Doctoral Dissertation

Parametric Optimization in Machine Learning

2011

Masayuki Karasuyama
烏山 昌幸

名古屋工業大学博士論文
甲第793号(課程修了による)
平成２３年３月２３日授与

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last few decades, a variety of *machine learning* algorithms have been developed as powerful tools for data analysis in various practical applications such as natural language processing, computer vision, search engine design and bioinfomatics. The *Support Vector Machine* (SVM) [15, 31, 112] is one of the most successful algorithms in machine learning. The SVM was originally developed for the classification problem and the underlying statistical and algorithmical framework has been extended to other learning tasks such as regression, domain description and learning to rank.

Most of machine learning algorithms are represented as the *mathematical optimization problem* which is formulated as the problem of minimizing (or maximizing) an objective function subject to some constraints. The objective function of the SVM is defined as a linear combination of a loss term and a regularization term. The loss term penalizes the miss-classification of the estimated classifier and the regularization term penalizes the complexity of the classifier. The balance of these two terms are adjusted by a constant so-called *regularization parameter.*

The optimization problem is usually solved by iterative algorithms such as the interior point method and the active set method except for the simple case that the analytical closed form of the solutions can be derived. Since these algorithms iteratively improve the accuracy of the solution, the computational cost can be large especially when they are applied to large data sets. Moreover, in many situations of machine learning, one needs to solve a sequence of optimization problems. For example, in practical applications of the SVM, one has to solve many optimization problems for various different values of the regularization parameter in order to select a classifier with good performance.

In this paper, we focus on the *parametric programming* [40] technique in optimization. It has been developed for solving a sequence of optimization problems parametrized by a set of *problem parameters* which are constant in the optimization process. Unlike the above mentioned iterative algorithms, it provides a framework for computing the closed analytical form of the solutions for a class of optimization problems when the problem parameters change continuously. The remarkable features of parametric programming approach are:

- The continuous changes of optimal solutions can be analytically investigated
- It helps to develop a computationally efficient algorithm for updating solution

It is well-known that the change of optimal solutions are represented as *piece-wise linear* functions of the problem parameters for a class of optimization problems [12, 92] and then it can be efficiently traced by solving linear systems.

One of the most popular applications of parametric programming in machine learning literature is the *regularization path.* This algorithm allows us to trace the optimal solution with respect to the change of regularization parameter which controls the complexity of a model. The LARS algorithm [36] uses parametric programming like approach for variable selection of the least square regression. It has been shown that the slight modification of the LARS also leads the regularization path of $L1$ regularized regression (LASSO [109]). Inspired by LARS, [47] proposed regularization path for the SVM. Due to its computational efficiency for giving a full presentation of the solutions, the same technique has been applied to various models and situations [5, 46, 53, 69, 71, 73, 78, 95, 102, 111, 121, 124].

Another important parametric programming approach in machine learning is in an online learning scenario. In this situation, we need to update the trained model when some new observations arrive and/or some observations become obsolete. *Incremental decremental algorithms* [24, 33, 68, 79, 80] efficiently update the SVM solutions when a data point is added to or removed from training data set. This algorithm enables efficient computation by exploiting the piece-wise linearity of the SVM solutions.

In machine learning literature, these parametric programming approaches are sometimes called *solution path* algorithm or *path following* algorithm. This technique has been widely applied to various machine learning tasks other than those above [7, 33, 38, 42, 72, 107, 117]. Although all of the above examples exploit the piece-wise linearity of solutions, *nonlinear path following* is also studied so far [8, 61, 65, 66, 86, 88, 94, 116, 122]. However, since it is difficult to reveal the exact behavior of nonlinear solutions, most of these algorithms trace approximated path (typical approach is taking small steps to roughly trace the path).

In this paper, we further extend the parametric programming approach in machine learning to the following three directions:

- Incremental decremental learning by multi-parametric approach
- Solution path for instance-weighted learning
- Nonlinear path for a quadratic loss and a quadratic regularizer model

The first two directions are based on *multi-parametric programming* technique which changes multiple problem parameters simultaneously. This technique has not been fully applied to machine learning algorithms. We show this approach has various advantages for the above tasks. The third one considers nonlinear regularization path algorithm for a class of learning machines that have a quadratic loss and a quadratic regularizer. We develop an algorithm that can efficiently follow the piecewise nonlinear path by exploiting a specific form of nonlinear function.

We use the SVM as the basic learning algorithm throughout the paper. We thus briefly introduce the SVM in Chapter 2. The typical advantages of the SVM are as follows.

- Using the *kernel* function, it can estimate complex nonlinear models in a linear modeling framework.
- The optimal model is obtained by solving a convex quadratic optimization problem in which any local optima are guaranteed to be global optima.
- The final solutions are often highly sparse. This leads to efficient computation.

Due to the above reasons, the SVM is one of the standard tools for various machine learning tasks. Although we manly derive formulations of proposed approaches in the classification setting, the same approaches can be easily applied to other variants of the SVM such as regression and domain description.

In Chapter 3, we propose multiple incremental decremental algorithm which is the novel online learning algorithm for the SVM using multi-parametric approach. When we want to add and/or remove multiple data points, previous incremental decremental algorithm need to run repeatedly for each data point. Our proposed approach efficiently updates changes of multiple data points simultaneously. Some analyses and experimental results show that the proposed algorithm can substantially reduce the computational cost.

In Chapter 4, we consider the instance-weighted learning which implies each training instance has their weight or importance. This instance-weighted learning plays an important role in various machine learning tasks such as non-stationary data analysis, heteroscedastic data modeling, covariate shift adaptation, learning to rank and transductive learning. Those instance weights often change dynamically or adaptively, and thus the weighted SVM solution must be repeatedly computed. We introduce multi-parametric solution path algorithm for efficient update of the instance-weighted SVM. Moreover, through extensive experiments on various practical applications, we demonstrate the usefulness of the proposed algorithm.

In Chapter 5, we study the solution path of the learning machines that have quadratic loss and quadratic regularizer. Since the path of this class of learning machines is not piece-wise linear, we cannot apply usual parametric programing technique directory. In this case, each piece-wise segment is represented as a class of rational functions. We develop an algorithm that efficiently follow piece-wise nonlinear path using rational approximation approach. Our approach is highly accurate but faster than naive exhaustive search for regularization parameter.

Finally, we conclude our work, in Chapter 6.

Chapter 2

Support Vector Machines

Support Vector Machines (SVM) have attracted wide interest as the effective tools for pattern recognition. The SVM learns a linear model in a feature space through convex quadratic optimization. This formulation leads various computational advantages such as the kernel trick, sparseness of solutions and the absence of local optima (except for global optima). Although the original SVM is a binary classifier based on the large margin principle, the idea has also been applied to the other problems such as regression, domain description and ranking. Since throughout the paper we use the SVM as the basic learning machine, here, we briefly review the formulation of it, especially for classification and regression. The more comprehensive information of the SVMs can be found in [10,16,20,32,37,59,83,97,112].

2.1 Classification

Suppose we have a set of training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^p$ is the input and $y_i \in \{-1, +1\}$ is the output class label. Support Vector Machines (SVM) [15,31] learn the following discriminant function:

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \Phi(\boldsymbol{x}) + b,$$

in a feature space $\mathcal{F}$, where $\Phi : \mathcal{X} \to \mathcal{F}$ is a map from the input space $\mathcal{X}$ to the feature space $\mathcal{F}$, $\boldsymbol{w} \in \mathcal{F}$ is a coefficient vector, $b \in \mathbb{R}$ is a bias term, and $^\top$ denotes the transpose. The model parameter $\boldsymbol{w}$ and b can be obtained by solving an optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w}, b, \{\xi_i\}_{i=1}^n} \quad & \frac{1}{2}||\boldsymbol{w}||_2^2 + C \sum_{i=1}^n \xi_i \\ \text{s.t.} \quad & y_i \left(\boldsymbol{w}^\top \Phi(\boldsymbol{x}) + b\right) \geq 1 - \xi_i, \ \xi_i \geq 0, \ i = 1, \cdots, n, \end{aligned} \tag{2.1}$$

where $\frac{1}{2}||\boldsymbol{w}||^2$ is the regularization term, $||\cdot||_2$ denotes the Euclidean norm, $C \in \mathbb{R}^+$ is the trade-off parameter. Introducing Lagrange multipliers $\alpha_i, \rho_i \geq 0$, we can write the corresponding Lagrangian as

$$\begin{aligned} L &= \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\xi_i \\ &\quad -\sum_{i=1}^{n}\alpha_i\left\{y_i\left(\boldsymbol{w}^\top\Phi(\boldsymbol{x})+b\right)-1+\xi_i\right\}-\sum_{i=1}^{n}\rho_i\xi_i. \end{aligned} \tag{2.2}$$

Setting the derivatives w.r.t. primal variables $\boldsymbol{w}, b$ and ξ_i to zero, we can obtain

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{w}} = \mathbf{0} \;&\Leftrightarrow\; \boldsymbol{w} = \sum_{i=1}^{n}\alpha_i y_i \Phi(\boldsymbol{x}_i), \\ \frac{\partial L}{\partial b} = 0 \;&\Leftrightarrow\; \sum_{i=1}^{n}\alpha_i y_i = 0, \\ \frac{\partial L}{\partial \xi_i} = 0 \;&\Leftrightarrow\; \alpha_i = C - \rho_i, \end{aligned}$$

where $\mathbf{0}$ denotes the vector with all zeros. Substituting these equations into (2.2), we obtain the following dual problem:

$$\begin{aligned} \max_{\{\alpha_i\}_{i=1}^n} \quad & -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j Q_{ij} + \sum_{i=1}^{n}\alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} y_i\alpha_i = 0,\; 0 \leq \alpha_i \leq C,\; i = 1,\ldots,n, \end{aligned} \tag{2.3}$$

where

$$Q_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j),$$

and $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i)^T\Phi(\boldsymbol{x}_j)$ is a *reproducing kernel* [4]. The *Karush-Kuhn-Tucker (KKT) complementarity conditions* [17] are

$$\alpha_i\left\{y_i f(\boldsymbol{x}_i) - 1 + \xi_i\right\} = 0, \quad i = 1,\ldots,n, \tag{2.4a}$$

$$\xi_i(\alpha_i - C) = 0, \quad i = 1,\ldots,n. \tag{2.4b}$$

The optimal discriminant function $f : \mathcal{X} \rightarrow \mathbb{R}$ is formulated as

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}\alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b. \tag{2.5}$$

The bias term b can be obtained using KKT complementarity condition (2.4a).

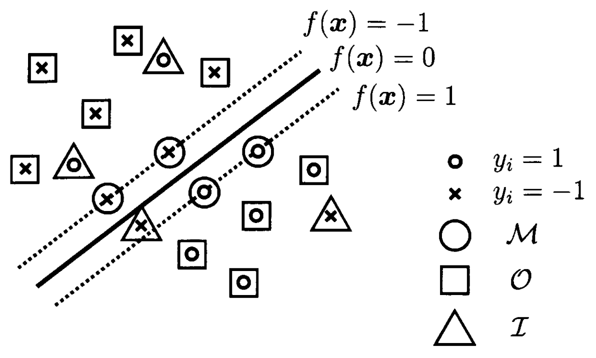


Figure 2.1: The partitioning of the data points in SVM. The data points which are in the set $\mathcal{M}$ (enclosed by the circle) are exactly on the margin. The data points which are in the set $\mathcal{O}$ (enclosed by the square) are in the outside the margin. The data points which are in the set $\mathcal{I}$ (enclosed by the triangle) are in the inside the margin.

The optimality conditions are summarized as follows:

$$y_i f(\boldsymbol{x}_i) \geq 1, \quad \text{if} \quad \alpha_i = 0, \tag{2.6a}$$

$$y_i f(\boldsymbol{x}_i) = 1, \quad \text{if} \quad 0 < \alpha_i < C, \tag{2.6b}$$

$$y_i f(\boldsymbol{x}_i) \leq 1, \quad \text{if} \quad \alpha_i = C, \tag{2.6c}$$

$$\sum_{i=1}^{n} y_i \alpha_i = 0. \tag{2.6d}$$

At the optimal solution, we can categorize the location of each data point using their parameter value as follows:

$$\mathcal{O} = \{i \mid \alpha_i = 0\}, \tag{2.7a}$$

$$\mathcal{M} = \{i \mid 0 < \alpha_i < C\}, \tag{2.7b}$$

$$\mathcal{I} = \{i \mid \alpha_i = C\}, \tag{2.7c}$$

where $\mathcal{O}$, $\mathcal{M}$, and $\mathcal{I}$ stand for 'Outside the margin' $(y_i f(\boldsymbol{x}_i) \geq 1)$, 'on the Margin' $(y_i f(\boldsymbol{x}_i) = 1)$, and 'Inside the margin' $(y_i f(\boldsymbol{x}_i) \leq 1)$, respectively (see Fig. 2.1).

2.2 Regression

The *support vector regression* (SVR) is a variant of SVM for regression problems [81, 84, 113].

The primal optimization problem for the SVR is defined by

$$\begin{aligned}
\min_{\boldsymbol{w},b,\{\xi_i,\xi_i^*\}_{i=1}^n} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^n C_i(\xi_i + \xi_i^*), \\
\text{s.t.} \quad & y_i - f(\boldsymbol{x}_i) \le \varepsilon + \xi_i, \\
& f(\boldsymbol{x}_i) - y_i \le \varepsilon + \xi_i^*, \\
& \xi_i, \xi_i^* \ge 0,\ i = 1, \dots, n,
\end{aligned} \tag{2.8}$$

where $\epsilon > 0$ is an insensitive-zone thickness. The SVR ignores the small error which is less than ε. This ε-insensitive loss leads sparseness property. The Lagrangian primal function of (2.8) is represented as

$$\begin{aligned}
L = {} & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^n (\xi_i + \xi_i^*) \\
& + \sum_{i=1}^n \beta_i\{y_i - \boldsymbol{w}^\top \Phi(\boldsymbol{x}_i) - b - \varepsilon - \xi_i\} \\
& + \sum_{i=1}^n \beta_i^*\{\boldsymbol{w}^\top \Phi(\boldsymbol{x}_i) + b - y_i - \varepsilon - \xi_i^*\} \\
& - \sum_{i=1}^n \rho_i \xi_i - \sum_{i=1}^n \rho_i^* \xi_i^*,
\end{aligned}$$

where $\beta_i, \beta_i^*, \rho_i, \rho_i^* \ge 0$ are the Lagrange multipliers. Setting the derivatives w.r.t. primal variables $\boldsymbol{w}, b, \xi_i, \xi_i^*$ to zero, we arrive at:

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{0} \Leftrightarrow \boldsymbol{w} = \sum_{i=1}^n (\beta_i - \beta_i^*)\Phi(\boldsymbol{x}_i), \tag{2.9a}$$

$$\frac{\partial L}{\partial b} = 0 \Leftrightarrow \sum_{i=1}^n (\beta_i - \beta_i^*) = 0, \tag{2.9b}$$

$$\frac{\partial L}{\partial \xi_i} = 0 \Leftrightarrow \beta_i = C - \rho_i, \tag{2.9c}$$

$$\frac{\partial L}{\partial \xi_i^*} = 0 \Leftrightarrow \beta_i^* = C - \rho_i^*. \tag{2.9d}$$

Using these equations and (2.9), we obtain the following dual optimization problem:

$$\begin{aligned}
\max_{\{\beta_i,\beta_i^*\}_{i=1}^n} \quad & -\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n (\beta_i - \beta_i^*)(\beta_j - \beta_j^*)K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
& -\varepsilon \sum_{i=1}^n |\beta_i - \beta_i^*| + \sum_{i=1}^n y_i(\beta_i - \beta_i^*) \\
\text{s.t.} \quad & \sum_{i=1}^n (\beta_i - \beta_i^*) = 0, \\
& 0 \le \beta_i, \beta_i^* \le C,\ i = 1, \cdots, n.
\end{aligned}$$

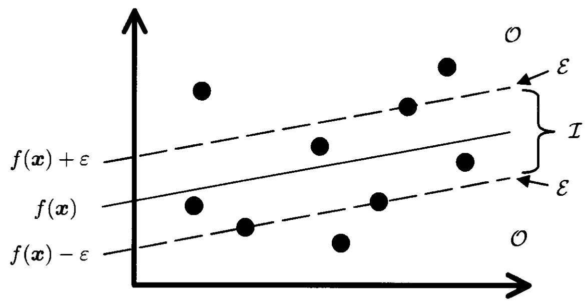


Figure 2.2: Partitioning of data points in SVR.

The Karush-Kuhn-Tucker (KKT) complementarity conditions are

$$\beta_i\{y_i - f(\boldsymbol{x}_i) - \varepsilon - \xi_i\} = 0, \tag{2.10a}$$

$$\beta_i^*\{f(\boldsymbol{x}_i) - y_i - \varepsilon - \xi_i^*\} = 0, \tag{2.10b}$$

$$\xi_i(C - \beta_i) = 0, \tag{2.10c}$$

$$\xi_i^*(C - \beta_i^*) = 0. \tag{2.10d}$$

First two conditions leads $\beta_i\beta_i^* = 0$. Because if $\beta_i > 0$ and $\beta_i^* > 0$, we can't satisfy (2.10a) and (2.10b) simultaneously. Using Lagrangian (2.9) and (2.9a)-(2.9d), and substituting $\alpha_i \equiv \beta_i - \beta_i^*$, we can obtain simplified dual formulation [32]:

$$\max_{\{\alpha_i\}_{i=1}^n} \quad -\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \alpha_i\alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \varepsilon\sum_{i=1}^n |\alpha_i| + \sum_{i=1}^n y_i\alpha_i \tag{2.11}$$

$$\text{s.t.} \quad \sum_{i=1}^n \alpha_i = 0, \ -C \le \alpha_i \le C, i = 1, \cdots, n.$$

The regression function $f : \mathcal{X} \to \mathbb{R}$ is formulated as

$$f(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b.$$

The bias term b can be obtained from KKT complementarity condition (2.10a) and (2.10b).

The KKT conditions can be summarized as below:

$$|y_i - f(\boldsymbol{x}_i)| \le \varepsilon, \ \text{if} \ \ \alpha_i = 0, \tag{2.12a}$$

$$|y_i - f(\boldsymbol{x}_i)| = \varepsilon, \ \text{if} \ \ 0 < |\alpha_i| < C, \tag{2.12b}$$

$$|y_i - f(\boldsymbol{x}_i)| \ge \varepsilon, \ \text{if} \ \ |\alpha_i| = C, \tag{2.12c}$$

$$\sum_{i=1}^n \alpha_i = 0. \tag{2.12d}$$

Then the training instances can be partitioned into the following three index sets (see Fig. 2.2):

$$\mathcal{O} = \{i : |y_i - f(\boldsymbol{x}_i)| \geq \varepsilon, |\alpha_i| = C\}, \tag{2.13a}$$

$$\mathcal{E} = \{i : |y_i - f(\boldsymbol{x}_i)| = \varepsilon, 0 < |\alpha_i| < C\}, \tag{2.13b}$$

$$\mathcal{I} = \{i : |y_i - f(\boldsymbol{x}_i)| \leq \varepsilon, \alpha_i = 0\}. \tag{2.13c}$$

Chapter 3

Incremental Decremental Learning

In online learning, we need to update the trained model when some new observations arrive and/or some observations become obsolete. If we want to add or remove single data point in the SVM, incremental decremental algorithm [24] can be used to update the model efficiently. However, to add and/or remove multiple data points, the computational cost of current update algorithm becomes inhibitive because we need to repeatedly apply it for each data point. In this chapter, we develop an extension of incremental decremental algorithm which efficiently works for simultaneous update of multiple data points. Some analyses and experimental results show that the proposed algorithm can substantially reduce the computational cost. Our approach is especially useful for online SVM learning in which we need to remove old data points and add new data points in a short amount of time. The discussion in this chapter has appeared in [54, 57, 58].

3.1 Introduction

For online learning, incremental decremental algorithm of the SVM was previously proposed in [24], and the approach was adapted to other variants of kernel machines [33, 68, 79, 80]. When a single data point is added and/or removed, these algorithms can efficiently update the trained model without re-training it from scratch. Although these algorithms were developed in different context, they can be considered as instances of parametric programming or path-following [47]. Parametric programming [2, 40] is an optimization technique for solving a series of parameterized optimization problems. Recently, in the machine learning literature, path-following was used for various purposes [7, 46, 47, 107, 117]. In the incremental and decremental algorithms, one solves a solution path with respect to the coefficient parameter corresponding to the data point to be added or removed. When we add and/or remove multiple data points using these algorithms, one must repeat the updating operation for each single data point. It often requires too much computational cost for real-time online learning. In what follows, we refer this conventional algorithm as *single incremental decremental algorithm* or *single update algorithm.*

In this chapter, we develop a multiple incremental decremental algorithm of the SVM. The proposed algorithm can update the trained model more efficiently when multiple data points are added and/or removed simultaneously. We develop the algorithm by introducing *multi-parametric programming* [89] from the optimization literature. We consider a path-following problem in the multi-dimensional space spanned by the coefficient parameters corresponding to the set of data points to be added or removed. In this chapter, we call our proposed algorithm as *multiple incremental decremental algorithm* or *multiple update algorithm.* Throughout the chapter, we discuss multiple update algorithm for support vector classification. However, the idea is easily applicable to other kernel machines, e.g., for regression or outlier detection tasks. Although we do not describe the detail of other kernel machines, the derivation is much the same as classification case.

The total computational cost of parametric programming is roughly proportional to the number of *breakpoints* on the solution path. In the repeated use of single update algorithm for each data point, one follows the coordinate-wise solution path in the multi-dimensional coefficient parameter space. On the other hand, in multiple update algorithm, we establish a direction in the multi-dimensional coefficient parameter space so that the total length of the path becomes much shorter than the coordinate-wise one. Because the number of breakpoints in the shorter path followed by our algorithm is less than that in the longer coordinate-wise path, we can gain relative computational efficiency. Fig. 3.3 schematically illustrates our main idea.

3.2 Single Incremental Decremental SVM

In this section, we briefly review the conventional single incremental decremental SVM [24]. Using the SV sets (2.7b) and (2.7c), we can expand $y_i f(\boldsymbol{x}_i)$ as

$$y_i f(\boldsymbol{x}_i) = \sum_{j \in \mathcal{M}} Q_{ij}\alpha_j + \sum_{j \in \mathcal{I}} Q_{ij}\alpha_j + y_i b,$$

where $Q_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. When a new data point $(\boldsymbol{x}_c, y_c)$ is added, we increase the corresponding new parameter α_c from 0 while keeping the optimal conditions of the other parameters satisfied.

Let us denote the amount of the change of each variable with an operator Δ. To satisfy (2.6b) and the equality constraint of the dual (2.3), we need

$$\begin{aligned} Q_{ic}\Delta\alpha_c + \sum_{j\in\mathcal{M}} Q_{ij}\Delta\alpha_j + y_i\Delta b &= 0,\ i \in \mathcal{M}, \\ y_c\Delta\alpha_c + \sum_{j\in\mathcal{M}} y_j\Delta\alpha_j &= 0. \end{aligned}$$

Solving this linear system with respect to $\Delta\alpha_i, i \in \mathcal{M}$, and Δb, we obtain the update direction of the parameters. We update the parameters in that direction with the largest step length under the constraint that no element moves across $\mathcal{M}, \mathcal{I}$ and $\mathcal{O}$. In other words, the step length is chosen so that KKT optimality conditions are kept satisfied.

If we update the parameters as above, we encounter a point at which the three index sets $\mathcal{M}, \mathcal{I}$ and $\mathcal{O}$ must be updated. For example, if there is a data point i such that $\alpha_i > 0, \Delta\alpha_i < 0, i \in \mathcal{M}$, the parameter α_i is decreased toward 0. And, if α_i becomes 0 in the update process as above, we need to move the index i from $\mathcal{M}$ to $\mathcal{O}$. After updating the index sets $\mathcal{M}, \mathcal{I}$ and $\mathcal{O}$, we repeat the process until the new data point satisfies the optimality condition. Decremental algorithm can be derived similarly, in which the target parameter moves toward 0.

3.3 Multiple Incremental Decremental SVM

Suppose we add m new data points and remove ℓ data points simultaneously. Let us denote the index set of new adding data points and removing data points as

$$\mathcal{A} = \{n+1, n+2, \cdots, n+m\} \text{ and } \mathcal{R} \subset \{1, \cdots, n\},$$

respectively, where $|\mathcal{R}| = \ell$. We remove the elements of $\mathcal{R}$ from the sets $\mathcal{M}, \mathcal{I}$ and $\mathcal{O}$ (i.e. $\mathcal{M} \leftarrow \mathcal{M} \setminus \mathcal{R}$, $\mathcal{I} \leftarrow \mathcal{I} \setminus \mathcal{R}$ and $\mathcal{O} \leftarrow \mathcal{O} \setminus \mathcal{R}$).

When $m = 1, \ell = 0$ or $m = 0, \ell = 1$, our method corresponds to the conventional single incremental decremental algorithm. We initially set $\alpha_i = 0, \forall i \in \mathcal{A}$. If we have $y_i f(\boldsymbol{x}_i) > 1, i \in \mathcal{A}$, we can append these indices to $\mathcal{O}$ and remove them from $\mathcal{A}$ because these points already satisfy the optimality condition (2.6a). Similarly, we can append the indices $\{i \mid y_i f(\boldsymbol{x}_i) = 1, i \in \mathcal{A}\}$ to $\mathcal{M}$ and remove them from $\mathcal{A}$. In addition, we can remove the points $\{i \mid \alpha_i = 0, i \in \mathcal{R}\}$ because they already have no influence on the model.

Let us define $\boldsymbol{y} = [y_1, \cdots, y_{n+m}]^\top$, $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_{n+m}]^\top$, and $\boldsymbol{Q} \in \mathbb{R}^{(n+m)\times(n+m)}$, where (i,j)-th entry of $\boldsymbol{Q}$ is Q_{ij}. Unlike single incremental decremental algorithm, we need to determine the directions of $\boldsymbol{\Delta\alpha}_{\mathcal{A}}$ and $\boldsymbol{\Delta\alpha}_{\mathcal{R}}$. These directions have a critical influence on the computational cost. For $\boldsymbol{\Delta\alpha}_{\mathcal{R}}$, we simply trace the shortest path to $\boldsymbol{0}$, i.e.,

$$\boldsymbol{\Delta\alpha}_{\mathcal{R}} = -\eta\boldsymbol{\alpha}_{\mathcal{R}}, \tag{3.1}$$

where $\eta \geq 0$ is a step length. For $\boldsymbol{\Delta\alpha}_{\mathcal{A}}$, we do not know the optimal value of $\boldsymbol{\alpha}_{\mathcal{A}}$ beforehand. To determine this direction, we may be able to use some optimization techniques (e.g. Newton method). However, such methods usually need additional computational burden. In this paper, we simply take

$$\boldsymbol{\Delta\alpha}_{\mathcal{A}} = \eta(C\boldsymbol{1} - \boldsymbol{\alpha}_{\mathcal{A}}). \tag{3.2}$$

This would become the shortest path if $\alpha_i = C, \forall i \in \mathcal{A}$, at optimality.

When we move parameters $\alpha_i, \forall i \in \mathcal{A} \cup \mathcal{R}$, the optimality conditions of the other parameters must be kept satisfied. From $y_i f(\boldsymbol{x}_i) = 1, i \in \mathcal{M}$, and the equality constraint of dual (2.3), we need

$$\sum_{j\in\mathcal{A}} Q_{ij}\Delta\alpha_j + \sum_{j\in\mathcal{R}} Q_{ij}\Delta\alpha_j + \sum_{j\in\mathcal{M}} Q_{ij}\Delta\alpha_j + y_i\Delta b = 0, \ i \in \mathcal{M}, \tag{3.3}$$

$$\sum_{j\in\mathcal{A}} y_j\Delta\alpha_j + \sum_{j\in\mathcal{R}} y_j\Delta\alpha_j + \sum_{j\in\mathcal{M}} y_j\Delta\alpha_j = 0. \tag{3.4}$$

Using matrix notation, (3.3) and (3.4) can be written as

$$\boldsymbol{M}\begin{bmatrix} \Delta b \\ \boldsymbol{\Delta\alpha}_{\mathcal{M}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{y}_{\mathcal{A}}^\top & \boldsymbol{y}_{\mathcal{R}}^\top \\ \boldsymbol{Q}_{\mathcal{M},\mathcal{A}} & \boldsymbol{Q}_{\mathcal{M},\mathcal{R}} \end{bmatrix}\begin{bmatrix} \boldsymbol{\Delta\alpha}_{\mathcal{A}} \\ \boldsymbol{\Delta\alpha}_{\mathcal{R}} \end{bmatrix} = \boldsymbol{0}, \tag{3.5}$$

where

$$\boldsymbol{M} = \begin{bmatrix} 0 & \boldsymbol{y}_{\mathcal{M}}^\top \\ \boldsymbol{y}_{\mathcal{M}} & \boldsymbol{Q}_{\mathcal{M}} \end{bmatrix}.$$

From the definitions of the index sets in (2.7a)-(2.7c), the following inequality constraints must also be satisfied:

$$0 \leq \alpha_i + \Delta\alpha_i \leq C, \quad i \in \mathcal{M}, \tag{3.6a}$$

$$y_i\{f(\boldsymbol{x}_i) + \Delta f(\boldsymbol{x}_i)\} > 1, \quad i \in \mathcal{O}, \tag{3.6b}$$

$$y_i\{f(\boldsymbol{x}_i) + \Delta f(\boldsymbol{x}_i)\} < 1, \quad i \in \mathcal{I}. \tag{3.6c}$$

Since we removed the indices $\{i \mid f(\boldsymbol{x}_i) \geq 1\}$ from $\mathcal{A}$, we obtain

$$y_i\{f(\boldsymbol{x}_i) + \Delta f(\boldsymbol{x}_i)\} < 1, \qquad i \in \mathcal{A}. \tag{3.7}$$

During the process of moving $\alpha_i, i \in \mathcal{A}$, to C from 0, if the inequality (3.7) becomes equality for any i, we can append the point to $\mathcal{M}$ and remove it from $\mathcal{A}$. On the other hand, if (3.7) holds after α_i becomes C, the point moves to $\mathcal{I}$. The region that satisfies (3.6) and (3.7) corresponds to *critical region (CR)* in the parametric programming literature [89].

We decide the update direction by solving the linear system (3.5) while monitoring inequalities (3.6) and (3.7). Substituting (3.1) and (3.2) to (3.5), we obtain the update direction

$$\begin{bmatrix} \Delta b \\ \Delta \boldsymbol{\alpha}_{\mathcal{M}} \end{bmatrix} = \eta \boldsymbol{\phi}, \tag{3.8}$$

where

$$\boldsymbol{\phi} = -\boldsymbol{M}^{-1} \begin{bmatrix} \boldsymbol{y}_{\mathcal{A}}^{\top} & \boldsymbol{y}_{\mathcal{R}}^{\top} \\ \boldsymbol{Q}_{\mathcal{M},\mathcal{A}} & \boldsymbol{Q}_{\mathcal{M},\mathcal{R}} \end{bmatrix} \begin{bmatrix} C\boldsymbol{1} - \boldsymbol{\alpha}_{\mathcal{A}} \\ -\boldsymbol{\alpha}_{\mathcal{R}} \end{bmatrix}. \tag{3.9}$$

In this paper, we assume that the kernel matrix $\boldsymbol{Q}$ is positive definite. Then the matrix $\boldsymbol{M}$ is invertible[1] . If $\boldsymbol{Q}$ is positive semi-definite, $\boldsymbol{M}$ may not be invertible. In the later experiments, we use one practical heuristic to circumvent this problem: adding small positive constant to the diagonal of the kernel matrix.

To determine the step length η, we need to check inequalities (3.6) and (3.7). Using vector notation and the Hadamard product $\odot$ (element-wise product [98]), we can write

$$\boldsymbol{y} \odot \Delta \boldsymbol{f} = \eta\, \boldsymbol{\psi}, \tag{3.10}$$

where

$$\boldsymbol{\psi} = [\; \boldsymbol{y} \;\; \boldsymbol{Q}_{:,\mathcal{M}} \;]\; \boldsymbol{\phi} + \boldsymbol{Q}_{:,\mathcal{A}}(C\boldsymbol{1} - \boldsymbol{\alpha}_{\mathcal{A}}) - \boldsymbol{Q}_{:,\mathcal{R}}\boldsymbol{\alpha}_{\mathcal{R}}, \tag{3.11}$$

and the subscription ":" of $\boldsymbol{Q}$ denotes the index of all the elements $\{1, \cdots, n+m\}$.

Since (3.8) and (3.10) are linear function of η, we can calculate the set of the largest step length η's for each i at which the inequalities (3.6) and (3.7) becomes equality for

[1]Precisely speaking, we have only to assume that the submatrix $\boldsymbol{Q}_{\mathcal{M}}$ is strictly positive definite in the subspace $\{\boldsymbol{z} \in \mathbb{R}^{|\mathcal{M}|} | \boldsymbol{y}_{\mathcal{M}}^{\top}\boldsymbol{z} = 0\}$ because it is the necessary and sufficient condition of $\boldsymbol{M}$ to be non-singular.

i. The size of such η's is $|\mathcal{M}| \times 2 + |\mathcal{O}| + |\mathcal{I}| + |\mathcal{A}|$ and we define this set as $\mathcal{H}$. We determine the step length as follows:

$$\eta = \min(\{\tilde{\eta} \mid \tilde{\eta} \in \mathcal{H},\ \tilde{\eta} \geq 0\} \cup \{1\}).$$

If η becomes 1, we terminate the algorithm because all the new data points in $\mathcal{A}$ and existing points in $\mathcal{M}, \mathcal{O}$ and $\mathcal{I}$ satisfy the optimality conditions and $\boldsymbol{\alpha}_{\mathcal{R}}$ is $\mathbf{0}$. Once we decide η, we can update $\boldsymbol{\alpha}_{\mathcal{M}}$ and b using (3.8), and $\boldsymbol{\alpha}_{\mathcal{A}}$ and $\boldsymbol{\alpha}_{\mathcal{R}}$ using (3.1) and (3.2). In the path-following literature, the points at which the size of linear system (3.5) is changed are called *breakpoints*. If the ith data point reaches the bound of any one of the constraints (3.6) and (3.7) we need to update $\mathcal{M}, \mathcal{O}$ and $\mathcal{I}$. After updating, we re-calculate $\boldsymbol{\phi}, \boldsymbol{\psi}$ to determine the next step length.

3.3.1 Empty Margin

We need to establish the way of dealing with empty margin $\mathcal{M}$ [2]. In such case, we can not obtain the bias from $y_i f(\boldsymbol{x}_i) = 1, i \in \mathcal{M}$. Then we can only obtain the interval of the bias from

$$\begin{aligned} y_i f(\boldsymbol{x}_i) &> 1, \quad && i \in \mathcal{O}, \\ y_i f(\boldsymbol{x}_i) &< 1, \quad && i \in \mathcal{I} \cup \mathcal{A}. \end{aligned}$$

To keep these inequality constraints, the bias term must be in

$$\max_{i \in \mathcal{L}} y_i g_i \leq b \leq \min_{i \in \mathcal{U}} y_i g_i, \tag{3.12}$$

where

$$g_i = 1 - \sum_{i \in \mathcal{I}} \alpha_i Q_{ij} - \sum_{i \in \mathcal{A}} \alpha_i Q_{ij} - \sum_{i \in \mathcal{R}} \alpha_i Q_{ij},$$

and

$$\begin{aligned} \mathcal{L} &= \{i \mid i \in \mathcal{O}, y_i = +1\} \cup \{i \mid i \in \mathcal{I} \cup \mathcal{A}, y_i = -1\}, \\ \mathcal{U} &= \{i \mid i \in \mathcal{O}, y_i = -1\} \cup \{i \mid i \in \mathcal{I} \cup \mathcal{A}, y_i = +1\}. \end{aligned}$$

[2]In some active set based SVM solvers [96,100,114], similar situation can be happened. Most of these algorithms choose active set using heuristics for quick convergence (e.g., choosing most KKT violating point to be active). On the other hand, in incremental decremental algorithm, a point in $\mathcal{M}$ must be chosen to keep satisfying optimality conditions.

If this empty margin happens during the path-following, we look for the new data points which re-enter the margin. When the set $\mathcal{M}$ is empty, equality constraint (3.4) becomes

$$\sum_{i \in \mathcal{A}} y_i \Delta\alpha_i + \sum_{i \in \mathcal{R}} y_i \Delta\alpha_i = \eta\delta(\boldsymbol{\alpha}) = 0, \tag{3.13}$$

where

$$\delta(\boldsymbol{\alpha}) = \sum_{i \in \mathcal{A}} y_i (C - \alpha_i) - \sum_{i \in \mathcal{R}} y_i \alpha_i.$$

We take two different strategies depending on $\delta(\boldsymbol{\alpha})$.

First, if $\delta(\boldsymbol{\alpha}) \neq 0$, we can not simply increase η from 0 while keeping (3.13) satisfied. Then we need new margin data point m_1 which enables the equality constraint to be satisfied. The index m_1 is either

$$i_{low} = \operatorname*{argmax}_{i \in \mathcal{L}} y_i g_i \text{ or } i_{up} = \operatorname*{argmax}_{i \in \mathcal{U}} y_i g_i.$$

If $i_{low}, i_{up} \in \mathcal{O} \cup \mathcal{I}$, we can update b and $\mathcal{M}$ as follows:

$$\delta(\boldsymbol{\alpha}) > 0 \;\Rightarrow\; b = y_{i_{up}} g_{i_{up}},\ \mathcal{M} = \{i_{up}\}, \tag{3.14a}$$

$$\delta(\boldsymbol{\alpha}) < 0 \;\Rightarrow\; b = y_{i_{low}} g_{i_{low}},\ \mathcal{M} = \{i_{low}\}. \tag{3.14b}$$

By setting the bias terms as above, equality condition

$$\eta\delta(\boldsymbol{\alpha}) + y_{m_1} \Delta\alpha_{m_1} = 0$$

is satisfied. If $i_{low} \in \mathcal{A}$ or $i_{up} \in \mathcal{A}$, we can put either of these points to margin by setting the bias term as $b = y_{i_{low}} g_{i_{low}}$ or $b = y_{i_{up}} g_{i_{up}}$.

On the other hand, if $\delta(\boldsymbol{\alpha}) = 0$, we can increase η while keeping (3.13) satisfied. Then, we increase η until the upper bound and the lower bound of the bias (3.12) take the same value (the bias term can be uniquely determined). When we increase η, g_i changes linearly:

$$\begin{aligned} \Delta g_i(\eta) &= -\sum_{j \in \mathcal{A}} \Delta\alpha_j Q_{ij} - \sum_{j \in \mathcal{R}} \Delta\alpha_j Q_{ij} \\ &= \eta\{-\sum_{j \in \mathcal{A}} (C - \alpha_j) Q_{ij} + \sum_{j \in \mathcal{R}} \alpha_j Q_{ij}\}. \end{aligned}$$

Since each $y_i(g_i + \Delta g_i(\eta))$ may intersect, we need to consider the following piece-wise linear boundaries:

$$\begin{aligned} u(\eta) &= \max_{i \in \mathcal{U}} y_i (g_i + \Delta g_i(\eta)), \\ l(\eta) &= \min_{j \in \mathcal{L}} y_j (g_j + \Delta g_j(\eta)). \end{aligned}$$

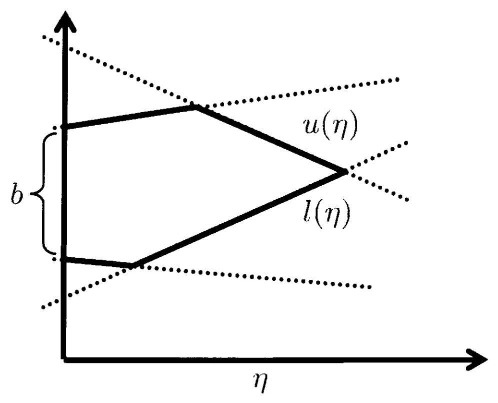


Figure 3.1: An illustration of the bias in empty margin case. Dotted lines represent $y_i(g_i + \Delta g_i(\eta))$, for each i. Solid lines are the upper bound and the lower bound of the bias. The bias term is uniquely determined when $u(\eta)$ and $l(\eta)$ intersect.

Fig. 3.1 shows an illustration of these functions. We trace the upper bound and the lower bound until two bounds become the same value.

3.3.2 Initialization

An initial SVM parameter can be obtained from conventional batch SVM solver. One of the most widely used batch algorithm is Sequential Minimal Optimization (SMO) [90]. We can also use our multiple incremental algorithm to find an initial solution. As we will see in the later experiments, incremental approach can obtain accurate solution as close as floating-point precision. We choose two data points $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2)\}$ from training data set that satisfy $y_1 = +1, y_2 = -1$. Analytical solution is obtained by

$$\begin{aligned}\alpha_1 &= \max\Big(0, \min\big(C, \frac{1}{Q_{22} - y_1 y_2 Q_{12}}\big)\Big), \\ \alpha_2 &= -y_1 y_2 \alpha_1.\end{aligned}$$

Then, we add remaining data points as the adding data points $\mathcal{A} = \{2, 3, \cdots, n\}$. This basic strategy has already proposed in [79]. However, our proposed method is more efficient than the approach in [79] because we can add $n-2$ data points simultaneously without repeatedly applying incremental operation to each data point.

3.3.3 Algorithm and Computational Complexity

Now, we describe entire procedure of our algorithm in Fig. 3.2.

At each breakpoint, we solve the linear system (3.9) with size $|\mathcal{M}|+1$ using Cholesky factor $\boldsymbol{L}$ of $\boldsymbol{Q}_{\mathcal{M}}$ (see Appendix A). The cost is $O(|\mathcal{M}|^2)$ because we use the *Cholesky*

```
arguments:
    Optimal parameters $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_n]^T, b$
    SV sets $\mathcal{M}, \mathcal{O}, \mathcal{I}$
    Adding indices $\mathcal{A}$ and removing indices $\mathcal{R}$
end arguments

function MID-SVM($\boldsymbol{\alpha}, b, \mathcal{M}, \mathcal{O}, \mathcal{I}, \mathcal{A}, \mathcal{R}$)
    Perform Cholesky factorization of kernel matrix:
    $\boldsymbol{Q}_{\mathcal{M}} = \boldsymbol{L}^T\boldsymbol{L}$
    repeat
        if $\mathcal{M}$ is empty then
            $\eta \leftarrow$ EmptyMargin
        else
            Solve linear system (3.9) to calculate $\boldsymbol{\phi}$
            using Cholesky factor $\boldsymbol{L}$
            Calculate $\boldsymbol{\psi}$ by (3.11)
            Calculate a set of step lengths $\mathcal{H}$ by checking
            inequalities (3.6) and (3.7)
            $\eta \leftarrow \min(\{\tilde{\eta} \mid \tilde{\eta} \in \mathcal{H},\ \tilde{\eta} \geq 0\} \cup \{1\})$
            $[\ \boldsymbol{\alpha}_{\mathcal{M}}^T\ b\ ] \leftarrow [\ \boldsymbol{\alpha}_{\mathcal{M}}^T\ b\ ] + \eta\ \boldsymbol{\phi}$
        end if
        $\boldsymbol{\alpha}_{\mathcal{A}} \leftarrow \boldsymbol{\alpha}_{\mathcal{A}} + \eta(C\mathbf{1} - \boldsymbol{\alpha}_{\mathcal{A}})$
        $\boldsymbol{\alpha}_{\mathcal{R}} \leftarrow \boldsymbol{\alpha}_{\mathcal{R}} - \eta\boldsymbol{\alpha}_{\mathcal{R}}$
        Update $\mathcal{M}, \mathcal{O}, \mathcal{I}, \mathcal{A}$ depending on the event type
        Update $\boldsymbol{L}$ (Cholesky factor rank-one update)
    until $\eta$ becomes 1
end function

function EmptyMargin
    if $\delta(\boldsymbol{\alpha}) \neq 0$ then
        if $i_{up} \in \mathcal{A}$ or $i_{low} \in \mathcal{A}$ then
            Set bias as $b \leftarrow y_{i_{up}} g_{i_{up}}$ or $b \leftarrow y_{i_{low}} g_{i_{low}}$
        else
            Set bias term $b$ as (3.14)
        end if
        $\eta \leftarrow 0$
    else
        Trace $u(\eta)$ and $l(\eta)$ until the bias term can
        be determined uniquely or $\eta$ becomes 1
    end if
    return $\eta$
end function
```

Figure 3.2: Pseudo-code of the multiple incremental decremental SVM

decomposition rank-one update [45] (see Appendix B) except the first step in $O(|\mathcal{M}|^3)$ [3]. Although the size of $\mathcal{M}$ changes at each breakpoint, to make our analysis easy, we assume $|\mathcal{M}|$ is constant in the entire process[4]. To update $y_i f(\boldsymbol{x}_i)$ we need to calculate (3.10) which takes $O((n+m) \times (|\mathcal{M}|+m+l))$ cost (Note that we need to calculate $\boldsymbol{\psi}$ in (3.11) which represents the change of $y_i f(\boldsymbol{x}_i)$. However, since the function value on the margin points do not change, we have only to compute $n + m - |\mathcal{M}|$ elements of $\boldsymbol{\psi}$.). Since we are interested in the situation where the number of adding or removing data points are relatively smaller than the training sample size, i.e., $m, l \ll n$, this cost is roughly $O(n|\mathcal{M}|)$. The step length calculation takes $O(n)$ cost. From these analyses, we see that each iteration roughly needs $O(|\mathcal{M}|^2+n|\mathcal{M}|+n)$ computations. Since $n > |\mathcal{M}|$, this can be considered as $O(n|\mathcal{M}|)$.

From these considerations, we see the computational cost of multiple update algorithm is approximately $O(\mathcal{B}n|\mathcal{M}| + n|\mathcal{M}|^2)$, where $\mathcal{B}$ is the number of breakpoints. Since the $O(n|\mathcal{M}|^2)$ computations are needed only once at initialization, $O(\mathcal{B}n|\mathcal{M}|)$ is the main computational burden in practice. Thus the number of breakpoints $\mathcal{B}$ is an important factor of the computational cost.

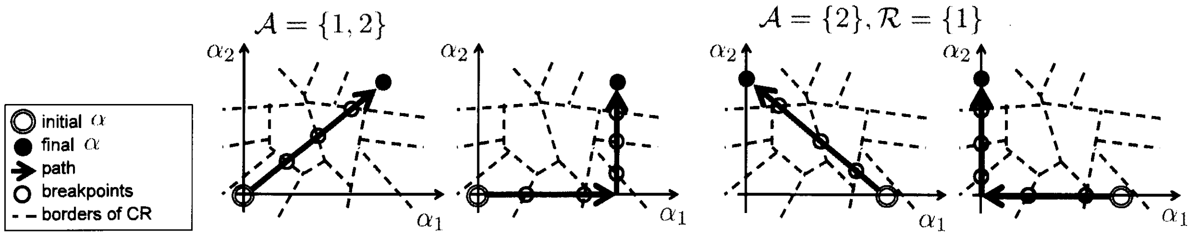
To analyze the relative computational efficiency of our multiple update algorithm to conventional single update algorithm, let us introduce the following assumptions:

- The number of breakpoints is proportional to the total length of the path.
- The path obtained by our algorithm is the shortest one.

The first assumption means that the breakpoints are uniformly distributed on the path. The second assumption holds for the removing parameters $\boldsymbol{\alpha}_{\mathcal{R}}$ because we know that we should move $\boldsymbol{\alpha}_{\mathcal{R}}$ to $\mathbf{0}$. On the other hand, for some of $\boldsymbol{\alpha}_{\mathcal{A}}$, the second assumption does not necessarily hold because we do not know the optimal $\boldsymbol{\alpha}_{\mathcal{A}}$ beforehand. In particular, if the point $i \in \mathcal{A}$ which was located inside the margin before the update moved to $\mathcal{M}$ during the update (i.e. the equality (3.7) holds), the path with respect to this parameter is not really the shortest one.

[3]As another way, we can also update $\boldsymbol{M}^{-1}$ directly using *block matrix inversion formula* [98] which is based on the Sherman-Morrison-Woodbury formula. However this approach sometimes runs into numerical difficulties. It is well known that the Cholesky factor approach is numerically more stable than this approach [39, 99].

[4]This simplification is employed to evaluate the computational cost of many path-following or (closely related) active-set based optimization algorithms (e.g., [47]).



(a) Adding 2 data points. (b) Adding and Removing 1 data point

Figure 3.3: The schematic illustration of the difference of path length and the number of breakpoints. Each polygonal region enclosed by dashed lines represents the region in which $\mathcal{M}, \mathcal{I}, \mathcal{O}$ and $\mathcal{A}$ are constant (CR: critical region). The intersection of the path and the borders are the breakpoints. The update of matrices and vectors at the breakpoints are the main computational cost of path-following. In the case of Fig. 3.3(a), we add 2 data points. If optimal $\alpha_1 = \alpha_2 = C$, our proposed algorithm can trace shortest path to optimal point from the origin (left plot). On the other hand, single incremental algorithm moves one coordinate at a time (right plot). Fig. 3.3(b) shows the case that we add and remove 1 data point, respectively. In this case, if $\alpha_2 = C$, our algorithm can trace shortest path to $\alpha_1 = 0, \alpha_2 = C$ (left plot), while single incremental algorithm again moves one coordinate at a time (right plot).

To simplify the discussion further, let us consider only the case of $|\mathcal{A}| = m > 0$ and $|\mathcal{R}| = 0$ (the same discussion holds for other cases too). In this simplified scenario, the ratio of the number of breakpoints of multiple update algorithm to that of repeated use of single update algorithm is

$$\|\boldsymbol{\alpha}_{\mathcal{A}}\|_2 : \|\boldsymbol{\alpha}_{\mathcal{A}}\|_1,$$

where $\| \bullet \|_2$ is ℓ_2 norm and $\| \bullet \|_1$ is ℓ_1 norm. Fig. 3.3 illustrates the concept in the case of $m = 2$. If we consider only the case of $\alpha_i = C, \forall i \in \mathcal{A}$, the ratio is simply $\sqrt{m} : m$.

3.4 Experiments

We compared the computational cost of the proposed multiple incremental decremental algorithm (MID-SVM) with (repeated use of) single incremental decremental algorithm [24] (SID-SVM) and with the LIBSVM [25], the-state-of-the-art batch SVM solver based on sequential minimal optimization algorithm (SMO). We wrote our own codes for SID and MID in C++ format. For matrix computations (e.g. matrix vector multiplication), we used LAPACK [3] routines. The parameters $\alpha_i, i \in \{1, \cdots, n\}$, and b were initialized to be optimal for the training set before incremental decremental operation. We compared the CPU time for adding and/or deleting several data points.

In LIBSVM, we examined several tolerances for termination criterion: $\varepsilon = \{10^{-3}, 10^{-6}, 10^{-9}\}$. When we use LIBSVM for online-learning, alpha seeding [34, 70] sometimes works well. The basic idea of alpha seeding is to use the parameters before the update as the initial parameter. In alpha seeding, we need to take care of the fact that the summation constraint $\boldsymbol{\alpha}^\top \boldsymbol{y} = 0$ may not be satisfied after removing α's in $\mathcal{R}$. In that case, we simply re-distribute

$$\delta = \sum_{i \in \mathcal{R}} \alpha_i y_i$$

uniformly to the in-bound α_i, $i \in \{i \mid 0 < \alpha_i < C\}$. If δ cannot be distributed to in-bound α's, it is also distributed to other α's. If we still can not distribute δ by this way, we did not use alpha-seeding.

For kernel function, we used RBF kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma ||\boldsymbol{x}_i - \boldsymbol{x}_j||^2)$. In this paper, we assume that the kernel matrix $\boldsymbol{K}$ is positive definite. If the kernel matrix happens to be singular, which typically arise when there are two or more identical data points in $\mathcal{M}$, our algorithm may not work. As long as we know, this degeneracy problem is not fully solved in path-following literature. Many heuristics are proposed to circumvent the problem. In the experiments described below, we use one of them: adding small positive constant to the diagonal elements of kernel matrix. We set this constant as 10^{-6}. For fair comparison we do not use any previously computed kernel information during the following incremental decremental process in all three algorithms (SID, MID and SMO). During incremental decremental process, we compute kernel information whenever needed, and they are kept in cache. In the LIBSVM we can specify cache size of kernel matrix. We set this cache size enough large to store the entire matrix. All results presented in this section are average of 10 runs.

3.4.1 Artificial Data

First, we used simple artificial data set to see the computational cost for various number of adding and/or removing points. We generated data points $(\boldsymbol{x}, y) \in \mathbb{R}^2 \times \{+1, -1\}$ using normal distributions:

$$\begin{aligned} p(\boldsymbol{x} \mid y = +1) &= \frac{1}{2}\mathcal{N}(\boldsymbol{\mu}_1^{(1)}, \boldsymbol{\Sigma}_1^{(1)}) + \frac{1}{2}\mathcal{N}(\boldsymbol{\mu}_2^{(1)}, \boldsymbol{\Sigma}_2^{(1)}), \\ p(\boldsymbol{x} \mid y = -1) &= \mathcal{N}(\boldsymbol{\mu}^{(2)}, \boldsymbol{\Sigma}^{(2)}), \end{aligned}$$

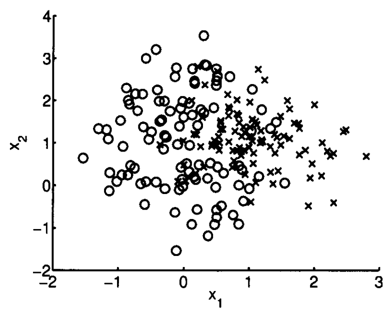
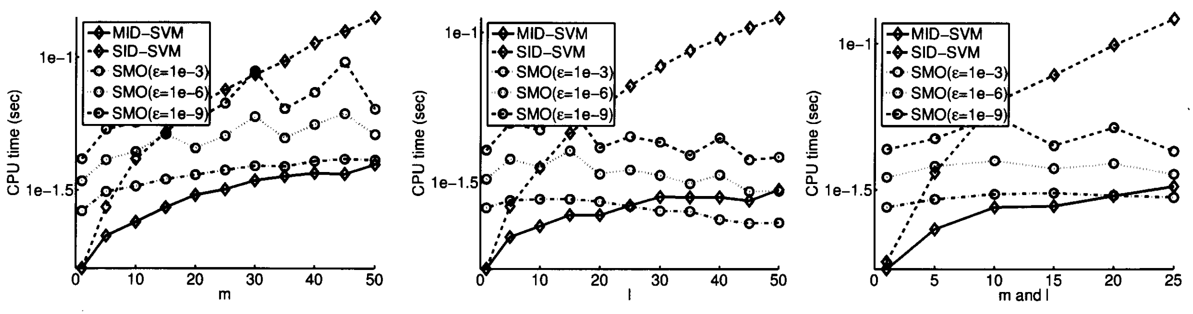


Figure 3.4: Artificial data set for incremental decremental learning. For graphical simplicity, we plot only a part of the data points. The cross points are generated from a mixture of two Gaussians while the circle points come from a single Gaussian. Two classes have equal prior probabilities.

where,

$$\boldsymbol{\mu}_1^{(1)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \ \Sigma_1^{(1)} = \begin{bmatrix} 0.5 & -0.1 \\ -0.1 & 0.5 \end{bmatrix},$$
$$\boldsymbol{\mu}_2^{(1)} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \ \Sigma_2^{(1)} = \begin{bmatrix} 0.5 & 0.1 \\ 0.1 & 0.5 \end{bmatrix},$$
$$\boldsymbol{\mu}^{(2)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ \Sigma^{(2)} = \begin{bmatrix} 0.5 & -0.1 \\ -0.1 & 0.5 \end{bmatrix}.$$

Two classes have equal prior probabilities, and Fig.3.4 shows generated data points. The size of initial data points is $n = 500$. We set the regularization parameter $C = 10$ and kernel parameter $\gamma = 1$. As discussed, adding or removing the data points with $\alpha_i = 0$ at optimal can be performed with almost no cost. Thus, to make clear comparison, we restrict the adding and/or removing points as those with $\alpha_i = C$ at optimal. Fig. 3.5 shows the log plot of the CPU time. We examined several scenarios: (a) adding $m \in \{1, \cdots, 50\}$ data points, (b) removing $\ell \in \{1, \cdots, 50\}$ data points, (c) adding $m \in \{1, \cdots, 25\}$ data points and removing $\ell \in \{1, \cdots, 25\}$ data points simultaneously. The horizontal axis is the number of adding and/or removing data points. When $m = 1$ or $\ell = 1$, SID-SVM and MID-SVM are identical. We see that MID-SVM is significantly faster than SID-SVM when m or ℓ is more than 1. The relative difference of SID-SVM and MID-SVM grows as the m and/or ℓ increase because MID-SVM can add or remove multiple data points simultaneously while SID-SVM merely iterates the algorithm $m + \ell$ times. In this experimental setting, the CPU time of SMO does not change largely because m and ℓ are relatively smaller than n. Fig. 3.6 shows the number of breakpoints of SID-SVM and MID-SVM along with the theoretical number

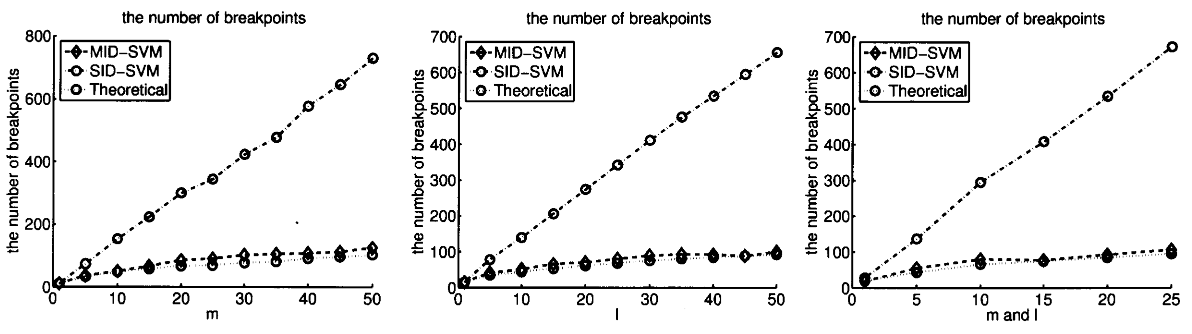


(a) Adding m data points.

(b) Removing ℓ data points.

(c) Adding m data points and removing ℓ data points, simultaneously ($m = \ell$).

Figure 3.5: Log plot of the CPU time (artificial data set)



(a) Adding m data points.

(b) Removing ℓ data points.

(c) Adding m data points and removing ℓ data points, simultaneously ($m = \ell$).

Figure 3.6: The number of breakpoints (artificial data set)

of breakpoints of the MID-SVM in Section 3.3.3 (e.g., for scenario (a), the number of breakpoints of SID-SVM multiplied by $\sqrt{m}/m$). The results are very close to the theoretical one.

3.4.2 Real Data

We also used real data sets to evaluate efficiency of our proposed method. Table 3.1 shows data sets statistics. These data sets are obtainable from LIBSVM site [25]. We obtained scaled version from the site and randomly extracted subset of original data set. In this experiments, we compare the CPU time of adding 10 data points and removing 10 data points simultaneously. We investigated various settings of the regularization

Table 3.1: Data sets for incremental decremental learning
(p is the number of features)

Data set	n	p
diabetes	500	8
svmguide3	1000	21
a2a	2000	123
covtype	2000	54
ijcnn1	4000	22

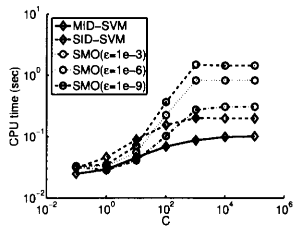
parameter $C \in \{10^{-1}, 10^{0}, \cdots, 10^{5}\}$ and kernel parameter $\gamma \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$. Since we do not need to do anything when the corresponding parameters are 0, we chose adding or removing data points whose optimal parameters are not 0.

Figs. 3.7-3.11 show CPU time of each data set. Each figure has 4 plots corresponding to different settings of kernel parameter γ. The horizontal axis of each plot is the regularization parameter C.

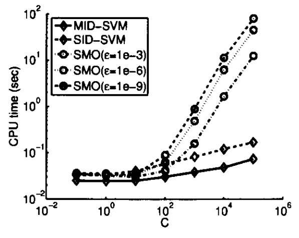
In many cases, our proposed algorithm is faster than the others, especially when C is large and γ is small. However, when γ is relatively large, MID-SVM is sometimes slower than the SMO (e.g., see Fig. 3.9(a)).

Unlike previous experiments, our algorithm did not take the shortest path in adding data points. However, in all results in Fig. 3.7-3.10, MID-SVM was much faster than SID-SVM. In Fig. 3.13, we compared the shortest path $\boldsymbol{\Delta\alpha}_{\mathcal{A}} = \eta(\boldsymbol{\alpha}^{*}_{\mathcal{A}} - \boldsymbol{\alpha}_{\mathcal{A}})$ with our path $\boldsymbol{\Delta\alpha}_{\mathcal{A}} = \eta(C\mathbf{1} - \boldsymbol{\alpha}_{\mathcal{A}})$. Here, $\boldsymbol{\alpha}^{*}_{\mathcal{A}}$ denotes optimal value of $\boldsymbol{\alpha}_{\mathcal{A}}$ (of course, we usually do not know this value beforehand). We used diabetes data set and set $C = 1000, \gamma = 1$. In this case, $\boldsymbol{\alpha}^{*}_{\mathcal{A}}$ becomes far away from $\boldsymbol{\alpha}^{*}_{\mathcal{A}} = C\mathbf{1}$. Fig. 3.13(a) shows an example: histogram of $\alpha_i, i \in \mathcal{A}$, when $m = 50$. We added $m \in \{1, \cdots, 50\}$ data points and took the average of 10 runs. In Fig. 3.13(b) and 3.13(c), MID-SVM(S) means MID-SVM with the shortest path. Although MID-SVM(S) is faster than usual MID-SVM in Fig. 3.13(b), MID-SVM is substantially faster than SID-SVM. Fig. 3.13(c) shows the number of breakpoints of each algorithm. We see that the number of breakpoints of MID-SVM(S) are close to that in the (ideal) shortest path.

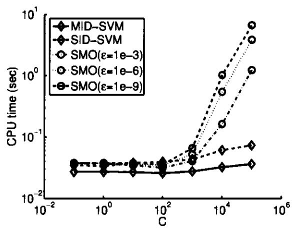
In MID-SVM and SID-SVM, when the size of set $\mathcal{M}$ is large, the computational cost of solving associated linear systems becomes large. Fig. 3.12 shows the sizes of $\mathcal{M}, \mathcal{O}$ and $\mathcal{I}$ for a2a data set. From Fig. 3.9 and Fig. 3.12(a), we see that the CPU time of update algorithm is large when $|\mathcal{M}|$ is large. In our experiments, $|\mathcal{M}|$ sometimes



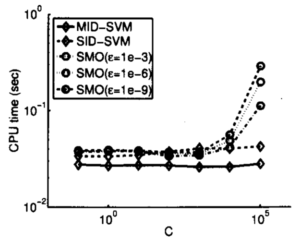
(a) $\gamma = 10^0$



(b) $\gamma = 10^{-1}$

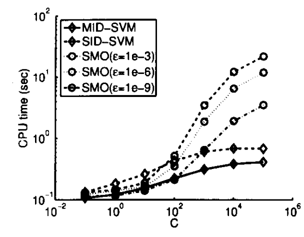


(c) $\gamma = 10^{-2}$

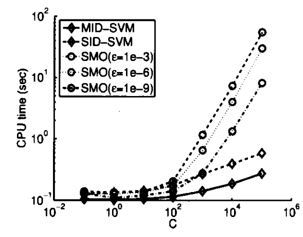


(d) $\gamma = 10^{-3}$

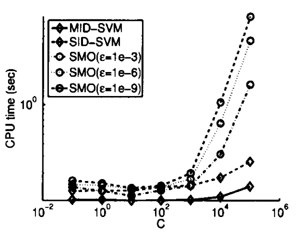
Figure 3.7: Log plot of the CPU time (diabetes data set)



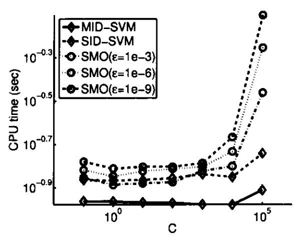
(a) $\gamma = 10^0$



(b) $\gamma = 10^{-1}$



(c) $\gamma = 10^{-2}$

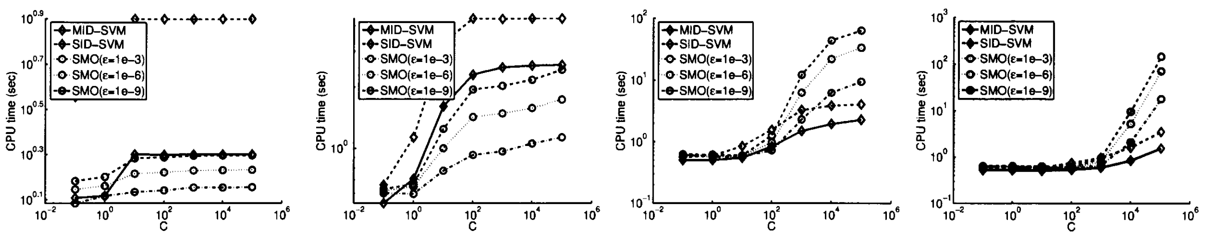


(d) $\gamma = 10^{-3}$

Figure 3.8: Log plot of the CPU time (svmguide3 data set)

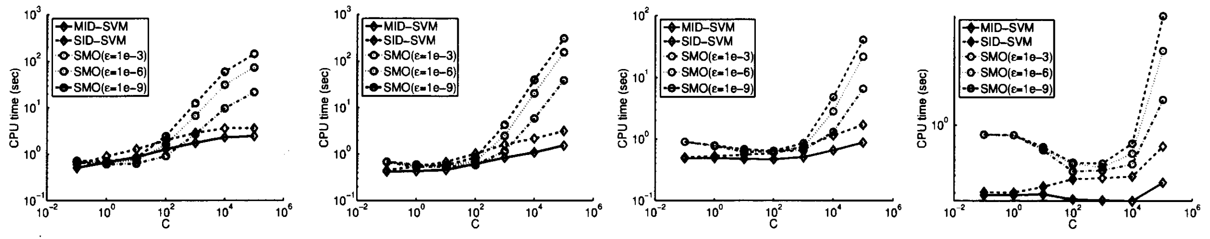
becomes large when both C and γ were large. On the other hand, it is well known that the computational cost of the SMO algorithm becomes large when C is large [16]. We see this in our results.

We also compared the numerical accuracy of the solutions in three algorithms by comparing KKT optimality violation. The accuracy of SMO algorithm can be explicitly specified by arbitrary setting the termination criterion tolerance ε. The accuracy of update algorithm (both SID-SVM and MID-SVM) depends on the accuracy of the linear system solution. Fig. 3.14 shows the maximum violation of KKT conditions on diabetes data set ($\gamma = 1$). We see the accuracy of the SMO does not change with C. The violation of MID-SVM becomes large when C is large. In our implementation, real number is represented as double precision floating-point number (about 15 digit precision). In many cases, the number of digits of some α_i are same as or close to the number of digits of C. Considering these facts, the accuracy of MID-SVM is close to the floating-point accuracy limitation. However, in update algorithm, the computational errors may be accumulated in updating Cholesky factor, especially when the matrix is close to singular (In this paper, since we add positive constant to diagonal of kernel matrix, this problem



(a) $\gamma = 10^0$ (b) $\gamma = 10^{-1}$ (c) $\gamma = 10^{-2}$ (d) $\gamma = 10^{-3}$

Figure 3.9: Log plot of the CPU time (a2a data set)



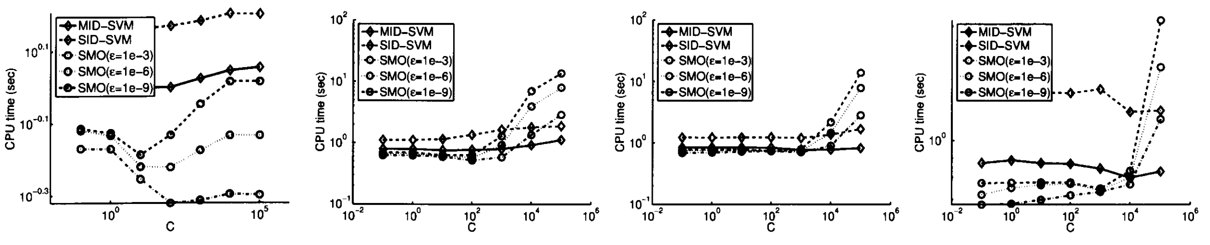
(a) $\gamma = 10^0$ (b) $\gamma = 10^{-1}$ (c) $\gamma = 10^{-2}$ (d) $\gamma = 10^{-3}$

Figure 3.10: Log plot of the CPU time (covtype data set)

did not occur). We can avoid such accumulation using *refresh strategy*: by re-calculating the matrix from scratch, for example, every 100 steps (However, in this case, we need $O(|\mathcal{M}|^3)$ computations in every 100 steps).

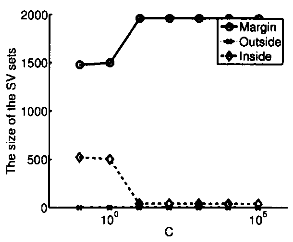
3.4.3 Application to Online Time Series Learning

The SVM has been applied to many time series problems and has shown good performance (e.g. [28,64,84,108]). We applied the proposed algorithm to an online time series problem, in which we update the model when some new observations arrive (adding the

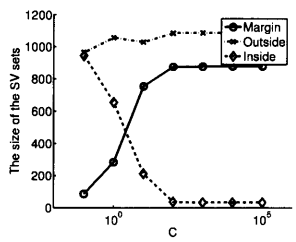


(a) $\gamma = 10^0$ (b) $\gamma = 10^{-1}$ (c) $\gamma = 10^{-2}$ (d) $\gamma = 10^{-3}$

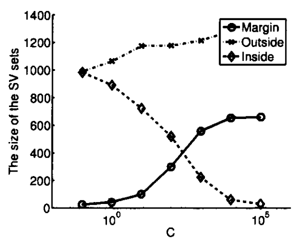
Figure 3.11: Log plot of the CPU time (ijcnn data set)



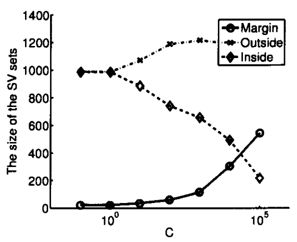
(a) $\gamma = 10^0$



(b) $\gamma = 10^{-1}$

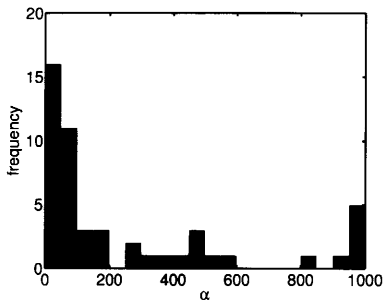


(c) $\gamma = 10^{-2}$

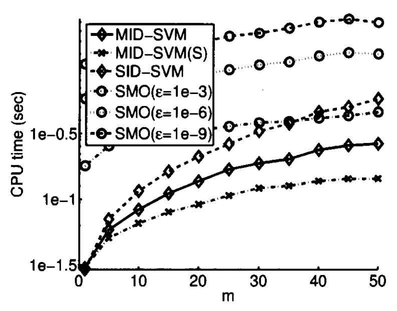


(d) $\gamma = 10^{-3}$

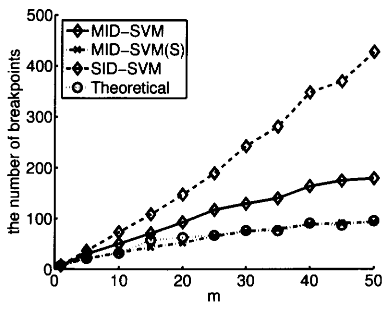
Figure 3.12: The size of the SV sets (a2a data set)



(a) Histogram of $\alpha_i, i \in \mathcal{A}$



(b) Log plot of CPU time



(c) The number of breakpoints

Figure 3.13: Comparison of the shortest path with $\Delta\alpha_i = C, i \in \mathcal{A}$ (diabetes data set, $C = 1000, \gamma = 1$)

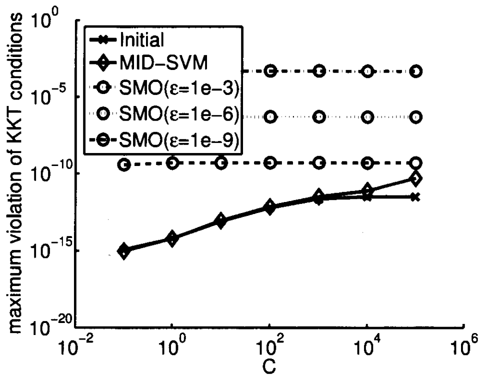


Figure 3.14: Maximum violation of KKT conditions (diabetes data sets, $\gamma = 1$)

new ones and removing the obsolete ones). We use the following two data sets:

- Beer data set: This set consists of monthly beer production records in Australia from Jan 1956 to Aug 1995. The task is to predict whether the production in the next month increases or decreases from the previous 12 months production records

($\boldsymbol{x} \in \mathbb{R}^{12}$). The size of initial data points is $n = 451$ and we set $m = \ell = 12$.

- Fisher river data set: In this data set, the task is to predict whether the mean daily flow of the river increases or decreases using the previous 7 days temperature, precipitation and flow ($\boldsymbol{x}_i \in \mathbb{R}^{21}$). This data set contains the observations from Jan 1 1988 to Dec 31 1991. The size of the initial data points is $n = 1423$ and we set $m = \ell = 30$ (about a month).

Beer data set is obtainable from Time Series Data Library [49]. Fisher river data set is available at StatLib [82]. We normalized each dimension of $\boldsymbol{x}$ to $[0, 1]$. We add new m data points and remove old ℓ data points. Unlike previous two experiments, we did not choose adding or removing data points by its parameter at optimality (the corresponding parameters may be 0).

Fig.3.15-3.18 shows the elapsed CPU times and 10-fold cross-validation error of each setting. Fig. 3.15 and Fig. 3.17 show that our algorithm is faster than the others, especially in large C. Cross-validation error in Fig. 3.16 and Fig. 3.18 indicate that the relative computational cost of our proposed algorithm is especially low for the hyperparameters with good generalization performances in these application problems.

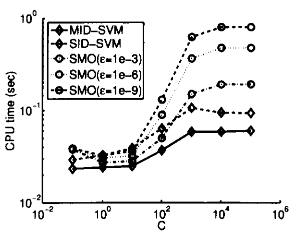
3.4.4 Application to Cross-Validation

Cross-Validation (CV) [104] is a commonly used technique to estimate the generalization performance of a model. The CV procedure can be naturally implemented by decremental algorithm. Like previous works [24, 79], we initially calculate parameters for the entire data set. Then, in the leave-ℓ-out CV (equivalent to n/ℓ-fold CV), we remove ℓ data points from the initial parameters at each fold.

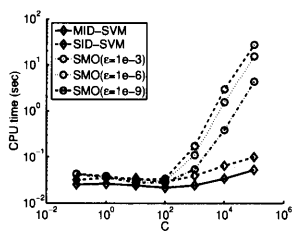
Fig.3.19 shows the CPU time of the leave-ℓ-out CV for the diabetes data set (Table 3.1). The horizontal axis of each plot indicates the number of leaving out data point ℓ. We examined several settings with $\ell \in \{1, 5, 10, 20, 50\}$[5] and $C \in \{10^0, 10^1, 10^2, 10^3\}$. All results do not include the CPU time of calculating initial solution (parameters for all data points). RBF kernel parameter is $\gamma = 0.1$ and all algorithms uses common kernel cache during n/ℓ-fold learning.

In Fig.3.19, MID-SVM was faster than the other algorithms in these settings. Note, in SID-SVM, that the entire n/ℓ-fold learning always requires n decremental operations

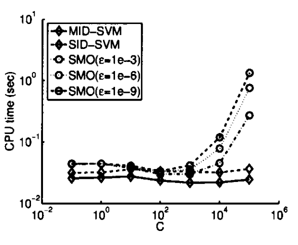
[5]Each of which corresponds to $500, 100, 50, 25, 10$-fold CV.



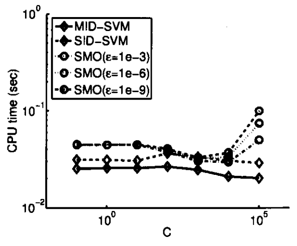
(a) $\gamma = 10^0$



(b) $\gamma = 10^{-1}$

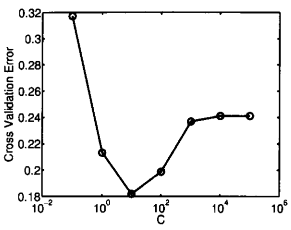


(c) $\gamma = 10^{-2}$

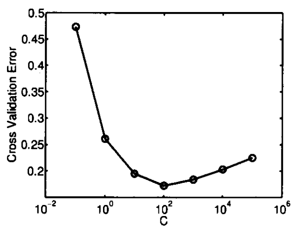


(d) $\gamma = 10^{-3}$

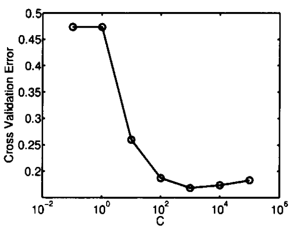
Figure 3.15: Log plot of the CPU time (Beer data set)



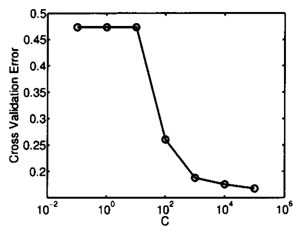
(a) $\gamma = 10^0$



(b) $\gamma = 10^{-1}$

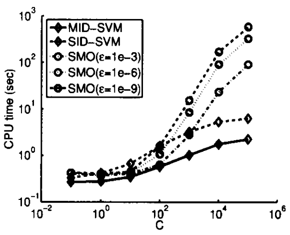


(c) $\gamma = 10^{-2}$

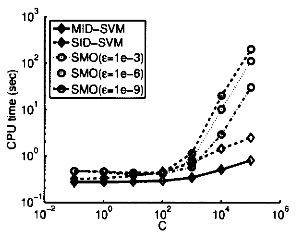


(d) $\gamma = 10^{-3}$

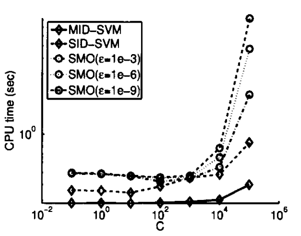
Figure 3.16: Cross-validation error (Beer data set)



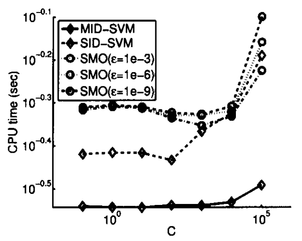
(a) $\gamma = 10^0$



(b) $\gamma = 10^{-1}$

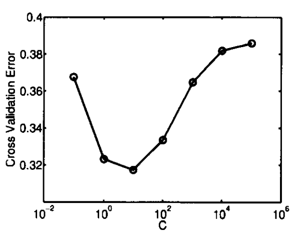


(c) $\gamma = 10^{-2}$

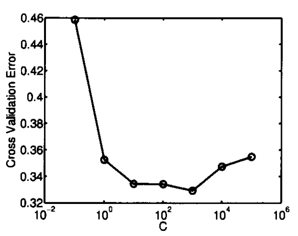


(d) $\gamma = 10^{-3}$

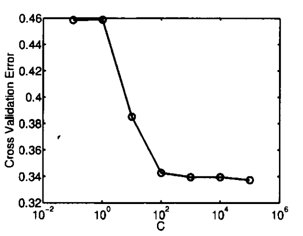
Figure 3.17: Log plot of the CPU time (Fisher river data set)



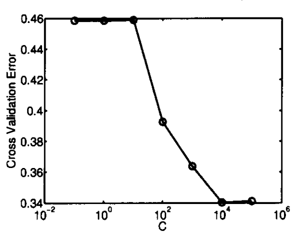
(a) $\gamma = 10^0$



(b) $\gamma = 10^{-1}$

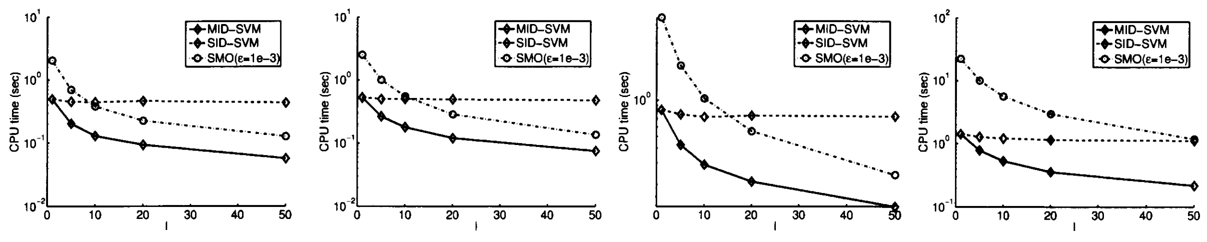


(c) $\gamma = 10^{-2}$



(d) $\gamma = 10^{-3}$

Figure 3.18: Cross-validation error (Fisher river data set)



(a) $C = 10^0$ (b) $C = 10^1$ (c) $C = 10^2$ (d) $C = 10^3$

Figure 3.19: Log plots of the CPU time of leave-ℓ-out CV

because it needs ℓ times decremental for each fold, Therefore, for any ℓ, the computational cost of SID-SVM is same as that needed in leave-one-out CV ($\ell = 1$).

3.5 Multiple Update for SVR

Here, we briefly introduce a formulation of multiple incremental decremental learning for the SVR. The derivation is almost the same in the classification case. This is an extension of our previous work which is for acceleration of the cross-validation procedure of the SVR [58].

From the definition (2.13b) and the KKT conditions, the points $i \in \mathcal{E}$ must satisfy $|y_i - f(\boldsymbol{x}_i)| = \varepsilon$. This equation can be re-written as

$$|y_i - \sum_{j \in \mathcal{E}} \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{j \in \mathcal{O}} \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - b| = \varepsilon, \; i \in \mathcal{E}. \tag{3.15}$$

Similarly, the equality constraint of (2.11) is re-written as

$$\sum_{i \in \mathcal{E}} \alpha_i + \sum_{i \in \mathcal{O}} \alpha_i = 0. \tag{3.16}$$

Using vector notations: $\boldsymbol{y} = [y_1, \cdots, y_n]^\top$, $\boldsymbol{f} = [f(\boldsymbol{x}_1), \cdots, f(\boldsymbol{x}_n)]^\top$, $\boldsymbol{s} = \text{sign}(\boldsymbol{y} - \boldsymbol{f})$, we can rewrite (3.15) and (3.16) as the following linear system of equations:

$$\begin{bmatrix} 0 & \boldsymbol{1}^\top \\ \boldsymbol{1} & \boldsymbol{K}_{\mathcal{E}} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha}_{\mathcal{E}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{1}^\top \boldsymbol{\alpha}_{\mathcal{O}} \\ \boldsymbol{K}_{\mathcal{E},\mathcal{O}} \boldsymbol{\alpha}_{\mathcal{O}} - \boldsymbol{y}_{\mathcal{E}} + \varepsilon \boldsymbol{s}_{\mathcal{E}} \end{bmatrix} = \boldsymbol{0}, \tag{3.17}$$

where $\boldsymbol{K}$ is a kernel matrix $\boldsymbol{K}_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and we abbreviate $\boldsymbol{K}_{\mathcal{E},\mathcal{E}}$ as $\boldsymbol{K}_{\mathcal{E}}$.

As in the case of classification, let us denote the index set of new adding data points and removing data points as

$$\begin{aligned} \mathcal{A} &= \{n+1, \ldots, n+m\} \\ \mathcal{R} &\subset \{1, \ldots, n\}, \end{aligned}$$

where $|\mathcal{R}| = \ell$ and we remove these index sets from $\mathcal{E}, \mathcal{I}$ and $\mathcal{O}$. Let $\boldsymbol{\Delta\alpha}_{\mathcal{A}}$ and $\boldsymbol{\Delta\alpha}_{\mathcal{R}}$ denote the change of $\boldsymbol{\alpha}_{\mathcal{A}}$ and $\boldsymbol{\alpha}_{\mathcal{R}}$ respectively, then we need

$$\begin{bmatrix} 0 & \mathbf{1}^\top \\ \mathbf{1} & \boldsymbol{K}_{\mathcal{E}} \end{bmatrix} \begin{bmatrix} \Delta b \\ \boldsymbol{\Delta\alpha}_{\mathcal{E}} \end{bmatrix} + \begin{bmatrix} \mathbf{1}^\top & \mathbf{1}^\top \\ \boldsymbol{K}_{\mathcal{E},\mathcal{A}} & \boldsymbol{K}_{\mathcal{E},\mathcal{R}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta\alpha}_{\mathcal{A}} \\ \boldsymbol{\Delta\alpha}_{\mathcal{R}} \end{bmatrix} = \mathbf{0}, \tag{3.18}$$

in order to let (3.17) hold for $\alpha_i, i \in \mathcal{E}$ and b. From the definitions of index sets in (2.13c), (2.13b), and (2.13a), following inequality constraints must also be satisfied:

$$\begin{array}{rlll} & s_i(\alpha_i + \Delta\alpha_i) & \leq C, & i \in \mathcal{E}, \\ & s_i(\alpha_i + \Delta\alpha_i) & \geq 0, & i \in \mathcal{E}, \\ & s_i(y_i - f(\boldsymbol{x}_i) - \Delta f(\boldsymbol{x}_i)) & \geq \varepsilon, & i \in \mathcal{O}, \\ -\varepsilon \leq & y_i - f(\boldsymbol{x}_i) - \Delta f(\boldsymbol{x}_i) & \leq \varepsilon, & i \in \mathcal{I}, \end{array} \tag{3.19}$$

where $s_i = \mathrm{sign}(y_i - f(\boldsymbol{x}_i))$. While we satisfy these conditions (3.18) and (3.19), parameters keep satisfying optimality conditions. The direction of $\boldsymbol{\Delta\alpha}_{\mathcal{A}}$ and $\boldsymbol{\Delta\alpha}_{\mathcal{R}}$ are

$$\begin{aligned} \boldsymbol{\Delta\alpha}_{\mathcal{A}} &= \eta(C\mathbf{1} - \boldsymbol{\alpha}_{\mathcal{A}}), \\ \boldsymbol{\Delta\alpha}_{\mathcal{R}} &= -\eta\boldsymbol{\alpha}_{\mathcal{R}}, \end{aligned}$$

where η is a step length.

Using these formulas, multiple update of the SVR can be constructed in the same manner as the classification case.

3.6 Conclusion

In this chapter, we have developed multiple incremental decremental algorithms for the SVM. Unlike previous formulations [24,33,68,79,80], our approach can add and/or remove multiple data points simultaneously. We experimentally showed the efficiency of multiple update algorithm using artificial and several common benchmark data sets. Moreover, we derived the ratio of the number of breakpoints on some reasonable assumptions. We also verified this ratio is actually achieved in some practical settings. As applications of our approach, we provided accelerations of the online time series learning and the cross validation procedure. Results showed our approach is faster than single update algorithm and hot start of the SMO algorithm.

Chapter 4

Multi-Parametric Solution-Path for Weighted SVMs

An *instance-weighted* variant of the support vector machine (SVM) has attracted considerable attention recently since they are useful in various machine learning tasks such as non-stationary data analysis, heteroscedastic data modeling, transfer learning, learning to rank, and transduction. An important challenge in these scenarios is to overcome the computational bottleneck—instance weights often change dynamically or adaptively, and thus the weighted SVM solutions must be repeatedly computed. In this chapter, we develop an algorithm that can efficiently and exactly update the weighted SVM solutions for arbitrary change of instance weights. Technically, this contribution can be regarded as an extension of the conventional *solution-path* algorithm for a single regularization parameter to multiple instance-weight parameters. However, this extension gives rise to a significant problem that *breakpoints* (at which the solution path turns) have to be identified in high-dimensional space. To facilitate this, we introduce a parametric representation of instance weights. We also provide a geometric interpretation in weight space using a notion of *critical region*: a polyhedron in which the current affine solution remains to be optimal. Then we find breakpoints at intersections of the solution path and boundaries of polyhedrons. Through extensive experiments on various practical applications, we demonstrate the usefulness of the proposed algorithm. The discussion in this chapter has appeared in [56].

4.1 Instance-weighted Learning

The most fundamental principle of machine learning would be the *empirical risk minimization*, i.e., the sum of empirical losses over training instances is minimized:

$$\min \sum_i L_i,$$

where L_i denotes the empirical loss for the i-th training instance. This empirical risk minimization approach was proved to produce *consistent* estimators [112]. On the other hand, one may also consider an *instance-weighted* variant of empirical risk minimization:

$$\min \sum_i C_i L_i,$$

where C_i denotes the weight for the i-th training instance. This weighted variant plays an important role in various machine learning tasks:

- **Non-stationary data analysis:** When training instances are provided in a sequential manner under changing environment, smaller weights are often assigned to older instances for imposing some 'forgetting' effect [22,85].

- **Heteroscedastic data modeling:** A supervised learning setup where the noise level in output values depends on input points is said to be *heteroscedastic*. In heteroscedastic data modeling, larger weights are often assigned to instances with smaller noise variance [63]. The traditional *Gauss-Markov theorem* [1] forms the basis of this idea.

- **Covariate shift adaptation, transfer learning, and multi-task learning:** A supervised learning situation where training and test *inputs* follow different distributions is called covariate shift. Under covariate shift, using the *importance* (the ratio of the test and training input densities) as instance weights assures the consistency of estimators [101]. Similar importance-weighting ideas can be applied also to transfer learning (where data in one domain is transferred to another domain) [51] and multi-task learning (where multiple learning problems are solved simultaneously by sharing training instances) [13].

- **Learning to rank and ordinal regression:** The goal of ranking (a.k.a. ordinal regression) is to give an ordered list of items based on their relevance [48,77]. In practical ranking tasks such as information retrieval, users are often not interested in the entire ranking list, but only in the top few items. In order to improve the prediction accuracy in the top of the list, larger weights are often assigned to higher-ranked items [119].

- **Transduction and semi-supervised learning:** Transduction is a supervised learning setup where the goal is not to learn the entire input-output mapping, but only to estimate the output values for pre-specified unlabeled input points [112]. A popular approach to transduction is to label the unlabeled samples using the current estimator, and then modify the estimator using the 'self-labeled' samples [52,91]. In this procedure, smaller weights are usually assigned to the self-labeled samples than the originally-labeled samples due to their high uncertainty.

A common challenge in the research of instance-weighted learning has been to overcome the computational issue. In many of these tasks, instance weights often change dynamically or adaptively, and thus the instance-weighted solutions must be repeatedly computed. For example, in on-line learning, every time when a new instance is observed, all the instance weights must be updated in such a way that newer instances have larger weights and older instances have smaller weights. Model selection in instance-weighted learning also poses a considerable computational burden. In many of the above scenarios, we only have qualitative knowledge about instance weights. For example, in the aforementioned ranking problem, we only know that higher-ranked items should have larger weights than lower-ranked items, but it is often difficult to know how large or small these weights should be. The problem of selecting the optimal weighting patterns is an instance of model selection, and many instance-weighted solutions with various weighting patterns must be computed in the model selection phase. The goal of this chapter is to alleviate the computational bottleneck of instance-weighted learning.

The SVM minimizes a regularized empirical risk:

$$\min R + C \sum_i L_i,$$

where R is a regularization term and $C \geq 0$ controls the trade-off between the regularization effect and the empirical risk minimization. We consider an instance-weighted variant of SVM, which we refer to as the *weighted SVM* (WSVM) [74,75,120]:

$$\min R + \sum_i C_i L_i.$$

For ordinary SVM, the *solution path algorithm* was proposed [47], which allows efficient computation of SVM solutions for all C by utilizing the piecewise-linear structure of the solutions w.r.t. C. This technique is known as *parametric programming* in the optimization community [2, 11, 12, 92], and has been applied to various machine learning tasks recently [5, 7, 33, 38, 42, 46, 53, 69, 71–73, 78, 95, 102, 107, 111, 117, 121, 124]; the *incremental-decremental SVM algorithm*, which efficiently follows the piecewise-linear solution path when some training instances are added or removed from the training set, is also based on the same parametric programming technique [24, 57, 68].

The solution path algorithms described above have been developed for problems with a *single* hyper-parameter. Recently, attention has been paid to studying solution-path tracking in two-dimensional hyper-parameter space. For example, [117] developed a path-following algorithm for regularization parameter C and an insensitive zone thickness ε in *support vector regression* [81,84,113]. [93] studied a path-following algorithm for regularization parameter λ and quantile parameter τ in *kernel quantile regression* [106]. However, these works are highly specialized to specific problem structure of bivariate path-following, and it is not straightforward to extend them to more than two hyper-parameters. Thus, the existing approaches may not be applicable to path-following of WSVM, which contains n-dimensional instance-weight parameters $\boldsymbol{c} = [C_1, \dots, C_n]^\top$, where n is the number of training instances.

In order to go beyond the limitation of the existing approaches, we derive a general solution path algorithm for efficiently computing the solution path of *multiple* instance-weight parameters $\boldsymbol{c}$ in WSVM. This extension involves a significant problem that *breakpoints* (at which the solution path turns) have to be identified in high-dimensional space. To facilitate this, we introduce a parametric representation of instance weights. We also provide a geometric interpretation in weight space using a notion of *critical region* from the studies of *multi-parametric programming* [41,89]. A critical region is a polyhedron in which the current *affine* solution remains to be optimal (see Fig. 4.1). This enables us to find breakpoints at intersections of the solution path and the boundaries of polyhedrons.

4.2 Problem Formulation

In this section, we review the definition of the *weighted support vector machine* (WSVM) and its optimality conditions. For the moment, we focus on binary classification scenarios. Later in Section 4.5, we extend our discussion to more general scenarios such as regression, ranking, and transduction.

4.2.1 WSVM

Let us consider a binary classification problem. Denote n training instances as $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^p$ is the input and $y_i \in \{-1, +1\}$ is the output label.

SVM [15, 31] is a learning algorithm of a linear decision boundary

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \Phi(\boldsymbol{x}) + b$$

in a feature space $\mathcal{F}$, where $\Phi : \mathcal{X} \to \mathcal{F}$ is a map from the input space $\mathcal{X}$ to the feature space $\mathcal{F}$, $\boldsymbol{w} \in \mathcal{F}$ is a coefficient vector, $b \in \mathbb{R}$ is a bias term, and $^\top$ denotes the transpose. The parameters $\boldsymbol{w}$ and b are learned as

$$\min_{\boldsymbol{w}, b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C \sum_{i=1}^n [1 - y_i f(\boldsymbol{x}_i)]_+, \tag{4.1}$$

where $\frac{1}{2}\|\boldsymbol{w}\|_2^2$ is the regularization term, $\|\cdot\|$ denotes the Euclidean norm, C is the trade-off parameter, and

$$[z]_+ = \max\{0, z\}.$$

$[1 - y_i f(\boldsymbol{x}_i)]_+$ is the so-called *hinge-loss* for the i-th training instance.

WSVM is an extension of the ordinary SVM so that each training instance possesses its own weight [74, 75, 120]:

$$\min_{\boldsymbol{w}, b} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^n C_i [1 - y_i f(\boldsymbol{x}_i)]_+, \tag{4.2}$$

where C_i is the weight for the i-th training instance. WSVM includes the ordinary SVM as a special case when $C_i = C$ for $i = 1, \ldots, n$. The primal optimization problem (4.2) is expressed as the following quadratic program:

$$\begin{aligned} \min_{\boldsymbol{w}, b, \{\xi_i\}_{i=1}^n} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^n C_i \xi_i, \\ \text{s.t.} \quad & y_i f(\boldsymbol{x}_i) \geq 1 - \xi_i, \ \xi_i \geq 0, \ i = 1, \ldots, n. \end{aligned} \tag{4.3}$$

The goal of this paper is to derive an algorithm that can efficiently compute the sequence of WSVM solutions for arbitrary weighting patterns of $\boldsymbol{c} = [C_1, \ldots, C_n]^\top$.

4.2.2 Optimization in WSVM

Here we review basic optimization issues of WSVM which are used in the following section.

Introducing Lagrange multipliers $\alpha_i \geq 0$ and $\rho_i \geq 0$, we can write the *Lagrangian* of (4.3) as

$$L = \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{n} C_i\xi_i - \sum_{i=1}^{n} \alpha_i\{y_i f(\boldsymbol{x}_i) - 1 + \xi_i\} - \sum_{i=1}^{n} \rho_i\xi_i. \tag{4.4}$$

Setting the derivatives of the above Lagrangian w.r.t. the primal variables $\boldsymbol{w}$, b, and ξ_i to zero, we obtain

$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{w}} &= \boldsymbol{0} \;\Leftrightarrow\; \boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \Phi(\boldsymbol{x}_i),\\
\frac{\partial L}{\partial b} &= 0 \;\Leftrightarrow\; \sum_{i=1}^{n} \alpha_i y_i = 0,\\
\frac{\partial L}{\partial \xi_i} &= 0 \;\Leftrightarrow\; \alpha_i = C_i - \rho_i,\; i = 1, \ldots, n,
\end{aligned}$$

where $\boldsymbol{0}$ denotes the vector with all zeros. Substituting these equations into (4.4), we arrive at the following dual problem:

$$\begin{aligned}
\max_{\{\alpha_i\}_{i=1}^{n}} \quad & -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j Q_{ij} + \sum_{i=1}^{n} \alpha_i\\
\text{s.t.} \quad & \sum_{i=1}^{n} y_i\alpha_i = 0, \; 0 \leq \alpha_i \leq C_i,
\end{aligned} \tag{4.5}$$

where $Q_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The discriminant function $f : \mathcal{X} \to \mathbb{R}$ is represented in the following form:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b.$$

KKT conditions are summarized as follows:

$$y_i f(\boldsymbol{x}_i) \geq 1, \quad \text{if} \quad \alpha_i = 0, \tag{4.6a}$$

$$y_i f(\boldsymbol{x}_i) = 1, \quad \text{if} \quad 0 < \alpha_i < C_i, \tag{4.6b}$$

$$y_i f(\boldsymbol{x}_i) \leq 1, \quad \text{if} \quad \alpha_i = C_i, \tag{4.6c}$$

$$\sum_{i=1}^{n} y_i\alpha_i = 0. \tag{4.6d}$$

We define the following three index sets for later use:

$$\mathcal{O} = \{i \mid \alpha_i = 0\}, \tag{4.7a}$$

$$\mathcal{M} = \{i \mid 0 < \alpha_i < C_i\}, \tag{4.7b}$$

$$\mathcal{I} = \{i \mid \alpha_i = C_i\}. \tag{4.7c}$$

The geometric interpretation of these sets is also provided by Fig. 2.1.

4.3 Solution-Path Algorithm for WSVM

The path-following algorithm for the ordinary SVM [47] computes the entire solution path for the single regularization parameter C. In this section, we develop a path-following algorithm for the vector of weights $\boldsymbol{c} = [C_1, \ldots, C_n]^\top$. Our proposed algorithm keeps track of the optimal α_i and b when the weight vector $\boldsymbol{c}$ is changed.

4.3.1 Analytic Expression of WSVM Solutions

Let

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}, \ \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \ \text{and } \boldsymbol{Q} = \begin{bmatrix} Q_{11} & \cdots & Q_{1n} \\ \vdots & \ddots & \vdots \\ Q_{n1} & \cdots & Q_{nn} \end{bmatrix}.$$

Then, using the index sets (4.7b) and (4.7c), we can expand one of the KKT conditions, (4.6b), as

$$\boldsymbol{Q}_{\mathcal{M}}\boldsymbol{\alpha}_{\mathcal{M}} + \boldsymbol{Q}_{\mathcal{M},\mathcal{I}}\boldsymbol{c}_{\mathcal{I}} + \boldsymbol{y}_{\mathcal{M}}b = \mathbf{1}, \tag{4.8}$$

where $\mathbf{1}$ denotes the vector with all ones. Similarly, another KKT condition (4.6d) is expressed as

$$\boldsymbol{y}_{\mathcal{M}}^\top\boldsymbol{\alpha}_{\mathcal{M}} + \boldsymbol{y}_{\mathcal{I}}^\top\boldsymbol{c}_{\mathcal{I}} = 0. \tag{4.9}$$

Let

$$\boldsymbol{M} = \begin{bmatrix} 0 & \boldsymbol{y}_{\mathcal{M}}^\top \\ \boldsymbol{y}_{\mathcal{M}} & \boldsymbol{Q}_{\mathcal{M}} \end{bmatrix}.$$

Then (4.8) and (4.9) can be compactly expressed as the following system of $|\mathcal{M}| + 1$ linear equations, where $|\mathcal{M}|$ denotes the number of elements in the set $\mathcal{M}$:

$$\boldsymbol{M}\begin{bmatrix} b \\ \boldsymbol{\alpha}_{\mathcal{M}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{y}_{\mathcal{I}}^\top \\ \boldsymbol{Q}_{\mathcal{M},\mathcal{I}} \end{bmatrix}\boldsymbol{c}_{\mathcal{I}} = \begin{bmatrix} 0 \\ \mathbf{1} \end{bmatrix}. \tag{4.10}$$

Solving (4.10) w.r.t. b and $\boldsymbol{\alpha}_{\mathcal{M}}$, we obtain

$$\begin{bmatrix} b \\ \boldsymbol{\alpha}_{\mathcal{M}} \end{bmatrix} = -\boldsymbol{M}^{-1}\begin{bmatrix} \boldsymbol{y}_{\mathcal{I}}^\top \\ \boldsymbol{Q}_{\mathcal{M},\mathcal{I}} \end{bmatrix}\boldsymbol{c}_{\mathcal{I}} + \boldsymbol{M}^{-1}\begin{bmatrix} 0 \\ \mathbf{1} \end{bmatrix}, \tag{4.11}$$

where we implicitly assumed that $\boldsymbol{M}$ is invertible[1] . Since b and $\boldsymbol{\alpha}_{\mathcal{M}}$ are *affine* w.r.t. $\boldsymbol{c}_{\mathcal{I}}$, we can calculate the change of b and $\boldsymbol{\alpha}_{\mathcal{M}}$ by (4.11) as long as the weight vector $\boldsymbol{c}$ is changed continuously. By the definition of $\mathcal{I}$ and $\mathcal{O}$, the remaining parameters $\boldsymbol{\alpha}_{\mathcal{I}}$ and $\boldsymbol{\alpha}_{\mathcal{O}}$ are merely given by

$$\boldsymbol{\alpha}_{\mathcal{I}} = \boldsymbol{c}_{\mathcal{I}}, \tag{4.12}$$

$$\boldsymbol{\alpha}_{\mathcal{O}} = \boldsymbol{0}. \tag{4.13}$$

A change of the index sets $\mathcal{M}$, $\mathcal{O}$, and $\mathcal{I}$ is called an *event.* As long as no event occurs, the WSVM solutions for all $\boldsymbol{c}$ can be computed by (4.11)–(4.13) since all the KKT conditions (4.6a)–(4.6d) are still satisfied. However, when an event occurs, we need to check the violation of the KKT conditions. Below, we address the issue of event detection when $\boldsymbol{c}$ is changed.

4.3.2 Event Detection

Suppose we want to change the weight vector from $\boldsymbol{c}^{(\text{old})}$ to $\boldsymbol{c}^{(\text{new})}$ (see Fig. 4.1). This can be achieved by moving the weight vector $\boldsymbol{c}^{(\text{old})}$ toward the direction of $\boldsymbol{c}^{(\text{new})} - \boldsymbol{c}^{(\text{old})}$.

Let us write the line segment between $\boldsymbol{c}^{(\text{old})}$ and $\boldsymbol{c}^{(\text{new})}$ in the following parametric form

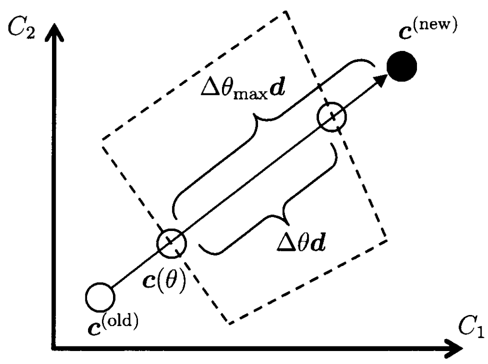
$$\boldsymbol{c}(\theta) = \boldsymbol{c}^{(\text{old})} + \theta\left(\boldsymbol{c}^{(\text{new})} - \boldsymbol{c}^{(\text{old})}\right),\ \theta \in [0, 1],$$

where θ is a parameter. This parametrization allows us to derive a path-following algorithm between arbitrary $\boldsymbol{c}^{(\text{old})}$ and $\boldsymbol{c}^{(\text{new})}$ by considering the change of the solutions when θ is moved from 0 to 1. Suppose we are currently at $\boldsymbol{c}(\theta)$ on the path, and the current solution is $(b, \boldsymbol{\alpha})$. Let

$$\boldsymbol{\Delta c} = \Delta\theta\left(\boldsymbol{c}^{(\text{new})} - \boldsymbol{c}^{(\text{old})}\right),\ \Delta\theta \geq 0, \tag{4.14}$$

where the operator Δ represents the amount of change of each variable from the current value. If $\Delta\theta$ is increased from 0, we may encounter a point at which some of the KKT conditions (4.6a)–(4.6c) do not hold. This can be checked by investigating the following

[1]The invertibility of the matrix $\boldsymbol{M}$ is assured if and only if the submatrix $\boldsymbol{Q}_{\mathcal{M}}$ is positive definite in the subspace $\{\boldsymbol{z} \in \mathbb{R}^{|\mathcal{M}|} \mid \boldsymbol{y}_{\mathcal{M}}^\top \boldsymbol{z} = 0\}$. We assume this technical condition here. A notable exceptional case is that $\mathcal{M}$ is empty—we will discuss how to cope with this case in detail in Section 4.3.3.



C_2
$c^{(\text{new})}$
$\Delta\theta_{\max}\boldsymbol{d}$
$\Delta\theta\boldsymbol{d}$
$\boldsymbol{c}(\theta)$
$\boldsymbol{c}^{(\text{old})}$
C_1

Figure 4.1: The schematic illustration of path-following in the space of $\boldsymbol{c} \in \mathbb{R}^2$. In this plot, the WSVM solution is updated from $\boldsymbol{c}^{(\text{old})}$ to $\boldsymbol{c}^{(\text{new})}$. Suppose we are currently at $\boldsymbol{c}(\theta)$. The vector $\boldsymbol{d}$ represents the update direction $\boldsymbol{c}^{(\text{new})} - \boldsymbol{c}^{(\text{old})}$, and the polygonal region enclosed by dashed lines indicates the current critical region. Although $\boldsymbol{c}(\theta) + \Delta\theta_{\max}\boldsymbol{d}$ seems to directly lead the solution to $\boldsymbol{c}^{(\text{new})}$, the maximum possible update from $\boldsymbol{c}(\theta)$ is $\Delta\theta\boldsymbol{d}$; otherwise the KKT conditions are violated. To go beyond the border of the critical region, we need to update the index sets $\mathcal{M}$, $\mathcal{I}$, and $\mathcal{O}$ to fulfill the KKT conditions.

conditions.

$$\begin{cases} y_i f(\boldsymbol{x}_i) + y_i \Delta f(\boldsymbol{x}_i) \geq 1, & i \in \mathcal{O}, \\ \alpha_i + \Delta\alpha_i > 0, & i \in \mathcal{M}, \\ \alpha_i + \Delta\alpha_i - (C_i + \Delta C_i,) < 0, & i \in \mathcal{M}, \\ y_i f(\boldsymbol{x}_i) + y_i \Delta f(\boldsymbol{x}_i) \leq 1, & i \in \mathcal{I}. \end{cases} \tag{4.15}$$

The set of inequalities (4.15) defines a convex polyhedron, called a *critical region* in the multi-parametric programming literature [89]. The event points lie on the border of critical regions, as illustrated in Fig. 4.1.

We detect an event point by checking the conditions (4.15) along the solution path as follows. Using (4.11), we can express the changes of b and $\boldsymbol{\alpha}_{\mathcal{M}}$ as

$$\begin{bmatrix} \Delta b \\ \boldsymbol{\Delta\alpha}_{\mathcal{M}} \end{bmatrix} = \Delta\theta\boldsymbol{\phi}, \tag{4.16}$$

where

$$\boldsymbol{\phi} = -\boldsymbol{M}^{-1} \begin{bmatrix} \boldsymbol{y}_{\mathcal{I}}^\top \\ \boldsymbol{Q}_{\mathcal{M},\mathcal{I}} \end{bmatrix} (\boldsymbol{c}_{\mathcal{I}}^{(\text{new})} - \boldsymbol{c}_{\mathcal{I}}^{(\text{old})}). \tag{4.17}$$

Furthermore, $y_i \Delta f(\boldsymbol{x}_i)$ is expressed as

$$\begin{aligned} y_i \Delta f(\boldsymbol{x}_i) &= \begin{bmatrix} y_i & \boldsymbol{Q}_{i,\mathcal{M}} \end{bmatrix} \begin{bmatrix} \Delta b \\ \boldsymbol{\Delta\alpha}_{\mathcal{M}} \end{bmatrix} + \boldsymbol{Q}_{i,\mathcal{I}} \boldsymbol{\Delta c}_{\mathcal{I}} \\ &= \Delta\theta\psi_i, \end{aligned} \tag{4.18}$$

where

$$\psi_i = \begin{bmatrix} y_i & \boldsymbol{Q}_{i,\mathcal{M}} \end{bmatrix} \boldsymbol{\phi} + \boldsymbol{Q}_{i,\mathcal{I}}(\boldsymbol{c}_{\mathcal{I}}^{(\text{new})} - \boldsymbol{c}_{\mathcal{I}}^{(\text{old})}). \tag{4.19}$$

Let us denote the elements of the index set $\mathcal{M}$ as

$$\mathcal{M} = \{m_1, \ldots, m_{|\mathcal{M}|}\}.$$

Substituting (4.16) and (4.18) into the inequalities (4.15), we can obtain the maximum step-length with no event occurrence as

$$\Delta\theta = \min_{i\in\{1,\ldots,|\mathcal{M}|\}, j\in\mathcal{I}\cup\mathcal{O}} \left\{ -\frac{\alpha_{m_i}}{\phi_{i+1}}, \frac{C_{m_i} - \alpha_{m_i}}{\phi_{i+1} - d_{m_i}}, \frac{1 - y_j f(\boldsymbol{x}_j)}{\psi_j} \right\}_+, \tag{4.20}$$

where ϕ_i denotes the i-th element of $\boldsymbol{\phi}$ and $d_i = C_i^{(\text{new})} - C_i^{(\text{old})}$. We used $\min_i\{z_i\}_+$ as a simplified notation of $\min_i\{z_i \mid z_i \geq 0\}$. Based on this largest possible $\Delta\theta$, we can compute $\boldsymbol{\alpha}$ and b along the solution path by (4.16).

At the border of the critical region, we need to update the index sets $\mathcal{M}$, $\mathcal{O}$, and $\mathcal{I}$. For example, if α_i ($i \in \mathcal{M}$) reaches 0, we need to move the element i from $\mathcal{M}$ to $\mathcal{O}$. Then the above path-following procedure is carried out again for the next critical region specified by the updated index sets $\mathcal{M}$, $\mathcal{O}$, and $\mathcal{I}$, and this procedure is repeated until $\boldsymbol{c}$ reaches $\boldsymbol{c}^{(\text{new})}$.

4.3.3 Empty Margin

In the above derivation, we have implicitly assumed that the index set $\mathcal{M}$ is not empty—when $\mathcal{M}$ is empty, we can not use (4.16) because $\boldsymbol{M}^{-1}$ does not exist.

When $\mathcal{M}$ is empty, the KKT conditions (4.6) can be re-written as

$$\sum_{j\in\mathcal{I}} Q_{ij}C_j + y_i b \geq 1, \quad i \in \mathcal{O}, \tag{4.21a}$$

$$\sum_{j\in\mathcal{I}} Q_{ij}C_j + y_i b \leq 1, \quad i \in \mathcal{I}, \tag{4.21b}$$

$$\sum_{i\in\mathcal{I}} y_i C_i = 0. \tag{4.21c}$$

Although we can not determine the value of b uniquely only from the above conditions, (4.21a) and (4.21b) specify the range of optimal b:

$$\max_{i\in\mathcal{L}} y_i g_i \leq b \leq \min_{i\in\mathcal{U}} y_i g_i, \tag{4.22}$$

where

$$\begin{aligned}
g_i &= 1 - \sum_{j \in \mathcal{I}} Q_{ij} C_j, \\
\mathcal{L} &= \{i \mid i \in \mathcal{O}, y_i = 1\} \cup \{i \mid i \in \mathcal{I}, y_i = -1\}, \\
\mathcal{U} &= \{i \mid i \in \mathcal{O}, y_i = -1\} \cup \{i \mid i \in \mathcal{I}, y_i = 1\}.
\end{aligned}$$

Let

$$\delta \equiv \sum_{i \in \mathcal{I}} y_i d_i,$$

where

$$d_i = C_i^{(\text{new})} - C_i^{(\text{old})}.$$

When $\delta = 0$, the step size $\Delta\theta$ can be increased as long as the inequality (4.22) is satisfied. Violation of (4.22) can be checked by monitoring the upper and lower bounds of the bias b (which are piecewise-linear w.r.t. $\Delta\theta$) when $\Delta\theta$ is increased

$$\begin{aligned}
u(\Delta\theta) &= \max_{i \in \mathcal{U}} y_i(g_i + \Delta g_i(\Delta\theta)), \\
\ell(\Delta\theta) &= \min_{i \in \mathcal{L}} y_i(g_i + \Delta g_i(\Delta\theta)),
\end{aligned} \tag{4.23}$$

where

$$\Delta g_i(\Delta\theta) = -\Delta\theta \sum_{j \in \mathcal{I}} Q_{ij} d_j.$$

On the other hand, when $\delta \neq 0$, $\Delta\theta$ can not be increased without violating the equality condition (4.21c). In this case, an instance with index

$$i_{\text{low}} = \underset{i \in \mathcal{L}}{\operatorname{argmax}}\, y_i g_i$$

or

$$i_{\text{up}} = \underset{i \in \mathcal{U}}{\operatorname{argmin}}\, y_i g_i$$

actually enters the index set $\mathcal{M}$. If the instance (we denote its index by m) comes from the index set $\mathcal{O}$, the following equation must be satisfied for keeping (4.21c) satisfied:

$$\Delta\theta\delta = -\Delta\alpha_m y_m.$$

Since $\Delta\theta > 0$ and $\Delta\alpha_m > 0$, we have

$$\mathrm{sign}(\delta) = \mathrm{sign}(-y_m).$$

On the other hand, if the instance comes from the index set $\mathcal{I}$,

$$\Delta\theta\delta = y_m(\Delta C_m - \Delta\alpha_m)$$

must be satisfied. Since $\Delta\theta > 0$ and $\Delta C_m - \Delta\alpha_m > 0$, we have

$$\mathrm{sign}(\delta) = \mathrm{sign}(y_m).$$

Considering these conditions, we arrive at the following updating rules for b and $\mathcal{M}$:

$$\begin{array}{lll} \delta > 0 & \Rightarrow & b = y_{i_{\mathrm{up}}} g_{i_{\mathrm{up}}},\ \mathcal{M} = \{i_{\mathrm{up}}\}, \\ \delta < 0 & \Rightarrow & b = y_{i_{\mathrm{low}}} g_{i_{\mathrm{low}}},\ \mathcal{M} = \{i_{\mathrm{low}}\}. \end{array} \tag{4.24}$$

Note that we also need to remove i_{up} and i_{low} from $\mathcal{O}$ and $\mathcal{I}$, respectively.

4.3.4 Computational Complexity

The entire pseudo-code of the proposed WSVM path-following algorithm is described in Fig. 4.2.

The computational complexity at each iteration of our path-following algorithm is the same as that for the ordinary SVM (i.e., the single-C formulation) [47]. Thus, our algorithm inherits a superior computational property of the original path-following algorithm.

The linear system (4.17) can be solved using Cholesky factor $\boldsymbol{L}$ of $\boldsymbol{Q}_{\mathcal{M}}$ (see Appendix A). Moreover, we can update $\boldsymbol{L}$ from the previous one at each event point can be carried out efficiently with $O(|\mathcal{M}|^2)$ computational cost based on the *Cholesky decomposition rank-one update* [45] (see Appendix B) or the *block-matrix inversion formula* [98]. Thus, the computational cost required for identifying the next event point is $O(n|\mathcal{M}|)$.

It is difficult to state the number of iterations needed for complete path-following because the number of events depends on the sensitivity of the model and the data set. Several empirical results suggest that the number of events linearly increases w.r.t. the data set size [46,47,117]; our experimental analysis given in Section 4.4 also showed the same tendency. This implies that path-following is computationally highly efficient—indeed, in Section 4.4, we will experimentally demonstrate that the proposed path-following algorithm is faster than an alternative approach in one or two orders of magnitude.

```
 1: arguments:
 2:     Optimal parameters $\boldsymbol{\alpha}$ and $b$ for $\boldsymbol{c}^{(\text{old})}$
 3:     Sets $\mathcal{M}$, $\mathcal{O}$, $\mathcal{I}$, and Cholesky factor $\boldsymbol{L}$ of $\boldsymbol{Q}_{\mathcal{M}}$
 4:     New weight vector $\boldsymbol{c}^{(\text{new})}$
 5: end arguments

 6: function WSVM-PATH($\boldsymbol{\alpha}, b, \boldsymbol{c}^{(\text{old})}, \mathcal{M}, \mathcal{O}, \mathcal{I}, \boldsymbol{L}, \boldsymbol{c}^{(\text{new})}$)
 7:     $\theta \leftarrow 0,\ \boldsymbol{c} \leftarrow \boldsymbol{c}^{(\text{old})}$
 8:     while $\theta \neq 1$ do
 9:         if $\mathcal{M}$ is empty then
10:             $\Delta\theta \leftarrow$ EMPTYMARGIN
11:         else
12:             Calculate $\boldsymbol{\phi}$ by (4.17) using Cholesky factor $\boldsymbol{L}$
13:             Calculate $\boldsymbol{\psi}$ by (4.19)
14:             Calculate $\Delta\theta$ by (4.20)
15:         end if
16:         If $\theta + \Delta\theta > 1$, then $\Delta\theta \leftarrow 1 - \theta$
17:         Update $\boldsymbol{\alpha}$, $b$, and $\boldsymbol{c}$ by step length $\Delta\theta$
18:         $\theta \leftarrow \theta + \Delta\theta$
19:         Update $\mathcal{M}$, $\mathcal{O}$, and $\mathcal{I}$ depending on the event type
20:         Update $\boldsymbol{L}$ (Cholesky factor rank-one update)
21:     end while
22: end function

23: function EMPTYMARGIN
24:     if $\delta(\boldsymbol{\alpha}) \neq 0$ then
25:         Set bias term $b$ by (4.24)
26:         $\Delta\theta \leftarrow 0$
27:     else
28:         Trace $u(\Delta\theta)$ and $\ell(\Delta\theta)$ in (4.23) until $u(\Delta\theta) = \ell(\Delta\theta)$
29:     end if
30:     return $\Delta\theta$
31: end function
```

Figure 4.2: Pseudo-code of the proposed WSVM path-following algorithm.

4.4 Experiments

In this section, we illustrate the empirical performance of the proposed WSVM path-following algorithm in a toy example and two real-world applications. We compared the computational cost of the proposed path-following algorithm with the *sequential minimal optimization* (SMO) algorithm [90] when the instance weights of WSVM are changed in various ways. In particular, we investigated the CPU time of updating solutions from some $\boldsymbol{c}^{(\mathrm{old})}$ to $\boldsymbol{c}^{(\mathrm{new})}$.

In the path-following algorithm, we assume that the optimal parameter $\boldsymbol{\alpha}$ as well as the Cholesky factor $\boldsymbol{L}$ of $\boldsymbol{Q}_{\dot{\mathcal{M}}}$ for $\boldsymbol{c}^{(\mathrm{old})}$ has already been obtained. In the SMO algorithm, we used the old optimal parameter $\boldsymbol{\alpha}$ as the initial starting point (i.e., the 'hot' start) after making them *feasible* using the *alpha-seeding strategy* [34]. We set the tolerance parameter in the termination criterion of SMO to 10^{-3}. Our implementation of the SMO algorithm is based on *LIBSVM* [25]. To circumvent possible numerical instability, we added small positive constant 10^{-6} to the diagonals of the matrix $\boldsymbol{Q}$. In all the experiments, we used the Gaussian kernel

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\gamma}{p}\|\boldsymbol{x} - \boldsymbol{x}'\|^2\right), \tag{4.25}$$

where γ is a hyper-parameter and p is the dimensionality of $\boldsymbol{x}$.

4.4.1 Illustrative Example

First, we illustrate the behavior of the proposed path-following algorithm using an artificial data set. Consider a binary classification problem with the training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathbb{R}^2$ and $y_i \in \{-1, +1\}$. Let us define the sets of indices of positive and negative instances as $\mathcal{K}_{-1} = \{i | y_i = -1\}$ and $\mathcal{K}_{+1} = \{i | y_i = +1\}$, respectively. We assume that the loss function is defined as

$$\sum_i v_i I(y_i f(\boldsymbol{x}_i) \leq 0), \tag{4.26}$$

where $v_i \in \{1, 2\}$ is the cost of misclassifying the instance $(\boldsymbol{x}_i, y_i)$, and $I(\cdot)$ is the indicator function. Let $\mathcal{D}_1 = \{i | v_i = 1\}$ and $\mathcal{D}_2 = \{i | v_i = 2\}$, i.e., $\mathcal{D}_2$ is the set of instance indices which have stronger influence on the overall test error than $\mathcal{D}_1$.

To be consistent with the above error metric, it would be natural to assign a smaller weight C_1 for $i \in \mathcal{D}_1$ and a larger weight C_2 for $i \in \mathcal{D}_2$ when training SVM. However, naively setting $C_2 = 2C_1$ is not generally optimal because the hinge loss is used in SVM training, while the 0-1 loss is used in performance evaluation (see (4.26)). In the following experiments, we fixed the Gaussian kernel width to $\gamma = 1$ and the instance weight for $\mathcal{D}_2$ to $C_2 = 10$, and we changed the instance weight C_1 for $\mathcal{D}_1$ from 0 to 10. Thus, the change of the weights is represented as

$$\begin{bmatrix} c_{\mathcal{D}_1}^{(\text{old})} \\ c_{\mathcal{D}_2}^{(\text{old})} \end{bmatrix} = \begin{bmatrix} 0 \\ 10 \end{bmatrix} \text{ and } \begin{bmatrix} c_{\mathcal{D}_1}^{(\text{new})} \\ c_{\mathcal{D}_2}^{(\text{new})} \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \end{bmatrix}.$$

The two-dimensional input $\{\boldsymbol{x}_i\}_{i=1}^n$ were generated from the following distribution:

$$\boldsymbol{x}_i \sim \begin{cases} \mathcal{N}\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}\right) & \text{if } i \in \mathcal{K}_{+1} \cap \mathcal{D}_1, \\ \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\right) & \text{if } i \in \mathcal{K}_{+1} \cap \mathcal{D}_2, \\ \mathcal{N}\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}\right) & \text{if } i \in \mathcal{K}_{-1} \cap \mathcal{D}_1, \\ \mathcal{N}\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\right) & \text{if } i \in \mathcal{K}_{-1} \cap \mathcal{D}_2. \end{cases} \tag{4.27}$$

Fig. 4.3 shows the generated instances for $n = 400$, in which instances in the above four cases have the equal size $n/4$. Before feeding the generated instances into algorithms, we normalized the inputs in $[0, 1]^2$.

Fig. 4.4 shows piecewise-linear paths of some of the solutions α_i for $C_1 \in [0, 10]$ when $n = 400$. The left graph includes the solution paths of three representative parameters α_i for $i \in \mathcal{D}_1$. All three parameters increase as C_1 grows from zero, and one of the parameters (denoted by the dash-dotted line) suddenly drops down to zero at around $C_1 = 7$. Another parameter (denoted by the solid line) also sharply drops down at around $C_1 = 9$, and the last one (denoted by the dashed line) remains equal to C_1 until C_1 reaches 10. The right graph includes the solution paths of three representative parameters α_i for $i \in \mathcal{D}_2$, showing that their behavior is substantially different from that for $\mathcal{D}_1$. One of the parameters (denoted by the dash-dotted line) fluctuates significantly, while the other two parameters (denoted by the solid and dashed lines) are more stable and tend to increase as C_1 grows.

An important advantage of the path following algorithm is that the path of the validation error can be traced as well (see Fig. 4.5). First, note that the path of the

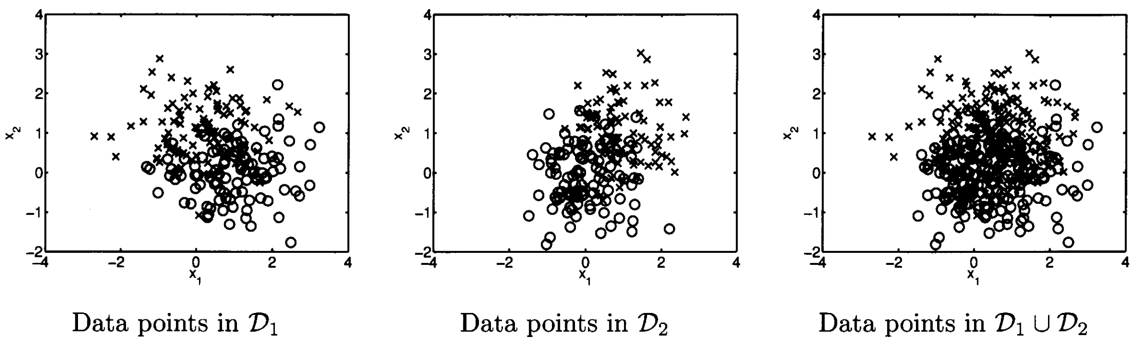


Figure 4.3: Artificial data set generated by the distribution (4.27). The crosses and circles indicate the data points in $\mathcal{K}_{-1}$ (negative class) and $\mathcal{K}_{+1}$ (positive class), respectively. The left plot shows the data points in $\mathcal{D}_1$ (misclassification cost is 1), the middle plot shows the data points in $\mathcal{D}_2$ (misclassification cost is 2), and the right plots show their union.

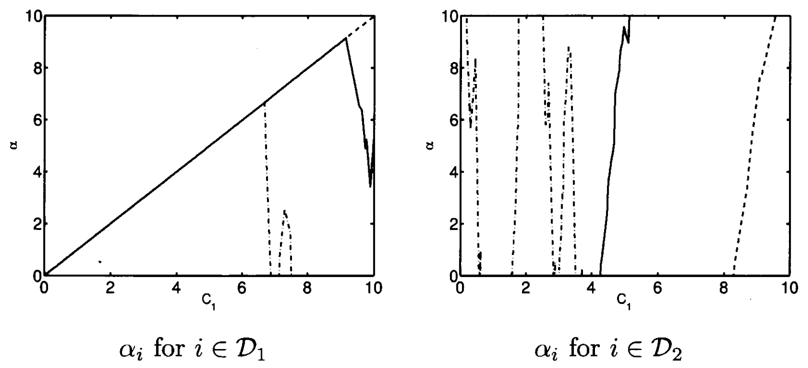


Figure 4.4: Examples of piecewise-linear paths of α_i for the artificial data set. The weights are changed from $C_i = 0$ to 10 for $i \in \mathcal{D}_1$ (for $i \in \mathcal{D}_2$, $C_i = 10$ is unchanged). The left and right plots show the paths of three representative parameters α_i for $i \in \mathcal{D}_1$, and for $i \in \mathcal{D}_2$, respectively.

validation error (26) has piecewise-constant form because the 0-1 loss changes only when the sign of $f(\boldsymbol{x})$ changes. In our path-following algorithm, the path of $f(\boldsymbol{x})$ also has piecewise-linear form because $f(\boldsymbol{x})$ is linear in their parameters $\boldsymbol{\alpha}$ and b. Exploiting the piecewise linearity of $f(\boldsymbol{x})$, we can exactly detect the point at which the sign of $f(\boldsymbol{x})$ changes. These points correspond to the breakpoints of the piecewise-constant validation-error path. Fig. 4.5 illustrates the relationship between the piecewise-linear path of $f(\boldsymbol{x})$ and the piecewise-constant validation-error path. Fig. 4.6 shows an example of piecewise-constant validation-error path when C_1 is increased from 0 to 10, indicating that the lowest validation error was achieved at around $C_1 = 4$.

Finally, we investigated the computation time when the solution path from $C_1 = 0$ to 10 was computed. For comparison, we also investigated the computation time of the

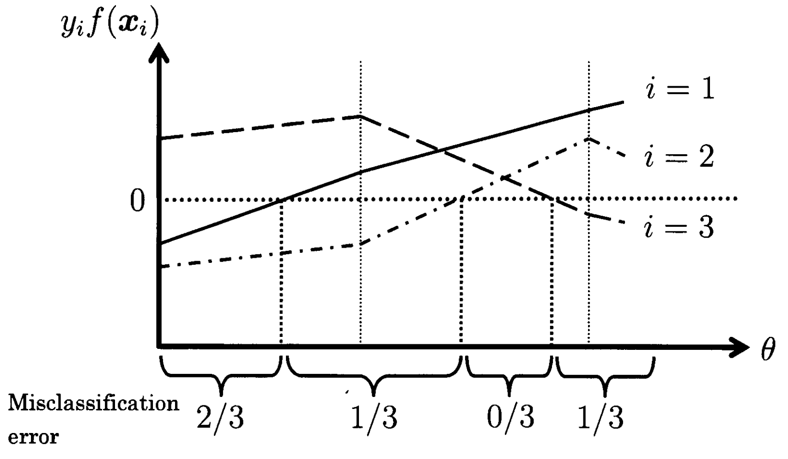


Figure 4.5: A schematic illustration of validation-error path. In this plot, the path of misclassification error rate $\frac{1}{3}\sum_{i=1}^{n} I(y_i f(\boldsymbol{x}_i)) \leq 0)$ for the 3 validation instances (x_1, y_1), (x_2, y_2), and (x_3, y_3) are depicted. The horizontal axis indicates the parameter θ and the vertical axis denotes $y_i f(\boldsymbol{x}_i)$, $i = 1, 2, 3$. The path of the validation error has piecewise–constant form because the 0-1 loss changes only when $f(\boldsymbol{x}_i) = 0$. The breakpoints of the piecewise-constant validation-error path can be exactly detected by exploiting the piecewise linearity of $f(\boldsymbol{x}_i)$.

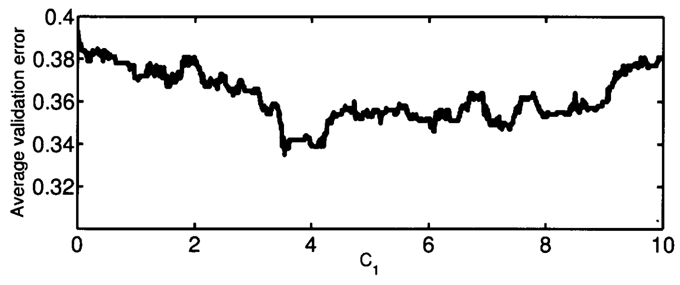


Figure 4.6: An example of the validation-error path for 1000 validation instances of the artificial data set. The number of training instances is 400 and the Gaussian kernel with $\gamma = 1$ is used.

SMO algorithm when the solutions at every breakpoint were computed. We considered the following four cases: the number of training instances was $n = 400$, 800, 1200, and 1600. For each n, we generated 10 data sets and the average and standard deviation over 10 runs are reported. Table 4.1 describes the results, showing that our path-following algorithm is faster than the SMO algorithm in one or two orders of magnitude; the difference between the two methods becomes more significant as the training data size n grows.

The table also includes the number of events and the average number of elements in the margin set $\mathcal{M}$ (see Eq.(4.7b)). This shows that the number of events increases almost linearly in the sample size n, which well agrees with the empirical results reported

Table 4.1: The experimental results of the artificial data set.
The average and the standard deviation (in brackets) over 10 runs are reported.

n	CPU time (sec.)		#events	mean $\|\mathcal{M}\|$
	path	SMO		
400	0.03 (0.00)	0.39 (0.01)	326.70 (7.17)	3.07 (0.03)
800	0.08 (0.00)	2.84 (0.12)	635.30 (17.47)	3.27 (0.02)
1200	0.19 (0.00)	10.63 (0.38)	997.60 (26.85)	3.38 (0.05)
1600	0.35 (0.01)	28.11 (0.77)	1424.00 (31.27)	3.50 (0.02)

in [47], [46], and [117]. The average number of elements in the set $\mathcal{M}$ increases very mildly as the sample size n grows.

4.4.2 Online Time-series Learning

In online time-series learning, larger (resp. smaller) weights should be assigned to newer (resp. older) instances. For example, in [22], the following weight function is used:

$$C_i = C_0 \frac{2}{1 + \exp(a - 2a \times \frac{i}{n})}, \tag{4.28}$$

where C_0 and a are hyper-parameters and the instances are assumed to be sorted along the time axis (the most recent instance is $i = n$). Fig. 4.7 shows the profile of the weight function (4.28) when $C_0 = 1$. In online learning, we need to update parameters when new observations arrive, and all the weights must be updated accordingly (see Fig. 4.8).

We investigated the computational cost of updating parameters when several new observations arrive. The experimental data are obtained from the *NASDAQ composite index* between January 2, 2001 and December 31, 2009. As [22] and [29], we transformed the original closing prices using the Relative Difference in Percentage (RDP) of the price and the exponential moving average (EMA).

Extracted features are listed in Table 4.2 (see [22] for more details). Our task is to predict the sign of RDP+5 using EMA15 and four lagged-RDP values (RDP-5, RDP-10, RDP-15, and RDP-20). RDP values which exceed ± 2 standard deviations are replaced with the closest marginal values. We have an initial set of training instances with size $n = 2515$. The inputs were normalized in $[0, 1]^p$, where p is the dimensionality of the input $\boldsymbol{x}$. We used the Gaussian kernel (4.25) with $\gamma \in \{10, 1, 0.1\}$, and the weight parameter a in (4.28) was set to 3. We first trained WSVM using the initial set of instances. Then we added 5 instances to the previously trained WSVM and removed

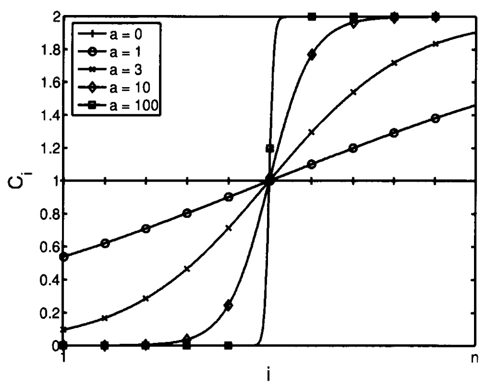


Figure 4.7: The weight functions for financial time-series forecasting. The horizontal axis is the index of training instances which is sorted according to time (the most recent instance is $i = n$). If we set $a = 0$, all the instances are weighted equally. This weighting strategy is proposed in [22].

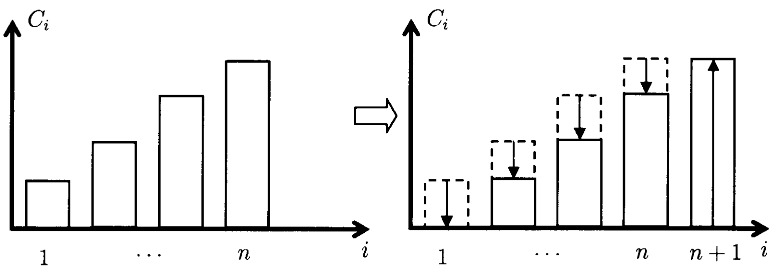


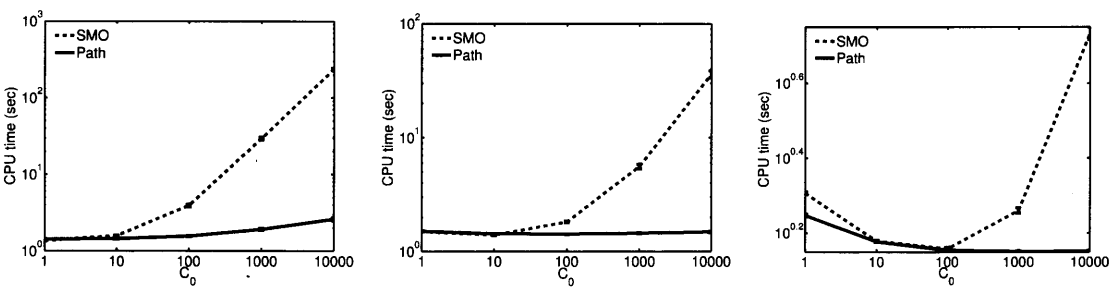
Figure 4.8: A schematic illustration of the change of weights in time-series learning. The left plot shows the fact that larger weights are assigned to more recent instances. The right plot describes a situation where we receive a new instance ($i = n + 1$). In this situation, the oldest instance ($i = 1$) is deleted by setting its weight to zero, the weight of the new instance is set to be the largest, and the weights of the rest of the instances are decreased accordingly.

the oldest 5 instances by decreasing their weights to 0. This does not change the size of the training data set, but the entire weights need to be updated as illustrated in Fig. 4.8. We iterated this process 5 times and compared the total computational costs of the path-following algorithm and the SMO algorithm. For fair comparison, we cleared the cache of kernel values at each update before running the algorithms.

Fig. 4.9 shows the average CPU time over 10 runs for $C_0 \in \{1, 10, 10^2, 10^3, 10^4\}$, showing that the path-following algorithm is much faster than the SMO algorithm especially for large C_0.

Table 4.2: Features for financial forecasting
($p(i)$ is the closing price of the ith day and $\mathrm{EMA}_k(i)$ is the k-day exponential moving average of the ith day.

Feature	Formula
EMA15	$p(i) - \mathrm{EMA}_{15}(i)$
RDP-5	$(p(i) - p(i-5))/p(i-5) * 100$
RDP-10	$(p(i) - p(i-10))/p(i-10) * 100$
RDP-15	$(p(i) - p(i-15))/p(i-15) * 100$
RDP-20	$(p(i) - p(i-20))/p(i-20) * 100$
RDP+5	$(\overline{p(i+5)} - \overline{p(i)})/\overline{p(i)} * 100$
	$\overline{p(i)} = \mathrm{EMA}_3(i)$



(a) $\gamma = 10$ (b) $\gamma = 1$ (c) $\gamma = 0.1$

Figure 4.9: CPU time comparison for online time-series learning using NASDAQ composite index.

4.4.3 Model Selection in Covariate Shift Adaptation

Covariate shift is a situation in supervised learning where the input distributions change between the training and test phases but the conditional distribution of outputs given inputs remains unchanged [101]. Under covariate shift, standard SVM and SVR are biased, and the bias caused by covariate shift can be asymptotically canceled by weighting the loss function according to the *importance* (i.e., the ratio of training and test input densities).

Here, we apply importance-weighted SVMs to *brain-computer interfaces* (BCIs) [35]. A BCI is a system which allows for a direct communication from man to machine via brain signals. Strong non-stationarity effects have been often observed in brain signals between training and test sessions, which could be modeled as covariate shift [105]. We used the BCI datasets provided by the Berlin BCI group [19], containing 24 binary classification tasks. The input features are 4-dimensional preprocessed *electroencephalogram* (EEG) signals, and the output labels correspond to the 'left' and 'right' commands. The size of training datasets is around 500 to 1000, and the size of test datasets is around 200 to 300.

Although the importance-weighted SVM tends to have lower bias, it in turns has larger estimation variance than the ordinary SVM [101]. Thus, in practice, it is desirable to slightly 'flatten' the instance weights so that the trade-off between bias and variance is optimally controlled. Here, we changed the instance weights from the uniform values to the importance values using the proposed path-following algorithm, i.e., the instance weights were changed from

$$C_i^{(\mathrm{old})} = C_0,\ i = 1, \ldots, n,$$

to

$$C_i^{(\mathrm{new})} = C_0 \frac{p_{test}(\boldsymbol{x}_i)}{p_{train}(\boldsymbol{x}_i)},\ i = 1, \ldots, n.$$

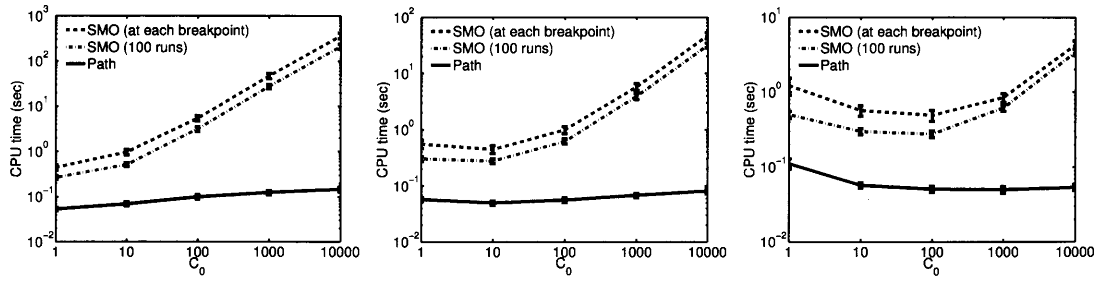
The importance values $\frac{p_{test}(\boldsymbol{x}_i)}{p_{train}(\boldsymbol{x}_i)}$ were estimated by the method proposed in [53], which directly estimates the density ratio without going through density estimation of $p_{test}(\boldsymbol{x})$ and $p_{train}(\boldsymbol{x})$.

For comparison, we ran the SMO algorithm at (i) each breakpoint of the solution path, and (ii) 100 weight vectors taken uniformly in $[C_i^{(\mathrm{old})}, C_i^{(\mathrm{new})}]$. We used the Gaussian kernel and the inputs were normalized in $[0, 1]^p$, where p is the dimensionality of $\boldsymbol{x}$.

Fig. 4.10 shows the average CPU time and its standard deviation. We examined several settings of hyper-parameters $\gamma \in \{10, 1, \ldots, 10^{-2}\}$ and $C_0 \in \{1, 10, 10^2, \ldots, 10^4\}$. The horizontal axis of each plot represents C_0. The graphs show that our path-following algorithm is faster than the SMO algorithm in all cases. While the SMO algorithm tended to take longer time for large C_0, the CPU time of the path-following algorithm did not increase with respect to C_0.

4.5 Beyond Classification

So far, we focused on classification scenarios. Here we show that the proposed path-following algorithm can be extended to various scenarios including regression, ranking, and transduction.



(a) $\gamma = 10$ (b) $\gamma = 1$ (c) $\gamma = 0.1$

Figure 4.10: CPU time comparison for covariate shift adaptation using BCI data.

4.5.1 Regression

The *support vector regression* (SVR) is a variant of SVM for regression problems [81,84,113].

Formulation

The primal optimization problem for the weighted SVR (WSVR) is defined by

$$\begin{aligned}\min_{\boldsymbol{w},b,\{\xi_i,\xi_i^*\}_{i=1}^n} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^n C_i(\xi_i + \xi_i^*), \\ \text{s.t.} \quad & y_i - f(\boldsymbol{x}_i) \leq \varepsilon + \xi_i, \\ & f(\boldsymbol{x}_i) - y_i \leq \varepsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0, \ i = 1, \dots, n,\end{aligned}$$

where $\epsilon > 0$ is an insensitive-zone thickness. Note that, as in the classification case, WSVR is reduced to the original SVR when $C_i = C$ for $i = 1, \dots, n$. Thus, WSVR includes SVR as a special case.

The corresponding dual problem is given by

$$\begin{aligned}\max_{\{\alpha_i\}_{i=1}^n} \quad & -\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \alpha_i\alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \varepsilon\sum_{i=1}^n |\alpha_i| + \sum_{i=1}^n y_i\alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^n \alpha_i = 0, \ -C_i \leq \alpha_i \leq C_i, \ i = 1, \dots, n.\end{aligned}$$

The final solution, i.e., the regression function $f : \mathcal{X} \to \mathbb{R}$, is in the following form:

$$f(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b.$$

The KKT conditions for the above dual problem are given as

$$|y_i - f(\boldsymbol{x}_i)| \leq \varepsilon, \quad \text{if} \quad \alpha_i = 0, \tag{4.29a}$$

$$|y_i - f(\boldsymbol{x}_i)| = \varepsilon, \quad \text{if} \quad 0 < |\alpha_i| < C_i, \tag{4.29b}$$

$$|y_i - f(\boldsymbol{x}_i)| \geq \varepsilon, \quad \text{if} \quad |\alpha_i| = C_i, \tag{4.29c}$$

$$\sum_{i=1}^{n} \alpha_i = 0. \tag{4.29d}$$

Then the training instances can be partitioned into the following three index sets (see Fig. 2.2):

$$\begin{aligned}
\mathcal{O} &= \{i : |y_i - f(\boldsymbol{x}_i)| \geq \varepsilon, |\alpha_i| = C_i\}, \\
\mathcal{E} &= \{i : |y_i - f(\boldsymbol{x}_i)| = \varepsilon, 0 < |\alpha_i| < C_i\}, \\
\mathcal{I} &= \{i : |y_i - f(\boldsymbol{x}_i)| \leq \varepsilon, \alpha_i = 0\}.
\end{aligned}$$

Let

$$\boldsymbol{K}^{\mathcal{E}} = \begin{bmatrix} 0 & \boldsymbol{1}^\top \\ \boldsymbol{1} & \boldsymbol{K}_{\mathcal{E}} \end{bmatrix} \quad \text{and} \quad \boldsymbol{s} = \begin{bmatrix} \operatorname{sign}(y_1 - f(\boldsymbol{x}_1)) \\ \vdots \\ \operatorname{sign}(y_n - f(\boldsymbol{x}_n)) \end{bmatrix}.$$

Then, from (4.29), we obtain

$$\begin{aligned}
\begin{bmatrix} b \\ \boldsymbol{\alpha}_{\mathcal{E}} \end{bmatrix} &= -(\boldsymbol{K}^{\mathcal{E}})^{-1} \begin{bmatrix} \boldsymbol{1}_{\mathcal{O}}^\top \\ \boldsymbol{K}_{\mathcal{E},\mathcal{O}} \end{bmatrix} \boldsymbol{c}_{\mathcal{O}} + (\boldsymbol{K}^{\mathcal{E}})^{-1} \begin{bmatrix} 0 \\ \boldsymbol{y}_{\mathcal{E}} - \varepsilon \boldsymbol{s}_{\mathcal{E}} \end{bmatrix}, \\
\boldsymbol{\alpha}_{\mathcal{O}} &= \operatorname{diag}(\boldsymbol{s}_{\mathcal{O}}) \boldsymbol{c}_{\mathcal{O}}, \\
\boldsymbol{\alpha}_{\mathcal{O}} &= \boldsymbol{0}.
\end{aligned}$$

where $\operatorname{diag}(\boldsymbol{s}_{\mathcal{O}})$ indicates the diagonal matrix with its diagonal part $\boldsymbol{s}_{\mathcal{O}}$. These functions are *affine* w.r.t. $\boldsymbol{c}$, so we can easily detect an event point by monitoring the inequalities in (4.29). We can follow the solution path of SVR by using essentially the same technique as SVM classification (and thus the details are omitted).

Experiments on Regression

As an application of WSVR, we consider a heteroscedastic regression problem, where output noise variance depends on input points. In heteroscedastic data modeling, larger (resp. smaller) weights are usually assigned to instances with smaller (resp. larger) variances. Since the point-wise variances are often unknown in practice, they should also be estimated from data. A standard approach is to alternately estimate the weight vector $\boldsymbol{c}$ based on the current WSVR solution and update the WSVR solutions based on the new weight vector $\boldsymbol{c}$ [63].

We set the weights as

$$C_i = C_0 \frac{\widehat{\sigma}}{|e_i|}, \tag{4.30}$$

where $e_i = y_i - \widehat{f}(\boldsymbol{x}_i)$ is the residual of the instance $(\boldsymbol{x}_i, y_i)$ from the current fit $\widehat{f}(\boldsymbol{x}_i)$, and $\widehat{\sigma}$ is an estimate of the common standard deviation of the noise computed as $\widehat{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^n e_i^2}$. We employed the following procedure for the heteroscedastic data modeling:

Step1: Training WSVR with uniform weights (i.e., $C_i = C_0, i = 1, \ldots, n.$).

Step2: Update weights by (4.30) and update the solution of WSVR accordingly. Repeat this step until $\frac{1}{n}\sum_{i=1}^n |(e_i^{(\mathrm{old})} - e_i)/e_i^{(\mathrm{old})}| \leq 10^{-3}$ holds, where $e^{(\mathrm{old})}$ is the previous training error.

We investigated the computational cost of Step2. We applied the above procedure to the well-known *Boston housing* data set. The sample size is 506 and the number of features is $p = 13$. The inputs were normalized in $[-1, 1]^p$. We randomly sampled $n = 404$ instances from the original data set, and the experiments were repeated 10 times. We used the Gaussian kernel (4.25) with $\gamma \in \{10, 1, 0.1\}$. The insensitive zone thickness in WSVR was fixed to $\varepsilon = 0.05$.

Each plot of Fig. 4.11 shows the CPU time comparison for $C_0 \in \{1, 10, \ldots, 10^4\}$, and Fig. 4.12 shows the number of iterations performed in Step2. Our path-following approach is faster than the SMO algorithm especially for large C_0.

4.5.2 Ranking

Recently, the problem of *learning to rank* has attracted wide interest as a challenging topic in machine learning and information retrieval [77]. Here, we focus on a method called the *ranking SVM* (RSVM) [48].

Formulation

Assume that we have a set of n triplets $\{(\boldsymbol{x}_i, y_i, q_i)\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^p$ is a feature vector of an item and $y_i \in \{r_1, \ldots, r_q\}$ is a relevance of $\boldsymbol{x}_i$ to a query q_i. The relevance

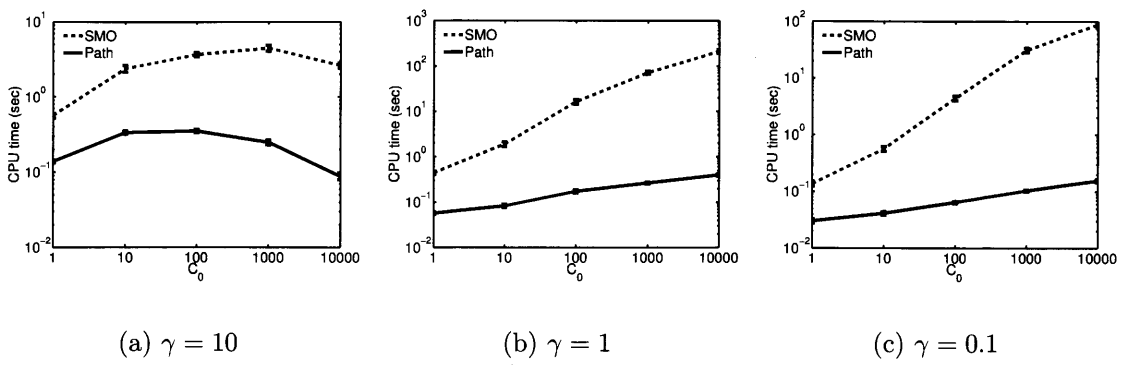


Figure 4.11: CPU time comparison for heteroscedastic modeling using Boston housing data.

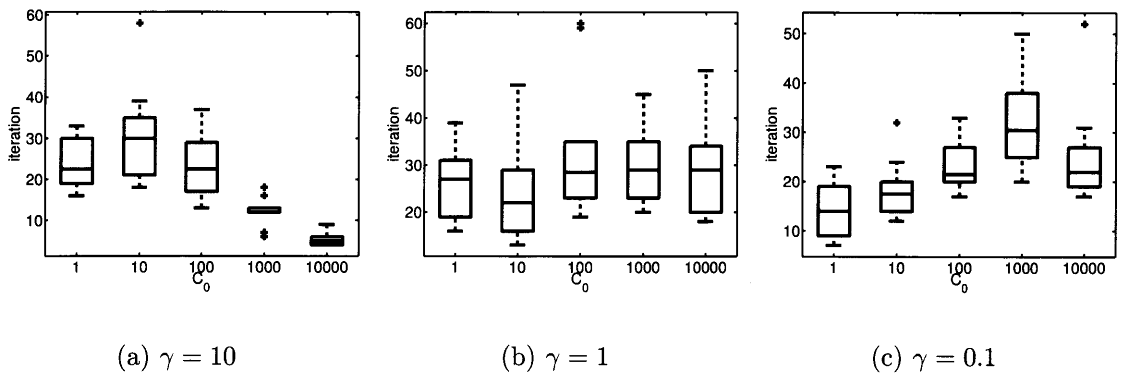


Figure 4.12: The number of weight updates for Boston housing data.

has an order of the preference $r_q \succ r_{q-1} \succ \cdots \succ r_1$, where $r_q \succ r_{q-1}$ means that r_q is preferred to r_{q-1}. The goal is to learn a ranking function $f(\boldsymbol{x})$ which returns a larger value for a preferred item. More precisely, for items $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ such that $q_i = q_j$, we want the ranking function $f(\boldsymbol{x})$ to satisfy

$$y_i \succ y_j \quad \Leftrightarrow \quad f(\boldsymbol{x}_i) > f(\boldsymbol{x}_j).$$

Let us define the following set of pairs:

$$\mathcal{P} = \{(i,j) \mid y_i \succ y_j, q_i = q_j\}.$$

RSVM solves the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w}, \{\xi_{ij}\}_{(i,j)\in\mathcal{P}}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C \sum_{(i,j)\in\mathcal{P}} \xi_{ij} \\ \text{s.t.} \quad & f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j) \geq 1 - \xi_{ij}, \ (i,j) \in \mathcal{P}. \end{aligned}$$

In practical ranking tasks such as information retrieval, a pair of items with highly different preference levels should have a larger weight than those with similar preference levels. Based on this prior knowledge, [23] and [119] proposed to assign different weights C_{ij} to different relevance pairs $(i,j) \in \mathcal{P}$. This is a *cost-sensitive* variant of RSVM whose primal problem is given as

$$\begin{aligned} \min_{\boldsymbol{w}, \{\xi_{ij}\}_{(i,j)\in\mathcal{P}}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \sum_{(i,j)\in\mathcal{P}} C_{ij}\xi_{ij} \\ \text{s.t.} \quad & f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j) \geq 1 - \xi_{ij}, \ (i,j) \in \mathcal{P}. \end{aligned}$$

Since this formulation is interpreted as a WSVM for pairs of items $(i,j) \in \mathcal{P}$, we can easily apply our multi-parametric path approach. This leads to the following dual problem:

$$\begin{aligned} \max_{\{\alpha_i\}_{i=1}^m} \quad & -\frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m \alpha_i\alpha_j\widetilde{Q}_{ij} + \sum_{i=1}^m \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C_i, \ i = 1, \ldots, m, \end{aligned}$$

where

$$\widetilde{Q}_{ij} = K_{k_i,k_j} - K_{k_i,\ell_j} - K_{\ell_i,k_j} + K_{\ell_i,\ell_j}.$$

Then, the ranking function f is given as

$$f(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i (K(\boldsymbol{x}, \boldsymbol{x}_{k_i}) - K(\boldsymbol{x}, \boldsymbol{x}_{\ell_i})).$$

The index partitioning of the optimal solution is given as follows:

$$\begin{aligned} \mathcal{O} &= \{i \mid f(\boldsymbol{x}_{k_i}) - f(\boldsymbol{x}_{\ell_i}) \geq 1, \alpha_i = 0\}, \\ \mathcal{M} &= \{i \mid f(\boldsymbol{x}_{k_i}) - f(\boldsymbol{x}_{\ell_i}) = 1, 0 < \alpha_i < C_i\}, \\ \mathcal{I} &= \{i \mid f(\boldsymbol{x}_{k_i}) - f(\boldsymbol{x}_{\ell_i}) \leq 1, \alpha_i = C_i\}. \end{aligned}$$

Using these sets, we can compute the solution path for changing weight vector $\boldsymbol{c}$. Note that the solution path algorithm for the cost-sensitive RSVM is regarded as an extension of the previous work by [5], in which the solution path for the standard RSVM was studied.

In this paper, we consider a model selection problem for the weighting pattern $\{C_{ij}\}_{(i,j)\in\mathcal{P}}$. We assume that the weighting pattern is represented as

$$C_{ij} = C_{ij}^{(\text{old})} + \theta(C_{ij}^{(\text{new})} - C_{ij}^{(\text{old})}), \ (i,j) \in \mathcal{P}, \ \theta \in [0,1], \tag{4.32}$$

where

$$C_{ij}^{(\mathrm{old})} = C_0,\ (i,j) \in \mathcal{P}, \tag{4.33}$$

$$C_{ij}^{(\mathrm{new})} = (2^{y_i} - 2^{y_j})C_0,\ (i,j) \in \mathcal{P}, \tag{4.34}$$

and C_0 is the common regularization parameter[2] . We follow the multi-parametric solution path from $\{C_{ij}^{(\mathrm{old})}\}_{(i,j)\in\mathcal{P}}$ to $\{C_{ij}^{(\mathrm{new})}\}_{(i,j)\in\mathcal{P}}$ and the best θ is selected based on the validation performance.

The performance of ranking algorithms is usually evaluated by some information-retrieval measures such as the *normalized discounted cumulative gain* (NDCG) [50]. Consider a query q and define $q(j)$ as the index of the j-th largest item among $\{f(\boldsymbol{x}_i)\}_{i\in\{i|q_i=q\}}$. The NDCG at position k for a query q is defined as

$$\mathrm{NDCG@k} = \mathrm{Z}\sum_{\mathrm{j}=1}^{\mathrm{k}} \begin{cases} 2^{y_{q(j)}} - 1, & j = 1, \\ \frac{2^{y_{q(j)}}-1}{\log(j)}, & j > 1, \end{cases} \tag{4.35}$$

where Z is a constant to normalize the NDCG in $[0, 1]$. Note that the NDCG value in (4.35) is defined using only the top k items and the rest are ignored. The NDCG for multiple queries are defined as the average of (4.35).

The goal of our model selection problem is to choose θ with the largest NDCG value. As explained below, we can identify θ that attains the exact maximum NDCG value for validation samples by exploiting the piecewise linearity of the solution path. The NDCG value changes only when there is a change in the top k ranking, and the rank of two items $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ changes only when $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$ cross. Then change points of the NDCG can be exactly identified because $f(\boldsymbol{x})$ changes in piecewise-linear form. Fig. 4.13 schematically illustrates piecewise-linear paths and the corresponding NDCG path for validation samples. The validation NDCG changes in piecewise-constant form, and change points are found when there is a crossing between two piecewise-linear paths.

Experiments on ranking

We used the OHSUMED data set from the LETOR package (version 3.0) provided by Microsoft Research Asia [76]. We used the query-level normalized version of the data set containing 106 queries. The total number of query-document pairs is 16140, and

[2]In [26], ranking of each item is also incorporated to define the weighting pattern. However, these weights depend on the current ranking, and it might change during training. We thus, for simplicity, introduce the weighting pattern (4.34) that depends only on the difference of the preference levels.

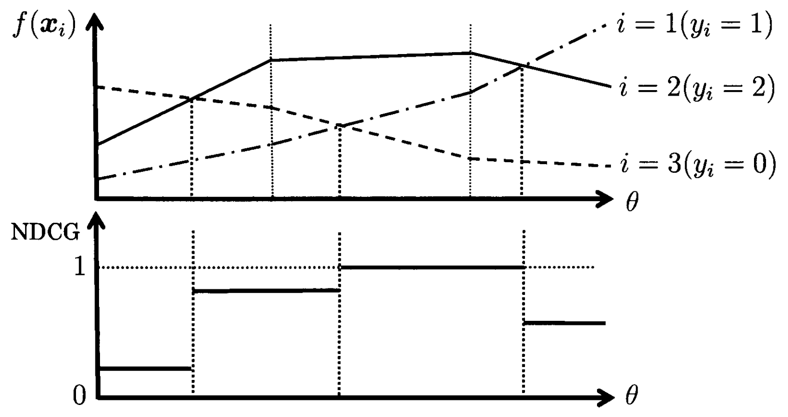
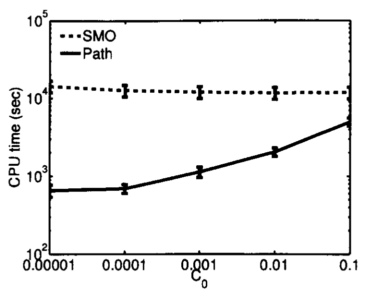


Figure 4.13: The schematic illustration of the NDCG path. The upper plot shows outputs for 3 items which have different levels of preferences y. The bottom plot shows the changes of the NDCG. Since the NDCG depends on the sorted order of items, it changes only when two lines of the upper plot intersect.

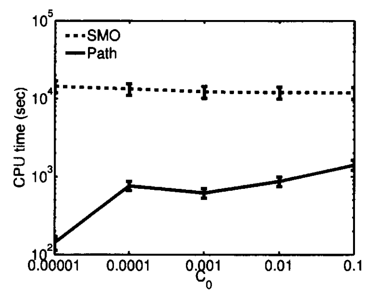
the number of features is $p = 45$. The data set provided is originally partitioned into 5 subsets, each of which has training, validation, and test sets for 5-fold cross validation. Here, we only used the training and the validation sets.

We compared the CPU time of our path algorithm and the SMO algorithm to change $\{C_{ij}\}_{(i,j)\in\mathcal{P}}$ from flat ones (4.33) to relevance weighted ones (4.34). We need to modify the SMO algorithm to train the model without the explicit bias term b. The usual SMO algorithm updates selected two parameters per iteration to ensure that the solution satisfies the equality constraint derived from the optimality condition of b. Since RSVM has no bias term, the algorithm is adapted to update one parameter per iteration [115]. We employed the update rule of [115] to adapt the SMO algorithm to RSVM and we chose the maximum violating point as an update parameter. This strategy is analogous to the maximum violating-pair working set selection of [62] in ordinary SVM. Since it took relatively large computational time, we ran the SMO algorithm only at 10 points uniformly taken in $[C_{ij}^{(\text{old})}, C_{ij}^{(\text{new})}]$. We considered every pair of initial weight $C_0 \in \{10^{-5}, \ldots, 10^{-1}\}$ and Gaussian width $\gamma \in \{10, 1, 0.1\}$. The results, given in Fig. 4.14, show that the path algorithm is faster than the SMO in all of the settings.

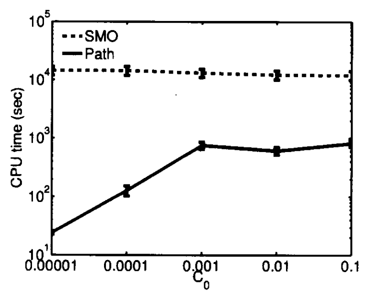
The CPU time of the path algorithm in Fig. 4.14(a) increases as C_0 increases because the number of breakpoints and the size of the set $\mathcal{M}$ also increase. Since our path algorithm solves a linear system with size $|\mathcal{M}|$ using $O(|\mathcal{M}|^2)$ update in each iteration, practical computational time depends on $|\mathcal{M}|$ especially in large data sets. In the case of RSVM, the maximum value of $|\mathcal{M}|$ is the number of pairs of training documents $m = |\mathcal{P}|$.



(a) $\gamma = 10$



(b) $\gamma = 1$



(c) $\gamma = 0.1$

Figure 4.14: CPU time comparison for RSVM.

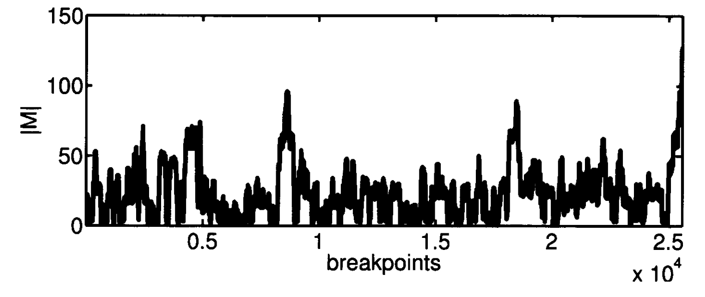


Figure 4.15: The number of instances on the margin $|\mathcal{M}|$ in ranking experiment.

For each fold of the OHSUMED data set, m = 367663, 422716, 378087, 295814, and 283484. If $|\mathcal{M}| \approx m$, a large computational cost may be needed for updating the linear system. However, as Fig. 4.15 shows, the size $|\mathcal{M}|$ is at most about one hundred in this setup.

Fig. 4.16 shows the example of the path of validation NDCG@10. Since the NDCG depends on the sorted order of documents, direct optimization is rather difficult [77]. Using our path algorithm, however, we can detect the exact behavior of the NDCG by monitoring the change of scores $f(\boldsymbol{x})$ in the validation data set. Then we can find the best weighting pattern by choosing θ with the maximum NDCG for the validation set.

4.5.3 Transduction

In *transductive inference* [112], we are given unlabeled instances along with labeled instances. The goal of transductive inference is not to estimate the true decision function, but to classify the given unlabeled instances correctly. The *transductive SVM* (TSVM) [52] is one of the most popular approaches to transductive binary classification. The objective of the TSVM is to maximize the classification margin for both labeled and

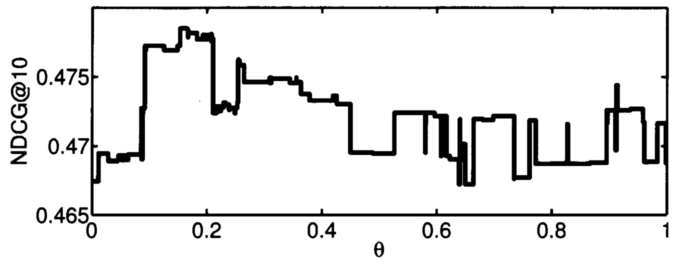


Figure 4.16: The change of NDCG@10 for $\gamma = 0.1$ and $C = 0.01$. The parameter θ in the horizontal axis is used as $\boldsymbol{c}^{(\text{old})} + \theta(\boldsymbol{c}^{(\text{new})} - \boldsymbol{c}^{(\text{old})})$.

unlabeled instances.

Formulation

Suppose we have k unlabeled instances $\{\boldsymbol{x}_i^*\}_{i=1}^k$ in addition to n labeled instances $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. The optimization problem of TSVM is formulated as

$$\min_{\{y_i^*,\xi_i^*\}_{i=1}^k, \boldsymbol{w}, b, \{\xi_i\}_{i=1}^n} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^n \xi_i + C^*\sum_{j=1}^k \xi_j^* \tag{4.36}$$
$$\text{s.t.} \quad \begin{array}{ll} y_i(\boldsymbol{w}^\top \boldsymbol{\Phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, & i = 1, \ldots, n, \\ y_j^*(\boldsymbol{w}^\top \boldsymbol{\Phi}(\boldsymbol{x}_j) + b) \geq 1 - \xi_j^*, & j = 1, \ldots, k, \\ \xi_i \geq 0, & i = 1, \ldots, n, \\ \xi_j^* \geq 0, & j = 1, \ldots, k, \end{array}$$

where C and C^* are the regularization parameters for labeled and unlabeled data, respectively, and $y_j^* \in \{-1, +1\}, j = 1, \ldots, k$, are the labels of the unlabeled instances $\{\boldsymbol{x}_i^*\}_{i=1}^k$. Note that (4.36) is a combinatorial optimization problem with respect to $\{y_j^*\}_{j\in\{1,\ldots,k\}}$. The optimal solution of (4.36) can be found if we solve binary SVMs for all possible combinations of $\{y_j^*\}_{j\in\{1,\ldots,k\}}$, but this is computationally intractable even for moderate k. To cope with this problem, [52] proposed an algorithm which approximately optimizes (4.36) by solving a series of WSVMs. The subproblem is formulated by assigning temporarily estimated labels $\widehat{y}_j^*$ to unlabeled instances:

$$\min_{\{\xi_i^*\}_{i=1}^k, \boldsymbol{w}, b, \{\xi_i\}_{i=1}^n} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^n \xi_i + C_-^* \sum_{j\in\{j|\widehat{y}_j^*=-1\}} \xi_j^* + C_+^* \sum_{j\in\{j|\widehat{y}_j^*=1\}} \xi_j^* \tag{4.37}$$
$$\text{s.t.} \quad \begin{array}{ll} y_i(\boldsymbol{w}^\top \boldsymbol{\Phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, & i = 1, \ldots, n, \\ \widehat{y}_j^*(\boldsymbol{w}^\top \boldsymbol{\Phi}(\boldsymbol{x}_j) + b) \geq 1 - \xi_j^*, & j = 1, \ldots, k, \\ \xi_i \geq 0, & i = 1, \ldots, n, \\ \xi_j^* \geq 0, & j = 1, \ldots, k, \end{array}$$

where C_-^* and C_+^* are the weights for unlabeled instances for $\{j \mid \widehat{y}_j^* = -1\}$ and $\{j \mid \widehat{y}_j^* = +1\}$, respectively. The entire algorithm is given as follows (see [52] for details):

Step1: Set the parameters C, C^*, and k^+, where k^+ is defined as

$$k^+ = k \times \frac{|\{j \mid y_j = +1, j = 1, \ldots, n\}|}{n}.$$

k^+ is defined so that the balance of positive and negative instances in the labeled set is equal to that in the unlabeled set.

Step2: Optimize the decision function using only the labeled instances and compute the decision function values $\{f(\boldsymbol{x}_j^*)\}_{j=1}^k$. Assign positive label $y_j^* = 1$ to the top k^+ unlabeled instances in decreasing order of $f(\boldsymbol{x}_j^*)$, and negative label $y_j^* = -1$ to the remaining instances. Set C_-^* and C_+^* to some small values (see [52] for details).

Step3: Train SVM using all the instances (i.e., solve (4.37)). Switch the labels of a pair of positive and negative unlabeled instances if the objective value (4.36) is reduced, where the pair of instances are selected based on $\{\xi_j^*\}_{j\in\{1,\ldots,k\}}$ (see [52] for details). Iterate this step until no data pair decreases the objective value.

Step4: Set $C_-^* = \min(2C_-^*, C^*)$ and $C_+^* = \min(2C_+^*, C^*)$. If $C_-^* \geq C^*$ and $C_+^* \geq C^*$, terminate the algorithm. Otherwise return to Step3.

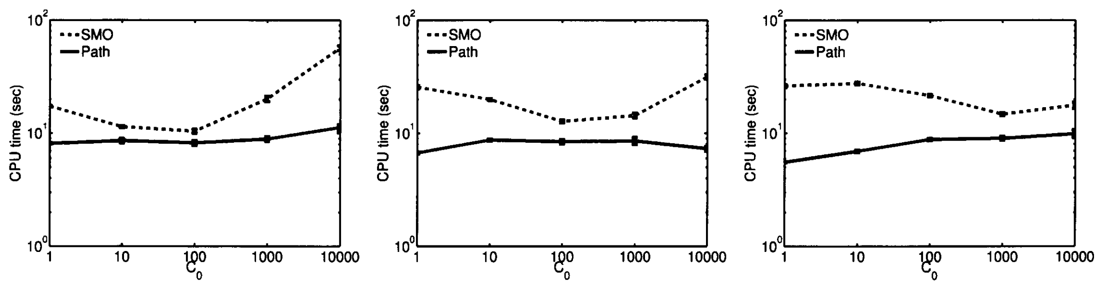
Our path-following algorithm can be applied to Step3 and Step4 for improving computational efficiency. Step3 can be carried out via path-following as follows:

Step3(a) Choose a pair of positive instance $\boldsymbol{x}_m^*$ and negative instance $\boldsymbol{x}_{m'}^*$.

Step3(b) After removing the positive instance $\boldsymbol{x}_m^*$ by decreasing its weight parameter C_m from C_+^* to 0, add the instance $\boldsymbol{x}_m^*$ as a negative one by increasing C_m from 0 to C_-^*.

Step3(c) After removing the negative instance $\boldsymbol{x}_{m'}^*$ by decreasing its weight parameter $C_{m'}$ from C_-^* to 0, add the instance $\boldsymbol{x}_{m'}^*$ as a positive one by increasing $C_{m'}$ from 0 to C_+^*.

Note that the steps 3(b) and 3(c) for switching the labels may be merged into a single step. Step4 also can be carried out by our path-following algorithm.



(a) $\gamma = 10$ (b) $\gamma = 1$ (c) $\gamma = 0.1$

Figure 4.17: CPU time comparison for the transductive SVM.

Experiments on Transduction

We compare the computation time of the proposed path-following algorithm and the SMO algorithm for Step3 and Step4 of TSVM. We used the *spam* data set obtained from the *UCI machine learning repository* [6]. The sample size is 4601, and the number of features is $p = 57$. We randomly selected 10% of data set as labeled instances, and the remaining 90% were used as unlabeled instances. The inputs were normalized in $[0, 1]^p$.

Fig. 4.17 shows the average CPU time and its standard deviation over 10 runs for the Gaussian width $\gamma \in \{10, 1, 0.1\}$ and $C \in \{1, 10, 10^2, \dots, 10^4\}$. The figure shows that our algorithm is consistently faster than the SMO algorithm in all of these settings.

4.6 Discussion

Another important advantage of the proposed approach beyond computational efficiency is that the exact solution path is represented in piecewise linear form. In SVM (and its variants), the decision function $f(\boldsymbol{x})$ also has a piecewise linear form because $f(\boldsymbol{x})$ is linear in the parameters $\boldsymbol{\alpha}$ and b. It enables us to compute the entire path of the validation errors and to select the model with the minimum validation error. Let $\mathcal{V}$ be the validation set and the validation loss is defined as $\sum_{i \in \mathcal{V}} \ell(y_i, f(\boldsymbol{x}_i))$. Suppose that the current weight is expressed as $\boldsymbol{c} + \theta \boldsymbol{d}$ in a critical region $\mathcal{R}$ using a parameter $\theta \in \mathbb{R}$, and the output has the form $f(\boldsymbol{x}_i) = a_i \theta + b_i$ for some scalar constants a_i and b_i. Then the minimum validation error in the current critical region $\mathcal{R}$ can be identified by solving the following optimization problem:

$$\min_{\theta \in \mathbb{R}} \sum_{i \in \mathcal{V}} \ell(y_i, a_i \theta + b_i) \ \text{ s.t. } \boldsymbol{c} + \theta \boldsymbol{d} \in \mathcal{R}. \tag{4.38}$$

After following the entire solution-path, the best model can be selected among the candidates in the finite number of critical regions. In the case of 0-1 loss, i.e.,

$$\ell(y_i, f(\boldsymbol{x}_i)) = I\{y_i \mathrm{sgn}(f(\boldsymbol{x}_i)) = -1\},$$

the problem (4.38) can be solved by monitoring all the points at which $f(\boldsymbol{x}_i) = 0$ (see Fig. 4.5). Furthermore, if the validation error is measured by squared loss

$$\ell(y_i, f(\boldsymbol{x}_i)) = (y_i - f(\boldsymbol{x}_i))^2,$$

the problem (4.38) can be analytically solved in each critical region. As another interesting example, we described how to find the maximum validation NDCG in ranking problem (see Section 4.5.2). In the case of NDCG, the problem (4.38) can be solved by monitoring all the intersections of $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$ such that $y_i \neq y_j$ (see Fig. 4.13)[3] .

Incremental-decremental SVM [24, 57, 68, 79, 80] exploits the piecewise linearity of the solutions. It updates SVM solutions efficiently when instances are added or removed from the training set. The incremental and decremental operations can be implemented using our instance-weight path approach. If we want to add an instance $(\boldsymbol{x}_i, y_i)$, we increase C_i from 0 to some specified value. Conversely, if we want to remove an instance $(\boldsymbol{x}_j, y_j)$, we decrease C_j to 0. The paths generated by these two approaches are different in general. The instance-weight path keeps the optimality of all the instances including currently adding and/or removing ones. On the other hand, the incremental-decremental algorithm does not satisfy the optimality of adding and/or removing ones until the algorithm terminates. When we need to guarantee the optimality at intermediate points on the path, instance-weight path is more useful.

In the parametric programming approach, numerical instabilities sometimes cause computational difficulty. In practical implementation, we usually update several quantities such as $\boldsymbol{\alpha}$, b, $yf(\boldsymbol{x}_i)$, and $\boldsymbol{L}$ from the previous values without calculating them from scratch. However, the rounding error may be accumulated at every breakpoints. We can avoid such accumulation using the *refresh strategy*: re-calculating variables from scratch, for example, every 100 steps (note that, in this case, we need $O(n|\mathcal{M}|^2)$ computations in every 100 steps). Fortunately, such numerical instabilities rarely occurred in our experiments, and the accuracy of the KKT conditions of the solutions were kept high enough.

[3]In NDCG case, the "min" is replaced with "max" in the optimization problem (4.38).

Another (but related) numerical difficulty arises when the matrix $\boldsymbol{M}$ is close to singular. [117] pointed out that if the matrix is singular, the update is no longer unique. To the best of our knowledge, this degeneracy problem is not fully solved in path-following literature. Many heuristics are proposed to circumvent the problem, and we used one of them in the experiments: adding small positive constant to the diagonal elements of kernel matrix. Other strategies are also discussed in [43, 87, 118].

Scalability of our algorithm depends on the size of $\mathcal{M}$ because a linear system with $|\mathcal{M}|$ unknowns must be solved at each breakpoint. Although we can update the Cholesky factor by $O(|\mathcal{M}|^2)$ cost from the previous one, iterative methods such as conjugate-gradients may be more efficient than the direct matrix update when $|\mathcal{M}|$ is fairly large. When $|\mathcal{M}|$ is small the parametric programming approach can be applied to relatively large data sets such as more than tens of thousands of instances.

4.7 Conclusion

In this chapter, we developed solution-path algorithm for the instance-weighted SVMs. Our formulation can be considered as a generalization of the regularization path algorithm [47] for the changes of instance-weights. We have shown that this extension enables to apply solution-path algorithm to various machine learning tasks. Inheriting computational efficiency of the original regularization path algorithm, our approach efficiently updates the solutions. In the experiments, we demonstrated efficiency of our approach in several practical applications.

Chapter 5

Nonlinear Regularization Path for $L2$ Loss SVM

Regularization path algorithms have been proposed to deal with model selection problem in several machine learning approaches. These algorithms allow to compute the entire path of solutions for every value of regularization parameter using the fact that their solution paths have piecewise linear form. In this chapter, we extend the applicability of regularization path algorithm to a class of learning machines that have quadratic loss and quadratic penalty term. This class contains several important learning machines such as squared hinge loss SVM and modified Huber loss SVM. We first show that the solution paths of this class of learning machines have piecewise nonlinear form, and piecewise segments between two breakpoints are characterized by a class of rational functions. Then we develop an algorithm that can efficiently follow the piecewise nonlinear path by solving these rational equations. To solve these rational equations, we use rational approximation technique with quadratic convergence rate, and thus, our algorithm can follow the nonlinear path much more precisely than existing approaches such as predictor-corrector type nonlinear-path approximation. We show the algorithm performance on some artificial and real data sets. The preliminary version of this chapter has appeared in [55].

5.1 Regularization path and loss functions

In this chapter we study binary classification problem with training data points $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^p$ is the input and $y_i \in \{1, -1\}$ is the output class label. We consider a linear discriminant function:

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \Phi(\boldsymbol{x}),$$

where Φ maps the data into some feature space $\mathcal{F}$. A large class of learning machines are formulated as minimization of the following regularized risk function:

$$\min_{\boldsymbol{w}} \quad \sum_{i=1}^{n} \ell(y_i, f(\boldsymbol{x}_i)) + \lambda J(\boldsymbol{w}), \tag{5.1}$$

where ℓ is a loss function, J is a penalty function, and $\lambda > 0$ is a regularization parameter. In order to obtain a discriminant function with good generalization property, we need to select a good λ that appropriately controls the model complexity (model selection). A typical model selection approach is to specify a list of candidate values of λ and then apply cross validation to select the best one. However, this procedure is time-consuming since we need to solve many optimization problems in various settings.

Regularization path algorithm [47] provides an efficient approach to model selection problem. It allows to compute the entire solution path for a range of regularization parameters λ. The regularization path algorithm is built on an optimization technique called *parametric programming* also a.k.a. *path-following* [2,40]. Recently, in the machine learning literature, path-following was used for various purposes (e.g. [7,24,46,57,68,79, 80,107,117]). Most regularization path algorithms in the literature were developed by exploiting the fact that the solution paths in a class of learning machines have *piecewise linear* form. For example, the support vector machine (SVM) [112] characterized by the following hinge loss:

$$\ell(y_i, f(\boldsymbol{x}_i)) = \max(0, 1 - y_i f(\boldsymbol{x}_i)) \tag{5.2}$$

and ℓ_2-norm penalty term: $J(\boldsymbol{w}) = \frac{1}{2}\|w\|_2^2$ is shown to have piecewise linear regularization path [47]. Recently, [95] showed that the regularization path has piecewise linear form if the loss function ℓ is a piecewise quadratic function and the penalty term $J(\boldsymbol{w})$ is a piecewise linear function. The LASSO [109] is a typical example of this class of learning machine. In optimization literature, [92] has derived more general sufficient conditions for piecewise linearity in quadratic and linear programming problems.

In other class of learning machines, the solution path may have nonlinear form. *Predictor corrector* approach [2] is usually adopted for general nonlinear path-following. In predictor corrector approach, the predictor step and the corrector step are iterated: the predictor step approximates the nonlinear (curved) solution path (in many cases, using Taylor expansion), while the corrector step projects the predicted solution to the solution space so that it satisfies the optimality conditions. This approach have been applied to some learning problems [8,66,88]. Some other approaches are also proposed in machine learning literature. For example, [94] proposed second-order approximation algorithm for nonlinear regularization path, in which small step is taken and the approximation is updated at each iteration. As another example, [116] derived an updating formula to obtain the path of solutions along the change of the kernel parameter (such as standard deviation in Gaussian kernel). In these methods, we can only obtain roughly approximated nonlinear path of solutions. If we want these nonlinear approximated solution path to be accurate, the algorithm would be computationally demanding because we need to take very small steps.

In this study we consider a class of learning machines that have quadratic loss and quadratic penalty. This class contains several important learning machines such as *squared hinge loss SVM* and *modified Huber loss SVM*. These loss functions are formulated as

- Squared hinge loss:

$$\ell(y, f(\boldsymbol{x})) = \max(0, 1 - yf(\boldsymbol{x}))^2. \tag{5.3}$$

- Modified Huber loss [123]:

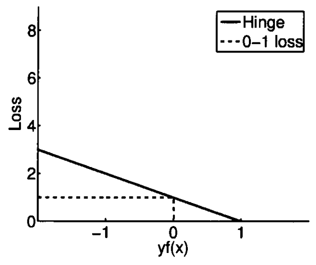
$$\ell(y, f(\boldsymbol{x})) = \begin{cases} 0, & yf(\boldsymbol{x}) > 1, \\ (1 - yf(\boldsymbol{x}))^2, & yf(\boldsymbol{x}) \in [-1, 1], \\ -4yf(\boldsymbol{x}), & yf(\boldsymbol{x}) < -1. \end{cases} \tag{5.4}$$

  Another formulation of Huber-type loss function proposed in [27]:

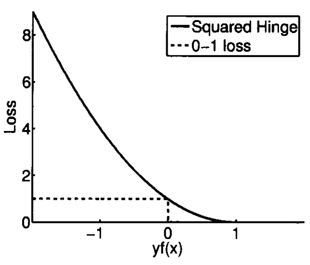
$$\ell(y, f(\boldsymbol{x})) = \begin{cases} 0, & yf(\boldsymbol{x}) > 1 + \varepsilon, \\ \frac{(1+\varepsilon - yf(\boldsymbol{x}))^2}{4\varepsilon}, & |1 - yf(\boldsymbol{x})| \le \varepsilon, \\ 1 - yf(\boldsymbol{x}), & yf(\boldsymbol{x}) < 1 - \varepsilon, \end{cases} \tag{5.5}$$

  where $\varepsilon > 0$ is a parameter. If $\varepsilon \to 0$, this loss function approaches to the hinge loss.

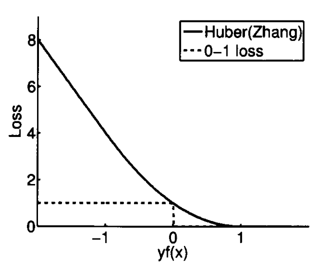
Fig. 5.1 shows these loss functions along with the 0-1 loss. These quadratic loss functions are sometimes preferred to the hinge loss. For example, it is known that this type of loss functions are suited to estimate conditional probability $P(Y = 1 \mid X = \boldsymbol{x})$ (see e.g., [9,123]). Another advantage of these loss functions is their differentiability. Some primal SVM solvers [14,27,60] require differentiable objective function.



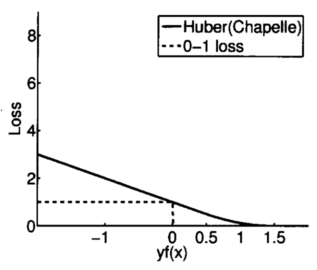
(a) Hinge loss (5.2)



(b) Squared hinge loss (5.3)



(c) Modified Huber loss (5.4)



(d) Modified Huber loss (5.5) ($\varepsilon = 0.5$)

Figure 5.1: Loss functions for classification

Unfortunately, the regularization paths of this class of learning machines (quadratic loss + quadratic penalty) do not exhibit piecewise linear form anymore. To extend the applicability of regularization path algorithm, we develop a nonlinear regularization path algorithm for this class of learning machines. We first show that the solution path of this class of learning machine is represented as piecewise nonlinear form, and the piecewise segment of solutions between two breakpoints are characterized by a class of rational function. The breakpoint itself can be identified solving the rational equations. Then we develop an efficient algorithm that can efficiently follow the piecewise nonlinear path by solving these rational equations. To solve these rational equations, we introduce a rational approximation technique with quadratic convergence rate used in rank-one-update of eigenvalue decompositions [18]. Note that our algorithm is NOT a predictor-corrector type approach. While predictor corrector approach can only follow nonlinear path with rather rough approximation, our algorithm can compute quite accurate path of solutions because we use an efficient iterative procedure with quadratic convergence rate.

5.2 SVM with Huber Type loss

In this paper, we set the penalty term as $J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2$ and the loss function as the following general quadratic loss function:

$$\ell(y, f(\boldsymbol{x})) = \begin{cases} 0, & yf(\boldsymbol{x}) > \rho, \\ (\rho - yf(\boldsymbol{x}))^2, & yf(\boldsymbol{x}) \in [\rho - h, \rho], \\ 2h(\rho - yf(\boldsymbol{x})) - h^2, & yf(\boldsymbol{x}) < \rho - h, \end{cases} \tag{5.6}$$

where $\rho > 0$ and $h > 0$. The loss function (5.6) can represent the previous three quadratic loss functions (5.3)-(5.5) by specifying (ρ, h). If we set $(\rho, h) = (1, \infty), (1, 2)$ and $(1 + \varepsilon, 2\varepsilon)$, the loss function (5.6) is reduced to (5.3),(5.4) and (5.5), respectively[1] . The optimization problem (5.1) is now written as

$$\begin{aligned} \min_{\boldsymbol{w}, \{\xi_i\}_{i=1}^n} \quad & \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \frac{1}{2}\sum_{i=1}^n \phi(\xi_i), \\ \text{s.t.} \quad & \rho - y_i f(\boldsymbol{x}_i) \le \xi_i, \\ & \xi_i \ge 0, i = 1, \cdots, n, \end{aligned}$$

where

$$\phi(\xi) = \begin{cases} \xi_i^2, & \xi \in [0, h], \\ 2h\xi_i - h^2, & \xi > h. \end{cases}$$

We derive the dual problem using the same approach in [30]. Introducing Lagrange multipliers $\alpha_i, \eta_i \ge 0$, $i = 1, \cdots, n$, we can write the corresponding Lagrangian as

$$\begin{aligned} L = & \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \frac{1}{2}\sum_{i=1}^n \phi(\xi_i) + \\ & \sum_{i=1}^n \alpha_i\{\rho - y_i \boldsymbol{w}^\top \Phi(\boldsymbol{x}_i) - \xi_i\} - \sum_{i=1}^n \eta_i \xi_i. \end{aligned} \tag{5.7}$$

Setting the derivatives w.r.t. primal variables $\boldsymbol{w}$ and ξ_i to zero, we obtain

$$\frac{\partial L}{\partial \boldsymbol{w}} = \mathbf{0} \quad \Leftrightarrow \quad \boldsymbol{w} = \frac{1}{\lambda}\sum_{i=1}^n \alpha_i y_i \Phi(\boldsymbol{x}_i), \tag{5.8}$$

$$\begin{aligned} \frac{\partial L}{\partial \xi_i} = 0 \quad \Leftrightarrow & \quad \frac{1}{2}\frac{\partial \phi(\xi_i)}{\partial \xi_i} = \alpha_i + \eta_i, \\ \Leftrightarrow & \quad \begin{cases} \xi_i = \alpha_i + \eta_i, & \xi_i \in [0, h], \\ h = \alpha_i + \eta_i, & \xi_i > h. \end{cases} \end{aligned} \tag{5.9}$$

[1]To represent (5.5) by (5.6), we further need to multiply (5.6) by $1/(4\varepsilon)$. This difference can be absorbed by the scale of regularization parameter.

Substituting (5.8) into (5.7), we obtain

$$\begin{aligned} L &= -\frac{1}{2\lambda}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j Q_{ij} + \rho\sum_{i=1}^{n}\alpha_i \\ &\quad + \sum_{i=1}^{n}\{\frac{1}{2}\phi(\xi_i) - (\alpha_i + \eta_i)\xi_i\}, \end{aligned}$$

where $Q_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Using (5.9), we can eliminate ξ_i from Lagrangian. If $\xi_i \in [0, h]$, we have

$$\begin{aligned} \frac{1}{2}\phi(\xi_i) - (\alpha_i + \eta_i)\xi_i &= \frac{1}{2}\xi_i^2 - (\alpha_i + \eta_i)\xi_i \\ &= -\frac{1}{2}(\alpha_i + \eta_i)^2. \end{aligned}$$

On the other hand, if $\xi_i > h$,

$$\begin{aligned} \frac{1}{2}\phi(\xi_i) - (\alpha_i + \eta_i)\xi_i &= h\xi_i - \frac{1}{2}h^2 - h\xi_i \\ &= -\frac{1}{2}(\alpha_i + \eta_i)^2. \end{aligned}$$

Then, the dual problem is represented as

$$\begin{aligned} \max_{\boldsymbol{\alpha},\boldsymbol{\eta}} \quad & W(\boldsymbol{\alpha}, \boldsymbol{\eta}) = \\ & -\frac{1}{2\lambda}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j Q_{ij} \\ & +\rho\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}(\alpha_i + \eta_i)^2, \\ \text{s.t.} \quad & \alpha_i, \eta_i \geq 0,\ i = 1, \cdots, n, \\ & \alpha_i + \eta_i \leq h,\ i = 1, \cdots, n, \end{aligned}$$

where $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_n]^\top$ and $\boldsymbol{\eta} = [\eta_1, \cdots, \eta_n]^\top$. Since $\alpha_i, \eta_i \geq 0$, an inequality $W(\boldsymbol{\alpha}, \boldsymbol{\eta}) \leq W(\boldsymbol{\alpha}, \mathbf{0})$ holds for every feasible $\boldsymbol{\alpha}$ and $\boldsymbol{\eta}$. Therefore, we can delete $\boldsymbol{\eta}$ and the dual problem is finally written as

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & W(\boldsymbol{\alpha}) = \\ & -\frac{1}{2\lambda}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j Q_{ij} \\ & +\rho\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\alpha_i^2, \qquad (5.10) \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq h,\ i = 1, \cdots, n. \end{aligned}$$

The discriminant function $f : \mathcal{X} \to \mathbb{R}$ is formulated as

$$f(\boldsymbol{x}) = \frac{1}{\lambda}\Big(\sum_{i=1}^{n} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x})\Big).$$

5.3 Nonlinear Regularization Path

In this section we derive nonlinear regularization path algorithm for quadratic loss SVM.

5.3.1 Optimal Solution and Regularization Parameter

At the optimal, α_i, $i = 1, \cdots, n$, satisfies the following first-order optimality conditions (KKT conditions):

$$\begin{aligned} \frac{\partial W}{\partial \alpha_i} &= -\frac{1}{\lambda}\sum_{j=1}^{n} Q_{ij}\alpha_j + \rho - \alpha_i \\ &= -y_i f(\boldsymbol{x}_i) + \rho - \alpha_i \begin{cases} \geq 0, & \alpha_i = h, \\ = 0, & \alpha_i \in (0, h), \\ \leq 0, & \alpha_i = 0. \end{cases} \end{aligned} \tag{5.11}$$

Using these relationships, we define the following index sets:

$$\begin{aligned} \mathcal{L} &= \{i \mid y_i f(\boldsymbol{x}_i) \leq \rho - h,\ \alpha_i = h\}, \\ \mathcal{C} &= \{i \mid y_i f(\boldsymbol{x}_i) = \rho - \alpha_i,\ \alpha_i \in (0, h)\}, \\ \mathcal{R} &= \{i \mid y_i f(\boldsymbol{x}_i) \geq \rho,\ \alpha_i = 0\}. \end{aligned} \tag{5.12}$$

The regularization path algorithm keeps track of these sets while the regularization parameter λ is perturbed.

In what follows, the subscription by an index set, such as $\boldsymbol{v}_{\mathcal{C}}$ for a vector $\boldsymbol{v} \in \mathbb{R}^n$, indicates a subvector of $\boldsymbol{v}$ whose elements are indexed by $\mathcal{C}$. Similarly, the subscription by two index sets, such as $\boldsymbol{M}_{\mathcal{C},\mathcal{L}}$ for a matrix $\boldsymbol{M} \in \mathbb{R}^{n \times n}$, denotes a submatrix whose rows are indexed by $\mathcal{C}$ and the columns are indexed by $\mathcal{L}$. Principal submatrix such as $\boldsymbol{Q}_{\mathcal{C},\mathcal{C}}$ is abbreviated as $\boldsymbol{Q}_{\mathcal{C}}$.

The KKT conditions (5.11) for $i \in \mathcal{C}$ can be written as

$$\sum_{j \in \mathcal{C}} Q_{ij}\alpha_j + \lambda\alpha_j = \rho\lambda - h\sum_{j \in \mathcal{L}} Q_{ij}, \ i \in \mathcal{C}.$$

Using matrix notation, it is written as

$$\Big(\boldsymbol{Q}_{\mathcal{C}} + \lambda \boldsymbol{I}\Big)\boldsymbol{\alpha}_{\mathcal{C}} = \rho\lambda\boldsymbol{1} - h\boldsymbol{Q}_{\mathcal{C},\mathcal{L}}\boldsymbol{1}, \tag{5.13}$$

where (i,j)-th entry of $\boldsymbol{Q}$ is Q_{ij} and $\boldsymbol{I}$ is an identity matrix with appropriate size. Let the eigenvalue decomposition (EVD) of $\boldsymbol{Q}_{\mathcal{C}}$ be

$$\boldsymbol{Q}_{\mathcal{C}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top,$$

where $\boldsymbol{\Sigma} \in \mathbb{R}^{|\mathcal{C}|\times|\mathcal{C}|}$ is a diagonal matrix whose i-th diagonal entry σ_i is the i-th eigenvalue of $\boldsymbol{Q}_{\mathcal{C}}$ and $\boldsymbol{U} \in \mathbb{R}^{|\mathcal{C}|\times|\mathcal{C}|}$ is an orthogonal matrix whose i-th column is the i-th eigenvector of $\boldsymbol{Q}_{\mathcal{C}}$. It is easy to show that the EVD of $\boldsymbol{Q}_{\mathcal{C}} + \lambda\boldsymbol{I}$ is explicitly obtained as

$$\boldsymbol{Q}_{\mathcal{C}} + \lambda\boldsymbol{I} = \boldsymbol{U}(\boldsymbol{\Sigma} + \lambda\boldsymbol{I})\boldsymbol{U}^\top, \tag{5.14}$$

Using (5.13) and (5.14), we can compute $\boldsymbol{\alpha}_{\mathcal{C}}$ as follows:

$$\boldsymbol{\alpha}_{\mathcal{C}} = \boldsymbol{U}(\boldsymbol{\Sigma} + \lambda\boldsymbol{I})^{-1}\boldsymbol{U}^\top\Big\{\rho\lambda\boldsymbol{1} - h\boldsymbol{Q}_{\mathcal{C},\mathcal{L}}\boldsymbol{1}\Big\}. \tag{5.15}$$

Let us denote the index of the set $\mathcal{C}$ as

$$\mathcal{C} = \{c_1, \cdots, c_{\ell_{\mathcal{C}}}\},$$

where $\ell_{\mathcal{C}} = |\mathcal{C}|$. Then, we can write (5.15) by element-wise notation:

$$\alpha_{c_i} = \sum_{j=1}^{\ell_{\mathcal{C}}}\sum_{k=1}^{\ell_{\mathcal{C}}} \frac{u_{ik}u_{jk}\{\rho\lambda - hq^{\mathcal{L}}_{c_j}\}}{\sigma_k + \lambda}, \quad i = 1, \cdots, \ell_{\mathcal{C}},$$

where u_{ij} is a (i,j)-th entry of $\boldsymbol{U}$ and $q^{\mathcal{L}}_i = \sum_{j\in\mathcal{L}} Q_{ij}$. This equation can be reduced to the following form:

$$\alpha_{c_i} = \rho - \sum_{k=1}^{\ell_{\mathcal{C}}} \frac{\zeta_{ik}}{\sigma_k + \lambda}, \quad i = 1, \cdots, \ell_{\mathcal{C}}, \tag{5.16}$$

where

$$\zeta_{ik} = u_{ik}\sum_{j=1}^{\ell_{\mathcal{C}}} u_{jk}(\rho\sigma_k + hq^{\mathcal{L}}_{c_j}).$$

Using (5.16), $y_i f(\boldsymbol{x}_i)$ can be written as

$$y_i f(\boldsymbol{x}_i) = \frac{1}{\lambda}\left(d_i - \sum_{k=1}^{\ell_{\mathcal{C}}} \frac{\omega_{ik}}{\sigma_k + \lambda}\right), \tag{5.17}$$

Table 5.1: Event categorization

Event	The change of index	The change of inequality
type 1	$i \in \mathcal{C}$ migrates to $\mathcal{R}$	$\alpha_i > 0$ to $\alpha_i = 0$
type 2	$i \in \mathcal{C}$ migrates to $\mathcal{L}$	$\alpha_i < h$ to $\alpha_i = h$
type 3	$i \in \mathcal{R}$ migrates to $\mathcal{C}$	$y_i f(\boldsymbol{x}_i) > \rho$ to $y_i f(\boldsymbol{x}_i) = \rho$
type 4	$i \in \mathcal{L}$ migrates to $\mathcal{C}$	$y_i f(\boldsymbol{x}_i) < \rho - h$ to $y_i f(\boldsymbol{x}_i) = \rho - h$

where

$$\begin{aligned} d_i &= \rho \sum_{j \in \mathcal{C}} Q_{ij} + h q_i^{\mathcal{L}}, \\ \omega_{ik} &= \sum_{\ell=1}^{\ell_{\mathcal{C}}} Q_{ic_\ell} \zeta_{\ell k}. \end{aligned}$$

The above derivation indicates that, if we have complete information on the index sets $\mathcal{L}$, $\mathcal{C}$, and $\mathcal{R}$, the set of model parameters $\{\alpha_i\}_{i=1}^n$ can be represented as a function of the regularization parameter λ. In particular, for a data point $i \in \mathcal{C}$, the corresponding parameter α_i is formulated by a rational function (5.16).

5.3.2 Event Detection

Equation (5.16) holds only when the indices in the sets $\mathcal{C}$, $\mathcal{L}$ and $\mathcal{R}$ are not changed. The change of these indices is called an *event*, and a λ is called an *event point* if there is an event at λ. Events in our path-following algorithm are categorized into four types in Table 5.1. Each type of events is relevant to the inequality constraints in the definitions of the sets $\mathcal{C}$, $\mathcal{L}$ and $\mathcal{R}$ in (5.12). In the case of piecewise linear path, event points are easily detected by solving linear equations. In our nonlinear path, however, we need to solve nonlinear equations to detect the event points. To this end, we introduce rational approximation approach. Rational approximation has been used in the context of rank-one-update of EVD [18].

Here, we consider how to detect an event point when we decrease λ from the current value λ_0 (the same discussion holds when we increase λ). For the type 1 in Table 5.1, we need to find λ^* such that $\alpha_{c_i} > 0, c_i \in \mathcal{C}$, becomes $\alpha_{c_i} = 0$. Let us define $t = -\lambda$. Then, we increase t from $t_0 = -\lambda_0$ until we find t^* such that

$$\rho - \sum_{k=1}^{\ell_{\mathcal{C}}} \frac{\zeta_{ik}}{\sigma_k - t^*} = 0, \; t^* \in (t_0, 0). \tag{5.18}$$

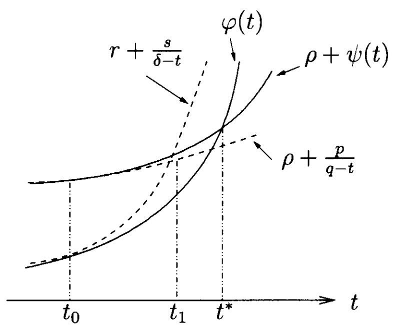


Figure 5.2: Rational approximation for the type 1 in Table 5.1. Note that $\rho+\psi_\zeta(t) > \varphi_\zeta(t), t \in (t_0, t^*)$. The approximated solution t_1 can be computed via a quadratic equation. Iterating this, we can obtain a sequence of approximated solution with quadratic convergence.

If this equation has multiple solutions, we choose the minimum one as t^*. We note the fact that (5.18) is similar to the *secular equation* which often arises in rank-one-update problem of the EVD [18, 44, 45]. In this paper we introduce *rational approximation* approach [18] for solving (5.18). Let us define

$$\psi_\zeta(t) \equiv \sum_{k \in \{k | \zeta_{ik} < 0\}} \frac{-\zeta_{ik}}{\sigma_k - t}, \quad \varphi_\zeta(t) \equiv \sum_{k \in \{k | \zeta_{ik} > 0\}} \frac{\zeta_{ik}}{\sigma_k - t}.$$

Using these functions, (5.18) is represented as

$$\rho + \psi_\zeta(t^*) = \varphi_\zeta(t^*), \quad t^* \in (t_0, 0).$$

Since the kernel matrix $\boldsymbol{Q}$ is positive semi-definite, $\sigma_k \geq 0$, $k = 1, \cdots, \ell_C$. Therefore, we see that both $\rho + \psi_\zeta(t)$ and $\varphi_\zeta(t)$ are increasing convex functions of $t \in (t_0, 0)$. We approximate ψ_ζ and φ_ζ by their lower and upper bounds (see Fig. 5.2):

$$\psi_\zeta(t) > \frac{p}{q-t}, \quad \varphi_\zeta(t) < r + \frac{s}{\delta - t}, \quad t \in (t_0, 0), \tag{5.19}$$

where $\delta = \min\{\sigma_k \mid \zeta_{ik} > 0\}$. These upper and lower bound functions are the 1st order local approximations to ψ_ζ and φ_ζ at t_0, respectively. The four parameters p, q, r and s are defined to satisfy these requirements, *i.e.*, they are defined as

$$\begin{aligned} p &= \frac{\{\psi_\zeta(t_0)\}^2}{\psi'_\zeta(t_0)}, & r &= \varphi_\zeta(t_0) - \Delta \varphi'_\zeta(t_0), \\ q &= t_0 + \frac{\psi_\zeta(t_0)}{\psi'_\zeta(t_0)}, & s &= \Delta^2 \varphi'_\zeta(t_0), \end{aligned} \tag{5.20}$$

where $\psi'_\zeta(t) = \partial \psi_\zeta(t)/\partial t$, $\varphi'_\zeta(t) = \partial \varphi_\zeta(t)/\partial t$, and $\Delta = \delta - t_0$. Then, inequalities (5.19) hold as in [18]. Using the approximation by these simple rational functions in (5.19), we compute an approximate solution t_1 by solving

$$\rho + \frac{p}{q - t_1} = r + \frac{s}{\delta - t_1}.$$

This equation can be reduced to a quadratic equation, and we choose the minimum one in $(t_0, 0)$ as t_1. In case we have no solution in $(t_0, 0)$, the point $i \in \mathcal{C}$ is disregarded for the moment because it has no chance to define the next event point. Given t_1, we iterate the same procedure to obtain new approximate solution $t_2 \in (t_1, 0)$. As a consequence, we can produce a sequence $\{t_k\}$ that approaches t^* from the left.

For the type 2 in Table 5.1, we need to find λ^* such that $\alpha_{c_i} < h, c_i \in \mathcal{C}$, becomes $\alpha_{c_i} = h$. We increase t from t_0 until we find t^* such that

$$\rho - \sum_{k=1}^{\ell_C} \frac{\zeta_{ik}}{\sigma_k - t^*} = h, \ t^* \in (t_0, 0).$$

This can be written as

$$\rho + \psi_\zeta(t^*) = \phi_\zeta(t^*) + h, \ t^* \in (t_0, 0),$$

We use the following bounds:

$$\psi_\zeta(t) < r + \frac{s}{\delta - t}, \ \varphi_\zeta(t) > \frac{p}{q - t}, \ t \in (t_0, 0),$$

where $\delta = \min\{\sigma_k \mid \zeta_{ik} < 0\}$. Since $\rho + \psi_\zeta(t) < \varphi_\zeta(t) + h, \ t \in (t_0, t^*)$, we need to set the upper bound to ψ_ζ and the lower bound to φ_ζ. Note that p, q, r and s are computed by alternating ψ_ζ and φ_ζ in (5.20):

$$p = \frac{\{\varphi_\zeta(t_0)\}^2}{\varphi'_\zeta(t_0)}, \quad r = \psi_\zeta(t_0) - \Delta\psi'_\zeta(t_0),$$
$$q = t_0 + \frac{\varphi_\zeta(t_0)}{\varphi'_\zeta(t_0)}, \quad s = \Delta^2\psi'_\zeta(t_0).$$

For the type 3 of Table 5.1, we have to detect λ^* such that $y_i f(\boldsymbol{x}_i) > \rho, \ i \in \mathcal{R}$, becomes $y_i f(\boldsymbol{x}_i) = \rho$. Using (5.17), we obtain the following equation:

$$d_i - \sum_{k=1}^{\ell_C} \frac{\omega_{ik}}{\sigma_k - t^*} = -\rho t^*, \ t^* \in (t_0, 0). \tag{5.21}$$

As in the previous two types of cases, we define the following functions:

$$\psi_\omega(t) \equiv \sum_{k \in \{k \mid \omega_{ik} < 0\}} \frac{-\omega_{ik}}{\sigma_k - t}, \ \varphi_\omega(t) \equiv \sum_{k \in \{k \mid \omega_{ik} > 0\}} \frac{\omega_{ik}}{\sigma_k - t}.$$

Then, (5.21) can be written as

$$d_i + \psi_\omega(t^*) + \rho t^* = \varphi_\omega(t^*),\ t^* \in (t_0, 0). \tag{5.22}$$

Both sides of equation are increasing convex functions of $t \in (t_0, 0)$. Since $d_i + \psi_\omega(t) + \rho t > \varphi_\omega(t)$, $t \in (t_0, t^*)$, we replace ψ_ω by its lower bound and φ_ω by its upper bound:

$$d_i + \frac{p}{q - t_1} + \rho t_1 = r + \frac{s}{\delta - t_1}, \tag{5.23}$$

where $\delta = \min\{\sigma_k \mid \omega_{ik} > 0\}$. p, q, r and s are computed by (5.20) with ψ_ζ replaced by ψ_ω and φ_ζ replaced by φ_ω. We can easily solve (5.23) because it is reduced to a cubic equation. To detect the event, we only need the minimum solution of that cubic equation in $(t_0, 0)$. Once we obtain t_1, we can iterate rational approximation in the same way as the type 1 and 2.

For the last type 4 of Table 5.1, we solve

$$d_i - \sum_{k=1}^{\ell_C} \frac{\omega_{ik}}{\sigma_k - t} = -(\rho - h)t^*,\ t^* \in (t_0, 0),$$

to find λ^* where $y_i f(\boldsymbol{x}_i) < \rho - h$, $i \in \mathcal{L}$, reaches a boundary $y_i f(\boldsymbol{x}_i) = \rho - h$. This can be written as

$$d_i + \psi_\omega(t^*) + (\rho - h)t^* = \varphi_\omega(t^*),\ t^* \in (t_0, 0). \tag{5.24}$$

When $h \leq 1$, both sides of the equation are increasing convex functions of $t \in (t_0, 0)$. Even if $h > 1$, we can make the increasing convex functions by moving the term $(\rho - h)t^*$ to the right hand side. In this case, we approximate ψ_ω by its upper bound and φ_ω by its lower bound using the rational approximation.

By checking all the four types in Table 5.1, we obtain λ^*'s for each inequality constraint. The event point is determined as the maximum λ^* among those candidates.

5.3.3 Advantages of The Rational Approximation and Cutoff Strategy

Rational approximation generates approximate solution sequence $\{t_k\}$ for a nonlinear equation expressed by two monotonic convex functions. The sequence $\{t_k\}$ can reach $t_k \in [t^* - \varepsilon, t^*]$ by the finite number of iterations for arbitrary small $\varepsilon > 0$. This convergence is guaranteed for any starting point $t_0 < 0$ (note, in contrast, that Newton method may be trapped in periodic cycle). Furthermore, if we assume that the gradient of nonlinear equation is not 0 at t^*, we can prove quadratic convergence of the approximation (we provide the proof in Appendix C).

Since $\{t_k\}$ is a monotonically increasing sequence, it approaches t^* from the left. Exploiting this property, we can sometimes terminate iteration for approximation before convergence without affecting the accuracy of the event detection. Suppose we have obtained $\lambda^*_{\max} = -t^*_{\min}$ which is the maximum λ^* among some of the inequalities in Table 5.1. When we investigate the next inequality, we can terminate the approximation at the i-th iteration if t_i becomes larger than $t^*_{\min}$. This is because we only need the minimum t^* to detect the event. We refer to this early termination strategy as *cutoff* (Note that this strategy has no effects on the accuracy of solutions).

5.3.4 Empty Set $\mathcal{C}$ and Initialization

When we have no data points in $\mathcal{C}$, α_i is either $\alpha_i = 0$, $i \in \mathcal{R}$, or $\alpha_i = 1$, $i \in \mathcal{L}$. Then, $y_i f(\boldsymbol{x}_i)$ can be written as

$$y_i f(\boldsymbol{x}_i) = \frac{h}{\lambda}\Big(\sum_{j \in \mathcal{L}} Q_{ij}\Big).$$

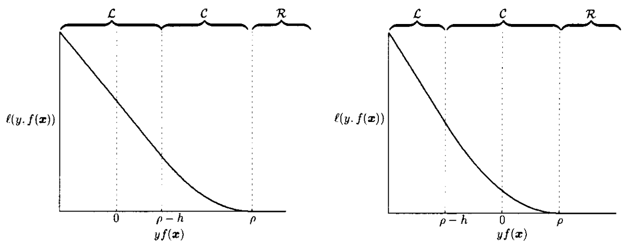
Using this, we can easily check the type 3 and 4 in Table 5.1.

We can use the optimal solution $\boldsymbol{\alpha}$ for any $\lambda > 0$ as a starting point of the regularization path. Although optimal $\boldsymbol{\alpha}$ at initial λ can be obtained by directly solving the optimization problem (5.10) using, for instance, active set method there is a more appealing approach for initialization. We can find a trivial solution for sufficiently large λ, and we may easily obtain the initial solution by following the path with decreasing λ. We explain how to obtain those trivial solutions in the following three cases: (i) $0 < \rho - h$, (ii) $0 > \rho - h > -\infty$, (iii) $h = \infty$ (squared hinge loss).

When $\lambda = \infty$, optimal $\boldsymbol{w}$ is obviously $\boldsymbol{0}$, and then $y_i f(\boldsymbol{x}_i) = 0$ for $i = 1, \cdots, n$. Thus, in the first case (i) $0 < \rho - h$, all the data points are in $\mathcal{L}$ (see Fig. 5.3(a)). We search the first event point λ_1 so that $i \in \mathcal{L}$ moves to $\mathcal{C}$ using the same approach to empty $\mathcal{C}$.

In the second case (ii), all the data points are in $\mathcal{C}$ as $\lambda \to \infty$ (see Fig. 5.3(b)). Then $y_i f(\boldsymbol{x}_i)$ must be in the following range:

$$y_i f(\boldsymbol{x}_i) \in [\rho - h, \rho], \quad i = 1, \cdots, n. \tag{5.25}$$



(a) case (i): $0 < \rho - h$ (b) case (ii): $0 > \rho - h > -\infty$

Figure 5.3: The loss function and $\rho - h$

On the other hand, from $\alpha_i \in [0, h]$, we know $y_i f(\boldsymbol{x}_i)$ has the following bounds:

$$y_i f(\boldsymbol{x}_i) = \frac{1}{\lambda}\sum_{j=1}^{n} Q_{ij}\alpha_j \in \frac{h}{\lambda}[\sum_{j=1}^{n}\min(0, Q_{ij}), \sum_{j=1}^{n}\max(0, Q_{ij})]$$

If the inequalities

$$\begin{aligned}\rho - h &\leq \frac{h}{\lambda}\sum_{j=1}^{n}\min(0, Q_{ij}), \quad i = 1, \cdots, n, \\ \rho &\geq \frac{h}{\lambda}\sum_{j=1}^{n}\max(0, Q_{ij}), \quad i = 1, \cdots, n,\end{aligned}$$

are hold, optimal solution of such λ satisfies (5.25). We can easily calculate λ which satisfies the above inequalities.

In the third case (iii), as in the previous case, all the data points are in $\mathcal{C}$ as $\lambda \to \infty$. However since α_i has no upper bound, we can not apply the same strategy as the previous case. We use the following lower bound:

$$\begin{aligned}\alpha_{\mathcal{C}_i} &= \rho - \sum_{k=1}^{\ell_{\mathcal{C}}}\frac{\zeta_{ik}}{\sigma_k + \lambda} \\ &\geq \rho - \sum_{k\in\{k|\zeta_{ik}>0\}}\frac{\zeta_{ik}}{\sigma_{\min} + \lambda},\end{aligned}$$

where $\sigma_{\min} = \min\{\sigma_k \mid k = 1, \cdots, \ell_{\mathcal{C}}\}$. Since this lower bound monotonically approaches to ρ as $\lambda \to \infty$, all $\alpha_{\mathcal{C}_i}$'s are positive at some large λ. We can find such λ by the following:

$$\max\left\{\frac{\sum_{k\in\{k|\zeta_{ik}>0\}}\zeta_{ik}}{\rho} - \sigma_{\min} \,\middle|\, i = 1, \cdots, \ell_{\mathcal{C}}\right\}.$$

If we set $\rho = h$, then $y_i f(\boldsymbol{x}_i) = 0$ is a boundary between $\mathcal{L}$ and $\mathcal{C}$. In this case, detecting the first event is more difficult than the previous three cases. However, even if we can not find trivial initial solution, we can start our path algorithm from any λ and its optimal solution.

5.3.5 Computational Complexity

The major computational cost of each iteration of the regularization path involves

- The eigenvalue decomposition of $\ell_C \times \ell_C$ matrix $\boldsymbol{Q}_C$ with $O(\ell_C^3)$.
- Computing $\zeta_{ik}, i = 1, \cdots, \ell_C, k = 1, \cdots, \ell_C$. This needs $O(\ell_C^2)$ computations.
- Computing $\omega_{ik}, i \in \mathcal{R} \cup \mathcal{L}, k = 1, \cdots, \ell_C$. Since we need to compute $\sum_{\ell=1}^{\ell_C} Q_{ic_\ell}\zeta_{\ell k}$ for each ω_{ik}, it takes $O(\ell_C^2(\ell_R + \ell_L))$, where $\ell_R = |\mathcal{R}|$ and $\ell_L = |\mathcal{L}|$.
- Solving nonlinear equations to detect the event. We solve $2\ell_C + \ell_R + \ell_L = O(n)$ equations using the rational approximation. In each iteration of the rational approximation, we have $O(\ell_C)$ computation (mainly for re-calculating α_i or $y_i f(\boldsymbol{x}_i)$). If we assume that the rational approximation terminates at the I_{ra}-th iteration, the total cost is roughly $O(I_{ra}\ell_C n)$.

Thus, the approximate complexity for one iteration of the regularization path is $O(\ell_C^2 n + I_{ra}\ell_C n)$. Note that ℓ_C changes each iteration. If we assume I_{ra} is small enough, it can be considered as $O(\ell_C^2 n)$. As we will see in later experiments, the rational approximation converges very quickly. $O(\ell_C^2 n)$ is similar to the cost of a single SVM training. Therefore, an event detection in our algorithm is as costly as re-training the SVM at each event point. However, empirical results in the next section suggests that our algorithm is an order of magnitude faster than the same number of the SVM re-training as the number of breakpoints (using SMO algorithm [90, 115]).

5.4 Experiments

To demonstrate our algorithm, we show some numerical results on artificial and real data sets. Using our algorithm, we traced the sequence of event points $\lambda_1 > \lambda_2 > \cdots > \lambda_L$ where λ_1 is the first event point computed by the initialization in Subsection 5.3.4 and λ_L is the first event point which becomes smaller than 10^{-5}. The Gaussian RBF kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$ is used. We set kernel parameter as $\gamma = 1/d$, where d is the number of features (This is a default setting in LIBSVM [25]). In the event detection, we iterate rational approximation until approximation error becomes less than 10^{-12}. We investigated the performances of our algorithm for modified Huber loss function and squared hinge loss. For the former loss function $(\rho, h) = (1.1, 0.2)$ and for the latter loss function $(\rho, h) = (1, \infty)$.

We compared the CPU time of our algorithm with the SMO (Sequential Minimal Optimization) algorithm [90]. Since we did not use the explicit bias term for simplicity, the dual problem (5.10) has no equality constraints. Then, the SMO algorithm is adapted to optimize only one parameter α_i per iteration [115]. We select updating index i by

$$i = \begin{cases} i_{\mathrm{up}}, & \text{if} \quad g_{i_{\mathrm{up}}} > -g_{i_{\mathrm{down}}}, \\ i_{\mathrm{down}}, & \text{if} \quad -g_{i_{\mathrm{down}}} > g_{i_{\mathrm{up}}}, \end{cases}$$

where $g_i = \partial W / \partial \alpha_i$ and

$$i_{\mathrm{up}} = \operatorname*{argmax}_{j \in \{j | \alpha_j < 1\}} g_j, \; i_{\mathrm{down}} = \operatorname*{argmax}_{j \in \{j | \alpha_j > 0\}} -g_j.$$

The SMO algorithm stops when $|g_i| < 10^{-6}$ are satisfied. We confirmed that the solutions which is obtained by our λ-path algorithm satisfied this condition at all of the breakpoints. We ran the SMO algorithm at L regularization parameters $\lambda^{-1} = \{10^{p_1}, \cdots, 10^{p_L}\}$ where L is the number of the events in regularization path. We set $p_1 = \log_{10} \lambda_1^{-1}$ and uniformly took L values from $[p_1, 4]$. We used the alpha seeding approaches in the SMO, i.e., solution at the previous C is used to produce initial estimates of α_i's. We examined *direct alpha reuse* and *scaling all alphas* strategies (see [34] for detail).

Our regularization path algorithm were mostly implemented in C++. For efficient matrix computations (e.g. matrix vector multiplication or the eigenvalue decomposition), we used LAPACK [3] routine. On the other hand, the SMO algorithm was written solely by C++ on the basis of the-state-of-the-art SVM solver LIBSVM [25]. In both algorithms, we computed and cached the entire kernel matrix at the beginning.

5.4.1 Artificial Data

First, we used simple artificial data set. We generated data points $(\boldsymbol{x}, y) \in \mathbb{R}^2 \times \{+1, -1\}$ using the 2-dimensional Normal distributions:

$$\begin{aligned} p(\boldsymbol{x} \mid y = +1) &= \frac{1}{2}\mathcal{N}(\boldsymbol{\mu}_1^+, \boldsymbol{\Sigma}_1^+) + \frac{1}{2}\mathcal{N}(\boldsymbol{\mu}_2^+, \boldsymbol{\Sigma}_2^+), \\ p(\boldsymbol{x} \mid y = -1) &= \mathcal{N}(\boldsymbol{\mu}^-, \boldsymbol{\Sigma}^-), \end{aligned}$$

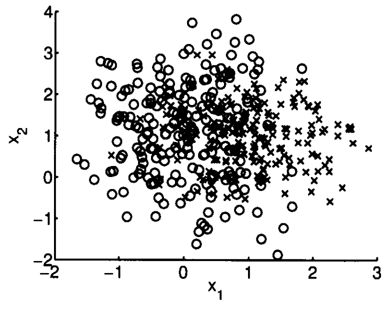


Figure 5.4: An example of artificial data set

where,

$$\boldsymbol{\mu}_1^+ = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \; \boldsymbol{\Sigma}_1^+ = \begin{bmatrix} 0.5 & -0.1 \\ -0.1 & 0.5 \end{bmatrix},$$
$$\boldsymbol{\mu}_2^+ = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \; \boldsymbol{\Sigma}_2^+ = \begin{bmatrix} 0.5 & 0.1 \\ 0.1 & 0.5 \end{bmatrix},$$
$$\boldsymbol{\mu}^- = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \; \boldsymbol{\Sigma}^- = \begin{bmatrix} 0.5 & -0.1 \\ -0.1 & 0.5 \end{bmatrix}.$$

We generated $n \in \{100, 200, 400\}$ training data points and the sizes of each class is set to be $n/2$. We normalized each dimension of $\boldsymbol{x}$ to $[0, 1]$. Fig.5.4 shows an example of data set when $n = 400$. For each size n, we generated 10 data sets to alleviate random sampling effect and computed results as average of 10 runs.

Table 5.2 and 5.3 show the results of the modified Huber SVM. Table 5.2 compares the CPU times on modified Huber SVM, where figures in the table are the average (and the standard deviations in the round bracket) of 10 runs. We refer to our regularization path algorithm as λ-path in Table 5.2. We see that λ-path is much faster than the SMO algorithm (we observed, as is well known, the SMO took relatively longer time when $C = \lambda^{-1}$ was large [16] (data not shown)). When we did not use cutoff strategy, λ-path becomes much slower.

Table 5.3 shows the number of the events L and the mean number of iterations in a rational approximation per one nonlinear equation. Some authors suggested that the number of the events appears to be roughly proportional to the number of training points [46, 47, 117]. Although this is only from empirical observations, we also see that the number of the events increased linearly with n in this simple artificial data sets. Even if we did not use cutoff strategy, iterations of rational approximation were only about 10-15 iterations. With the use of cutoff strategy, the average number of iterations

Table 5.2: Computational cost of the modified Huber SVM ($\rho = 1.1, h = 0.2$) for artificial data (sec.)

n	λ-path	λ-path	SMO	SMO	SMO
		no cutoff	from scratch	direct alpha reuse	scaling all alphas
100	0.28 (0.03)	2.80 (0.33)	68.44 (20.41)	49.40 (16.08)	37.73 (12.27)
200	1.10 (0.07)	13.43 (1.08)	548.36 (105.30)	358.15 (80.13)	271.89 (59.33)
400	4.72 (0.33)	57.06 (2.87)	4021.58 (535.73)	2285.46 (261.51)	1635.52 (224.54)

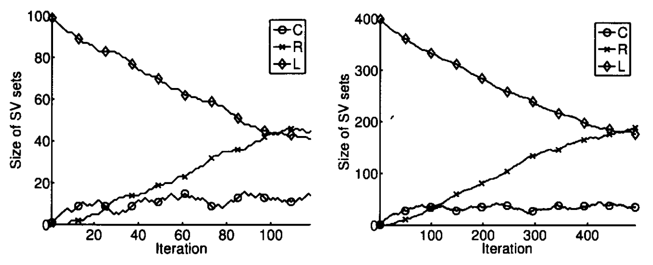
Table 5.3: The number of the events and the mean number of iterations of a rational approximation of the modified Huber SVM ($\rho = 1.1, h = 0.2$) for artificial data. (The figures in the table are the average (and the standard deviation) of 10 runs)

n	#events L	iteration	iteration (no cutoff)
100	150.00 (14.79)	1.20 (0.04)	11.99 (0.51)
200	286.80 (15.45)	1.12 (0.01)	14.72 (0.46)
400	532.80 (33.02)	1.07 (0.01)	16.28 (0.44)

became close to 1.

Fig. 5.5 shows how the sizes of index sets $\mathcal{C}, \mathcal{R}$ and $\mathcal{L}$ change in the λ-path. Each plot is one of the 10 runs of $n = 100$ and 400. The two plots for $n = 100$ and $n = 400$ look similar except their scale.

Table 5.4 and 5.5 show the results of the squared hinge SVM. We see similar results to the modified Huber SVM case. Our λ-path algorithm is faster than the SMO. Since the squared hinge SVM does not have the set $\mathcal{L}$, the number of event L reduced from the modified Huber SVM case.



(a) $n = 100$ (b) $n = 400$

Figure 5.5: The sizes of index sets $\mathcal{L}, \mathcal{C}$ and $\mathcal{R}$ in the regularization path for artificial data.

Table 5.4: Computational cost of the squared hinge SVM ($\rho = 1, h = \infty$) for artificial data (sec.)

n	λ-path	λ-path no cutoff	SMO from scratch	SMO direct alpha reuse	SMO scaling all alphas
100	0.18 (0.03)	0.70 (0.22)	185.59 (28.89)	178.05 (28.66)	130.13 (22.48)
200	1.48 (0.20)	4.25 (1.35)	1302.44 (201.31)	1207.30 (186.03)	852.27 (93.04)
400	16.27 (1.37)	28.43 (3.55)	8763.64 (966.75)	7669.12 (824.68)	4962.62 (509.71)

Table 5.5: The number of the events and the mean number of iterations of a rational approximation of the squared hinge SVM ($\rho = 1, h = \infty$) for artificial data.
(The figures in the table are the average (and the standard deviation) of 10 runs)

n	#events L	iteration	iteration (no cutoff)
100	42.10 (8.80)	1.82 (0.21)	14.28 (1.39)
200	83.40 (16.59)	1.40 (0.15)	14.15 (1.74)
400	145.10 (18.58)	1.37 (0.12)	12.92 (1.01)

5.4.2 Real Data

We also apply our algorithm to 6 real world data sets in Table 5.6. These data sets are available from LIBSVM site [25]. In all data sets, each dimension of $\boldsymbol{x}$ is normalized to $[-1, 1]$. We randomly sampled n data points from original data set 10 times (we set n be approximately 80% of the original number of data points).

Table 5.7 and 5.8 show the results of the modified Huber SVM. Table 5.7 shows the CPU time of each algorithm. Our algorithm is much faster than the SMO algorithm in all the data sets. Table 5.8 shows the number of the events L and the mean number of

Table 5.6: Real data sets (The figures in the parentheses are the size of original data set)

	n	d
sonar	166 (208)	60
heart	216 (270)	13
australian	552 (690)	14
diabetes	614 (768)	8
fourclass	689 (862)	2
german	800 (1000)	24

Table 5.7: Computational cost of the modified Huber SVM ($\rho = 1.1, h = 0.2$) for real data (sec.)

	λ-path	SMO	SMO	SMO
		from scratch	direct alpha reuse	scaling all alphas
sonar	1.23 (0.08)	9.24 (0.78)	5.25 (0.35)	6.21 (0.57)
heart	2.62 (0.14)	28.97 (6.69)	16.92 (4.55)	20.38 (4.98)
australian	82.81 (7.13)	4294.12 (638.70)	2371.00 (340.46)	2519.12 (366.11)
diabetes	54.01 (3.03)	26987.53 (2189.93)	15310.13 (1320.48)	14316.77 (1274.24)
fourclass	50.22 (1.18)	5019.78 (458.86)	3607.11 (302.86)	2319.08 (210.31)
german	313.91 (15.62)	4118.85 (264.45)	2159.13 (146.79)	2409.60 (164.76)

Table 5.8: The number of the events and the mean number of iterations of the rational approximation of the modified Huber SVM ($\rho = 1.1, h = 0.2$) for real data sets.

	#events L	iteration
sonar	279.40 (9.55)	1.14 (0.01)
heart	424.20 (7.05)	1.09 (0.01)
australian	1274.90 (30.90)	1.05 (0.00)
diabetes	1298.80 (27.32)	1.08 (0.00)
fourclass	1634.00 (14.21)	1.04 (0.00)
german	1821.80 (25.07)	1.04 (0.01)

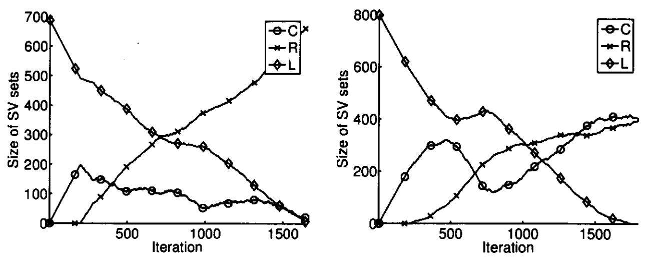
iterations in a rational approximation. We see that the number of the events is about 2-3 times n and the iteration of rational approximation is very small.

Fig. 5.6 shows the size of index sets of fourclass and german data sets. Although the size n of these 2 data sets are not much different, the changing patterns of the set sizes are very different. These differences affect to the computational cost of the regularization path (see Section IV).

Table 5.9 and 5.10 are the results of the squared hinge SVM. Here again, we obtain similar results to the modified Huber SVM case. The results demonstrate efficiency of our algorithm.

5.5 Conclusion

In this chapter, we derive nonlinear regularization path algorithm for the quadratic loss SVMs. Our algorithm efficiently detects the event points using rational approximation. As the advantages of rational approximation, we have shown quadratic convergence



(a) fourclass (b) german

Figure 5.6: The size of index sets $\mathcal{L}, \mathcal{C}$ and $\mathcal{R}$ in the regularization path for real data sets.

rate and cutoff strategy. The experimental results have demonstrated efficiency of our approach compared to naive grid search. Unlike predictor corrector and hot start approaches, our algorithm reveals substantially exact behavior of solutions.

Table 5.9: Computational cost of the squared hinge SVM ($\rho = 1, h = \infty$) for real data (sec.)

	λ-path	SMO from scratch	SMO direct alpha reuse	SMO scaling all alphas
sonar	1.11 (0.03)	3.49 (0.48)	2.69 (0.26)	2.67 (0.29)
heart	2.83 (0.08)	13.75 (2.64)	9.88 (1.78)	10.50 (1.98)
australian	84.61 (2.47)	3498.90 (697.69)	2825.95 (563.69)	2275.22 (395.63)
diabetes	120.69 (4.41)	30884.68 (1550.91)	28118.80 (1344.21)	21276.58 (1041.52)
fourclass	185.34 (6.57)	3403.16 (179.35)	3801.39 (146.05)	1800.37 (132.66)
german	339.70 (9.00)	1383.52 (162.89)	993.90 (138.92)	928.53 (121.83)

Table 5.10: The number of the events and the mean number of iterations of the rational approximation of the squared hinge SVM ($\rho = 1, h = \infty$) for real data sets.

	#events L	iteration
sonar	93.30 (4.52)	1.52 (0.09)
heart	146.60 (6.67)	1.41 (0.05)
australian	468.70 (22.32)	1.18 (0.05)
diabetes	353.70 (20.84)	1.22 (0.04)
fourclass	690.00 (12.90)	1.07 (0.01)
german	504.40 (19.52)	1.23 (0.05)

Chapter 6

Conclusions

In this paper, we have studied parametric optimization approaches for various machine learning tasks. Our approach enables efficient calculation of the exact solution when we have to solve a sequence of parametrized optimization problems.

In Chapter 3, we proposed multiple incremental decremental algorithm for the SVM. This approach is much faster than conventional (single) incremental decremental algorithm when multiple data points are added to and/or removed from training data set. The reason is that our approach can update the changes of multiple data points simultaneously by multi-parametric approach. We have also provided analytical evidences for efficiency of our algorithm on some reasonable assumptions. Experimental results showed that our approach was actually efficient in several practical settings such as online learning and cross-validation.

In Chapter 4, we developed an efficient algorithm for updating solutions of instance-weighted SVMs. Our algorithm was built upon multi-parametric programming techniques, and it is an extension of existing single-parameter path-following algorithms to multiple parameters. We experimentally demonstrated the computational advantage of the proposed algorithm on a wide range of applications including on-line time-series analysis, heteroscedastic data modeling, covariate shift adaptation, ranking, and transduction.

In Chapter 5, we proposed nonlinear regularization path algorithm for a class of learning machines that have quadratic loss and quadratic regularizer which we call quadratic loss SVM. We developed an accurate and efficient nonlinear path following algorithm using rational approximation technique. A unique point of our approach is that it reveals behavior of nonlinear solution path exactly. Experiments showed that our algorithm can efficiently trace the exact regularization path of the quadratic loss SVMs compared to naive exhaustive search.

Although we focused only on quadratic programming (QP) machines in this paper, similar algorithms can be developed for linear programming (LP) machines. It is well-known in the parametric programming literature that the solution of LP and QP have piecewise linear form if a linear function of hyper-parameters appears in the constant term of the constraints and/or the linear part of the objective function (see [12,40,89,92] for more details). Indeed, the parametric LP technique [40] has already been applied to several machine learning problems [73,121,124]. One of our future works is to apply the multi-parametric approach to these LP machines.

In Chapter 4, we studied the changes of instance-weights of various types of SVMs. There are many other situations in which a path of multiple hyper-parameters can be exploited. For instance, the application of the multi-parametric path approach to the following problems would be interesting future works:

- Different (functional) margin SVM:

$$\min_{\boldsymbol{w},b,\{\xi_i\}_{i=1}^n} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^n \xi_i,$$
$$\text{s.t. } y_i f(\boldsymbol{x}_i) \geq \delta_i - \xi_i,\ \xi_i \geq 0,\ i = 1,\dots,n,$$

  where $\delta_i \in \mathbb{R}$ is a margin rescaling parameter. [26] indicated that this type of parametrization can be used to give different costs to each pair of items in ranking SVM.

- SVR with different insensitive-zone thickness. Although usual SVR has the common insensitive-zone thickness ε for all instances, different thickness ε_i for every instance can be assigned:

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_i,\xi_i^*\}_{i=1}^n} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^n (\xi_i + \xi_i^*), \\ \text{s.t.} \quad & y_i - f(\boldsymbol{x}_i) \leq \varepsilon_i + \xi_i, \\ & f(\boldsymbol{x}_i) - y_i \leq \varepsilon_i + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0,\ i = 1,\dots,n. \end{aligned}$$

  In the case of common thickness, the optimal ε is known to be asymptotically proportional to the noise variance [67,103]. In the case of heteroscedastic noise, it would be reasonable to set different ε_i, each of which is proportional to the variance of each (iteratively estimated) y_i.

- The weighted lasso. The lasso solves the following quadratic programming problem [110]:

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \sum_{j=1}^{p} \boldsymbol{x}_j \beta_j\|_2^2 + \lambda \sum_{j=1}^{p} |\beta_j|,$$

  where λ is the regularization parameter. A weighted version of the lasso has been considered in [125]:

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \sum_{j=1}^{p} \boldsymbol{x}_j \beta_j\|_2^2 + \lambda \sum_{j=1}^{p} w_j |\beta_j|,$$

  where w_j is an individual weight parameters for each β_j. The weights are adaptively determined by $w_j = |\hat{\beta}_j|^{-\gamma}$, where $\hat{\beta}_j$ is an initial estimation of β_j and $\gamma > 0$ is a parameter. A similar weighted parameter-minimization problem has also been considered in [21] in the context of signal reconstruction. They considered the following weighted ℓ_1-minimization problem[1] :

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^n} \quad & \sum_{i=1}^{p} w_i |x_i| \\ \text{s.t.} \quad & \boldsymbol{y} = \boldsymbol{\Phi x}. \end{aligned}$$

  where $\boldsymbol{y} \in \mathbb{R}^m$, $\boldsymbol{\Phi} \in \mathbb{R}^{m \times n}$, and $m < n$. The goal of this problem is to reconstruct a sparse signal $\boldsymbol{x}$ from the measurement vector $\boldsymbol{y}$ and sensing matrix $\boldsymbol{\Phi}$. The constraint linear equations have infinitely many solutions and the simplest explanation of $\boldsymbol{y}$ is desirable. To estimate better sparse representation, they proposed an iteratively re-weighting strategy for estimating w_i.

In order to apply the multi-parametric approach to these problems, we need to determine the search direction of the path in the multi-dimensional hyper-parameter space. In many situations search directions can be estimated from data.

Parametric optimization derives analytical expression of solutions. As we have seen throughout the paper, it brings the following advantages:

- Efficiency

  The calculation of analytical solutions are often faster than the iterative application of optimization algorithm. We have shown that several machine learning tasks can be accelerated by this property in Chapter 3, 4 and 5.

[1]Note that $\boldsymbol{x}$ and $\boldsymbol{y}$ in the above equation have different meanings from other parts of this paper.

- Accuracy

  Since we calculate solutions from analytical expression, calculated solutions are highly accurate. In piecewise linear case, it depends on the accuracy of linear system solver. Many linear algebra packages (e.g., LAPACK [3]) provide routines to calculate accurate solution of linear system efficiently. In Chapter 3, we confirmed accuracy of our approach is close to floating-point precision.

- Continuity

  From the function form of the analytical solution, we can know continuous changes of solutions. Using this property, behavior of performance measures for models are also easily monitored. We have shown several examples of prediction performance monitoring in Chapter 4.

As mentioned in this chapter, there seems to be further research directions which we can exploit these advantages. In the future work, we plan to consider applying our approach to those other machine learning tasks.

Acknowledgments

This thesis would not have been a reality without collaboration with several people, the feedback they have provided and their support. I would like to convey my appreciation to all the people who have shared my work and life throughout these years spent in Nagoya Institute of Technology.

I would like to express my deep sense of appreciation to Prof. Masaru Sugiyama at Nagoya Institute of Technology , who is the chief examiner of my thesis, for his guidance, support, and advice to my research. Acknowledgments also go to Prof. Nobuhiro Inuzuka, Prof. Keiichi Tokuda and Prof. Tadashi Wadayama at Nagoya Institute of Technology for their careful review on this thesis.

I am also deeply indebted to Prof. Ryohei Nakano at Chubu university. Without him I would not have started anything of my research. He provided me overall guidance throughout the process of research.

I am deeply grateful to my current supervisor Assoc. Prof. Ichiro Takeuchi. I have benefited not only from his knowledge and great insight as an adviser and a scientist, but also from his kindness and patience. Most of my publications that are an essential part of this thesis have never completed without his help.

While the instance-weighted solution-path project (in Chapter 4), Prof. Masashi Sugiyama at Tokyo Institute of Technology gave me thoughtful and detailed feedback on my paper. I would like to express my gratitude to him for his valuable contribution to this research.

In addition, I would like to give my thanks to all current and past members of Prof. Nakano and Assoc. Prof. Takeuchi lab. for their participation in constructing fruitful discussion and meeting.

Finally, I thank my family and my friends for their long term support and help.

November, 2010

Masayuki Karasuyama

References

[1] A. Albert, *Regression and the Moore-Penrose Pseudoinverse.* New York and London: Academic Press, 1972.

[2] E. L. Allgower and K. Georg, "Continuation and path following," *Acta Numerica*, vol. 2, pp. 1–64, 1993.

[3] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen, *LAPACK Users' guide (third ed.).* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999.

[4] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American Mathematical Society*, vol. 68, pp. 337–404, 1950.

[5] K. Z. Arreola, T. Gärtner, G. Gasso, and S. Canu, "Regularization path for ranking SVM," in *ESANN 2008, 16th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 23-25, 2008, Proceedings*, 2008, pp. 415–420.

[6] A. Asuncion and D. Newman, "UCI machine learning repository," http://www.ics.uci.edu/mlearn/MLRepository.html, 2007.

[7] F. Bach, D. Heckerman, and E. Horvitz, "Considering cost asymmetry in learning classifiers," *Journal of Machine Learning Research*, vol. 7, pp. 1713–1741, 2006.

[8] F. R. Bach, R. Thibaux, and M. I. Jordan, "Computing regularization paths for learning multiple kernels," in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. Cambridge, MA: MIT Press, 2005, pp. 73–80.

[9] P. L. Bartlett and A. Tewari, "Sparseness vs estimating conditional probabilities: Some asymptotic results," *Journal of Machine Learning Research*, vol. 8, pp. 775–790, 2007.

[10] D. Basak, S. Pal, and D. C. Patranabis, "Support vector regression," *Neural Information Processing- Letters and Reviews*, vol. 11, no. 10, pp. 203–224, 2007.

[11] K. P. Bennett and E. J. Bredensteiner, "A parametric optimization method for machine learning," *INFORMS Journal on Computing*, vol. 9, no. 3, pp. 311–318, 1997.

[12] M. J. Best, "An algorithm for the solution of the parametric quadratic programming problem," Faculty of Mathematics, University of Waterloo, Tech. Rep. 82-24, 1982.

[13] S. Bickel, J. Bogojeska, T. Lengauer, and T. Scheffer, "Multi-task learning for HIV therapy screening," in *Proceedings of 25th Annual International Conference on Machine Learning (ICML2008)*, A. McCallum and S. Roweis, Eds., Jul. 5–9 2008, pp. 56–63.

[14] L. Bo, L. Wang, and L. Jiao, "Recursive finite newton algorithm for support vector regression in the primal," *Neural Comput.*, vol. 19, no. 4, pp. 1082–1096, 2007.

[15] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, D. Haussler, Ed. ACM Press, 1992, pp. 144–152.

[16] L. Bottou and C.-J. Lin, "Support vector machine solvers," in *Large Scale Kernel Machines*, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Eds. Cambridge, MA.: MIT Press, 2007, pp. 301–320.

[17] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge, UK: Cambridge University Press, 2004.

[18] J. Bunch, C. Nielsen, and D. Sorensen, "Rank-one modification of the symmetric eigenproblem," *Numerische Mathematik*, vol. 31, no. 1, pp. 31–48, 1979.

[19] W. Burde and B. Blankertz, "Is the locus of control of reinforcement a predictor of brain-computer interface performance?" in *Proceedings of the 3rd International Brain-Computer Interface Workshop and Training Course 2006.* Graz, Austria: Verlag der Technischen Universität, 2006.

[20] C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[21] E. Candes, M. Wakin, and S. Boyd, "Enhancing sparsity by reweighted ℓ_1 minimization," *Journal of Fourier Analysis and Applications*, vol. 14, no. 5-6, pp. 877–905, 2008.

[22] L. Cao and F. Tay, "Support vector machine with adaptive parameters in financial time series forecasting," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1506–1518, 2003.

[23] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon, "Adapting ranking SVM to document retrieval," in *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval.* New York, NY, USA: ACM, 2006, pp. 186–193.

[24] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *Advances in Neural Information Processing Systems*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds., vol. 13. Cambridge, Massachussetts: The MIT Press, 2001, pp. 409–415.

[25] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," 2001, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[26] O. Chapelle and S. Keerthi, "Efficient algorithms for ranking with SVMs," *Information Retrieval*, vol. 13, no. 3, pp. 201–215, 2010.

[27] O. Chapelle, "Training a support vector machine in the primal," *Neural Computation*, vol. 19, no. 5, pp. 1155–1178, 2007.

[28] B.-J. Chen, M.-W. Chang, and C.-J. Lin, "Load forecasting using support vector machines: A study on eunite competition 2001," *IEEE Transactions on Power Systems*, vol. 19, no. 4, pp. 1821–1830, 2004.

[29] W.-H. Chen, J.-Y. Shih, and S. Wu, "Comparison of support-vector machines and back propagation neural networks in forecasting the six major Asian stock markets," *Int. J. Electron. Financ.*, vol. 1, no. 1, pp. 49–67, 2006.

[30] W. Chu, S. S. Keerthi, and C. J. Ong, "Bayesian support vector regression using a unified loss function," *IEEE Transaction on Neural Networks*, vol. 15, no. 1, pp. 29–44, 2004.

[31] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[32] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and other kernel-based learning methods.*. Cambridge University Press, 2000.

[33] M. Davy, F. Desobry, A. Gretton, and C. Doncarli, "An online support vector machine for abnormal events detection," *Signal Processing*, vol. 86, no. 8, pp. 2009–2025, 2006.

[34] D. DeCoste and K. Wagstaff, "Alpha seeding for support vector machines," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2000, pp. 345–359.

[35] G. Dornhege, J. Millán, T. Hinterberger, D. McFarland, and K.-R. Müller, Eds., *Toward Brain Computer Interfacing.* Cambridge, MA, USA: MIT Press, 2007.

[36] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," Stanford University, Tech. Rep., 2002.

[37] T. Evgeniou, M. Pontil, and T. Poggio, "Regularization networks and support vector machines," *Advances in Computational Mathematics*, vol. 13, no. 1, pp. 1–50, 2000.

[38] S. Fine and K. Scheinberg, "Incremental learning and selective sampling via parametric optimization framework for svm," in *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. Cambridge, MA: MIT Press, 2002.

[39] ——, "Efficient SVM training using low-rank kernel representation," *Journal of Machine Learning Research*, vol. 2, pp. 243–264, 2001.

[40] T. Gal, *Postoptimal Analysis, Parametric Programming, and Related Topics.* Walter de Gruyter, 1995.

[41] T. Gal and J. Nedoma, "Multiparametric Linear Programming," *MANAGEMENT SCIENCE*, vol. 18, no. 7, pp. 406–422, 1972.

[42] P. Garrigues and L. E. Ghaoui, "An homotopy algorithm for the lasso with online observations," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., 2009, pp. 489–496.

[43] B. Gärtner, J. Giesen, M. Jaggi, and T. Welsch, "A combinatorial algorithm to compute regularization paths," *arXiv cs.LG.*, 2009.

[44] G. H. Golub, "Some modified eigenvalue problems," Stanford University, Stanford, CA, USA, Tech. Rep., 1971.

[45] G. H. Golub and C. F. Van Loan, *Matrix computations.* Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[46] L. Gunter and J. Zhu, "Efficient computation and model selection for the support vector regression," *Neural Computation*, vol. 19, no. 6, pp. 1633–1655, 2007.

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

[48] R. Herbrich, T. Graepel, and K. Obermayer, "Large margin rank boundaries for ordinal regression," in *Advances in Large Margin Classifiers*, A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, Eds. Cambridge, MA: MIT Press, 2000, pp. 115–132.

[49] R. Hyndman, "Time series data library," (n.d.), http://www.robhyndman.info/TSDL.

[50] K. Järvelin and J. Kekäläinen, "IR evaluation methods for retrieving highly relevant documents," in *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval.* New York, NY, USA: ACM, 2000, pp. 41–48.

[51] J. Jiang and C. Zhai, "Instance weighting for domain adaptation in NLP," in *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, 2007, pp. 264–271.

[52] T. Joachims, "Transductive inference for text classification using support vector machines," in *Proceedings of the 16th Annual International Conference on Machine Learning (ICML 1999).* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 200–209.

[53] T. Kanamori, S. Hido, and M. Sugiyama, "A least-squares approach to direct importance estimation," *Journal of Machine Learning Research*, vol. 10, pp. 1391–1445, Jul. 2009.

[54] M. Karasuyama and I. Takeuchi, "Multiple incremental decremental learning of support vector machines," *IEEE Transactions on Neural Networks*, vol. 21, no. 7, pp. 1048–1059, 2010.

[55] ——, "Nonlinear regularization path for the modified huber loss support vector machines," in *International Joint Conference on Neural Networks*, 2010, pp. 3099–3106.

[56] M. Karasuyama, N. Harada, M. Sugiyama, and I. Takeuchi, "Multi-parametric solution-path algorithm for instance-weighted support vector machines," 2010, arXiv:1009.4791.

[57] M. Karasuyama and I. Takeuchi, "Multiple incremental decremental learning of support vector machines," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds., 2009, pp. 907–915.

[58] M. Karasuyama, I. Takeuchi, and R. Nakano, "Efficient leave-m-out cross-validation of support vector regression by generalizing decremental algorithm," *New Generation Computing*, vol. 27, pp. 307–318, 2009.

[59] H. Kashima, T. Idé, T. Kato, and M. Sugiyama, "Recent advances and trends in large-scale kernel methods," *IEICE Transactions*, vol. 92-D, no. 7, pp. 1338–1353, 2009.

[60] S. S. Keerthi and D. DeCoste, "A modified finite newton method for fast solution of large scale linear SVMs," *Journal of Machine Learning Research*, vol. 6, pp. 341–361, 2005.

[61] S. Keerthi and S. Shevade, "A fast tracking algorithm for generalized LARS/LASSO," *IEEE Transactions on Neural Networks*, vol. 18, no. 6, pp. 1826–1830, 2007.

[62] S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy, "Improvements to platt's smo algorithm for svm classifier design," *Neural Computation*, vol. 13, no. 3, pp. 637–649, 2001.

[63] K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard, "Most likely heteroscedastic Gaussian process regression," in *Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007)*, Z. Ghahramani, Ed. Omnipress, 2007, pp. 393–400.

[64] K.-J. Kim, "Financial time series forecasting using support vector machines," *Neurocomputing*, vol. 55, no. 1-2, pp. 307–319, 2003.

[65] K. Koh, S.-J. Kim, and S. Boyd, "An interior-point method for large-scale ℓ_1-regularized logistic regression," *Journal of Machine Learning Research*, vol. 8, pp. 1519–1555, 2007.

[66] V. Krishnamurthy, S. D. Ahipasaoglu, and A. d'Aspremont, "A pathwise algorithm for covariance selection," in *NIPS 2009 Workshop on Optimization for Machine Learning*, 2009.

[67] J. Kwok and I. Tsang, "Linear dependency between ε and the input noise in ε-support vector regression," *IEEE Transactions on Neural Networks*, vol. 14, no. 3, pp. 544–553, 2003.

[68] P. Laskov, C. Gehl, S. Kruger, and K.-R. Muller, "Incremental support vector learning: Analysis, implementation and applications," *Journal of Machine Learning Research*, vol. 7, pp. 1909–1936, 2006.

[69] G. Lee and C. Scott, "The one class support vector machine solution path," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2, 2007, pp. II521–II524.

[70] M. M. Lee, S. S. Keerthi, C. J. Ong, and D. DeCoste, "An efficient method for computing leave-one-out error in support vector machines," *IEEE transaction on neural networks*, vol. 15, no. 3, pp. 750–757, 2004.

[71] Y. Lee and Z. Cui, "Characterizing the solution path of multicategory support vector machines," *Statistica Sinica*, vol. 16, no. 2, pp. 391–409, 2006.

[72] Y. Li, Y. Liu, and J. Zhu, "Quantile regression in reproducing kernel hilbert spaces," *Journal of the American Statistical Association*, vol. 102, no. 477, pp. 255–268, 2007.

[73] Y. Li and J. Zhu, "l_1-norm quantile regression," *Journal of Computational and Graphical Statistics*, vol. 17, no. 1, pp. 163–185, 2008.

[74] C.-F. Lin and S.-D. Wang, "Fuzzy support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 464–471, 2002.

[75] Y. Lin, Y. Lee, and G. Wahba, "Support vector machines for classification in nonstandard situations," *Machine Learning*, vol. 46, no. 1/3, pp. 191–202, 2002.

[76] T. Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li, "LETOR: Benchmark dataset for research on learning to rank for information retrieval," in *SIGIR '07: Proceedings of the Learning to Rank workshop in the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 2007.

[77] T.-Y. Liu, "Learning to rank for information retrieval," *Foundations and Trends in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009.

[78] G. Loosli, G. Gasso, and S. Canu, "Regularization paths for ν-SVM and ν-SVR," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4493, no. 3, pp. 486–496, 2007.

[79] J. Ma and J. Theiler, "Accurate online support vector regression," *Neural Computation*, vol. 15, no. 11, pp. 2683–2703, 2003.

[80] M. Martin, "On-line support vector machines for function approximation," Software Department, University Politecnica de Catalunya, Tech. Rep., 2002.

[81] D. Mattera and S. Haykin, "Support vector machines for dynamic reconstruction of a chaotic system," in *Advances in kernel methods: support vector learning*, B. Scholkopf, C. J. Burges, and A. J. Smola, Eds. Cambridge, MA, USA: MIT Press, 1999, pp. 211–241.

[82] M. Meyer, "Statlib," http://lib.stat.cmu.edu/index.php.

[83] K.-R. Müller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, "An introduction to kernel-based learning algorithms," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 181–201, 2001.

[84] K.-R. Müller, A. J. Smola, G. Rätsch, B. Schökopf, J. Kohlmorgen, and V. Vapnik, "Using support vector machines for time series prediction," in *Advances in kernel methods: support vector learning.* Cambridge, MA, USA: MIT Press, 1999, pp. 243–253.

[85] N. Murata, M. Kawanabe, A. Ziehe, K.-R. Müller, and S. Amari, "On-line learning in changing environments with applications in supervised and unsupervised learning," *Neural Networks*, vol. 15, no. 4-6, pp. 743–760, 2002.

[86] G. Obozinski, B. Taskar, and M. Jordan, "Joint covariate selection and joint subspace selection for multiple classification problems," *Statistics and Computing*, vol. 20, no. 2, pp. 231–252, 2010.

[87] C.-J. Ong, S. Shao, and J. Yang, "An improved algorithm for the solution of the regularization path of support vector machine," *IEEE Transactions on Neural Networks*, vol. 21, no. 3, pp. 451–462, 2010.

[88] M. Park and T. Hastie, "L1-regularization path algorithm for generalized linear models," *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, vol. 69, no. 4, pp. 659–677, 2007.

[89] E. N. Pistikopoulos, M. C. Georgiadis, and V. Dua, *Process Systems Engineering: Volume 1: Multi-Parametric Programming.* WILEY-VCH, 2007.

[90] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods — Support Vector Learning*, B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA: MIT Press, 1999, pp. 185–208.

[91] R. Raina, A. Battle, H. Lee, B. Packer, and A. Ng, "Self-taught learning: transfer learning from unlabeled data," in *Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007)*, Z. Ghahramani, Ed. Omnipress, 2007, pp. 759–766.

[92] K. Ritter, "On parametric linear and quadratic programming problems," in *Mathematical Programming: Proceedings of the International Congress on Mathematical Programming*, R. Cottle, M. L. Kelmanson, and B. Korte, Eds. Elsevier Science Publisher B.V., 1984, pp. 307–335.

[93] S. Rosset, "Bi-level path following for cross validated solution of kernel quantile regression," *Journal of Machine Learning Research*, vol. 10, pp. 2473–2505, 2009.

[94] ——, "Following curved regularized optimization solution paths," in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. Cambridge, MA: MIT Press, 2005, pp. 1153–1160.

[95] S. Rosset and J. Zhu, "Piecewise linear regularized solution paths," *Annals of Statistics*, vol. 35, pp. 1012–1030, 2007.

[96] K. Scheinberg, "An efficient implementation of an active set method for svms," *Journal of Machine Learning Research*, vol. 7, pp. 2237–2257, 2006.

[97] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning)*. The MIT Press, 2001.

[98] J. R. Schott, *Matrix Analysis For Statistics.* Wiley-Interscience, 2005.

[99] M. Seeger, "Low rank updates for the cholesky decomposition," University of California at Berkeley, Tech. Rep., 2004.

[100] A. Shilton, M. Palaniswami, D. Ralph, and A. Tsoi, "Incremental training of support vector machines," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 114–131, 2005.

[101] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of Statistical Planning and Inference*, vol. 90, no. 2, pp. 227–244, 2000.

[102] K. Sjostrand, M. Hansen, H. Larsson, and R. Larsen, "A path algorithm for the support vector domain description and its application to medical imaging," *Medical Image Analysis*, vol. 11, no. 5, pp. 417–428, 2007.

[103] A. J. Smola, N. Murata, B. Schölkopf, and K.-R. Müller, "Asymptotically optimal choice of ε-loss for support vector machines," in *Proceedings of the International Conference on Artificial Neural Networks*, ser. Perspectives in Neural Computing, L. Niklasson, M. Bodén, and T. Ziemke, Eds. Berlin: Springer, 1998, pp. 105–110.

[104] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 36, no. 2, pp. 111–147, 1974.

[105] M. Sugiyama, M. Krauledat, and K.-R. Müller, "Covariate shift adaptation by importance weighted cross validation," *Journal of Machine Learning Research*, vol. 8, pp. 985–1005, May 2007.

[106] I. Takeuchi, Q. V. Le, T. D. Sears, and A. J. Smola, "Nonparametric quantile estimation," *Journal of Machine Learning Research*, vol. 7, pp. 1231–1264, 2006.

[107] I. Takeuchi, K. Nomura, and T. Kanamori, “Nonparametric conditional density estimation using piecewise-linear solution path of kernel quantile regression,” *Neural Comput.*, vol. 21, no. 2, pp. 533–559, 2009.

[108] F. Tay and L. Cao, “Application of support vector machines in financial time series forecasting,” *Omega*, vol. 29, no. 4, pp. 309–317, 2001.

[109] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society (Series B)*, vol. 58, pp. 267–288, 1996.

[110] ——, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society, Series B*, vol. 58, no. 1, pp. 267–288, 1996.

[111] K. Tsuda, “Entire regularization paths for graph data,” in *Proceedings of 24th Annual International Conference on Machine Learning (ICML2007)*, 2007, pp. 919–926.

[112] V. N. Vapnik, *The Nature of Statistical Learning Theory.* Berlin, Germany: Springer-Verlag, 1995.

[113] V. Vapnik, S. E. Golowich, and A. Smola, “Support vector method for function approximation, regression estimation, and signal processing,” in *Advances in Neural Information Processing Systems 9*, M. Mozer, M. Jordan, and T. Petsche, Eds. MIT Press, 1996, pp. 281–287.

[114] S. Vishwanathan, A. Smola, and M. Murty, “SimpleSVM,” in *Proceedings of Twentieth International Conference on Machine Learning (ICML 2003), Washington, DC, USA.* AAAI Press, 2003, pp. 760–767.

[115] M. Vogt, “SMO algorithms for support vector machines without bias term,” Technische Universität Darmstadt, Tech. Rep., 2002.

[116] G. Wang, D.-Y. Yeung, and F. H. Lochovsky, “A kernel path algorithm for support vector machines,” in *Twenty-fourth International Conference on Machine Learning*, 2007, pp. 951–958.

[117] ——, “A new solution path algorithm in support vector regression,” *IEEE Transactions on Neural Networks*, vol. 19, no. 10, pp. 1753–1767, 2008.

[118] Z. Wu, A. Zhang, and C. Li, “Trace solution paths for SVMs via parametric quadratic programming,” in *KDD Workshop Report DMMT’08: data mining using matrices and tensors*, 2008.

[119] J. Xu, Y. Cao, H. Li, and Y. Huang, “Cost-sensitive learning of SVM for ranking,” in *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., vol. 4212. Springer, 2006, pp. 833–840.

[120] X. Yang, Q. Song, and Y. Wang, “A weighted support vector machine for data classification,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 21, no. 5, pp. 961–976, 2007.

[121] Y. Yao and Y. Lee, “Another look at linear programming for feature selection via methods of regularization,” Department of Statistics, The Ohio State University, Tech. Rep. 800, 2007.

[122] M. Yuan and H. Zou, "Efficient global approximation of generalized nonlinear ℓ_1-regularized solution paths and its applications," *Journal of the American Statistical Association*, vol. 104, no. 488, pp. 1562–1574, 2009.

[123] T. Zhang, "Statistical behavior and consistency of classification methods based on convex risk minimization," *Annals of Statistics*, vol. 32, no. 1, pp. 56–134, 2004.

[124] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani, "1-norm support vector machines," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. Cambridge, MA: MIT Press, 2004.

[125] H. Zou, "The adaptive lasso and its oracle properties," *Journal of the American Statistical Association*, vol. 101, no. 476, pp. 1418–1429, 2006.

Publications

Journal papers (Refereed)

[1] **M. Karasuyama** and I. Takeuchi, Multiple Incremental Decremental Learning of Support Vector Machines, *IEEE Transactions on Neural Networks*, vol. 21, no. 7, pp. 1048-1059. 2010.

[2] **M. Karasuyama**, I. Takeuchi and R. Nakano, Efficient Leave-m-out Cross-Validation of Support Vector Regression by Generalizing Decremental Algorithm, *New Generation Computing*, vol. 27, no. 4, Special Issue on Data-Mining and Statistical Science, pp.307-318, 2009.

[3] H. Moriguchi, I. Takeuchi, **M. Karasuyama**, S. Horikawa, Y. Ohta, T. Kodama, and H. Naruse, Adaptive kernel quantile regression for anomaly detection, *Journal of Advanced Computational Intelligence and Inteligent Informatics*, vol.13, no.3, pp.230-236, 2009.

International Conference Proceedings (Refereed)

[4] **M. Karasuyama**, and I. Takeuchi, Nonlinear Regularization Path for the Modified Huber loss Support Vector Machines, *In International Joint Conference on Neural Networks*, pp. 3099-3106, 2010.

[5] **M. Karasuyama** and I. Takeuchi, Multiple Incremental Decremental Learning of Support Vector Machines. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta eds., *Advances in Neural Information Processing Systems 22*, pp. 907-915. 2009.

[6] **M. Karasuyama**, I. Takeuchi and R. Nakano, Reducing SVR Support Vectors by Using Backward Deletion, *In Proceedings of the 12th international Conference on Knowledge-Based intelligent information and Engineering Systems*, Part III, Lecture Notes In Artificial Intelligence, vol. 5179. Springer-Verlag, Berlin, Heidelberg, pp. 76-83, 2008.

[7] **M. Karasuyama**, and R. Nakano, Optimizing Sparse Kernel Ridge Regression Hyperparameters Based on Leave-one-out Cross-validation, *In International Joint Conference on Neural Networks*, pp. 3463-3468, 2008.

[8] **M. Karasuyama**, and R. Nakano, Optimizing SVR Hyperparameters via Fast Cross-Validation using AOSVR, *In International Joint Conference on Neural Networks*, pp. 1186-1191, 2007.

[9] **M. Karasuyama**, D. Kitakoshi and R. Nakano, Revised Optimizer of SVR Hyperparameters Minimizing Cross-Validation Error, *In International Joint Conference on Neural Networks*, pp. 711-718, 2006.

Domestic Conference Proceedings (Non-refereed)

[10] **M. Karasuyama**, I. Takeuchi, A Study on the Suboptimal Solution Path Algorithm, IEICE technical report, IBISML, vol. 110, no. 265, IBISML2010-89, pp. 221-230, 2010-11 (in Japanese).

[11] T. Sugiura, K. Koide, T. Hongo, **M. Karasuyama**, I. Takeuchi, A Study on Simultaneous Feature Selection for Cost-Sensitive Classifiers Using Mixed-Norm Regularization, IEICE technical report, IBISML, vol. 110, no. 265, IBISML2010-70, pp. 83-90, 2010-11 (in Japanese).

[12] Y. Ichikawa, K. Isobe, **M. Karasuyama**, S. Izumi, I. Takeuchi, A Study on Multiple Multivariate Two-Sample Test for Gene Set Analysis using MST-based SVM Path-Following, IEICE technical report, IBISML, vol. 110, no. 265, IBISML2010-88, pp. 211-220, 2010-11 (in Japanese).

[13] **M. Karasuyama**, I. Takeuchi, A Study on the Exact Nonlinear Regularization Path for L2 Loss Support Vector Machines, IEICE technical report, IBISML, vol. 110, no. 76, IBISML2010-6, pp. 23-31, 2010-11 (in Japanese).

[14] N. Harada, T. Hasegawa, **M. Karasuyama**, I. Takeuchi, A Study on Ranking Model Optimization by Following Discrete Changes of an Evaluation Value IEICE technical report, Neurocomputing, vol. 109, no. 461, NC2009-166, pp. 461-466, 2010-3 (in Japanese).

[15] **M. Karasuyama**, I. Takeuchi, A Study on Regularization Path for the Support Vector Machine with a Huber Type Loss Function, *In Proceedings of Workshop on Informatics 2009* (WiNF2009), pp. 41-46, 2009-11 (in Japanese).

[16] **M. Karasuyama**, N. Harada, I. Takeuchi, A Study on Multi-dimensional Path Following for Weighted Kernel Machines, Technical Report on Information-Based Induction Sciences 2009 (IBIS2009), pp. 198-205, 2009-10 (in Japanese).

[17] N. Harada, **M. Karasuyama**, I. Takeuchi, A Study on an On-line Learning of Weighted Support Vector Machine, Tokai-Section Joint Conference on Electrical and Related Engineering, O-030, 2009-9 (in Japanese).

[18] T. Otubo, **M. Karasuyama**, I. Takeuchi, A Study on Forgetting Factor Path-following in Support Vector Regression, Tokai-Section Joint Conference on Electrical and Related Engineering, O-028, 2009-9 (in Japanese).

[19] A Study on Generalization of Decremental Algorithm for Support Vector Regression, **M. Karasuyama**, I. Takeuchi and R. Nakano, SIG-DMSM-A803-13, pp. 91-96, 2009-3 (in Japanese).

[20] **M. Karasuyama**, I. Takeuchi, R. Nakano, Gradient Based Two Dimensional Path Following for Kernel Machines, IEICE technical report, Neurocomputing, vol. 108, no. 372, NC2008-80, pp. 43-48, 2008-12 (in Japanese).

[21] **M. Karasuyama**, I. Takeuchi, R. Nakano, Optimizing SVR Hyperparameters via Fast Cross-Validation using AOSVR, IEICE technical report, Neurocomputing, vol. 107, no. 410, NC2007-73, pp. 13-18, 2007-12 (in Japanese).

Appendix

A Solving KKT Linear System

Here, we consider solving the following form of linear system:

$$\begin{bmatrix} 0 & \boldsymbol{y}^\top \\ \boldsymbol{y} & \boldsymbol{Q} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} r_1 \\ \boldsymbol{r}_2 \end{bmatrix},$$

where $\boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}$, $\boldsymbol{\alpha} \in \mathbb{R}^n$, $r_1 \in \mathbb{R}$. $\boldsymbol{r}_2 \in \mathbb{R}^n$ and $\boldsymbol{Q}$ is strictly positive definite. We can decompose above linear system as follows:

$$\boldsymbol{y}^\top \boldsymbol{\alpha} = r_1, \tag{A.1}$$

$$\boldsymbol{y}b + \boldsymbol{Q}\boldsymbol{\alpha} = \boldsymbol{r}_2. \tag{A.2}$$

From (A.2) we obtain

$$\boldsymbol{\alpha} = \boldsymbol{Q}^{-1}(\boldsymbol{r}_2 - \boldsymbol{y}b).$$

Substituting this equation into (A.1), we find

$$\begin{aligned} \boldsymbol{y}^\top \boldsymbol{Q}^{-1}(\boldsymbol{r}_2 - \boldsymbol{y}b) &= r_1 \\ b &= \frac{\boldsymbol{y}^\top \boldsymbol{Q}^{-1}\boldsymbol{r}_2 - r1}{\boldsymbol{y}^\top \boldsymbol{Q}^{-1}\boldsymbol{y}}. \end{aligned}$$

Consider the Cholesky factorization $\boldsymbol{Q} = \boldsymbol{L}\boldsymbol{L}^\top$ where $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ is a lower triangular matrix with positive diagonal entries. Let $\boldsymbol{\rho}_1 = \boldsymbol{L}^{-1}\boldsymbol{y}$ and $\boldsymbol{\rho}_2 = \boldsymbol{L}^{-1}\boldsymbol{r}_2$. Then the solution is represented as follows:

$$\begin{aligned} b &= \frac{\boldsymbol{\rho}_1^\top \boldsymbol{\rho}_2 - r_1}{\boldsymbol{\rho}_1^\top \boldsymbol{\rho}_1}, \\ \boldsymbol{\alpha} &= \boldsymbol{L}^{-\top}(\boldsymbol{\rho}_2 - \boldsymbol{\rho}_1 b). \end{aligned}$$

B Rank-one Cholesky Update

Since the size of $\mathcal{M}$ increases or decreases by one in each iteration, we use rank-one update of Cholesky factor $\boldsymbol{L}$ of $\boldsymbol{Q}_{\mathcal{M}}$. This requires $O(|\mathcal{M}|^2)$ computations instead of $O(|\mathcal{M}|^3)$

computations in usual Cholesky factorization [45]. If $\boldsymbol{Q}_{\mathcal{M}}$ is expanded as

$$\begin{bmatrix} \boldsymbol{Q}_{\mathcal{M}} & \boldsymbol{q}_c \\ \boldsymbol{q}_c^\top & Q_{cc} \end{bmatrix},$$

Cholesky factor of expanded matrix becomes

$$\begin{bmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{q}_c^\top \boldsymbol{L}^{-\top} & \sqrt{Q_{cc} - \boldsymbol{q}_c^\top \boldsymbol{L}^{-\top} \boldsymbol{L}^{-1} \boldsymbol{q}_c} \end{bmatrix}.$$

When we delete k-th row and k-th column of $\boldsymbol{Q}_{\mathcal{M}}$, we delete corresponding row of $\boldsymbol{L}$. However, then since the entries of $(j, j+1)$ for $j \geq k$ have non zero elements, $\boldsymbol{L}$ is no longer lower triangular matrix. We can eliminate these non zeros using Givens rotations with:

$$\begin{array}{c} \\ \\ \\ j \\ j+1 \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} & & j & j+1 & & \end{array} \\ \begin{pmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \cdots & c & -s & \cdots & 0 \\ 0 & \cdots & s & c & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 \end{pmatrix} \end{array},$$

where

$$c = \frac{L_{jj}}{\sqrt{L_{jj}^2 + L_{jj+1}^2}}, \qquad s = \frac{L_{jj+1}}{\sqrt{L_{jj}^2 + L_{jj+1}^2}}.$$

Multiplying this sparse orthogonal matrix (with $O(|\mathcal{M}|)$) from right side of $\boldsymbol{L}$, $(j+1) \times (j+1)$ leading principle submatrix of $\boldsymbol{L}$ becomes lower triangular. We iterate this operation until entire matrix becomes lower triangular.

C Convergence of Rational Approximation

We provide convergence proof of the rational approximation. We can prove it in almost same way as [18]. Although we will consider the case of solving (5.22) (i.e., type 3 in Table 5.1), other types can also be proved in similar way. Here, we employ simple notations such as $d = d_i, \psi_k = \psi_\omega(t_k), \psi^* = \psi_\omega(t^*), \psi' = \partial\psi_\omega(t)/\partial t$. In the proof, we assume that the following condition holds:

Assumption 1 $\psi^{*'} + \rho - \varphi^{*'} \neq 0$. Since $\psi^{*'} + \rho - \varphi^{*'} \leq 0$, Assumption 1 just means that $\psi^{*'} + \rho - \varphi^{*'} < 0$.

The following two theorems provide the convergence property of our algorithm.

Theorem 2 *Under Assumption 1, the sequence of the rational approximation $\{t_k\}_{k=1,2,\cdots}$ converges to t^* as $k \to \infty$. Even if the Assumption 1 does not hold, $\{t_k\}$ can reach $t^k \in [t^* - \varepsilon, t^*)$ by the finite number of iterations for arbitrary small $\varepsilon > 0$.*

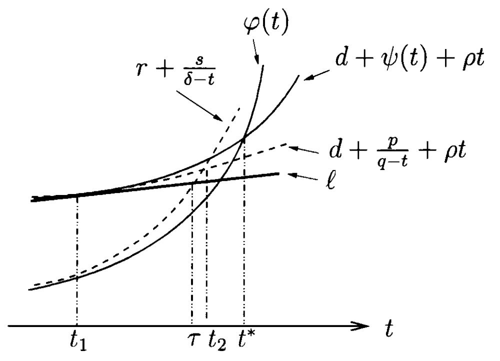


Figure C.1: The schematic illustration of the rational approximation. ℓ is a tangent line of $d+\psi(t)+\rho t$. From the convexity, ℓ becomes lower bound of $d+\psi(t)+\rho t$ and $d+\frac{p}{q-t}+\rho t$.

Theorem 3 *Under Assumption 1, if the sequence $\{t_k\}$ converges to t^*, the rational approximation has the quadratic rate of convergence for sufficiently large k.*

Proof of Theorem 2

Proof. Let $\beta \in (0,1)$ be a constant which is independent of the iteration k. We prove the following condition holds for any $t_1 \in (t_0, t^*)$:

$$t^* - t_2 \leq (1-\beta)(t^* - t_1),$$

where t_2 is obtained by one iteration of the rational approximation from t_1. Let τ satisfy

$$d + \psi_1 + \rho t_1 + (\psi_1' + \rho)(\tau - t_1) = r + \frac{s}{\delta - \tau}.$$

The left hand side represents the tangent line ℓ of $d+\psi(t)+\rho t$ at t_1 (see Fig. C.1). Let us define α as the angle between the line ℓ and the horizontal line. Then we see

$$\tan\alpha = \psi_1' + \rho = \frac{r + \frac{s}{\delta-\tau} - (d + \psi_1 + \rho t_1)}{\tau - t_1}.$$

From (5.20), we set $r = \varphi_1 - (\delta - t_1)\varphi_1'$ and $s = (\delta - t_1)^2\varphi_1'$. Substituting r and s into above equation, we obtain

$$\tau - t_1 = -\frac{d + \psi_1 + \rho t_1 - \varphi_1}{\varphi_1' + \rho} + \frac{\varphi_1'}{\psi_1' + \rho}\frac{\delta - t_1}{\delta - \tau}(\tau - t_1).$$

Arranging this equation, we have

$$\tau - t_1 = \frac{-g_1}{\gamma\varphi_1' - \psi_1' - \rho}(t^* - t_1) \tag{C.1}$$

where $\gamma = (\delta - t_1)/(\delta - \tau)$ and

$$g_1 = -\frac{d + \psi_1 + \rho t_1 - \varphi_1}{t^* - t_1}.$$

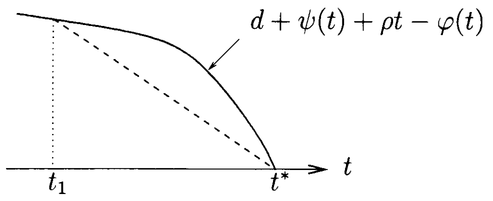


Figure C.2: The illustration of g_1
The slope of the dashed line is g_1.

g_1 can be interpreted as the slope of a line which passes through $(t_1, d + \psi_1 + \rho t_1 - \varphi_1)$ and $(t^*, 0)$ (see Fig. C.2). From the Assumption 1, g_1 has an upper bound which is smaller than 0, i.e.,

$$g_1 \le g_{\max} < 0.$$

If the Assumption 1 does not hold, we can maintain this inequality when t_1 is in $(t_0, t^* - \varepsilon]$ for arbitrary small $\varepsilon > 0$. Substituting this into (C.1), we obtain the following inequalities:

$$\tau - t_1 \ge \frac{-g_{\max}}{\gamma\varphi_1' - \psi_1' - \rho}(t^* - t_1) \ge \beta(t^* - t_1),$$

where

$$\beta = \frac{-g_{\max}}{\max_{t\in(t_0,t^*)}(\gamma\varphi'(t) - \psi'(t) - \rho)}.$$

Since $t_2 \ge \tau$, we have

$$\begin{aligned} t_2 - t_1 &\ge \beta(t^* - t_1) \\ -t^* + t_2 &\ge -(t^* - t_1) + \beta(t^* - t_1) \\ t^* - t_2 &\le (1 - \beta)(t^* - t_1). \end{aligned}$$

Finally, we need to prove $\beta \in (0, 1)$. First, we consider $\beta < 1$. It can be derived from the following inequalities:

$$\begin{aligned} \max_{t\in(t_0,t^*)} \left(\gamma\varphi'(t) - \psi'(t) - \rho\right) &> \max_{t\in(t_0,t^*)} \left(\varphi'(t) - \psi'(t) - \rho\right) \\ &\ge -g_{\max}. \end{aligned}$$

The first inequality comes from $\gamma > 1$ and $\psi'(t) \ge 0$. From the mean-value theorem, there exists at least one $\theta \in (t_0, t^*)$ such that $\psi'(\theta) + \rho - \varphi'(\theta) = g_{\max}$. Then the second inequality holds. Next, we consider the lower bound of β. From the monotonicity of ψ' and φ', we obtain

$$\max_{t_1\in(t_0,t^*)} \left(\gamma\varphi_1' - \psi_1' - \rho\right) < \frac{\delta - t_0}{\delta - t^*}\varphi^{*\prime} - \psi_0' - \rho.$$

Then we see β is in $(0, 1)$. □

Proof of Theorem 3

Proof. Let κ be a constant which is independent on the iteration. We show $|t_{k+1} - t^*| \leq \kappa |t_k - t^*|^2$ for sufficiently large k, when $t_k \to t^*$. Subtracting

$$d + \psi^* + \rho t^* = \varphi^*,$$

from

$$d + \frac{p}{q - t_2} + \rho t_2 = r + \frac{s}{\delta - t_2},$$

we obtain

$$\frac{p}{q - t_2} - \psi^* + \rho(t_2 - t^*) = r + \frac{s}{\delta - t_2} - \varphi^*.$$

Substituting p, q, r and s, this equation can be reduced to

$$\begin{aligned} &\psi_1 - \psi^* G(t_1) + \rho(t_2 - t^*) G(t_1) = \\ &\qquad \left\{ \varphi_1 - \varphi^* + \Delta \varphi_1' \left(\frac{t_2 - t_1}{\delta - t_2} \right) \right\} G(t_1), \end{aligned} \tag{C.2}$$

where $G(t_1) = 1 + \frac{\psi_1'}{\psi_1}(t_1 - t_2)$ and $\Delta = \delta - t_1$. The left hand side of (C.2) can be written as

$$\begin{aligned} &\psi_1 - \psi^* G(t_1) + \rho(t_2 - t^*) G(t_1) = \\ &\qquad \frac{1}{\psi_1}(\psi_1^2 - \psi^* \psi_1 - \psi^* \psi_1' \varepsilon_1) + \frac{\psi^* \psi_1'}{\psi_1} \varepsilon_2 + \rho \varepsilon_2 G(t_1), \end{aligned}$$

where $\varepsilon_1 = t_1 - t^*$ and $\varepsilon_2 = t_2 - t^*$. Using the Taylor expansion of ψ_1 and ψ_1' around t^*, we obtain

$$\psi_1^2 - \psi^* \psi_1 - \psi^* \psi_1' \varepsilon_1 = \left\{ (\psi^{*'})^2 - \frac{1}{2} \psi^* \psi^{*''} \right\} \varepsilon_1^2 + O(\varepsilon_1^3).$$

Finally the left hand side of (C.2) becomes

$$\frac{\psi^* \psi_1'}{\psi_1} \varepsilon_2 + \rho \varepsilon_2 G(t_1) + O(\varepsilon_1^2). \tag{C.3}$$

Expanding φ_1 and φ_1' in the right hand side of (C.2), we obtain

$$\varphi^{*'} \varepsilon_2 \left(\frac{\Delta - \varepsilon_1}{\delta - t_2} \right) G(t_1) + O(\varepsilon_1^2). \tag{C.4}$$

Using (C.3) and (C.4), (C.2) is reduced to

$$\left\{ \frac{\psi^* \psi_1'}{\psi_1} + \rho G(t_1) - \varphi_1' \left(\frac{\Delta - \varepsilon_1}{\delta - t_2} \right)' G(t_1) \right\} \varepsilon_2 = O(\varepsilon_1^2).$$

Since $G(t_1) \to 1$ as $t_1 \to t^*$,

$$\begin{aligned} &\lim_{t_1 \to t^*} \left\{ \frac{\psi^* \psi_1'}{\psi_1} + \rho G(t_1) - \varphi_1' \left(\frac{\Delta - \varepsilon_1}{\delta - t_2} \right) G(t_1) \right\} = \\ &\qquad \psi^{*'} + \rho - \varphi^{*'} \neq 0 \end{aligned}$$

Then we can see $\varepsilon_2 = O(\varepsilon_1^2)$. □