, then the solution for the initial β_0 is:

$$\widehat{\beta}_0 = y_{(\lceil n/2 \rceil)},$$

where $\lceil \cdot \rceil$ is the ceiling function. According to the definitions of $\mathcal{L}, \mathcal{R}$ and $\mathcal{E}$, we have $|\mathcal{L}|=\lceil n/2 \rceil - 1$, $|\mathcal{R}| = n - \lceil n/2 \rceil$ and $|\mathcal{E}| = 1$. Based on the derived relationships, if $y_i > \widehat{\beta}_0$ then $\gamma_i = 1$; if $y_i < \widehat{\beta}_0$ then $\gamma_i = -1$. Let i^* denote the point in $\mathcal{E}$, i.e., $\mathcal{E} = \{i^*\}$. From (5.19), we get: if n is odd, then $\gamma_{i^*} = 0$; otherwise, $\gamma_{i^*} = 1$.

When C increases from 0, some s_j will become non-zero and join $\mathcal{A}$. If C is increased by an infinitesimal amount: $C = \Delta C \to 0^+$, exactly one s_j will be added into $\mathcal{A}$, because we currently have $|\mathcal{E}|$=1, and $|\mathcal{A}| = |\mathcal{E}|$ has to hold. To determine which s_j will join $\mathcal{A}$, we consider the following problem:

$$\min_{\beta_0, j \in \{1,...,m\}} \sum_{i=1}^{n} |y_i - \beta_0 - \Delta C \cdot \widehat{f}_j(\boldsymbol{x}_i)|. \tag{5.33}$$

According to the derived relationships in "Problem setup", problem (5.33) is equivalent to:

$$\min_{\beta_0, j \in \{1,...,m\}} \sum_{i=1}^{n} \gamma_i (y_i - \beta_0 - \Delta C \cdot \widehat{f}_j(\boldsymbol{x}_i)). \tag{5.34}$$

Let $\mathcal{A} = \{j^*\}$, and j^* is determined by:

$$j^* = \text{argmax}_{j\in\{1,\ldots,m\}} \sum_{i=1}^{n} \gamma_i \widehat{f}_j(\boldsymbol{x}_i).$$

Now, we have initial $\widehat{\beta}_0, \mathcal{A}, \mathcal{L}, \mathcal{R}$ and $\mathcal{E}$, and we also need the initial value for the non-negative dual variable η as $\Delta C \to 0^+$. According to (5.32), it is determined by:

$$\eta = \sum_{i=1}^{n} \gamma_i \widehat{f}_{j^*}(\boldsymbol{x}_i).$$

In the following, we use the superscript ℓ to indicate the iteration number. After the initial setup, $\ell = 0$ and $\mathcal{L}^\ell, \mathcal{R}^\ell, \mathcal{E}^\ell, \mathcal{A}^\ell$, $s_j^\ell, \gamma_i^\ell, \beta_0^\ell$ and η^ℓ are all available.

5.3.3 Solution Path

When C increases by an infinitesimal amount, the sets $\mathcal{L}, \mathcal{R}, \mathcal{E}$ and $\mathcal{A}$ do not change due to their continuity with respect to C. Therefore the structure of the primal system (5.29)–(5.30) does not change. The right derivatives of the primal variables with respect to C can be obtained by:

$$\frac{\Delta\beta_0}{\Delta C} + \sum_{j\in\mathcal{A}} \frac{\Delta s_j}{\Delta C} \widehat{f}_j(\boldsymbol{x}_i) = 0, \ i \in \mathcal{E} \tag{5.35}$$

$$\sum_{j\in\mathcal{A}} \frac{\Delta s_j}{\Delta C} = 1. \tag{5.36}$$

Since $|\mathcal{E}| = |\mathcal{A}|$, $\frac{\Delta\beta_0}{\Delta C}$ and $\frac{\Delta s_j}{\Delta C}$ are constant and can be uniquely determined, assuming the linear system is non-singular. Therefore, if C increases by a small enough amount, the primal variables and fitted values are linear functions of C:

$$\begin{aligned}
\beta_0 &= \beta_0^\ell + \frac{\Delta\beta_0}{\Delta C}(C - C^\ell), \\
s_j &= s_j^\ell + \frac{\Delta s_j}{\Delta C}(C - C^\ell), \\
f(\boldsymbol{x}_i) &= f^\ell(\boldsymbol{x}_i) + \frac{\Delta\beta_0}{\Delta C}(C - C^\ell) + \sum_{j\in\mathcal{A}} \frac{\Delta s_j}{\Delta C} \widehat{f}_j(\boldsymbol{x}_i)(C - C^\ell).
\end{aligned}$$

If the increase in C is large enough, some of the sets $\mathcal{L}, \mathcal{R}, \mathcal{E}$ and $\mathcal{A}$ may change. Specifically, one of the following two *events* may occur in the primal system:

- a point leaves $\mathcal{L}$ or $\mathcal{R}$ and joins $\mathcal{E}$, i.e., its residual $y_i - f(\boldsymbol{x}_i)$ changes from non-zero to zero; or
- an active s_j becomes inactive, i.e., s_j reduces from a positive value to zero.

To determine ΔC for the first *event*, we calculate for every $i \notin \mathcal{E}$:

$$\Delta\widehat{C}_i = \max\left\{-\frac{f^\ell(\boldsymbol{x}_i)}{\frac{\Delta\beta_0}{\Delta C} + \sum_{j\in\mathcal{A}} \frac{\Delta s_j}{\Delta C}\widehat{f}_j(\boldsymbol{x}_i)}, 0\right\}.$$

For the second *event*, we calculate

$$\Delta\widetilde{C_j} = \max\left\{-s_j^\ell/\frac{\Delta s_j}{\Delta C}, 0\right\}, \text{ for } j \in \mathcal{A}.$$

Therefore, the overall ΔC is

$$\Delta C \;=\; \min\{\, \Delta\widehat{C}_i \;(i \notin \mathcal{E}, \Delta\widehat{C}_i > 0),\; \Delta\widetilde{C}_j \;(j \in \mathcal{A}, \Delta\widetilde{C}_j > 0) \,\},$$

and we get the updated variable $s_j^{\ell+1}$, $\beta_0^{\ell+1}$ and $C^{\ell+1}$. If the first *event* occurs, $|\mathcal{E}|$ will increase by 1; if the second *event* occurs, $|\mathcal{A}|$ will decrease by 1. Therefore, it always leads to $|\mathcal{E}| = |\mathcal{A}| + 1$ after an *event* occurs. Since the relation $|\mathcal{E}| = |\mathcal{A}|$ does not hold, we have to restore it by taking one of the two *actions*:

- removing a point from $\mathcal{E}$; or
- adding a new s_j into $\mathcal{A}$.

Solving the dual system helps us determine which *action* to take.

Since we now have $|\mathcal{E}| = |\mathcal{A}| + 1$, in the dual system (5.31)–(5.32), there is one more unknown than the number of equations. In other words, there is one degree of freedom in the system, and we can express γ_i $(i \in \mathcal{E})$ as a linear function of η. The derivatives of γ_i with respect to η can be solved by the following equations:

$$\sum_{i\in\mathcal{E}} \frac{\Delta\gamma_i}{\Delta\eta} \;=\; 0, \tag{5.37}$$

$$\sum_{i\in\mathcal{E}} \frac{\Delta\gamma_i}{\Delta\eta}\widehat{f}_j(\boldsymbol{x}_i) \;=\; 1,\; j \in \mathcal{A}. \tag{5.38}$$

In (5.37)–(5.38), there are $|\mathcal{E}|$ unknowns and $|\mathcal{A}|+1$ equations, so the values of $\frac{\Delta\gamma_i}{\Delta\eta}$ ($i \in \mathcal{E}$) can be uniquely determined. To determine the $\Delta\eta$ for taking the first *action* (removing a point from $\mathcal{E}$), we calculate for every $i \in \mathcal{E}$:

$$\Delta\widehat{\eta}_i = \begin{cases} -\gamma_i^\ell / \frac{\Delta\gamma_i}{\Delta\eta}, & \text{if } \frac{\Delta\gamma_i}{\Delta\eta} > 0, \\ (1-\gamma_i^\ell)/\frac{\Delta\gamma_i}{\Delta\eta}, & \text{if } \frac{\Delta\gamma_i}{\Delta\eta} < 0, \\ -\infty, & \text{otherwise.} \end{cases}$$

On the other hand, since equation (5.32) must hold for $j \in \mathcal{A}$, we can determine the $\Delta\eta$ for taking the second *action* (adding a new s_j into $\mathcal{A}$) by calculating:

$$\Delta\widetilde{\eta}_j = \begin{cases} \frac{\eta^\ell - \sum_{i=1}^n \gamma_i^\ell \widehat{f}_j(\boldsymbol{x}_i)}{\sum_{i\in\mathcal{E}} \frac{\Delta\gamma_i}{\Delta\eta}\widehat{f}_j(\boldsymbol{x}_i) - 1}, & \text{if } \sum_{i\in\mathcal{E}} \frac{\Delta\gamma_i}{\Delta\eta}\widehat{f}_j(\boldsymbol{x}_i) - 1 < 0, \\ -\infty, & \text{otherwise,} \end{cases}$$

for every $j \notin \mathcal{A}$. When η reduces to $\eta^\ell + \Delta\widetilde{\eta}_j$, s_j will join $\mathcal{A}$. Since η can only decrease, the overall $\Delta\eta$ is

$$\Delta\eta = \max\{\ \Delta\widehat{\eta}_i (i \in \mathcal{E}), \Delta\widetilde{\eta}_j (j \notin \mathcal{A})\ \}, \tag{5.39}$$

and we get the updated dual variables $\gamma_i^{\ell+1}$, $\eta^{\ell+1}$ and the sets $\mathcal{L}^{\ell+1}$, $\mathcal{R}^{\ell+1}$, $\mathcal{E}^{\ell+1}$ and $\mathcal{A}^{\ell+1}$.

Once an *action* is taken, we restore $|\mathcal{E}| = |\mathcal{A}|$. The algorithm keeps increasing C and alternates between reaching the next *event* and taking an *action*, until η reduces to 0.

Between any two consecutive events the optimal solutions are linear in C; after an *event* occurs, the derivatives with respect to C are changed. Therefore, the solution path is piecewise linear in C, and each *event* corresponds to a kink on the path. Figure 2 shows the solution path for a randomly generated data set with 50 samples and 10 candidate radial basis kernels. For this particular data set, it takes 48 iterations to compute the entire solution path. We can see that the 10 candidate kernels are

selected one by one as the algorithm proceeds (or the value of C increases). The algorithm ends up with $\widehat{s}_j = 1$ for all j, which corresponds to the scenario where the constraint $\sum_{j=1}^{m} s_j \leq C$ is fully released.

For clarity, we summarize our path algorithm in Table 1.

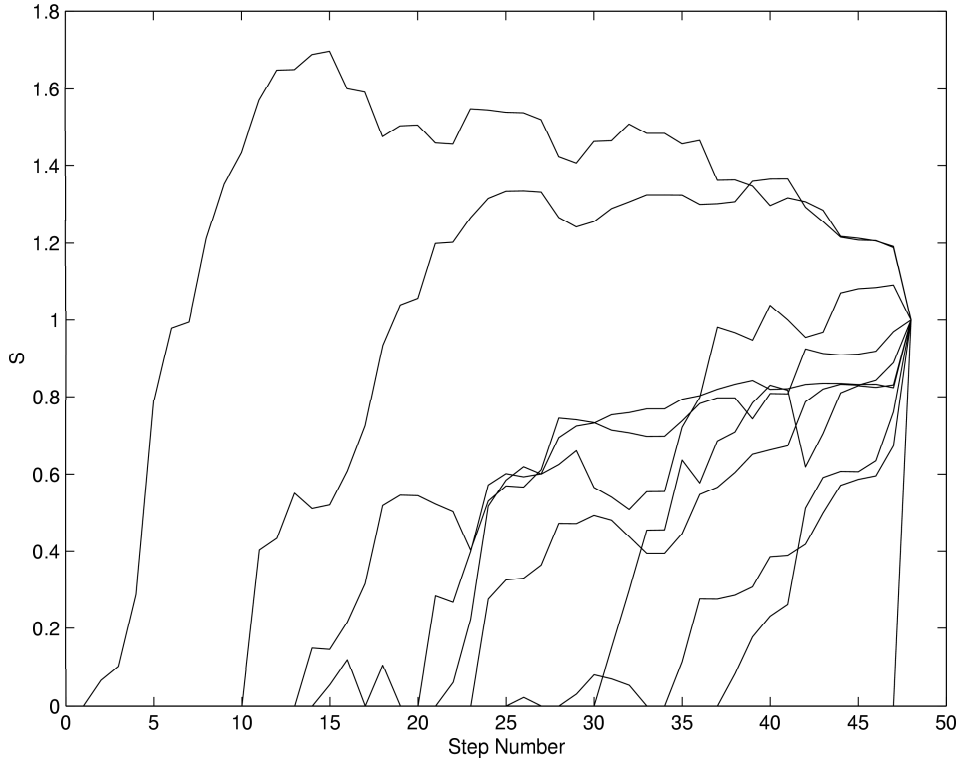

Figure 5.2: Piecewise linear solution path for a randomly generated data set

5.3.4 Computational Complexity

Our kernel learning method consists of two steps: solving a QP problem and running our solution path algorithm. The first problem has been extensively addressed in the literature, e.g., the SMO algorithm [55, 22] and the solution path algorithm [30]. In the second step, our solution path algorithm solves a series of LP problems. The major computation at each iteration is to solve the primal system (5.35)–(5.36) and the dual system (5.37)–(5.38). Since $|\mathcal{A}|$ is upper bounded by m, the computational

INITIALIZATION: Calculate β_0^0 and η^0; determine $\mathcal{A}^0, \mathcal{L}^0, \mathcal{R}^0, \mathcal{E}^0$ and set $\ell = 0$;
STEP 1: Solve the primal system (5.35)-(5.36);
STEP 2: Calculate ΔC ; identify the next event for the primal system and update the primal variables;
STEP 3: Solve the dual system (5.37)-(5.38);
STEP 4: Calculate $\Delta\eta$; identify the next event for the dual system and update the dual variables;
STEP 5: If the stopping criterion is met, then stop the algorithm;
STEP 6: Otherwise, let $\ell = \ell + 1$ and goto Step 1.

Table 5.1: Outline of the Algorithm

complexity seems to be $O(m^3)$ for each iteration. However, between any two consecutive iterations, only one element in the sets $\mathcal{L}$, $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{A}$ tends to get changed, so using inverse updating and downdating the computational complexity can be reduced to $O(m^2)$. It is difficult to predict the total number of iterations for completing the algorithm, but our experience suggests that $O(\max(n, m))$ is a reasonable estimate. The heuristic is that the algorithm needs m iterations to include all the s_j's into $\mathcal{A}$ and n iterations to move all the points to $\mathcal{E}$. Therefore, the overall computational cost of our solution path algorithm is approximately $O(\max(n, m)m^2)$.

5.4 Financial Market Forecasting

In this section, we apply our two-step kernel learning method to forecast the financial market and compare its performance to three other methods, specifically, the buy-and-hold strategy, the standard SVR and the kernel learning method proposed by Lanckriet et al. [42]. We chose to compare with [42] simply because none of the current kernel selection methods have their code publicly available, and the method in [42] seems to be the most convenient for implementation (and it also works well in practical problems). The experiments were based on historical prices of the S&P500 and the NASDAQ composite indices. All these methods were tested on 2,500 trading

days from 11/1997 to 10/2007, which covers for around 10 years.

From the daily closing prices of the S&P500 and the NASDAQ indices, we first calculated the 5 (weekly), 10 (biweekly), 20 (monthly), and 50-day (quarterly) moving averages, which are widely used technical indicators by traders. To illustrate our preprocessing procedure, let P_t denote the closing price on day t and $A_{t,T}$ be the T-day moving average on day t, where $A_{t,T}$ is calculated as:

$$A_{t,T} = \frac{1}{T} \sum_{k=t-T+1}^{t} P_k, \text{ for } T = 5, 10, 20, 50.$$

Notice that P_t can also be considered as $A_{t,1}$. Then we calculated the moving average log-return $R_{t,T}$ (including the daily log-return) for each day t:

$$R_{t,T} = \ln(A_{t,T}) - \ln(A_{t-T,T}), \text{ for } T = 1, 5, 10, 20, 50.$$

We chose to use the log-return, which is a more widely used measure in finance than the raw return, because the return tends to have a log-normal distribution, while the log-return has an approximate normal distribution. Since the log-return was used in our experiments, the cumulative log-return over a certain period was simply the summation of the daily log-returns over this period. Overall we had 5 time series: 1) $R_{t,1}$, 2) $R_{t,5}$, 3) $R_{t,10}$, 4) $R_{t,20}$, and 5) $R_{t,50}$. The output variable was $R_{t+1,1}$, the next day log-return.

Then we constructed the input features. For the jth time series, we extracted p_j data points to construct the input features. Specifically, let

$$\boldsymbol{x}_t^j = [R_{t-(p_j-1)T_j,T_j}, R_{t-(p_j-2)T_j,T_j}, \cdots, R_{t,T_j}], \text{ for } j = 1, \ldots, 5,$$

where $T_1 = 1$, $T_2 = 5$, $T_3 = 10$, $T_4 = 20$ and $T_5 = 50$. Notice that as T_j increases, the corresponding time series R_{t,T_j} becomes smoother, hence fewer data points were needed for $\boldsymbol{x}_t^j$. Specifically, we let $p_1 = 10$, $p_2 = 8$, $p_3 = 6$, $p_4 = 4$ and $p_5 = 4$. The

overall input features for day t were then:

$$\boldsymbol{x}_t = [\boldsymbol{x}_t^1, \boldsymbol{x}_t^2, \boldsymbol{x}_t^3, \boldsymbol{x}_t^4, \boldsymbol{x}_t^5].$$

The intuition is that different $\boldsymbol{x}_t^j$ represent different characteristics underlying the financial market. For example, $\boldsymbol{x}_t^1$ and $\boldsymbol{x}_t^2$ capture the short-term (daily and weekly) behaviors of the market, while $\boldsymbol{x}_t^4$ and $\boldsymbol{x}_t^5$ capture the long-term (monthly and quarterly) trends.

It is not clear a priori which features are important for predicting the next day return, neither how they should be combined to predict. We treated these features separately by applying separate kernels to them and used our method to automatically select and combine them to help us decide whether we should "long" or "short" for the next day. Since the radial basis kernel is the most popular kernel that gets used in practical problems and it has also been proved to be useful in financial market forecasting, for example, see [11, 40] and [37], we chose to use the radial basis kernel (5.5). The radial basis kernel contains a scaling parameter σ^2, which controls the "effective support" of the kernel function. To illustrate our method, we considered two possible values for σ^2. Specifically, for each feature $\boldsymbol{x}_t^j$ (which is a p_j-vector, $j = 1, \ldots, 5$), we considered two candidate radial basis kernel functions:

$$K_j(\boldsymbol{x}_t^j, \boldsymbol{x}_{t'}^j) = \exp(-\|\boldsymbol{x}_t^j - \boldsymbol{x}_{t'}^j\|^2/m_j^2) \text{ and } K_j(\boldsymbol{x}_t^j, \boldsymbol{x}_{t'}^j) = \exp(-\|\boldsymbol{x}_t^j - \boldsymbol{x}_{t'}^j\|^2/(10m_j^2)),$$

where m_j^2 is the median value of $\|\boldsymbol{x}_t^j - \bar{\boldsymbol{x}}^j\|^2$. Therefore, there were a total of $5 \times 2 = 10$ candidate kernel functions.

In evaluating the fitted model, we used a simple strategy to invest: let $\widehat{f}_t$ be the forecasted next day log-return on day t, if $\widehat{f}_t \geq 0$, we buy at the closing price on day t; otherwise, we short it. Then the log-return, $\widehat{R}_t$, associated with day t based on our

strategy, can be computed as:

$$\widehat{R}_t = \begin{cases} |R_{t+1,1}|, & \text{if } R_{t+1,1} \cdot \widehat{f}_t \geq 0; \\ -|R_{t+1,1}|, & \text{otherwise.} \end{cases}$$

Figure 3 illustrates how we selected the tuning parameters λ in (5.9) and C in (5.12) for our two-step procedure and how we evaluated the fitted model $\widehat{f}$ from our two-step procedure. Starting from 11/1997, we firstly trained models (via the solution path algorithm) on 150 consecutive data points (the training set); secondly, we applied the obtained models on the next 10 data points (the validation set) and selected the values of λ and C that had the best performance; thirdly, we combined 140 data points of the training set and all 10 data points of the validation set into a new set, which we call the "true training set," and trained the final model using the selected values of λ and C; lastly, we applied the final model on another 10 data points (the test set) and recorded its performance. We then shifted forward for 10 data points (or 10 trading days) and repeated the same procedure. We repeated this for 250 times, which correspond to 2,500 trading days until 10/2007. Notice that there was no overlap between any two test sets.

For the standard SVR, we used $\boldsymbol{x}_t$ as the input feature, which contains information from different time series without differentiating them, and considered one single radial basis kernel $K(\boldsymbol{x}_t, \boldsymbol{x}_{t'}) = \exp(-\|\boldsymbol{x}_t - \boldsymbol{x}_{t'}\|^2/\sigma^2)$. There were two tuning parameters, the λ as in equation (5.3) and the σ. For λ, we used the solution path algorithm that was developed in Gunter and Zhu (2006), which solves solutions for all possible values of λ, and the optimal λ was selected using a validation set. For σ, we first computed the median of $\|\boldsymbol{x}_t - \bar{\boldsymbol{x}}\|$, where $\boldsymbol{x}_t$ is the input feature vector of the tth training observation and $\bar{\boldsymbol{x}}$ is the corresponding mean. Denote it as $m_{\boldsymbol{x}}$. Then we searched the optimal σ over $(0.2m_{\boldsymbol{x}}, 0.4m_{\boldsymbol{x}}, \ldots, 2m_{\boldsymbol{x}})$. For the kernel learning method

by Lanckriet et al. [42], we selected the tuning parameters in a similar fashion. All selected final models were evaluated on the same test sets.

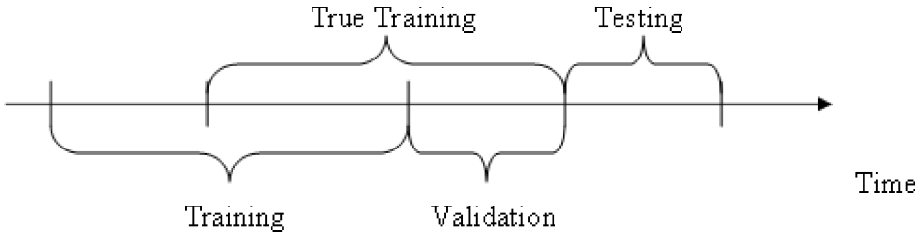

Figure 5.3: Training, validation and test sets

Since there are 5 trading days per week, the summation of log-returns on each test set represents the biweekly log-return. Table 2 shows the average biweekly log-returns among 250 test sets on the two markets from four different methods. The average biweekly log-returns of our method are 4.3 and 5.6 times of the buy-and-hold strategy on the S&P500 and the NASDAQ, respectively. We also used the paired t-test to compare the log-returns of our method against the buy-and-hold strategy. The p-value for the S&P500 is 0.03, and the p-value for the NASDAQ is 0.05. The standard SVR method and the kernel learning method by Lanckriet et al. [42] also performed better than the buy-and-hold strategy, but the improvements are not statistically significant based on the paired t-test. Figures 4 and 5 show the cumulative log-returns for all these methods on the two markets. We can see that the log-returns of our method have a consistently increasing trend over the 2,500 trading day period.

To further investigate the effects of different characteristics of the financial time series, we also recorded how each kernel was selected. Table 3 summarizes the results for our method and the kernel learning method in [42] on the S&P 500 index. As we can see, the two methods behaved very differently on this particular dataset. For our two-step kernel learning method, overall, the short-term (daily and weekly) trends

Method	Average Biweekly Log-Return (%)	p-value
S&P500:		
Buy-and-Hold	0.198 (0.217)	–
SVR	0.278 (0.216)	0.814
Kernel Learning	0.529 (0.231)	0.338
Two-Step Procedure	0.855 (0.231)	0.0296
NASDAQ:		
Buy-and-Hold	0.202 (0.351)	–
SVR	0.608 (0.374)	0.448
Kernel Learning	0.382 (0.339)	0.740
Two-Step Procedure	1.126 (0.361)	0.0504

Table 5.2: Performance on the S&P500 and NASDAQ indices. "Average biweekly return" is the average of log-returns over 250 different test sets, and the numbers in the parentheses are the corresponding standard errors. "p-values" were computed from the paired t-test against the buy-and-hold strategy.

had a bigger impact than the long-term (monthly and quarterly) trends in predicting the next day return, however, the long-term trends also played a significant role. On the other hand, the kernel learning method by Lanckriet et al. [42] tended to generate a more sparse model: Most of the time, it only selected the kernel that contained the daily information. Regarding the scaling parameter of the radial basis kernel, the value m_j^2 seemed to be preferred over $10m_j^2$. The results on the NASDAQ index are similar.

5.5 Discussion

In this paper, we have proposed a kernel learning method based on the support vector regression for financial market forecasting. Our method consists of two steps, where a similar idea was used by non-negative garrote [7] for variable selection in the setting of linear regression. In the first step, we fit a standard SVR using all candidate kernels; in the second step, we use scaling parameters, one for each candidate kernel, and search for a sparse linear combination of the kernels via the L_1-norm regularization. For the second step, we have also developed an efficient solution path algorithm that solves the optimal solutions for all possible values of the regularization

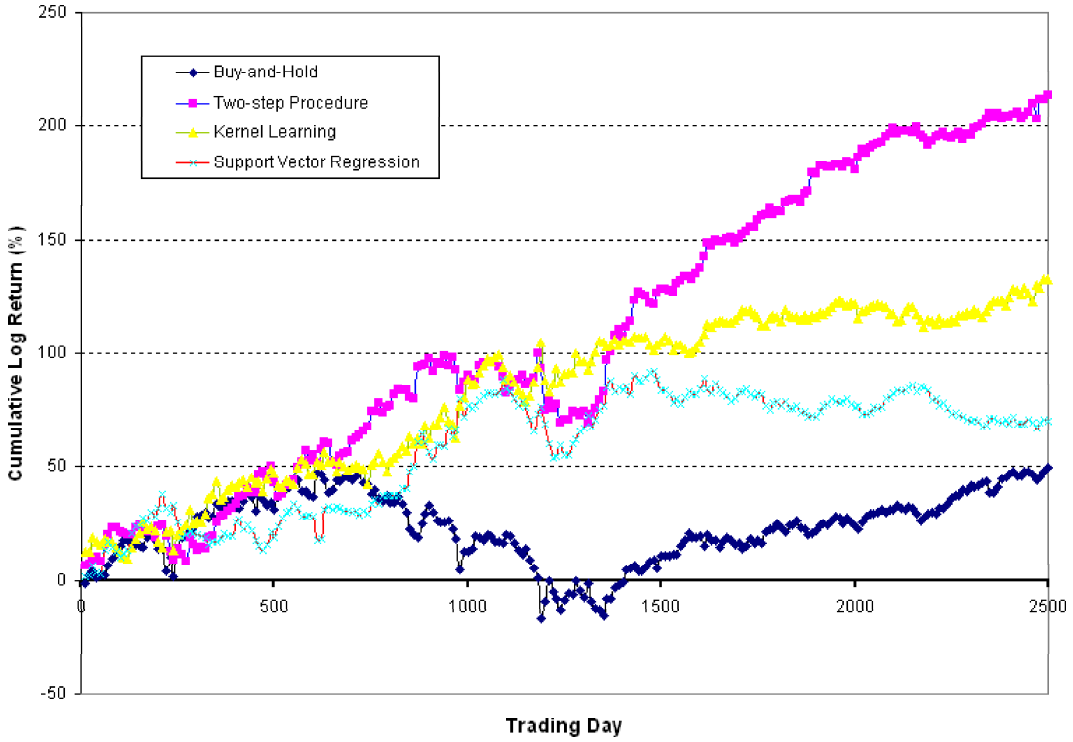

Figure 5.4: Cumulative log-returns of the four methods on the S&P500 index

parameter.

Our two-step kernel learning method shows promising results in forecasting the financial market. The trading strategy based on our method consistently outperforms the market, and the excess returns are statistically significant. Readers might be curious why we are publishing this work instead of making money for ourselves from the financial market. The main reason is that in the current experiment, transaction costs have not been considered. Suppose that the transaction cost takes off 0.1% of the capital for each trade, then overall it will eliminate 250% of the return over the 2,500-day trading period, which makes our method unprofitable. One may also argue that the transaction cost might become ignorable if our trading volume is large enough. However, large orders can also have impact on the market price and we do

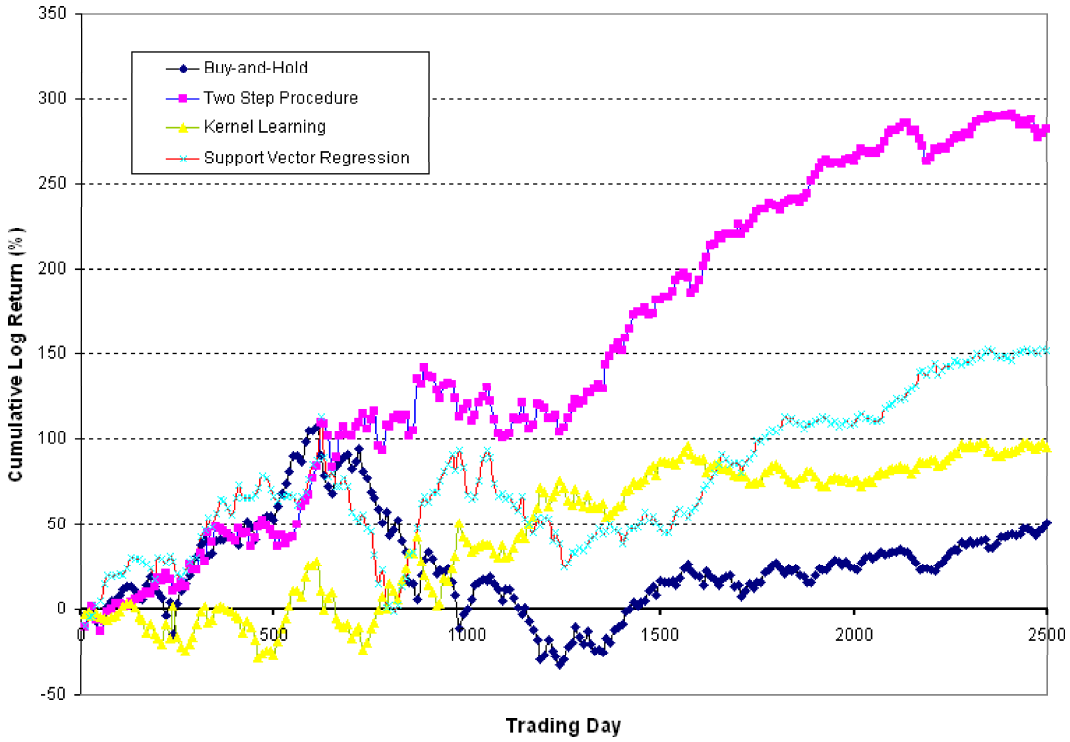

Figure 5.5: Cumulative log-returns of the four methods on the NASDAQ index

not have related data to discover the range of order sizes, within which we do not change the market behavior at its closing time. Although our method and the current trading strategy can not bring up economic benefits, they did demonstrate that the financial market is not completely efficient and is predictable to some degree, which is aligned with the conclusions made by many previous work.

Kernel	Two-Step Procedure: s_j	Frequency	Kernel Learning: s_j	Frequency
$K_1^{\mathbf{x}_1}$	1.2500 (0.2373)	250/250	0.9837 (0.0269)	250/250
$K_2^{\mathbf{x}_1}$	0.3710 (0.4015)	154/250	0.0001 (0.0013)	2/250
$K_1^{\mathbf{x}_2}$	0.7710 (0.4361)	218/250	0.0144 (0.0267)	82/250
$K_2^{\mathbf{x}_2}$	0.2386 (0.3884)	86/250	0.0000 (0.0000)	0/250
$K_1^{\mathbf{x}_3}$	0.5749 (0.5506)	157/250	0.0015 (0.0053)	24/250
$K_2^{\mathbf{x}_3}$	0.1582 (0.3474)	49/250	0.0000 (0.0000)	0/250
$K_1^{\mathbf{x}_4}$	0.3917 (0.5591)	97/250	0.0001 (0.0005)	5/250
$K_2^{\mathbf{x}_4}$	0.1682 (0.3722)	47/250	0.0000 (0.0000)	0/250
$K_1^{\mathbf{x}_5}$	0.2935 (0.4950)	74/250	0.0002 (0.0018)	4/250
$K_2^{\mathbf{x}_5}$	0.1745 (0.3722)	51/250	0.0000 (0.0000)	0/250

Table 5.3: Kernel selection on the S&P500 index. Each row corresponds to a candidate kernel; the subscript "1" corresponds to $\sigma^2 = m_j^2$ and the subscript "2" corresponds to $\sigma^2 = 10m_j^2$; the superscript indicates the part of the input features that the kernel is based on. The second column contains the average estimated combining coefficient s_j over the 250 different trials, and the numbers in the parentheses are the corresponding standard errors. The third column records the selection frequency for each kernel out of 250 trials.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] 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. In *Proceedings of National Academy of Sciences of the United States of America*, pages 6745–6750, 1999.

[2] F. Bach, G. Lanckriet, and M. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the 21st International Conference on Machine Learning*, page 6, 2004.

[3] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:509–522, 2002.

[4] P Billingsley. *Statistical Inference for Markov Processes*. University of Chicago Press, Chicago, 1961.

[5] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, 1992.

[6] P. Bradley and O. Mangasarian. Feature selection via concave minimization and support vector machines. In *International Conference on Machine Learning*, 1998.

[7] L. Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 37:373–384, 1995.

[8] A. Bruce and H. Gao. *Applied Wavelet Analysis with S-PLUS*. Springer Verlag, 1996.

[9] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[10] L. Cao and F. Tay. Financial forecasting using support vector machines. *Neural Computing & Applications*, 10:184–192, 2001.

[11] L. Cao and F. Tay. Support vector machine with adaptive parameters in financial time series forecasting. *IEEE Transactions on Neural Networks*, 14:1506–1518, 2003.

[12] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46:131–159, 2002.

[13] S. Chen, D. Donoho, and M. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal of Scientific Computing*, 20:33–61, 1998.

[14] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[15] N. Cristianini, J. Kandola, A. Elisseeff, and J.S. Taylor. On kernel target alignment. In *Innovations in Machine Learning: Theory and Applications*. Springer Verlag, 2006.

[16] D. Donoho, I. Johnstone, G. Kerkyachairan, and D. Picard. Wavelet shrinkage: asymptopia? (with discussion). *Journal of the Royal Statistical Society: Series B*, 57:201–337, 1995.

[17] B. Efron, I. Johnstone, T. Hastie, and R. Tibshirani. Least angle regression. *The Annals of Statistics*, 32:407–499, 2004.

[18] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. In *Advances in Large Margin Classifiers*. MIT Press, 1999.

[19] E. Fama. Efficient capital markets: A review of theory and empirical work. *Journal of Finance*, 25:383–417, 1970.

[20] E. Fama. Efficient capital markets: II. *Journal of Finance*, 46:1575–1617, 1991.

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

[22] R.E. Fan, P.H. Chen, and C.J. Lin. Working set seleciton using second order information for training support vector machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

[23] J. Friedman, T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. Discussion of "consistency in boosting" by w. jiang, g. lugosi, n. vayatis and t. zhang. *Annals of Statistics*, 32, 2004.

[24] F. Friedrichs and C. Igel. Evolutionary tuning of multiple svm parameters. *Neurocomputing*, 64:107–117, 2005.

[25] G. Fung, M. Dundar, J. Bi, and B. Rao. A fast iterative algorithm for fisher discriminant using heterogeneous kernels. In *Proceedings of the 21st International Conference on Machine Learning*, page 40, 2004.

[26] E. I. George and D. P. Foster. Calibration and empirical Bayes variable selection. *Biometrika*, 87:731–747, 2000.

[27] E. I. George and R. E. McCulloch. Variable selection via Gibbs sampling. *Journal of the American Statistical Association*, 88:881–889, 1993.

[28] T.V. Gestel, J. Suykens, D.E. Baestaens, A. Lambrechts, G. Lanckriet, B. Vandaele, B.D. Moor, and J. Vandewalle. Financial time series prediction using least squares support vector machines within the evidence framework. *IEEE Transactions on Neural Networks*, 12:809–821, 2001.

[29] T. Golub, D. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. Mesirov, H. Coller, M. Loh, J. Downing, and M. Caligiuri. Molecular classification of cancer: class discovery and class predicion by gene expression monitoring. *Science*, 286:531–536, 2000.

[30] L. Gunter and J. Zhu. Computing the solution path for the regularized support vector regression. In *Advances in Neural Information Processing Systems*, pages 483–490, 2006.

[31] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 2002.

[32] D. Harwood, M. Subbararao, H. Hakalahti, and L. Davis. A new class of edge preserving smoothing filters. *Pattern Recognition Letters*, 6:155–162, 1987.

[33] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5:1391–1415, 2004.

[34] T. Hastie, R. Tibshirani, and J. Friedman, editors. *The Elements of Statistical Learning*. Springer-Verlag, 2001.

[35] D. Hirshleifer. Investor psychology and asset pricing. *The Journal of Finance*, 4:1533–1597, 2001.

[36] A. Hoerl and R. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970.

[37] W. Huang, Y. Nakamori, and S.Y. Wang. Forecasting stock market movement direction with using support vector machine. *Computers and Operations Research*, 32:2513–2522, 2005.

[38] A. Hyvarinen, P. Hoyer, and E. Oja. Sparse coding shrinkage for image denoising. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 2, pages 859–864, 1998.

[39] S. Keerthi, V. Sindhwani, and O. Chapelle. An efficient method for gradient-based adaptation of hyperparameters in svm models. In *Advances in Neural Information Processing Systems*, pages 217–224, 2007.

[40] K. Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55:307–319, 2003.

[41] S. Kim, A. Magnani, and S. Boyd. Optimal kernel selection in kernel fisher discriminant analysis. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 465–472, 2006.

[42] G. Lanckriet, N. Cristianini, P. Bartlett, L. Ghaoui, and M. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[43] Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik. Learning algorithms for classification: a comparison on handwritten digit recognition. *Neural Networks: The Statistical Mechanics Perspective*, 1995.

[44] A.W. Lo and C. MacKinlay. *A Non-random Walk Down Wall Street.* Princeton University Press, 2001.

[45] A.W. Lo, H. Mamaysky, and J. Wang. Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. *Journal of Finance*, 55(4):1705–1765, 2000.

[46] B. Malkiel. *A Random Walk Down WAll Street.* W. W. Norton & Company, Inc., 1973.

[47] S. Mallat and Z. Zhang. Matching pursuit in a time-frequency dictionary. *IEEE Transactions on Signal Processing*, 41:3397–3415, 1993.

[48] S. Mika, B. Scholkopf, A. Smola, K. Muller, M. Scholz, and G. Ratsch. Kernel PCA and denoising in feature spaces. In *Advances in Neural Information Processing Systems*, volume 11, pages 537–542, 1999.

[49] S. Mukherjee, P. Tamayo, D. Slonim, A. Verri, T. Golub, J. Mesirov, and T. Poggio. Support vector machine classification of microarray data. *Technical Report, AI Memo 1677, MIT*, 1999.

[50] S. Mukherjee, P. Tamayo, D. Slonim, A. Verri, T. Golub, J.P. Mesirov, and T. Poggio. Support vector machine classification of microarray data. Technical report, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2000.

[51] A. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *International Conference on Machine Learning.* Morgan Kaufmann, 2004.

[52] H.N. Nguyen, S.Y. Ohn, and W.J. Choi. Combined kernel function for support vector machine and learning method based on evolutionary algorithm. In *Proceedings of the 11th International Conference on Neural Information Processing*, pages 1273–1278, 2007.

[53] C.S. Ong, A.J. Smola, and R.C. Williamson. Learning the kernel with hyperkernels. *Journal of Machine Learning Research*, 6:1043–1071, 2005.

[54] S. Palmer. *Vision Science – Photons to Phenomenology.* MIT Press, 1999.

[55] J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning.* MIT Press, 1999.

[56] A. Potti, S. Mukherjee, R. Petersen, H. Dressman, A. Bild, J. Koontz, R. Kratzke, M. Watson, M. Kelley, G. Ginsburg, M. West, D. Harpole, and J. Nevins. A genomic strategy to refine prognosis in early-stage non-small-cell lung cancer. *New England Journal of Medicine*, 355:570–580, 2006.

[57] S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. Mesirov, T. Poggio, W. Gerald, M. Loda, E. Lander, and T. Golub. Multiclass cancer diagnosis using tumor gene expression signature. In *Proceedings of National Academy of Sciences of the United States of America*, 2001.

[58] S. Rosset and J. Zhu. Piecewise linear regularized solution paths. *Technical Report, Department of Statistics, University of Michigan*, 2004.

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

[60] 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.

[61] M. Schulze and J. Pearce. A morphology-based filter structure for edge-enhancing smoothing. In *Proceedings of the IEEE International Conference on Image Processing*, pages 530–534, 1994.

[62] L. Shang, D. Huang, C. Zheng, and Z. Sun. Noise removal using a novel non-negative sparse coding shrinkage technique. *Neurocomputing*, 69:874–877, 2006.

[63] X. Shen and J. Ye. Adaptive model selection. *Journal of the American Statistical Association*, 97:210–221, 2002.

[64] A. Smola and B. Schoelkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2004.

[65] M. Song, C. Breneman, J. Bi, N. Sukumar, K. Bennett, S. Cramer, and N. Tugcu. Prediction of protein retention times in anion-exchange chromatography systems using support vector regression. *Journal of Chemical Information and Computer Sciences*, 2002.

[66] T. Takahashi and T. Kurita. Robust denoising by kernel PCA. In *Proceedings of the International Conference on Artificial Neural Networks*, volume 12, pages 739–744, 2002.

[67] A. Thompson, J. Brown, J. Kay, and D. Titterington. A study of methods of choosing the smoothing parameter in image restoration by regularization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:326–339, 1991.

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

[69] N. Troje and H. Bulthoff. Face recognition under varying poses: the role of texture and shape. *Vision Research*, 36:1761–1771, 1996.

[70] K. Tsuda and G. Ratsch. Image reconstruction by linear programming. *IEEE Transactions on Image Processing*, 14:737–744, 2005.

[71] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[72] G. Wahba. Support vector machines, reproducing kernel hilbert space and the randomized gacv. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

[73] G. Wahba, Y. Lin, and H. Zhang. Generalized approximate cross validation for support vector machines. In *Advances in Large Margin Classifiers*. MIT Press, 2000.

[74] G. Wang, D.Y. Yeung, and F.H. Lochovsky. A kernel path algorithm for support vector machines. In *Proceedings of the 24th International Conference on Machine Learning*, pages 951–958, 2007.

[75] X.X. Wang, S. Chen, D. Lowe, and C.J. Harris. Sparse support vector regression based on orthogonal forward selection for the generalised kernel model. *Neurocomputing*, 70:462–474, 2006.

[76] A. Weeks. *Fundamentals of Electronic Image Processing.* Wiley, 1996.

[77] H. Zhang and J. Malik. Learning a discriminative classifier using shape context distances. *IEEE Computer Vision and Pattern Recognition*, 1:242–247, 2003.

[78] P. Zhao and B. Yu. On model selection consistency of lasso. *Journal of Machine Learning Research*, 7:2541–2563, 2006.

[79] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. 1-norm svms. *Neural Information Processing Systems*, 16, 2004.

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

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