University of California

Los Angeles

Kernel Optimization and Distributed Learning Algorithms for Support Vector Machines

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical Engineering

by

Yumao Lu

2005

The dissertation of Yumao Lu is approved.

Chiara Sabatti

Stephen Jacobsen

Lieven Vandenberghe, Committee Co-chair

Vwani Roychowdhury, Committee Co-chair

University of California, Los Angeles

2005

This is dedicated

to Shanshan

and our upcoming baby.

Table of Contents

1 Introduction **1**

1.1 Support Vector Machine: an Overview 1

1.2 Learning with Unknown Kernels 10

1.3 Distributed Learning Algorithms 13

1.4 Maximum Likelihood Estimation of Gaussian Mixture Models with Known Variances 15

1.5 Contributions 16

2 Kernel Optimization for Multi-Class Support Vector Machines 19

2.1 Introduction 19

2.2 Formulations for Kernel Optimization 25

2.3 Application: Handwritten Digits Classfication 28

2.4 Application: Retinal Ganglion Cells Classification 31

2.5 Conclusion and Discussion 42

3 Kernel Optimization for Regression Estimation **43**

3.1 Introduction 43

3.2 Kernel Optimization 47

3.3 Numerical Experiments 50

3.4 Application: Retina Ganglion Cell Signal Encoding 56

3.5 Conclusion 59

4 Distributed Support Vector Machine 66

4.1 Introduction 66

4.2 Algorithm 67

4.3 Proof of Convergence 69

4.4 Performance Studies 71

4.5 Conclusions and Future Research 80

5 Parallel Randomized Support Vector Machine 84

5.1 Introduction 84

5.2 Support Vector Machine and the Sampling Lemma 85

5.3 Algorithm 89

5.4 Proof of the Average Convergence Rate 92

5.5 Simulations and Applications 95

5.6 Conclusions 98

6 Maximum Likelihood Estimation of Gaussian Mixture Models 100

6.1 Introduction 100

6.2 ML Estimation of Gaussian Mixture Distributions 103

6.3 Generalized Benders decomposition 111

6.4 Proof of Convergence 113

6.5 Demonstration 116

6.6 Conclusion and Future Research 117

7 Conclusion 119

7.1 Kernel Optimization 119

7.2 Distributed Learning 120

References **122**

List of Figures

2.1 A diagram of the one-against-one binary classifier (plotted in the solid line), the one-against-others binary classifiers (plotted in the dot-dash lines) and the relationship of their normal vectors. . . . 22

2.2 A set of random samples of the MNIST handwritten digits database. 29

2.3 Gray level indices of probabilities. The transition probability is indexed by intensity level. The highest intensity denotes probability 1 and lowest the intensity level denotes probability 0. This is one way of feature visualization. 35

2.4 The average conditional transition probabilities gray index map of neuron 6. Each image is corresponding to the average transition probability matrix $\frac{1}{N_6}\sum_{i=1}^{N_6} A_x^i, \forall x$, where N_6 is the number of training samples for the neuron 6. One may note that when $x \leq 3$, the neuron tends to jump down to state 1; when $x = 4, 5$, the neuron tends to remain or jump up to state 2 and when $x = 6, 7$, the neuron tends to jump up to state 3 or 4 37

2.5 The average conditional transition probabilities gray index map of neuron 9. Each image is corresponding to the average transition probability matrix $\frac{1}{N_9}\sum_{i=1}^{N_9} A_x^i, \forall x$, where N_9 is the number of training samples for the neuron 9. One may note that the neuron tends to jump to state 2 regardless of the input symbol x. 38

2.6 The average conditional transition probabilities gray index map of neuron 16. Each image is corresponding to the average transition probability matrix $\frac{1}{N_{16}} \sum_{i=1}^{N_{16}} A_x^i, \forall x$, where N_{16} is the number of training samples for the Neuron 16. One may note that the Neuron 16 has a broader distribution of transition probabilities that means it may go up to high energy state with relatively high probability. There is no jump-up with very high probability observed. 39

3.1 The testing result of problem 1. For the linear model, all kernels work very well. 53

3.2 The testing result of problem 2. For the quadratic model, except for the linear kernel, all kernels work very well. 53

3.3 The testing result of problem 3. For the cubic model, except for the linear and quadratic regressor, all kernels work very well. . . . 54

3.4 The testing result of problem 4. For a given polynomial model, the optimal kernel works much better than fixed kernels. 54

3.5 The testing result of problem 5. For the given polynomial model, the optimal kernel has the best performance. 55

3.6 The testing result of problem 6. For the Gaussian model, the combined kernel works very well. 55

3.7 The testing result of problem 7. For the difference of Gaussian model, the combined kernel works very well. 56

3.8 Average linear filter for neurons. The results match the previous obtained results using information theory by Zhong et al. [ZBJ05] 60

3.9 Dynamics of the encoding filter coefficients for neuron 1, 2, 3 and 4. From the left to the right, the w_{20}, w_{19}, w_{18} and w_{17} of neurons. are plotted against time. From the top to bottom are neuron 1, 2, 3, and 4. This figure demonstrates the non-stationary behavior of neurons. 61

3.10 Dynamics of the encoding filter coefficients for of neuron 5, 6, 7 and 8. From the left to the right, the w_{20}, w_{19}, w_{18} and w_{17} of neurons. are plotted against time. From the top to bottom are neuron 5, 6, 7, and 8. This figure demonstrates the non-stationary behavior of the neurons. 62

3.11 Dynamics of the encoding filter coefficients for of neuron 9, 10, 11 and 12. From the left to the right, the w_{20}, w_{19}, w_{18} and w_{17} of neurons. are plotted against time. From the top to bottom are neuron 9, 10, 11, and 12. This figure demonstrates the non-stationary behavior of the neurons. 63

3.12 Dynamics of the encoding filter coefficients for of neuron 13, 14, 15, 16 and 17. From the left to the right, the w_{20}, w_{19}, w_{18} and w_{17} of neurons. are plotted against time. From the top to bottom are neuron 13, 14, 15, 16 and 17. This figure demonstrates the non-stationary behavior of the neurons. 64

4.1 Demonstration of data distributions in iterations of the DSVM algorithm. Distributions of training vectors in 5 sites before iteration begins (the first row) and in iteration 1 to 4 (the 2nd to the 5th rows) are plotted with the local optimal classifiers. These figures show how all local classifiers converge to the global optimal classifier. 70

4.2 Diagram of a fully connected network with 5 sites. This is the densest network. 74

4.3 Diagram of a Round-Robin network with 5 sites. This the sparsest strongly connected network. 75

4.4 Diagram of a random strongly connected network. 75

4.5 Comparison of data accumulation of the parallel and distributed implementations. The first row shows the initial distribution of the training data in 5 sites. The 2nd row shows the data distribution in the last iteration (10th iteration) after a sequential implementation. The 3rd row shows the data distribution in the last iteration (8th iteration) after a parallel implementation. A unit variance Gaussian kernel is used in the DSVM algorithm. This figure shows that the algorithm converges without merging all the data and the sequential implementation achieves less data accumulation. 76

4.6 Computing time use the size of networks. *Left.* Elapsed computing time for a sequential implementation in a Round-Robin network. *Right.* Elapsed computing time for a parallel implementation in a fully connected network. Parallel implementation takes the advantage of the multiple servers and scales very well. 79

4.7 Data accumulation versus the size of networks. *Left.* Maximum number of training samples in a site after the convergence in a sequential implementation over a Round-Robin network. *Right.* Maximum number of training samples in a site after convergence in a parallel implementation over a fully connected network. Sequential implementation achieves less data accumulation. 79

4.8 Effect of the online implementation. *Top.* The number of training vectors, support vectors and the computing time of site 1 per iteration of the DSVM, implemented sequentially over a Round-Robin network. Here one iteration means that the solution in site 1 has been updated once. *Bottom.* The number of training vectors, support vectors and computing time of site 1 per iteration of the DSVM implemented parallel over a fully connected network. These two figures show that the initial increase of the computing time is mainly caused by the increasing of the number of support vectors. The later reduction in the computing time is the result of the online implementation. Comparing these two figures, one may observe that sequential implementation gain more than the parallel one from the online implementation. The reason is that in sequential implementation each site may get the most current α value as the input. 81

5.1 Weights of training vectors in iterations. Darker points denote higher weights. *Left.* The first iteration. *Middle.* The sixth iterations. *Right.* The last iteration. The figures demonstrate how those support vectors get higher and higher weights during iterations. 96

5.2 Number of violators and SVs found in each iteration of PRSVM. These figures demostrate the effect of using more servers. The system with more servers will find the violator and support vectors much faster than that with less servers. 99

6.1 Convergence of EM Algorithm and Benders Method (Example 1). 116

6.2 Convergence of EM Algorithm and Benders Method (Example 2). 117

List of Tables

2.1 Kernel selection and the testing error for the MNIST Database. The result shows that the Gaussian kernel K_3 always has a significantly large positive coefficient. If K_3 is involved in kernel combinations, the testing errors are low. 30

2.2 Comparison of the performance between the optimal kernel, fixed kernels of homogeneous features and a kernel of vectorized features, where $\overline{\mathrm{ERR}_{\mathrm{te}}}$ is the average testing error and $\sigma_{\mathrm{ERR}_{\mathrm{te}}}$ is the standard deviation of testing errors among 100 experiments. The result from the best individual feature kernel K_1 can be considered as the result of voting since there are only two candidates. The kernel K_{vec} from the vectorized feature has very bad result that is close to that of the worst kernel K_2. The optimal combined kernel achieves the best testing error. 41

3.1 Testing error of different kernels. The testing error e is the averaged ϵ-insensitive error defined in (3.19). This table shows that the optimized kernel in general has the best performance. Higher order polynomial kernels suffer from the curse of the dimensionality (see problem 1, 2 and 3) but lower order of kernels suffer from the limited regression capability (see problem 2, 3, 4, 5, 6, and 7). The optimal kernel has a good balance of different kernels. It may achieve much better performance than fixed kernels (see problem 4). 52

3.2 Comparison of testing errors of a linear kernel and the optimal kernel. Er_{K_1} and Er_K denote the average estimation error with a linear kernel and the optimal kernel respectively. The linear kernel works pretty well except for neuron 3 and 16, where the optimal kernel works much better. This shows that most neurons can be encoded by linear filters while neuron 3 and 16 can not be accurately encoded by linear filters. 58

4.1 A randomized initial distribution of training data of the toy example. 72

4.2 Algorithm performance in networks of different size. Experiments show that larger fully connected networks allow the advantage of shorter processing time but has the disadvantage of more training data accumulation. The result also suggests that the size of network has a limited effect on the communication overhead per exchange, measured by δ 73

4.3 Algorithm performance for different network topologies and type of implementations. Experiments show that denser network has the disadvantage of more data accumulation. Sequential implementation achieves much less data accumulation than parallel implementation. The best computing time is achieved by a randomly strong connected network. 74

4.4 Algorithm performance over the initial distribution of training data. In this table, σ denotes the standard deviation of initial training data distribution. This table shows that the unbalanced initial training vector distribution has very limited effect on the data accumulation (in terms of the total number of transferred vectors Δ) and the computing time e. 78

5.1 Algorithm performance comparison of SVM$^{\text{light}}$, RSVM and PRSVM. This table shows that the PRSVM really takes the advantage of the multiple servers. When the number of servers are limited, the salability of the PRSVM is good. Though the PRSVM is still not as good as the SVM$^{\text{light}}$, lack of a provable convergence rate makes the later one not always preferable. 97

Acknowledgments

I am extremely grateful to people who have helped me and inspired me throughout my education at UCLA. First and foremost, I would like to thank my advisors, Prof. Vwani Roychowdhury and Prof. Lieven Vandenberghe. Prof. Roychowdhury has been truly inspiring, encouraging and considerate. I would like to thank him for supporting my Ph.D. study in the past four years. My research progress benefited greatly from Prof. Roychowdhury's versatile knowledge and broad research interests. Without his support and encouragement, I could not have reached this far.

Prof. Vandenberghe is always nice, precise and willing to provide advice. He provided me very concrete and helpful advice in research topics, coding skills, writing styles and many other aspects. The four years at UCLA therefore became a comprehensive training in research, in writing and in thinking that may benefit me for a whole life. I am so lucky to be one of his students.

I am grateful for Dr. Hong Shen and Dr. Lin Hong at Siemens Corporate Research, who gave me the opportunity to get involved in an interesting and challenging project in medical image analysis and computer vision. During the ten months with them, I have learned how critical my knowledge acquired at UCLA could be for saving human's life. I would like to thank them for offering me the first opportunity in my life to work in industry.

Thanks to Prof. Sheila Nirenberg for providing me with a very interesting data set on retina neuron signal analysis. We were then able to apply our kernel optimization method to this interesting application and achieved good results.

I would like to thank my Mom for being my mentor and my best friend throughout the years. She has been proud of me no matter I succeeded or not.

For my wife, Shanshan Wu, I am so grateful. She has always been supportive. Thank her for everything she has done for me.

I am thankful for my friends, Hao Yang, Yichen Liu, Yan Li, Xiaolong Huang, Yuan Yao, Chang Liu, Ping Zhang, Xueli Liu, Yang Yan, Luyong Wang, Chao Yuan as well as my cousin, Li Pan and many others. There is no way to survive in this country without their friendship and support.

Vita

1977	Born, Wuhan, Hubei, P. R. China
1999	B.Eng. (Automatic Control), Huazhong University of Science and Technology, Wuhan, China
1999–2001	Teaching Assistant, Department of Industrial Engineering and Engineering Management, Hong Kong University of Science and Technology
1999–2001	Research Assistant, Department of Industrial Engineering and Engineering Management, Hong Kong University of Science and Technology
2001	M.Phil. (Industrial Engineering and Engineering Management), Hong Kong University of Science and Technology, Hong Kong, China
2001–2005	Research Assistant, Department of Electrical Engineering, University of California, Los Angeles, California
2003–2004	Teaching Assistant, Department of Electrical Engineering, University of California, Los Angeles, California
2004–2005	Temporary Technical Staff, Department of Intelligent Vision and Reasoning, Siemens Corporate Research, Ltd., Princeton, New Jersey

Abstract of the Dissertation

Kernel Optimization and Distributed Learning Algorithms for Support Vector Machines

by

Yumao Lu

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2005

Professor Vwani Roychowdhury, Co-chair

Professor Lieven Vandenberghe, Co-chair

The support vector machine (SVM) has been one of the most successful tools for pattern recognition and function estimation in the recent ten years. The underlying training problem can be formulated as a large quadratic programming problem that can be efficiently solved via existing decomposition algorithms. This assumes, however, that the kernel function and all the parameters are given and the training vectors can be locally accessed. These assumptions do not hold for classification problems with heterogeneous features or distributed training data. The purpose of this thesis is to address these problems by developing methods for kernel optimization and distributed learning.

Recent advances in kernel machine algorithms based on convex optimization have made it easier to incorporate information from training samples with little user interaction. As a first contribution, we generalize kernel estimation techniques for binary classification to multi-class classification problems. The kernel optimization problem for multi-class SVM is formulated as a semi-definite programming problem (SDP). A decomposition method is proposed to reduce the

computational complexity of solving the SDP. As a further step of generalization, we consider kernel optimization for support vector regression (SVR). The proposed kernel optimization methods are applied to retina ganglion cell neuron signal analysis.

The distributed SVM training algorithm proposed in this thesis is based on a simple idea of exchanging support vectors over a strongly connected network. The properties of the algorithm in various configurations have been analyzed. We also propose a randomized parallel SVM that uses randomized sampling and has a provably fast average convergence rate.

Finally, we discuss the maximum likelihood estimation of Gaussian mixture models with known variances. The problem is formulated as a bi-concave maximization problem. We work out the details of the generalized Benders decomposition for calculating the global optimum of the bi-convex problem. Simple numerical results are presented to demonstrate the advantage over the widely used Expectation-Maximization (EM) algorithm.

CHAPTER 1

Introduction

This chapter describes the fundamentals of support vector machines and the challenges underlying kernel estimation and distributed learning. The goal is to provide an overview of the basic concepts.

1.1 Support Vector Machine: an Overview

The Support Vector Machine (SVM) has recently become the most popular method for classification and regression in machine learning.

1.1.1 Support Vector Classification

One of the fundamental pattern recognition problems is the linear separation problem. We are seeking a hyperplane to separate a set of positively and negatively labeled training data. The hyperplane is defined by $w^T x - b = 0$, where the parameter $w \in \mathbf{R}^m$ is a vector orthogonal to the hyperplane and $b \in \mathbf{R}$ is the bias. The decision function is the hyperplane classifier

$$H(x) = \text{sign}(w^T x + b).$$

The hyperplane is designed such that $y_i(w^T x_i - b) \geq 1$, where $x_i \in \mathbf{R}^m$ is a training data point and $y_i \in \{+1, -1\}$ denotes the class of the vector x_i. The margin is defined by the distance of the two parallel hyperplanes $w^T x - b = 1$

and $w^T x - b = -1$, i.e. $2/||w||_2$. The margin is related to the generalization of the classifier [Vap95].

The support vector machine (SVM) classifier is computed by solving a quadratic programming problem. We maximize the margin over the parameters of the linear classifier, which is defined as follows:

$$\begin{array}{ll} \text{minimize} & (1/2)w^T w \\ \text{subject to} & y_i(w^T x_i - b) \geq 1,\ i = 1, ..., N \end{array} \tag{1.1}$$

The dual problem of problem (1.1) can be written as a quadratic programming (QP) problem

$$\begin{array}{ll} \text{maximize} & -(1/2)\alpha^T Q\alpha + \mathbf{1}^T\alpha \\ \text{subject to} & \alpha \geq 0,\ i = 1, ..., N, \\ & y^T\alpha = 0, \end{array} \tag{1.2}$$

where variables $\alpha \in \mathbf{R}^N$, sample labels $y \in \mathbf{R}^N$, the Gram matrix $Q \in \mathbf{R}^{N\times N}$ and $Q_{ij} = y_i y_j x_i^T x_j$.

The support vectors (SVs) are defined as the subset of the training vectors with non-zero dual multiplier α_i. By the complementary slackness condition,

$$\alpha_i[y_i(w^T x_i + b) - 1] = 0 \text{ for all } i = 1, ..., N, \tag{1.3}$$

in the optimum. So the SVs lie on the margin boundary.

One must note that the problem (1.1) is not feasible if the classes are not linearly separable. For those nonseparable problems, a set of slack variables ξ_i's is introduced. The SVM training problem for the nonseparable problem is defined as follows:

$$\begin{array}{ll} \text{minimize} & (1/2)w^T w + \gamma\mathbf{1}^T\xi \\ \text{subject to} & y_i(w^T x_i + b) \geq 1 - \xi_i,\ i = 1, ..., N \\ & \xi \geq 0 \end{array} \tag{1.4}$$

where the scalar γ is called the regularization parameter, and is usually empirically selected to reduce the testing error rate.

The dual of the problem (1.4) is shown as follows:

$$\begin{aligned} \text{maximize} \quad & -(1/2)\alpha^T Q\alpha + \mathbf{1}^T\alpha \\ \text{subject to} \quad & 0 \leq \alpha \leq \gamma\mathbf{1} \\ & y^T\alpha = 0. \end{aligned} \tag{1.5}$$

For the general SVM optimization problem, the complementary slackness condition has the form

$$\alpha_i[y_i(w^T x_i + b) - 1 + \xi_i] = 0 \text{ for all } i = 1, ..., N, \tag{1.6}$$

in the optimum. Therefore, SVs are the vectors that lie in between the margin boundary (including those on the boundary) and that are misclassified.

The selection of parameter γ can be referred to a more general framework: regularized risk function or structural risk function design [VL63, KW71]:

$$R_{\text{reg}} := R_{\text{emp}} + \frac{1}{\gamma}\Omega,$$

where R_{reg} denotes the structural risk, R_{emp} is the empirical risk and only depends on the training error [VW96], Ω is the stabilization or generalization term that reflects the prior information [SS98] and the parameter γ is used to balance these two terms.

A nonlinear kernel function can be used for nonlinear separation of the training data. In that case, the linear function $w^T x$ is replaced by a nonlinear function $\phi(x) : \mathbf{R}^m \rightarrow \mathbf{R}^n$, where m is the dimension of original training vectors and n is the lifted dimension. Linear kernels have $n = m$. The Gram matrix Q has therefore the component $Q_{ij} = y_i y_j \phi(x_i)^T \phi(x_j)$. A more general way is to replace $\phi(x)^T\phi(\tilde{x})$ by a kernel function $F(x, \tilde{x})$ such that $Q_{ij} = K_{ij} y_i y_j$ and

$K_{ij} = F(x_i, x_j)$, where $K \in \mathbf{R}^{N \times N}$ is the so-called kernel matrix. The nonlinear kernel may lift the dimension of training vectors to a higher dimension so that they can be separated linearly. If the kernel matrix K is positive definite, problem (1.5) is guaranteed to be strictly convex. Correspondingly we have the decision function of the form

$$H(x) = \text{sign}(\sum_{i=1}^{N} y_i \alpha_i F(x_i, x) + b),$$

where the bias b can be obtained by averaging

$$b = y_j - \sum_{i=1}^{N} y_i \alpha_i K_{ij}, \forall j$$

by the KKT condition (1.6).

1.1.2 Multi-Class Support Vector Machine

We often encounter classification problems with more than two classes. Researchers have generalized binary SVM to multi-class SVM, in which the label y_i may take k possible values. Various formulations have been developed with different definitions of classifiers and different geometric explanations. There are in general three groups of formulations: one-against-others, one-against-one and the DAGSVM method.

The idea of the one-against-others method [Fri96, Kre99] is to construct k binary classifiers, $w_l^T \phi(x) + b_l, l = 1, \ldots, k$, and to use the decision function

$$H(x) = \underset{l=1,\ldots,k}{\text{argmax}} (w_l^T \phi(x) + b_l).$$

On the other hand, the one-against-one method constructs $k(k-1)/2$ binary classifiers, $w_{ij}^T \phi(x) + b_{ij},\ i < j,\ i, j \in 1, \ldots, k$, and the testing can be based on the result of a voting process, namely

$$H(x) = \underset{l=1,\ldots,k}{\text{argmax}} \sum_{i \neq l} \text{sign}(w_{il}^T \phi(x) + b_{il}).$$

if we define $w_{ji} = -w_{ij},\ b_{ji} = -b_{ij},\ \forall i < j$.

The DAGSVM method is actually a modified one-against-others method. The difference is in the testing phase. Instead of voting, DAGSVM builds a generalized decision tree, a directed acyclic graph (DAG), that has $k(k-1)/2$ internal nodes and k leaves [PCS99]. A good comparison of these three formulations has been done by Hsu and Lin in 2002 [HL02].

A simplified one-against-other method, Crammer and Singer's method [CS01], is of special interest due to its simplicity and relatively better performance [HL02]. In Crammer and Singer's work, the decision function

$$H(x) = \underset{l=1,\ldots,k}{\operatorname{argmax}}(w_l^T \phi(x))$$

is used. The primal problem has the form:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\sum_{l=1}^{k} w_l^T w_l + \gamma \mathbf{1}^T \xi \\ \text{subject to} & w_{y_i}^T \phi(x_i) - w_l^T \phi(x_i) \leq (1 - d_i^l) - \xi_i,\ i = 1, \ldots, N,\ l = 1, \ldots, k \end{array} \tag{1.7}$$

where

$$d_i^l = \begin{cases} 1 & \text{if } y_i = l \\ 0 & \text{otherwise.} \end{cases} \tag{1.8}$$

This formula has only N slack variables $\xi_i, i = 1, \ldots, N$. The term $w_l^T w_l$ is still related to the generalization error [PCS99].

The dual problem of (1.7) is

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\alpha^T (K \otimes I)\alpha - d^T \alpha \\ \text{subject to} & \mathbf{1}^T \alpha_i = 0 \\ & \alpha \leq \gamma d, \end{array} \tag{1.9}$$

where $\otimes$ denotes the kronecker product, I is a k-by-k identity matrix, $\alpha_i \in \mathbf{R}^{k \times 1}$, $\alpha = [\alpha_1^T, \alpha_2^T, \ldots, \alpha_n^T]^T \in \mathbf{R}^{kN \times 1}$, $d = [d_1^T, d_2^T, \ldots, d_N^T]^T \in \mathbf{R}^{kN \times 1}$ is a constant vector and the l-th component of vector d_i, d_i^l, is defined in (1.8).

The decision function based on the dual solution is

$$\arg \max_{l=1,\ldots,k} \sum_{i=1}^{N} \alpha_i^l F(x_i, x),$$

where F is a predefined kernel function.

Our kernel learning technique proposed in Chapter 2 for multi-class SVM is based on Crammer and Singer's formulation.

1.1.3 Support Vector Regression

Regression is a further step of generalization that allows the label y_i to take any real value. The support vector regression (SVR) has been widely applied in financial forecasting [CT03], gene prediction [Xu05], time series [TBW03], signal coding/coding, density estimation [CHH04] and other functional estimations due to its flexibility of estimating real-valued functions. The basic idea is to minimize a pre-defined risk function over the parameters of the regressor.

To define a risk function, we start from the maximum likelihood estimation. The likelihood of samples $(x_1, y_1), \ldots, (x_N, y_N)$ given an underlying functional dependency f is given by

$$\text{Prob}((x_1, y_1), ..., (x_N, y_N)|f) = \Pi_{i=1}^{N} \text{Prob}(y_i|x_i, f)\text{Prob}(x_i). \tag{1.10}$$

To maximize the likelihood (1.10) over f is equivalent to minimizing the empirical risk R_{emp} if the loss function c is defined as

$$c(x, y, f(x)) = -\ln \text{Prob}(y|x, f),$$

since the empirical risk R_{emp} can be written as

$$R_{\text{emp}} := \frac{1}{N} \sum_{i=1}^{N} c(x_i, y_i, f(x_i)).$$

Therefore, one may easily see that the Laplacian loss function $|y - f(x)|$ corresponds to the density model $\exp(-|y - f(x)|)/2$; the Gaussian loss function $|y - f(x)|^2/2$ corresponds to the density model $\exp(-|y - f(x)|^2/2)/(2\sqrt{2\pi})$, the polynomial loss function $|y-f(x)|^d/d$ corresponds to the density $\frac{d}{2\Gamma(1/d)} \exp(-|y-f(x)|^d)$, and etc.

To preserve the property that the number of support vectors (SVs) is limited, Vapnik used the ϵ-insensitive loss function

$$c(x, y, f(x)) = |y_i - f(x_i)|_\epsilon \equiv \max\{0, |y - f(x)| - \epsilon\},$$

which does not penalize errors below the tolerance ϵ [Vap95]. We, again, start from the estimation of a linear function

$$f(x) = w^T x + b, \tag{1.11}$$

based on independent and identically distributed (IID) training samples

$$(x_1, y_1), \ldots, (x_N, y_N)$$

where $x_i \in \mathbf{R}^m$, $y_i \in \mathbf{R}$, $w \in \mathbf{R}^m$, $b \in \mathbf{R}$ and $f : \mathbf{R}^m \to \mathbf{R}$.

The total risk can be written as

$$R = \int c(f, x, y) d\text{Prob}(x, y),$$

where $\text{Prob}(x, y)$ is the joint probability of an observation pair (x, y) and c is the loss function. Since we never know the true $\text{Prob}(x, y)$, we may use a regularized risk function

$$R := R^\epsilon_{\text{emp}} + \frac{1}{\gamma}\Omega \tag{1.12}$$

where

$$R^\epsilon_{\text{emp}} := \frac{1}{N} \sum_{i=1}^{N} |y_i - f(x_i)|_\epsilon$$

is the empirical risk,

$$\Omega := \frac{1}{2}||w||^2$$

is the regularization term and γ is an empirically selected scalar used to balance these two terms. A small $||w||^2$ means that the linear regressor (1.11) is flat in the feature space. Simple geometry may verify that the flatter the function f, the larger the margin [Vap95].

To minimize (1.12), we have

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}w^T w + \gamma \mathbf{1}^T(\xi + \xi^*) \\ \text{subject to} \quad & w^T x + b - y \leq \epsilon + \xi \\ & y - w^T x - b \leq \epsilon + \xi^* \\ & \xi, \xi^* \geq 0, \end{aligned} \tag{1.13}$$

where $\xi \in \mathbf{R}^N$ and $\xi^* \in \mathbf{R}^N$ is the allowed error "above" and "below" the margin boundary.

The corresponding dual problem has the form

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}(\alpha^* - \alpha)^T K(\alpha^* - \alpha) - y^T(\alpha^* - \alpha) + \epsilon \mathbf{1}^T(\alpha^* + \alpha) \\ \text{subject to} \quad & \mathbf{1}^T(\alpha^* - \alpha) = 0 \\ & 0 \leq \alpha^*, \alpha \leq \gamma. \end{aligned} \tag{1.14}$$

where the dual variable $\alpha^*, \alpha \in \mathbf{R}^N$ and the training kernel matrix $K \in \mathbf{R}^{N \times N}$. The component of the kernel matrix K can be written as $K_{ij} = \phi(x_i)^T \phi(x_j)$. That is, we are actually estimating a function

$$f(x) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \phi(x_i)^T \phi(x) + b.$$

Kernel selection in regression therefore can be considered as regressor modeling.

1.1.4 Training Algorithms

The problem of training an SVM was developed by Vapnik in 1995 [Vap95, CV95]. The training problem is equivalent to a box constrained quadratic programming (QP), that can be solved by standard optimization algorithms [Vap95, CV95, Sch96]. These algorithms, however, are only able to solve some small and medium size problems since they require loading of the whole Gram matrix.

Osuna, in 1997, proposed a decomposition algorithm [OFG97b]. The basic idea is to separate the training vectors into two sets: a working set and a non-working set. Only the working set is solved. A nonsupport vector in the working set is then replaced by one in the nonworking set in each iteration such that the objective value keeps decreasing. This algorithm, however, requires the working set to contain all support vectors, which is an unrealistic assumption for large classification problems (say, a problem that has 50,000 training samples). This algorithm was improved in 1997 by Osuna, et al., themselves [OFG97a]. This improved decomposition algorithm is often called the chunking algorithm. The chunking algorithm chooses a working set arbitrarily, solves the subproblem and replaces a training vector in the working set by one in the nonworking set that violates the Karush-Kuhn-Tucker optimality conditions (KKT) such that the objective value can be improved. The advantage of the chunking algorithm is that the working set may be smaller than the set of support vectors.

In order to further increase the computing speed, Platt, in 1998, proposed a sequential minimization optimization (SMO) algorithm for training SVM, which later became the the standard algorithm for large scale SVM training problems. The basic idea of the SMO method is to simplify the numerical QP optimization by selecting two Lagrange multipliers, the largest possible optimization problem that can be solved analytically, to optimize in each step. Therefore, the SMO

can be considered a special case of chunking: the number of variables of the working set is fixed to two. Selection of the two multipliers depends on checking of the KKT conditions, which dominates the run time, especially when the kernel matrix is not sparse. Both chunking and SMO algorithms are decomposition algorithms that have been extensively studied [Joa98, CB01, FL02]. Among the different implementations, the $\text{SVM}^{\text{light}}$ has the fastest reported performance [Joa98] in selected applications. A parallel version of the $\text{SVM}^{\text{light}}$ was proposed by Zanghirati and Zanni in 2003 [ZZ03]. Another sequential algorithm is the successive overrelaxation (SOR), proposed by Mangasarian and Musicant in 1999 [MM99]. The SOR is a matrix splitting method used for boxed constrained QP that is essentially sequential. SOR also has several parallel versions [AJ86, BFM90].

1.2 Learning with Unknown Kernels

The traditional support vector machine assumes that the underlying kernel matrix is given. The assumption is not true if the feature sets are heterogeneous.

Homogenous features have uniform structure and composition throughout, such as pixels of images, frequencies of vocabularies, etc. Heterogeneous features, on the contrary, have varying scales. Heterogeneous features usually come from different modalities, multiple feature extraction approaches or distinct time scales.

One can always make homogenous feature measurements into a vector without worrying about the relative scaling problem. It is always a bad idea, however, to vectorize heterogeneous features since the "weaker" feature may dominate the feature space and the testing result may be worse than the case that only one set

of features are used. Centering and normalization does not help here because the metric is difficult to be introduced for heterogeneous features. Examples can be found in Section 2.4.

There are two groups of methods dealing with heterogeneous features: one is decision fusion; the other is feature fusion. Namely, decision fusion transforms the decision space and feature fusion transform the feature space. Voting (for example, Adaboost) [DHS00], decision tree [Val04] and Bayesian network [NH01] are all widely used decision fusion methods, which compose the final strong classifier based on the *independent* weak classifiers constructed from heterogeneous features. These methods suffer from the fact that each feature set is used independently to generate a weak classifier.

Functional combination of heterogeneous features is a way to make the final decision that depends on joint information of features. The main difficulty for functional combination is the parameter estimation. If an empirical posterior is given, one can minimize the KL divergency of the true posterior and the empirical posterior over the combined features [HKC04]. The weakness of this method is that there must be a good empirical posterior available that may not be true in reality.

If we apply a linear combination to p kernel matrices K_1, K_2, ..., K_p, formed from p sets of heterogeneous features, the resulting kernel matrix K can be written as

$$K = \mu_1 K_1 + \mu_2 K_2 + ... + \mu_p K_p,$$

where $\mu_.$'s are unknown linear coefficients.

Selection of Kernels

Except for the fact that the kernel matrix should be positive definite [Mer09], there is almost no consensus on how to select kernel matrices for a specific application. The question is "what is the best kernel for a given application?".

Selection of kernel should always reflect one's prior knowledge about the problem at hand [SS02]. There have been extensive research on properties of polynomial kernels [SSM98, VGS97], radial basis kernels [BGV92, GBV93, GBV93], ANOVA kernels [ONR95, SGV99, Wah90], and other kernels [Bur98, UCS99].

The kernel selection problem can be considered a special case of the heterogeneous features combination problem. Using different kernel function, the same feature set will be projected to different spaces. The projected features are no longer homogenous. We again may use a linear combination of different type of kernels with unknown coefficients to algorithmically select the best kernel. For details, please refer to Section 2.3.

Learning kernels for the binary support vector machine training problem has been attacked by Lanckriet, et al. in 2004 [LCB04]. Their method is to construct a semi-definite programming problem (SDP) by looking at the dual of problem (1.5).

In this thesis, we apply a similar technique to the more general SVM multiclass classification problem and functional estimation problem. We improve the efficiency of solving the underlying optimization problem by using a matrix decomposition method and provide results on multiple applications.

1.3 Distributed Learning Algorithms

Distributed learning is necessary if a centralized system is infeasible because of geographical, physical and computational reasons. The applications of distributed learning methods cover geoscience [PG98, JL99], defense system [Bas98, CZB97], imaging [LKT98] and etc. The popularity of distributed learning is due to the availability of advanced sensor networks, wired and wireless network communication technology [LCK04], and numerous successful distributed learning algorithms.

Current distributed learning algorithms can be summarized into four categories: Bayesian learning [FHL03], Dempster-Shafer theory [Bra00], fuzzy set theory [SH01] and neural networks [XLB02]. Bayesian learning accumulates distributed or temporal evidence, say x_1 and x_2, by calculating the posterior of a certain event E

$$p(E|x_1, x_2) = p(x_1|E, x_2)p(x_2|E)\frac{p(E)}{p(x_1, x_2)}.$$

In classification, E is usually a set of labeling results. The optimal result can be achieved by minimizing a Bayes risk function given the α and β error risk measurements. Bayesian learning needs a set of definitions of the likelihood functions, which requires the knowledge of the distribution family that may not be available in practical applications. Fuzzy set theory defines various set operations that are able to compromise the membership value in different sets. The fuzzy sets fusion is now becoming popular because of its capability of quantifying the events in the real world, say human language. The performance of fuzzy algorithms is, however, hard to evaluate because of its arbitrary representation of events. The Dempster-Shafer theory combines evidences by using a normalized orthogonal

sum

$$m_c(H) = \frac{\sum_{\forall X,Y:X\cap Y=H} m_1(X)m_2(Y)}{\sum_{\forall X,Y:X\cap Y=\emptyset} m_1(X)m_2(Y)},$$

where $m_i(.)$ is a basic probability assignment function for site i and H is an intersection of events set X and Y [Bra00]. That is, we can always calculate the "likelihood" of the intersection of distributed events. Again, it requires many pre-defined probabilities. Neural networks and hidden Markov models (HMM) work by approximating the real system behavior and usually require a sufficient amount of training data and training time. The optimality of a neural network is usually unknown, while an HMM may suffer from an inappropriate model structure.

The objectives of a distributed learning algorithm usually include robustness to noise, efficient representation for data of high-dimension and large volume, decision precision and certainty. The performance is usually measured by simulation comparison. Optimality criteria are seldom emphasized due to the complexity of the problem, and the arbitrary definition and assignment of likelihood functions, probabilities [Bra00], fuzzy sets association values [SH01], etc.

According to a statistical review [VMB01], eleven percent of distributed learning methods considered physically distributed data processing. For such problems, there are some additional objectives: robustness to network topology [ZL03] and efficient communications [VV97]. The information that is actually transferred could be the raw data, the features or local decisions.

Efficient Information Carrier: Support Vectors

Support vectors (SVs), namely the training data that have non-zero α values, lie physically inside the margin or are misclassified. Those vectors carry all the classification information of the local data set. Exchanging support vectors, instead

of moving all the data, is a natural way to fuse information among distributed sites. However, to the best of our knowledge, no research has been done using such a technique.

We exploit the idea of exchanging SVs in a deterministic way and a randomized way in our distributed learning algorithms in Chapter 4 and 5.

1.4 Maximum Likelihood Estimation of Gaussian Mixture Models with Known Variances

This is a relatively independent topic from the support vector machine. But it is rather interesting and related to global optimization.

Gaussian mixture models are a type of density model that comprises of a number of Gaussian functions. The density function has the form

$$f(y) = \sum_{j=1}^{n} w_j \frac{1}{\sqrt{2\pi s_j}} \exp\{-\frac{(y-\mu)^2}{2s_j}\}.$$

These component functions are combined to provide a multimodal density estimation. In a model inference problem, we are given a set of data and want to estimate the model parameters (w, μ, s) from these data. The maximum likelihood estimation is one of the most popular methods in parameter estimation. Unfortunately there is no efficient way to solve the globally optimal solution of the following maximal likelihood problem even when the variances s are known.

$$\begin{array}{ll} \text{maximize} & \sum_{k=1}^{N} \log \sum_{j=1}^{n} w_j \frac{1}{\sqrt{2\pi s_j}} \exp\{-\frac{(y-\mu)^2}{2s_j}\} \\ \text{subject to} & w \succeq 0, \mathbf{1}^T w = \mathbf{1}, \end{array}$$

where the variables are w and μ.

In Chapter 6, we reformulate the maximum likelihood estimation of Gaussian mixture model with known variances as an equivalent biconvex problem. All the

varieties of EM algorithms can be viewed as greedy solutions to this problem. One implication of the biconvexity of this formulation is that there exist global optimization methods that exploit this kind of structure. As an example, we worked out the details of generalized Benders decomposition [Geo72, FV93] and present numerical results for some simple cases.

1.5 Contributions

Our contributions in this thesis can be summarized as follows:

- ◇ We show that the SDP kernel learning technique can be efficiently applied in multi-class SVM training problems. The main obstacle is the big $Nk \times Nk$ matrix in the objectives, where N is number of training samples and k is the number of classes. We decomposed the SDP problem such that the size of each LMI constraint is the same as that in the binary problem, which consists of a $(N+1) \times (N+1)$ matrix, and there are only k such LMI constraints.

- ◇ We apply the kernel learning technique to the regression problem, that provides a unique way of "model-free" regression. A linear model can be verified by comparing the testing errors of a linear kernel with that of the optimal kernel. New pattern has been discovered via our method for the retina ganglion cell coding problems.

- ◇ We exploit a simple idea of exchanging support vectors over a strongly connected network as a distributed learning algorithm. The algorithm is proved to converge to the global optimal classifier in finite steps. Extensive numerical results are provided to analyze the performance of the algorithm in different network configurations, various initial data distribution, and

different implementation methods.

- ◇ Further adding a carefully designed sampling mechanism, we propose a novel parallel randomized support vector machine (PRSVM). We prove that our algorithm, on average, converges to the global optimum classifier/regressor in less than $(6\delta \ln(N + 6r(C-1)\delta)/C$ iterations, where δ denotes the underlying combinatorial dimension, N denotes the total number of training vectors, C denotes the number of working sites; and r denotes the size for a working set. The average convergence rate is faster than the randomized support vector machine training algorithm in [BDT01] by a factor of C. The numerical results on synthesized data and a real geometric database show that our algorithm has good scalability.

- ◇ The maximum likelihood estimation problem of Gaussian mixture models with known variance is formulated as a bi-concave maximization problem. This bi-concave structure can be exploited by global optimization methods, such as the generalized Benders decomposition. The details of the generalized Benders decomposition have been worked out. Simple numerical results are presented to demonstrate the potential advantage of our method over the widely used EM algorithm.

The report is organized in the following way. We first present an overview for the support vector machine and its kernel estimation and distributed learning problem as well as related researches in Chapter 1. Then, we present our formulation and applications of kernel learning technique for multi-class SVM and SVM regression estimation in Chapter 2 and Chapter 3 respectively. The distributed SVM method is presented in Chapter 4, followed by its randomized version in Chapter 5. Chapter 6, introduces global optimization methods for maximum

likelihood estimation of Gaussian mixture models with known variances. The last chapter, Chapter 7, concludes this thesis briefly.

CHAPTER 2

Kernel Optimization for Multi-Class Support Vector Machines

2.1 Introduction

2.1.1 Kernels

Support vector machines are also called kernel machines. A kernel function $F(x, \tilde{x})$ is a generalized dot product in a feature space:

$$F(x, \tilde{x}) = \phi(x)^T \phi(\tilde{x}),$$

where

$$\phi : \mathbf{R}^m \rightarrow \mathbf{R}^n$$

is a nonlinear mapping that lifts the training vector to a higher dimension space in which they can be scattered by a hyperplane. That is,

The kernel matrix K is defined as:

$$K_{ij} = F(x_i, x_j), i, j = 1, \ldots, N.$$

If both training and testing can be carried out using the form of K exclusively, we do not need the explicit map ϕ. Instead, we may directly design kernel functions.

There are several popular choices of nonlinear kernel functions: inhomoge-

neous polynomial kernels

$$F(x, \tilde{x}) = x^T \tilde{x} + 1,$$

Gaussian kernels

$$F(x, \tilde{x}) = \exp\left(-\frac{||x - \tilde{x}||^2}{2\sigma^2}\right),$$

and sigmoid kernels

$$F(x, \tilde{x}) = \tanh(\kappa(x^T \tilde{x}) + \theta),$$

where $\kappa > 0$ and $\theta < 0$.

2.1.2 A Formulation of Multi-Class Support Vector Machines

We consider a classification problem with $N_{\rm tr}$ training samples: $(x_1, y_1), \ldots, (x_{N_{\rm tr}}, y_{N_{\rm tr}})$, where $x_i \in \mathbf{R}^m, \forall i$ are the feature vectors, $y_i \in \{1, 2, \ldots, k\}, \forall i$ are the labels and k is the number of possible classes. A general multi-class classification training problem is to seek a classifier $H : \mathcal{X} \to \mathcal{Y}$ such that the structural risk

$$R = R_{\rm emp} + \frac{1}{\gamma}\Omega$$

is minimized, where Ω is the generalization term, $R_{\rm emp}$ is the empirical risk and a parameter γ is used to balance these two terms. The empirical risk can be expressed as the sum of the training errors. That is

$$R_{\rm emp} := \frac{1}{N_{\rm tr}} \sum_{i=1}^{N_{\rm tr}} (1 - \delta_{y_i}(H(x_i))),$$

where

$$\delta_y(\tilde{y}) = \begin{cases} 1, & \text{if } y = \tilde{y} \\ 0, & \text{other wise.} \end{cases}$$

In the one-against-others method, we construct k hyperplanes $y = w_l^T x + b_l, l = 1, \ldots, k$ and the decision function has the form

$$H(x) = \underset{l=1,\ldots,k}{\rm argmax}(w_l^T x + b_l).$$

That is, for a sample i, to exclude it from one class $l, l \neq y_i$, we need to compare a pair of binary classifiers, such that

$$(w_{y_i} - w_l)^T \phi(x_i) + (b_{y_i} - b_l) \geq 1, l \neq y_i. \tag{2.1}$$

Crammer and Singer [CS01] further omit the bias $b_l, l = 1, \ldots, k$ and allow N_{tr} slack variables for general linear non-separable cases. Their multi-class SVM primal problem has the form

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} \sum_{l=1}^{k} w_l^T w_l + \gamma \mathbf{1}^T \xi \\ \text{subject to} & w_{y_i}^T \phi(x_i) - w_l^T \phi(x_i) \geq (1 - \delta_l(y_i)) - \xi_i, i = 1, \ldots, N_{\text{tr}}, l = 1, \ldots, k, \end{array} \tag{2.2}$$

where w is the variable and ξ is the slack variable. The constraints can be considered inequalities (2.1) with added slack variables.

The corresponding dual problem may have the form

$$\begin{array}{ll} \text{minimize}_\alpha & \frac{1}{2} \alpha^T (K_{\text{tr}} \otimes I) \alpha - d^T \alpha \\ \text{subject to} & \mathbf{1}^T \alpha_i = 0 \\ & \alpha \leq \gamma d, \end{array} \tag{2.3}$$

where $\alpha = [\alpha_1^T, \alpha_2^T, \ldots, \alpha_{N_{\text{tr}}}^T]^T \in \mathbf{R}^{kN_{\text{tr}}}$ is the variable vector and $\alpha_i \in \mathbf{R}^k$ [CS01]. The operator $\otimes$ denotes the Kronecker product, I is a k-by-k identity matrix, $d = [d_1^T, d_2^T, \ldots, d_{N_{\text{tr}}}^T]^T \in \mathbf{R}^{kN_{\text{tr}} \times 1}$ is a constant vector, where the l-th component of vector d_i is $\delta_l(y_i)$.

The generalization term $\sum_{i=1}^{k} w_l^T w_l$ is inspired as followed. The margin of the binary classifier (2.1) is defined as $2/||w_i - w_j||$. Define

$$g_{ij} = w_i - w_j$$

and

$$b_{ij} = b_i - b_j.$$

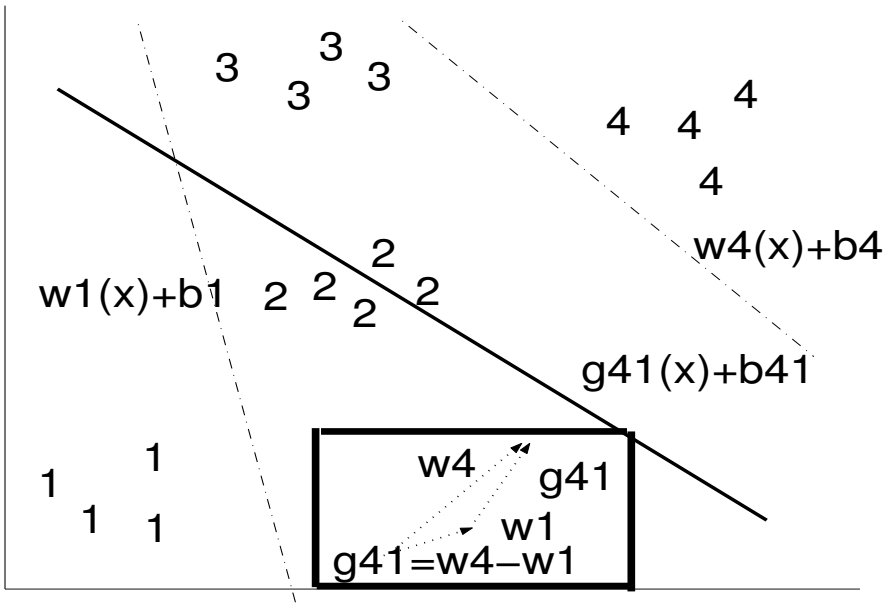

Figure 2.1: A diagram of the one-against-one binary classifier (plotted in the solid line), the one-against-others binary classifiers (plotted in the dot-dash lines) and the relationship of their normal vectors.

The hyperplane

$$g_{ij}^T \phi(x) + b_{ij} = 0$$

is also a binary classifier in the one-against-one method, where we have $k(k-1)/2$ such classifiers to compose the final multi-class classifier. In Figure 2.1, two binary one-against-others classifiers hyperplanes $w_1^T x + b_1 = 0$ and $w_4^T x + b_4 = 0$, are plotted in the dot-dash lines. The corresponding one-against-one classifier hyperplane $g_{41}^T x + b_{41} = 0$ are plotted in the solid line.

Platt, in 2000, showed that the generalization error Ω is proportional to the margin $\sum_{i<j} ||g_{ij}||$ in his modified one-against-one method, DAGSVM [PCS99]. One may observe that if $||w_i||^2 + ||w_j||^2$ is bounded, $||g_{ij}||^2$ is also bounded. That is, the generalization error will be limited by bounding $||w_i||^2 + ||w_j||^2$ from above.

Therefore, we can choose the generalization term to be

$$\Omega := \sum_{i=1}^{k} w_l^T w_l.$$

Our kernel optimization technique proposed in this chapter is based on Crammer and Singer's formulation for its simplicity and the slightly better performance comparing with other known multi-class SVM formulations [HL02].

2.1.3 Transduction Problem

In this chapter, we focus on the transduction problem. The transduction problem is the problem of completing the labeling of a partially labeled data set by estimating the classifier on the given points. That is, in a transduction problem, we do not intend to learn a general classification function. Instead, we are only interested in labeling the given unlabeled data points.

For a general multi-class problem, there are k classes and N data points, including N_{tr} unlabeled and training data and N_{t} testing data. Throughout this chapter, we use $X_{\text{tr}} \in \mathbf{R}^{N_{\text{tr}} \times m}$ to denote the labeled training set and $X_{\text{t}} \in \mathbf{R}^{N_{\text{t}} \times m}$ to denote the unlabeled testing set, where each row of the matrix X is a vector $x_i \in \mathbf{R}^m$. The label is stored in a vector $y \in \mathbf{R}^{N_{\text{tr}}}$. The kernel matrix

$$K = \begin{pmatrix} K_{\text{tr}} & K_{\text{tr,t}} \\ K_{\text{tr,t}}^T & K_{\text{t}} \end{pmatrix}, \tag{2.4}$$

where the block matrices K_{tr}, $K_{\text{tr,t}}$ and K_{t} are the optimal training, mixed and testing block that we want to learn.

Since the objective is only to complete the label of a finite testing set, there is no need to learn the general kernel function F. Instead, we only learn the kernel

matrix K. The estimated label for testing vector x_j can be written as

$$\hat{y_j} = \underset{l=1,\ldots,k}{\operatorname{argmax}} \sum_{i=1}^{N_{\mathrm{tr}}} (\alpha_i^l K_{\mathrm{tr,t}}(i,j)),$$

where α_i^l is the l-th component of vector α_i and $K_{\mathrm{tr,t}}(i,j)$ denotes the ij element of matrix $K_{\mathrm{tr,t}}$.

2.1.4 Challenges and Related Work

The selection of kernels should reflect our prior knowledge about the problem at hand [SS02]. An even better choice of kernel is one that not only reflects our prior information but also considers the current labeled training data.

Lanckriet, et al. in 2004 proposed a semi-definite programming (SDP) formulation to optimize kernel matrices for the binary SVM by applying duality [LCB04]. The detail is omitted since our derivation is similar but works for a more general problem.

It is not trivial to extend the technique to the multi-class SVM transduction problems. The main obstacle is the big $N_{\mathrm{tr}}k \times N_{\mathrm{tr}}k$ matrix in the objective, where N_{tr} is the number of training samples and k is the number classes.

In this chapter, we decompose the SDP problem such that the size of each linear matrix inequality (LMI) in the resulting problem is $(N_{\mathrm{tr}}+1)$-by-$(N_{\mathrm{tr}}+1)$ that is the same as that in the binary problem and there are only k such LMI constraints.

We also present two distinct applications. One is for handwritten digits classification. In this application, we apply the proposed method to select the best kernel matrix from a set of given matrices. The other application is the retina neuron signal source classification problem. The second application is used to demonstrate the ability and the advantage of combining heterogeneous features

by using the proposed method.

2.2 Formulations for Kernel Optimization

We start from Crammer and Singer's simplified version of one-against-other fixed kernel version [CS01] and derive our kernel learning formulation.

Let $\omega(K)$ denote the optimal value of the problem (2.3) for a given K. If the kernel matrix is unknown, we are trying to solve the following problem

$$\begin{array}{ll} \text{maximize} & \omega(K) \\ \text{subject to} & \mathbf{Tr}(K) = c \\ & K \succeq 0, \end{array} \tag{2.5}$$

where we add a normalization constraint $\mathbf{Tr}(K) = c$ for variable K.

Now we show that the multi-class SVM transduction problem can be formulated as an SDP problem if the underlying kernel matrix K is unknown.

Theorem 1 *The problem (2.5) is equivalent to the following SDP problem (2.6):*

$$\begin{array}{ll} \mathit{minimize}_{t,\nu,\lambda,K} & t \\ \mathit{subject\ to} & \begin{pmatrix} K_{\mathrm{tr}} \otimes I & d - \nu + B^T\lambda \\ (d - \nu + B^T\lambda)^T & t - 2\gamma d^T\nu \end{pmatrix} \succeq 0 \\ & K \succeq 0 \\ & \mathbf{Tr}(K) = c \\ & \nu \geq 0 \end{array} \tag{2.6}$$

where the identity matrix $I \in \mathbf{R}^{N_{\mathrm{tr}} \times N_{\mathrm{tr}}}$*, the constant matrix* $B \in \mathbf{R}^{N \times kN_{\mathrm{tr}}}$*,*

$$B = I \otimes \mathbf{1}^T,$$

and the unit vector $\mathbf{1} \in \mathbf{R}^k$*.*

Proof. Let $\omega(K)$ denote the optimal value of the problem (2.3) for a given K. We formulate $\omega(K)$ as an expression of K by applying the duality. The problem of maximizing $\omega(K)$ is then formulated as an SDP problem.

For any fixed kernel matrix K that satisfies the constraints $\mathbf{Tr}(K) = c$ and $K \succeq 0$, define the Lagrangian of problem (2.3) with α as variables by

$$L(\alpha, \nu, \lambda) = \alpha^T (K_{\mathrm{tr}} \otimes I)\alpha - 2d^T\alpha - 2\lambda^T B\alpha + 2\nu^T(\alpha - \gamma d),$$

where $\lambda \in \mathbf{R}^N$ and $\nu \in \mathbf{R}^{kN_{\mathrm{tr}}}$. By duality, the optimal objective value $\omega(K)$ of the problem (2.3) satisfies

$$\omega(K) = \max_{\nu \geq 0, \lambda} \min_{\alpha} L(\alpha, \nu, \lambda).$$

By setting $\frac{\partial L}{\partial \alpha} = 0$,

$$\alpha^* = (K_{\mathrm{tr}} \otimes I)^{-1}(d - \nu + B^T\lambda),$$

Note that $K \succeq 0$ infers $K \otimes I \succeq 0$. By substituting the optimal value of α into the Lagrangian, we have

$$\omega(K) = \max_{\nu \geq 0, \lambda} \{-(d - \nu + B^T\lambda)^T (K_{\mathrm{tr}} \otimes I)^{-1}(d - \nu + B^T\lambda) - 2\gamma d^T\nu\}.$$

Define $-t$ to be the lower bound of $\omega(K)$ over K. The problem

$$\begin{aligned} &\text{maximize} && \omega(K) \\ &\text{subject to} && K \succeq 0, \mathbf{Tr}(K) = c, \mu \geq 0 \end{aligned}$$

is equivalent to

$$\begin{aligned} &\text{maximize}_{t,\nu,\lambda,K} && -t \\ &\text{subject to} && -(d - \nu + B^T\lambda)^T (K_{\mathrm{tr}} \otimes I)^{-1}(d - \nu + B^T\lambda) - 2\gamma d^T\nu \geq -t \\ & && K \succeq 0 \\ & && \mathbf{Tr}(K) = c \\ & && \nu \geq 0. \end{aligned} \tag{2.7}$$

The result follows immediately after applying Schur complement to the first constraint.

2.2.1 A Decomposed Formulation

One may note that in problem (2.6), the linear matrix inequality (LMI) of the first constraint has dimension $(kN_{\rm tr}+1)$-by-$(kN_{\rm tr}+1)$. We now show that this constraint can be decomposed into k constraints of $(N_{\rm tr}+1)$-by-$(N_{\rm tr}+1)$ dimension.

Theorem 2 *The problem (2.6) is equivalent to the following SDP problem (2.8):*

$$\begin{array}{ll} \mathit{minimize}_{s,\nu,\lambda,K} & \mathbf{1}^T s \\ \mathit{subject\ to} & \left(\begin{array}{ll} K_{\rm tr} & d^l - \nu^l + \lambda \\ (d^l - \nu + \lambda)^T & s_l - 2\gamma (d^l)^T \nu^l \end{array} \right) \succeq 0, l = 1, \ldots, k \\ & K \succeq 0 \\ & \mathbf{Tr}(K) = c \\ & \nu^l \geq 0, l = 1, \ldots, k \end{array} \tag{2.8}$$

where $s \in \mathbf{R}^k$, of which the l-th component is denoted by s_l, $\nu^l = [\nu_1^l \nu_2^l \ldots \nu_{N_{\rm tr}}^l]^T$ and constant vector $d^l \in \mathbf{R}^{N_{\rm tr}}$ and the i-th component of vector d^l is $\delta_l(y_i)$.

Proof. We start from the first constraint of the problem (2.7):

$$(d - \nu + B^T\lambda)^T (K_{\rm tr} \otimes I)^{-1} (d - \nu + B^T\lambda) + 2\gamma d^T \nu \leq t.$$

Note that $(K_{\rm tr} \otimes I)^{-1} = K_{\rm tr}^{-1} \otimes I$. Let $(K_{\rm tr}^{-1})_{ij}$ denote ij elment of the matrix $K_{\rm tr}^{-1}$. We have

$$\begin{array}{ll} & (d - \nu + B^T\lambda)^T (K_{\rm tr} \otimes I)^{-1} (d - \nu + B^T\lambda) + 2\gamma d^T \nu \\ = & (d - \nu + B^T\lambda)^T (K_{\rm tr}^{-1} \otimes I) (d - \nu + B^T\lambda) + 2\gamma d^T \nu \\ = & \sum_{l=1}^k \sum_{i=1}^{N_{\rm tr}} \sum_{j=1}^{N_{\rm tr}} ((d_i^l - \nu_i^l + (B^l)^T\lambda_i)(d_j^l - \nu_j^l + (B^l)^T\lambda_j)(K_{\rm tr}^{-1})_{ij} + 2\gamma d_i^l \nu_i^l) \\ = & \sum_{l=1}^k ((d^l - \nu^l + (B^l)^T\lambda)^T K_{\rm tr}^{-1} (d^l - \nu^l + (B^l)^T\lambda) + 2\gamma (d^l)^T \nu^l) \end{array} \tag{2.9}$$

where $B^l \in \mathbf{R}^{N \times N_{\text{tr}}}$ is a constant matrix such that the i-th element of B^l is the $((i-1)k+l)$-th element of B. One may immediately note that $B^l = I, \forall l$. Therefore, the constraint $\sum_{l=1}^{k}(d^l - \nu^l + (B^l)^T\lambda)^T K_{\text{tr}}^{-1}(d^l - \nu^l + (B^l)^T\lambda) + 2\gamma(d^l)^T\nu^l \leq t$ holds if and only if

$$(d^l - \nu^l + \lambda)^T K_{\text{tr}}^{-1}(d^l - \nu^l + \lambda) + 2\gamma(d^l)^T\nu^l \leq s_l, \forall l \tag{2.10}$$

and

$$\sum_{l=1}^{k} s_l \leq t.$$

The result follows after we apply Schur complementary lemma to each of the inequalities of (2.10).

Throughout this chapter, the Sedumi [Stu99] Matlab toolbox is used to solve SDP problems and the MOSEK toolbox [AA00] is used to solve standard QP problems.

2.3 Application: Handwritten Digits Classfication

An immediate application of the proposed multi-class SVM kernel learning algorithm is the kernel selection. We applied this algorithm to the MNIST handwritten digits database [LBB98] for digits classificcation. The MNIST database of handwritten digits has a training set of 60,000 vectors. The digits have been size-normalized and centered in a fixed-size image. This database is a standard database online for benchmarking classification algorithms.

All the digits images are centered in a 28×28 image. A set of random samples of such images are given in Figure 2.2. We vectorize each image to a 781 vector. The problem is to classify all those 10 digits.

Assuming that there is no prior information to select kernels, we choose four

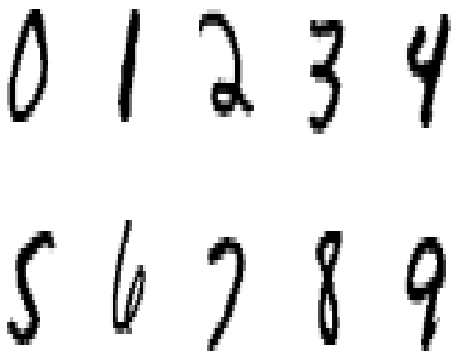

Figure 2.2: A set of random samples of the MNIST handwritten digits database.

candidate kernels: linear kernel

$$K_1(i,j) = x_i^T x_j,$$

quadratic kernel

$$K_2(i,j) = (1 + x_i^T x_j)^2,$$

Gaussian kernel

$$K_3(i,j) = e^{(x_1-x_2)^T(x_i-x_j)/(2\sigma^2)},$$

and sigmoid kernel

$$K_4(i,j) = \tanh(\kappa x_i^T x_j + \theta),$$

where $K_l(i,j)$ denotes the i-th row and j-th component of matrix K_l for all l. We fix the kernel parameter $\sigma = 0.5$, $\kappa = 1$ and $\theta = 0$ and the regularization parameter $\gamma = 16$ for simplicity. We tried all linear combinations of the four candidate kernels on a randomly selected set of 200 training samples and 100 testing samples. There are totally one four-kernel cases, four three-kernel cases, six two-kernel cases and four single-kernel cases. We did the same experiment for 100 times and present the average result in Table 2.1, where $\overline{\mathrm{ERR}_{\mathrm{te}}}$ denotes the average testing error rate.

Kernel	μ_1	μ_2	μ_3	μ_4	$\overline{\mathrm{ERR}_{\mathrm{te}}}$
$\mu_1 K_1 + \mu_2 K_2 + \mu_3 K_3 + \mu_4 K_4$	0.0939	-0.1986	1.5721	-0.4673	13.5%
$\mu_1 K_1 + \mu_2 K_2 + \mu_3 K_3$	-0.1769	-0.2336	1.4105	\	13.9%
$\mu_1 K_1 + \mu_2 K_2 + \mu_4 K_4$	-0.2249	0.0229	\	1.2020	22.0%
$\mu_1 K_1 + \mu_3 K_3 + \mu_4 K_4$	-0.1677	\	1.6535	-0.4858	13.4%
$\mu_2 K_2 + \mu_3 K_3 + \mu_4 K_4$	\	-0.1383	1.6070	-0.4687	14.0%
$\mu_1 K_1 + \mu_2 K_2$	1.0116	-0.0116	\	\	20.4%
$\mu_1 K_1 + \mu_3 K_3$	-0.4927	\	1.4927	\	14.8%
$\mu_1 K_1 + \mu_4 K_4$	0.0000	\	\	1.0000	22.0%
$\mu_2 K_2 + \mu_3 K_3$	\	-0.3345	1.3345	\	15.0%
$\mu_2 K_2 + \mu_4 K_4$	\	-0.0000	\	1.0000	23.1%
$\mu_3 K_3 + \mu_4 K_4$	\	\	1.4369	-0.4369	14.8%
K_1	\	\	\	\	20.6%
K_2	\	\	\	\	16.8%
K_3	\	\	\	\	14.7%
K_4	\	\	\	\	22.4%

Table 2.1: Kernel selection and the testing error for the MNIST Database. The result shows that the Gaussian kernel K_3 always has a significantly large positive coefficient. If K_3 is involved in kernel combinations, the testing errors are low.

The results in Table 2.1 show that the testing errors can be dramatically improved by including the candidate kernel matrix with large positive coefficient. That is to say, the SDP kernel optimization method may select the best kernel algorithmically.

2.4 Application: Retinal Ganglion Cells Classification

The proposed kernel learning method in multi-class SVM training problems has another important application: classification with heterogeneous features. The traditional fixed kernel method either assumes that all components in a feature vector are equally important or assigns a pre-defined weight to different components before applying kernel functions. For details, one may refer to the reference regarding locality-improved kernels [BB99, SSS98, BGL00]. Either way assumes that one has sufficient knowledge before the actual learning process, which is usually not true in practice. Our kernel method provides a way of combining heterogeneous features by jointly learning a set of weights. For example, suppose feature X_1 and X_2 are heterogeneous and we have no prior knowledge about their relative importance to the underlying classification problem. We may design a kernel matrix K such that $K = \mu_1 K_1 + \mu_2 K_2$, where K_1 and K_2 are proper kernels for feature X_1 and X_2. After solving the SDP problem, one gets an improved classifier as well as a naturally combined kernel matrix K. This method has been successfully applied in the mouse retina ganglion cell identification problem.

2.4.1 The Problem

The data set we analyze consists of signals from 17 retina ganglion cells from a mouse, recorded from a multi-electrode array. The visual stimulus is an 1.5 hour

long spatially uniform flicker, in which intensity values (ranging from 0 to 255) are drawn randomly from a Gaussian distribution with mean 127 and standard deviation 54 every 100 ms. There are 53,460 frames (1 frame = 0.1s). The response of the 17 cells are 17 spike trains. The firing rate ranges from 4.2 to 77.4 Hz.

Zhong, et al., classified the 17 neurons into 3 classes: Fast-ON, Slow-ON and OFF [ZBJ05] based on the mutual information of the stimulus and the neurons' response. We further discovered that there exists significant drifting in a neuron's response to the stimulus intensity level over time, based on SVM regression analysis with kernel learning technique (see Section 3.4 of Chapter 3). The drifting analysis suggests that shorter time series, say, 500 frames, shall be used to capture a neurons' behavior. An interesting question arises: given a segment of the spike train, can we classify it into one of the three classes? A special case is that, if one neuron from each classes is selected, can we identify which neuron it is based on its spike train?

We formally state our problem as follows. There are 315 (105×3) segments of spike train. Each segment contains 500 input symbols and 500 output symbols. The 315 pieces of signals are drawn from 3 neurons responses to 105 frames of input symbols, the intensities of the input image. Among the 315 samples, 180 samples are labeled and the rest 135 samples are not labeled. By "labeled", we mean the neuron that generates the spike train is given. The problem is to label the 135 unlabeled samples based on the training on the labeled 180 samples.

To further simplify the problem, we quantize the input image intensity into 3 bits. That is there are 8 different input symbols. The output symbols are the number of spikes in each frame.

Two heterogeneous features are extracted from the data set: one is the condi-

tional energy transition probabilities; and the other is the frequencies of the code words.

2.4.2 Conditional Transition Probability Matrices

By using the term "conditional energy transition probabilities", we assume that a neuron's firing behavior is based on its energy level, which depends on the previous energy level and the current condition (the input symbol). The energy level is only a logical concept and has nothing to do with real physical energy.

We first model a dynamic Bayesian network and then use each sample to train this network by the EM algorithm [DLR77]. The advantage of training the dynamic network separately using each sample piece of the spike train rather than the whole spike train is that the initial distribution π may neutralize the effect the drifting so that learned conditional transition probabilities are more accurate.

The dynamic Bayesian network is indeed a simplified input output hidden Markov model (IOHMM) without the output subnetwork [BF95]. We may also consider the model as an augmented hidden Markov model (HMM), since the only difference between the proposed model and an HMM is that the former one has n_{in} conditional transition matrices, where n_{in} is number of different input symbols, while a typical HMM has one transition matrix. We assume that there are $n_{\text{state}} = 10$ energy levels. That is, there are n_{state} hidden states, denoted by $s_i, i = 1, ..., n_{\text{state}}$. At level 0, the probability that there is no firing in the next frame is equal to 1. At level i, the number of spikes in the next frame follows a Poisson distribution with firing rate $\lambda_i = 2^{i-1}$. Since the observation o is discrete and finite and we know that $o_t \in \{0, 1, 2, ..., n_{\text{out}} - 1\}, t = 1, ..., T$ where $n_{\text{out}} = 40$

and $T = 500$. We normalize the observation probabilities such that

$$\sum_{o=0}^{n_{\text{out}}-1} \mathbf{Prob}(o|s) = 1.$$

The output probability matrix is denoted by $B \in \mathbf{R}^{n_{\text{state}} \times n_{\text{out}}}$, with component $B(o, s) = \mathbf{Prob}(o|s)$. By definition,

$$B(o,s) = \begin{cases} 0 & \text{if } s = 0 \\ \frac{(2^{s-1})^o e^{-2^{s-1}}/o!}{\sum_{o=0}^{n_{\text{out}}-1}(2^{s-1})^o e^{-2^{s-1}}/o!} & \text{otherwise.} \end{cases}$$

We always fix the pre-defined matrix B.

The conditional state transition matrix is denoted by $A_x \in \mathbf{R}^{n_{\text{state}} \times n_{\text{state}}}$ with component $A_x(s_i, s_j) = \mathbf{Prob}(s_j|s_i, x)$, where $x \in \{0, 2, ..., n_{\text{in}} - 1\}$ is an input symbol.

The training problem is a classic HMM training problem. That is, to maximize $\mathbf{Prob}(\mathcal{Y}, \mathcal{X}|\lambda)$ over the parameter $\lambda = (A_x, x = 0, ..., n_{\text{in}} - 1, B, \pi)$, where $\mathcal{X}$ and $\mathcal{Y}$ denotes a specific input and output sequence. This is a non-convex programming problem and a local optimum can be found by the EM algorithm. The Expectation-Maximization steps can be implemented as follows.

Expectation: To estimate the distribution $w_t(i, j) = \mathbf{Prob}(s_t = j|s_{t-1} = i)$. We have

$$\zeta_t(i,j) = \frac{\alpha_t(i) A_{x_t}(i,j) B(j, y_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{n_{\text{state}}} \sum_{j=1}^{n_{\text{state}}} \alpha_t(i) A_{x_t}(i,j) B(j, y_{t+1}) \beta_{t+1}(j)}$$

where $\alpha_{t+1}(i)$ and $\beta_t(i)$ can be calculated by forward backward induction as follows:

$$\alpha_{t+1}(j) = [\sum_{i=1}^{n_{\text{state}}} \alpha_t(i) A_{x_t}(i,j)] B(j, y_{t+1})$$

with initialization

$$\alpha_1(j) = \pi_j B(j, y_1), \forall j$$

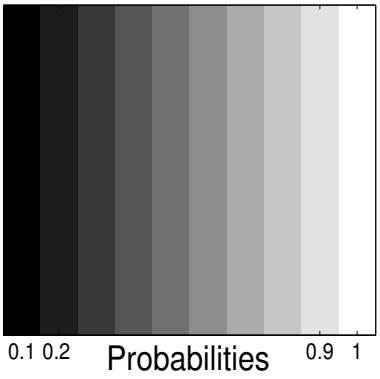

Figure 2.3: Gray level indices of probabilities. The transition probability is indexed by intensity level. The highest intensity denotes probability 1 and lowest the intensity level denotes probability 0. This is one way of feature visualization.

and

$$\beta_t(i) = \sum_{j=1}^{n_{\text{state}}} A_{x_t}(i,j) B(j, y_{t+1}) \beta_{t+1}(j)$$

with initialization

$$\beta_T(i) = 1, \forall i.$$

Maximization: To maximize the likelihood over the parameters $A_x, \forall x$ and π as follows:

$$\pi_i = \frac{\zeta_1(i,j)}{\sum_{j=1}^{n_{\text{state}}} \zeta_1(i,j)}$$

and

$$A_k(i,j) = \frac{\sum_{t:x_t=k} \zeta_t(i,j)}{\sum_{t:x_t=k} \sum_{j=1}^{n_{\text{state}}} \zeta_t(i,j)}.$$

We repeat the EM algorithm 10 times starting from randomized starting points and record the one with the largest log-likelihood to avoid the worst local optimum. So, we solve 3150 augmented HMM training problem totally. The learned conditional transition matrix serves as a feature in our later analysis.

To visualize the features, we plot the average conditional transition matrices of the three neurons. We use grey level to index the probabilities. That is, the highest intensity denotes probability 1 and the lowest intensity denotes probability 0. The grey level indices of probabilities are given in Figure 2.4.2. The three neurons' average conditional transition matrices are given in Figure 2.4-2.6. According to Zhong et al. [ZBJ05], neuron 6 belongs to the Fast-ON class, neuron 9 belongs to the OFF class and neuron 16 belongs to the Slow-ON class. Our conditional matrix further confirms Zhong's conclusion and obviously provides more information:

> Neuron 6: When the input image has intensity lower than level 4, the neuron's energy states tend to go down. That is, the firing rate is going down. Otherwise, the energy level jump up 1 or 2 steps with high probability.
>
> Neuron 9: Whatever the input intensity is, the energy level always goes to level 2 with very high probability. That is to say, the neuron tends to fire at a consistent rate.
>
> Neuron 16: When input intensity is low, the energy level tends to maintain the same up to level 3. When input intensity is high, the energy level tends to maintain the same up to level 5. There is no jump-up with very high probability observed as in Figure 2.6. This shows why neuron 16 is classified as Slow-ON.

The conditional transition matrix is an important feature for neuron classification, which characterizes the dynamics of neurons under different stimuli. Let matrix $X_1 \in \mathbf{R}^{N \times m_1}$ denote the feature data, where N is number of samples, and $m_1 = n_{\text{in}} n_{\text{state}}^2 = 800$ is the dimensionality of this feature. Each row of X_1 stores n_{in} transition probability matrices row by row.

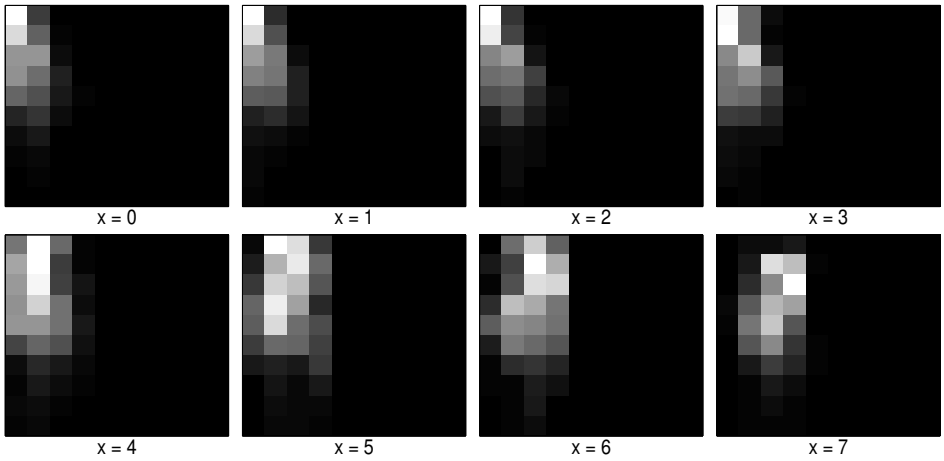

Figure 2.4: The average conditional transition probabilities gray index map of neuron 6. Each image is corresponding to the average transition probability matrix $\frac{1}{N_6}\sum_{i=1}^{N_6} A_x^i, \forall x$, where N_6 is the number of training samples for the neuron 6. One may note that when $x \leq 3$, the neuron tends to jump down to state 1; when $x = 4, 5$, the neuron tends to remain or jump up to state 2 and when $x = 6, 7$, the neuron tends to jump up to state 3 or 4

.

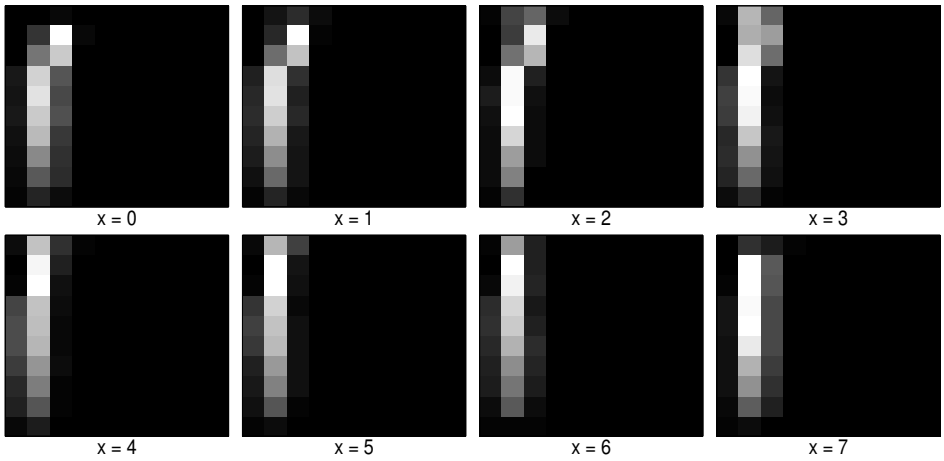

Figure 2.5: The average conditional transition probabilities gray index map of neuron 9. Each image is corresponding to the average transition probability matrix $\frac{1}{N_9}\sum_{i=1}^{N_9} A_x^i, \forall x$, where N_9 is the number of training samples for the neuron 9. One may note that the neuron tends to jump to state 2 regardless of the input symbol x.

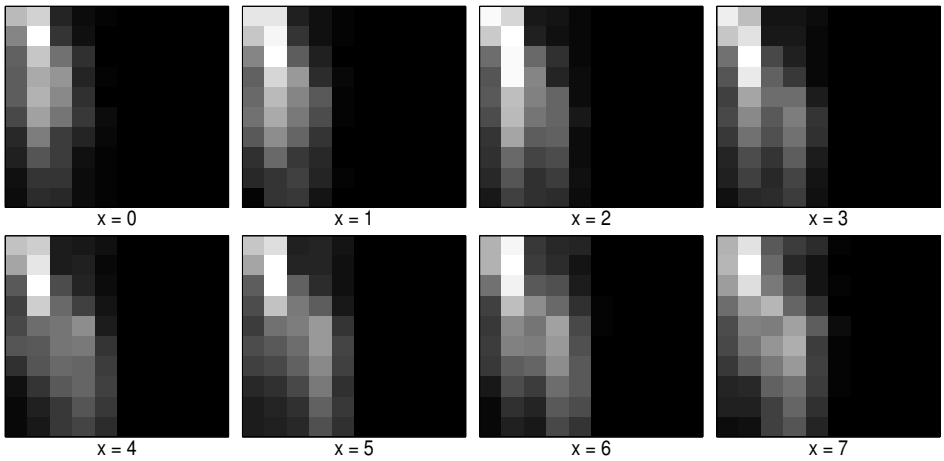

Figure 2.6: The average conditional transition probabilities gray index map of neuron 16. Each image is corresponding to the average transition probability matrix $\frac{1}{N_{16}} \sum_{i=1}^{N_{16}} A_x^i, \forall x$, where N_{16} is the number of training samples for the Neuron 16. One may note that the Neuron 16 has a broader distribution of transition probabilities that means it may go up to high energy state with relatively high probability. There is no jump-up with very high probability observed.

2.4.3 Frequency of Code Words

Recall that there are $n_{\text{out}} = 40$ different output symbols per frame, corresponding to different number of spikes in a frame. We say that the neuron signal has 40 codes from 0 to 39. Since the input stimuli are IID and the three neurons receive the same stimulus, the difference of the frequency of code words among the three neurons is only due to neuron's intrinsic difference. By using the frequency of code words, we ignore the order of the codes in a sample sequence. This feature emphasizes the pure firing rate.

Let $X_2 \in \mathbf{R}^{N \times n_{\text{out}}}$ denote the feature data extracted from frequency of codes. So, $X_2(i, j)$ denotes the number of frames where the neuron fires $j - 1$ spikes in i-th sample sequence.

2.4.4 Combining Features

The proposed kernel learning method enables us to combine the two features. We use a linear kernel for both features since the sample size is limited. That is

$$K_1 = X_1(X_1)^T$$

and

$$K_2 = X_2(X_2)^T.$$

A linear combination is used to combine the two features such that

$$K = \mu_1 K_1 + \mu_2 K_2.$$

We compare the testing error of the optimal kernel K and those of fixed kernel K_1 and K_2 in cross validation. We have 315 samples. Each time we randomly select 180 samples as training samples and 135 samples as testing samples. We repeat the procedure 100 times. We also vectorize the two feature sets X_1 and X_2

Kernel	$\overline{\mathrm{ERR}_{\mathrm{te}}}$	$\sigma_{\mathrm{ERR}_{\mathrm{te}}}$
$0.863K_1 + 0.137K_2$	0.082%	0.0023
K_1	0.311%	0.0037
K_2	2.948%	0.0107
K_{vec}	2.897%	0.0105

Table 2.2: Comparison of the performance between the optimal kernel, fixed kernels of homogeneous features and a kernel of vectorized features, where $\overline{\mathrm{ERR}_{\mathrm{te}}}$ is the average testing error and $\sigma_{\mathrm{ERR}_{\mathrm{te}}}$ is the standard deviation of testing errors among 100 experiments. The result from the best individual feature kernel K_1 can be considered as the result of voting since there are only two candidates. The kernel K_{vec} from the vectorized feature has very bad result that is close to that of the worst kernel K_2. The optimal combined kernel achieves the best testing error.

into a one matrix $X_{\mathrm{vec}} \in \mathbf{R}^{N\times(m_1+n_{\mathrm{out}})}$ after normalization. The corresponding linear kernel

$$K_{\mathrm{vec}} = X_{\mathrm{vec}}(X_{\mathrm{vec}})^T.$$

The average performance of the optimal kernel, the two fixed individual kernels and the kernel of the vectorized feature is summarized in Table 2.2. The results show that learning a kernel that combines heterogeneous features improves the classification accuracy significantly. Since there are only two candidates, the result from the best fixed kernel of individual feature can be considered the result of a voting process. This example clearly demonstrates the weakness of the voting method: it ignores the feature with higher testing error. The vectorized feature is even worse since the bad feature dominates the result and the good feature X_1 is wasted.

2.5 Conclusion and Discussion

In this chapter, we proposed a method to optimize the kernel matrix jointly with the classifier in a multi-class SVM training problem. We also presented a decomposition method. After decomposition, the proposed SDP problem has only $k-1$ more LMIs than similar formulation proposed by Lanckriet [LCB04] for binary SVM transduction problems. This result shows that the kernel optimization method for multi-class SVM training problems can be used as easily as for binary SVM problems.

The kernel learning method has been applied to two applications: kernel selection and combining heterogeneous features. Applications in real-world database show that the proposed method may have significant advantages in improving the quality of the resulting classifier .

The kernel selection method has been tested in the MNIST database. An obvious result is that the proposed kernel learning method may select the best multi-class kernel or kernel combination algorithmically.

The application on combining heterogeneous feature for the neuron identification and classification problem shows that the proposed method may significantly increase the classification accuracy. This method is superior to the widely used voting algorithm and the intuitive feature vectorization approach.

CHAPTER 3

Kernel Optimization for Regression Estimation

3.1 Introduction

Regression estimation is a very popular method in pattern recognition and signal processing. The basic idea is to minimize a cost function over the parameters of the regression function. The choice of the cost function is application dependent. For simplicity, in our research, we take the most commonly used ϵ-insensitive cost function [Vap95]:

$$|y - f(x)|_\epsilon \equiv \max\{0, |y - f(x)| - \epsilon\}. \tag{3.1}$$

The primal support vector machine regression problem can be considered a linear regression problem. To estimate a linear regressor

$$f(x) = w^T x + b, \tag{3.2}$$

one minimizes the empirical error with a regularization term

$$\frac{1}{2} w^T w + \gamma \sum_{i=1}^{N_{\text{tr}}} |y_i - f(x_i)|_\epsilon, \tag{3.3}$$

where $w \in \mathbf{R}^m$, $b \in \mathbf{R}$, $f(x) : \mathbf{R}^m \to \mathbf{R}$, N_{tr} denotes the number of training vectors and γ is an experimentally selected parameter. This problem can be

re-written in a quadratic programming formulation:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2} w^T w + \gamma \mathbf{1}^T (\xi + \xi^*) \\ \text{subject to} \quad & w^T x_i + b - y_i \leq \epsilon + \xi_i, i = 1, ..., N_{\text{tr}} \\ & y_i - w^T x_i - b \leq \epsilon + \xi_i^*, i = 1, ..., N_{\text{tr}} \\ & \xi, \xi^* \geq 0, \end{aligned} \tag{3.4}$$

where $\xi \in \mathbf{R}^{N_{\text{tr}}}$ and $\xi^* \in \mathbf{R}^{N_{\text{tr}}}$ are the allowed errors "above" and "below" the margin boundary.

The dual problem of problem (3.4) is

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2} (\alpha^* - \alpha)^T K_{\text{tr}} (\alpha^* - \alpha) - y^T (\alpha^* - \alpha) + \epsilon \mathbf{1}^T (\alpha^* + \alpha) \\ \text{subject to} \quad & \mathbf{1}^T (\alpha^* - \alpha) = 0 \\ & 0 \leq \alpha^*, \alpha \leq \gamma \mathbf{1}, \end{aligned} \tag{3.5}$$

where the dual variables $\alpha^*, \alpha \in \mathbf{R}_{\text{tr}}^N$, the training data $X_{\text{tr}} \in \mathbf{R}^{N_{\text{tr}} \times m_{\text{tr}}}$, each row of X_{tr} is a data vector $x_i \in \mathbf{R}^m$, the kernel matrix $K_{\text{tr}} \in \mathbf{R}^{N_{\text{tr}} \times N_{\text{tr}}}$ and its ij element $K_{\text{tr}}(i,j) = x_i^T x_j$.

We have the following complementary slackness conditions between optimal primal and dual solutions:

$$\begin{aligned} \alpha_i (\epsilon + \xi_i - y_i + w^T x_i + b) = 0, \\ \alpha_i^* (\epsilon + \xi_i + y_i - w^T x_i - b) = 0, \end{aligned} \tag{3.6}$$

and

$$(\gamma - \alpha_i) \xi_i = 0, \quad (\gamma - \alpha_i^*) \xi_i^* = 0. \tag{3.7}$$

3.1.1 Kernels

Using the kernel trick in support vector classification problems, we may lift the dimension of a feature vector x to higher dimension via a function $\phi(x) : \mathbf{R}^m \rightarrow$

$\mathbf{R}^n$. Define kernel function $F(x, \tilde{x}) = \phi(x)\phi(\tilde{x})$. We have the kernel matrix

$$K(i, j) = F(x_i, x_j),$$

where $K(i, j)$ denotes the ij element of the matrix K. The regressor has the form

$$f(x) = \sum_{i=1}^{N_{\text{tr}}} (\alpha_i - \alpha_i^*) F(x_i, x) + b.$$

That is to say, via the kernel trick, we may estimate a general nonlinear regressor. The definition of the kernel function $F(x, \tilde{x})$ also determines the regressor model $f(x)$.

Traditional support vector regression (SVR) assumes the kernel is predefined and fixed. Researches have been focusing on how to select the kernel matrix and determine the parameters. One approach is to use a heuristic selection of parameters [SMS99, MH99, Kwo01, CM04]. Multiple parameters can also be used to improve the performance but the SVM training algorithm needs to be executed multiple times for every selected parameter setting [CVB02]. These references provide general practical guidelines for parameter selection but the selection is far from the optimum and may not be suitable for specific applications. Another approach is to use meta-learning method to determine kernel and parameters [KBS02, CSB04]. The idea is to learn a set of data characteristics measurements with multiple settings of parameters and to construct a relationship between data sets and parameters by using statistics tools, say K-mean clustering. This method requires much additional information, namely, meta-data, whose definition however remains vague. That is, one may encounter the problem of selecting meta-data instead of selecting parameters.

The selection of the kernel function F is equivalent to regressor modeling and depends on the prior information of the problem we are solving. When the prior information is vague, a wrong model may result in large testing error.

3.1.2 Transduction Problem

A transduction problem is a problem of evaluating the function value on a set of testing data from a set of training data, for which the function value is given. That is, in a transduction problem, we do not intend to learn a general regressor. Instead, we are only interested in evaluating the function value on the testing data points.

For a regression problem, there are N data points, including $N_{\rm tr}$ training data and $N_{\rm t}$ testing data. Throughout this chapter, we use $X_{\rm tr} \in \mathbf{R}^{N_{\rm tr} \times m}$ to denote the labeled training set and $X_{\rm t} \in \mathbf{R}^{N_{\rm t} \times m}$ to denote the unlabeled testing set, where each row of the matrix $X_{\cdot}$ is a vector $x_i \in \mathbf{R}^m$. The given function value is stored in vector $y \in \mathbf{R}^{N_{\rm tr}}$. The kernel matrix

$$K = \begin{pmatrix} K_{\rm tr} & K_{\rm tr,t} \\ K_{\rm tr,t}^T & K_{\rm t} \end{pmatrix}, \tag{3.8}$$

where the block matrixes $K_{\rm tr}$, $K_{\rm tr,t}$ and $K_{\rm t}$ are the optimal training, mixed and testing block that we want to learn. Since the objective is only to complete the label of a finite testing set, there is no need to learn the general kernel function. Instead, we only learn the kernel matrix K.

The regression estimate $f : \mathbf{R}^{N_{\rm t} \times m} \to \mathbf{R}^{N_{\rm t}}$ for testing sample $X_{\rm t}$ takes the following form

$$f(X_{\rm t}) = (\alpha^* - \alpha)^T K_{\rm tr,t} + b\mathbf{1}. \tag{3.9}$$

By the complementary slackness conditions (3.6), b is

$$\begin{aligned} b &= y_i + \epsilon - (\alpha^* - \alpha)^T K_{\rm tr}(., i) \quad \text{if } 0 < \alpha_i < \gamma \\ b &= y_i - \epsilon - (\alpha^* - \alpha)^T K_{\rm tr}(., i) \quad \text{if } 0 < \alpha_i^* < \gamma, \end{aligned} \tag{3.10}$$

where $K_{\rm tr}(., i)$ denotes the i-th column of the matrix $K_{\rm tr}$. A useful observation is that $\alpha_i^* \alpha_i = 0$, since the corresponding constraints in problem (3.5) cannot be active at the same time.

In the chapter, we propose a novel method that is able to optimize the kernel matrix from the training samples. Our method can incorporate a broader family of regressor models, and algorithmically select a linear combination of them.

3.2 Kernel Optimization

Many techniques applied in support vector classification (SVC) can be used in support vector regression (SVR) because of the similarity of the underlying optimization problems. We extend Lanckriet's semi-definite programming (SDP) formulation for binary classification problem [LCB04] to support vector regression training problems.

Let $\omega(K)$ denote the optimal value of the problem (3.5) for a given K. If the kernel matrix is unknown, we are trying to maximize $\omega(K)$ over K:

$$\begin{array}{ll} \text{maximize} & \omega(K) \\ \text{subject to} & \mathbf{Tr}(K) = s \\ & K \succeq 0, \end{array} \tag{3.11}$$

where we add a normalization constraint $\mathbf{Tr}(K) = s$.

Now we show that the SVM regression estimation problem can be formulated as an SDP problem if the underlying kernel matrix K is unknown.

Theorem 3 *The problem (3.11) is equivalent to the following SDP problem (3.12):*

$$
\begin{array}{ll}
\textit{minimize} & t \\
\textit{subject to} & \left(\begin{array}{ll} K_{\mathrm{tr}} & y + \lambda \mathbf{1} + \nu - \upsilon - \delta + \eta \\ (y + \lambda \mathbf{1} + \nu - \upsilon - \delta + \eta)^T & t - 4\gamma \mathbf{1}^T(\delta + \eta) \end{array} \right) \succeq 0 \\
& -\epsilon \mathbf{1} + \nu + \upsilon - \delta - \eta = 0 \\
& K \succeq 0 \\
& \mathbf{Tr}(K) = s \\
& \nu, \upsilon, \delta, \eta \geq 0
\end{array}
\tag{3.12}
$$

where $t, \lambda, \nu, \upsilon, \delta, \eta, K$ *are variables,* $\lambda \in \mathbf{R}$ *and* $\nu, \upsilon, \delta, \eta \in \mathbf{R}^{N_{\mathrm{tr}}}$.

Proof. Let $\omega(K)$ denote the optimal value of the problem (3.5) as a function of K. We formulate $\omega(K)$ as an expression of K by applying duality. The $\omega(K)$ is then maximized over K using an SDP problem.

Define $\alpha^+ = \alpha^* + \alpha$ and $\alpha^- = \alpha^* - \alpha$. The problem (3.5) is equivalent to the following problem (3.13).

$$
\begin{array}{ll}
\mathrm{minimize}_{\alpha} & \frac{1}{2}\alpha^{-T} K_{\mathrm{tr}} \alpha^- - y^T \alpha^- + \epsilon \mathbf{1}^T \alpha^+ \\
\mathrm{subject\ to} & \mathbf{1}^T \alpha^- = 0 \\
& 0 \leq \alpha^+ + \alpha^- \leq 2\gamma \mathbf{1} \\
& 0 \leq \alpha^+ - \alpha^- \leq 2\gamma \mathbf{1}
\end{array}
\tag{3.13}
$$

For any fixed kernel matrix K that satisfies the constraints $\mathbf{Tr}(K) = s$ and $K \succeq 0$, define the Lagrangian of problem (3.5) by

$$
\begin{array}{ll}
& L(\alpha, \nu, \upsilon, \delta, \eta, \lambda) \\
= & \alpha^{-T} K_{\mathrm{tr}} \alpha^- + 2\epsilon \mathbf{1}^T \alpha^+ - 2y^T \alpha^- - 2\lambda \mathbf{1}^T \alpha^- - 2\nu^T(\alpha^+ + \alpha^-) \\
& -2\upsilon(\alpha^+ - \alpha^-) - 2\delta^T(2\gamma \mathbf{1} - \alpha^+ - \alpha^-) - 2\eta^T(2\gamma \mathbf{1} - \alpha^+ + \alpha^-).
\end{array}
\tag{3.14}
$$

By duality, the optimal objective value $\omega(K)$ of the problem (3.13) satisfies

$$
\omega(K) = \max_{\nu, \upsilon, \delta, \eta \geq 0, \lambda} \min_{\alpha^+, \alpha^-} L(\alpha, \nu, \upsilon, \delta, \eta, \lambda).
$$

Note $\omega(K)$ is bounded as α^+ goes to infinity if and only if

$$-\epsilon\mathbf{1} + \nu + \upsilon - \delta - \eta = 0.$$

By setting $\frac{\partial L}{\partial \alpha^-} = 0$,

$$\alpha^- = K_{\text{tr}}^{-1}(y + \lambda\mathbf{1} + \nu - \upsilon - \delta + \eta).$$

By substituting the optimal value of α^- into the Lagrangian, we have

$$\omega(K) = \max_{\nu,\upsilon,\delta,\eta\geq 0,\lambda} -(y+\lambda\mathbf{1}+\nu-\upsilon-\delta+\eta)^T K_{\text{tr}}^{-1}(y+\lambda\mathbf{1}+\nu-\upsilon-\delta+\eta) - 4\gamma\mathbf{1}^T(\delta+\eta).$$

Define $-t$ to be the lower bound of $\omega(K)$ over K. The problem

$$\begin{aligned} &\text{maximize} && \omega(K) \\ &\text{subject to} && K \succ 0, \mathbf{Tr}(K) = c \end{aligned}$$

is equivalent to

$$\begin{aligned} &\text{maximize} && -t \\ &\text{subject to} && -(y + \lambda\mathbf{1} + \nu - \upsilon - \delta + \eta)^T K_{tr}^{-1}(y + \lambda\mathbf{1} + \nu - \upsilon - \delta + \eta) \\ & && -4\gamma\mathbf{1}^T(\delta + \eta) \geq -t \\ & && -\epsilon\mathbf{1} + \nu + \upsilon - \delta - \eta = 0 \\ & && K \succ 0 \\ & && \mathbf{Tr}(K) = c \\ & && \nu, \upsilon, \delta, \eta \geq 0, \end{aligned} \tag{3.15}$$

where $t, \nu, \upsilon, \delta, \eta, \lambda, K$ are variables. The theorem now follows immediately after applying Schur complement to the first constraint.□

The estimated function value on the testing points has the following form

$$(\alpha^-)^T K_{\text{tr,t}} + b\mathbf{1} \tag{3.16}$$

where b can be expressed as:

$$\begin{aligned} b &= y_i + \epsilon - (\alpha^-)^T K_{\text{tr}}(., x_i) \quad \text{if } -\gamma < \alpha^- < 0 \\ b &= y_i - \epsilon - (\alpha^-)^T K_{\text{tr}}(., x_i) \quad \text{if } 0 < \alpha^- < \gamma, \end{aligned} \tag{3.17}$$

inferred from (3.10) and the fact that at least one of α_i^* and α_i is 0.

3.3 Numerical Experiments

We use several synthesized one-dimensional examples to examine the benefits of learning a linear combination of different kernels.

Fifty training vectors and fifty testing vectors are generated from the following model

$$y(x) = g_l(x) + \eta_l(x) \tag{3.18}$$

where $x, y \in \mathbf{R}$, $\eta_l(x) \in \mathbf{R}$ is zero mean uniformly distributed additive noise in $[-1, 1]$ and $g_l(x)$ is the regressor we want to learn. We made seven examples: a linear function

$$g_1(x) = x + 1,$$

a quadratic function

$$g_3(x) = 0.2(x^2 + 1),$$

a cubic function

$$g_3(x) = 0.04(x^3 + 1),$$

a polynomial function

$$g_4(x) = x^7/15000 + x^2/5,$$

another polynomial function

$$g_5(x) = (x^3 + 1)/25 - x^7/15000,$$

a Gaussian function

$$g_6(x) = 5\exp\{-x^2\},$$

and a difference of Gaussians function

$$g_7(x) = 8\left(\exp\{-x^2\} - 0.5\exp\{-\frac{x^2}{5}\}\right).$$

We test the polynomial kernels of order 1 to 7 and their optimal linear combination. The linear kernel is denoted by

$$K_1(i,j) = x_i^T x_j + b,$$

and polynomial kernels are denoted by

$$K_d(i,j) = (1 + x_i^T x_j)^d, d = 2, \ldots, 7.$$

Their linear combination is

$$K = \sum_{d=1}^{7} \mu_d K_d.$$

The testing error e is measured by averaging the ϵ-insensitive errors. That is

$$e := \frac{1}{N_t} \sum_{i=1}^{N_t} |y_i - f(x_i)|_\epsilon. \tag{3.19}$$

In this section, we fix $\epsilon = 1$ for all problems.

We determine the parameter γ by selecting the value $\gamma = 2^p, p = -5, ..., 50$ with the lowest testing error. We only present the result for the best γ.

The Sedumi [Stu99] Matlab toolbox was used to solve SDP problems and MOSEK [AA00] was used to solve standard SVM regression problems.

The testing errors of all kernels including the fixed polynomial kernels and their optimal combination for all seven problems are listed in Table 3.1.

To further demonstrate the accuracy of regression estimation of the traditional SVM regression method and the proposed kernel learning method, we present the testing plot in Figure 3.1-3.7.

The results have a few implications:

- If one or more candidate kernels work well (see problem 2 and 3), the learned optimal linear combination kernel usually achieves the best testing result that the best candidate kernel achieves.

Prob.	e							
l	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K
1	0.020	0.020	0.066	0.092	0.092	0.092	0.092	0.067
2	1.129	0.042	0.128	0.000	0.108	0.063	0.086	0.000
3	0.327	0.328	0.022	0.042	0.042	0.042	0.042	0.000
4	1.305	0.486	0.159	0.159	0.042	0.153	0.111	0.020
5	0.188	0.177	0.188	0.136	0.098	0.107	0.044	0.042
6	0.926	0.798	0.789	0.439	0.446	0.304	0.341	0.368
7	0.836	0.808	0.840	0.757	0.755	0.577	0.559	0.572

Table 3.1: Testing error of different kernels. The testing error e is the averaged ϵ-insensitive error defined in (3.19). This table shows that the optimized kernel in general has the best performance. Higher order polynomial kernels suffer from the curse of the dimensionality (see problem 1, 2 and 3) but lower order of kernels suffer from the limited regression capability (see problem 2, 3, 4, 5, 6, and 7). The optimal kernel has a good balance of different kernels. It may achieve much better performance than fixed kernels (see problem 4).

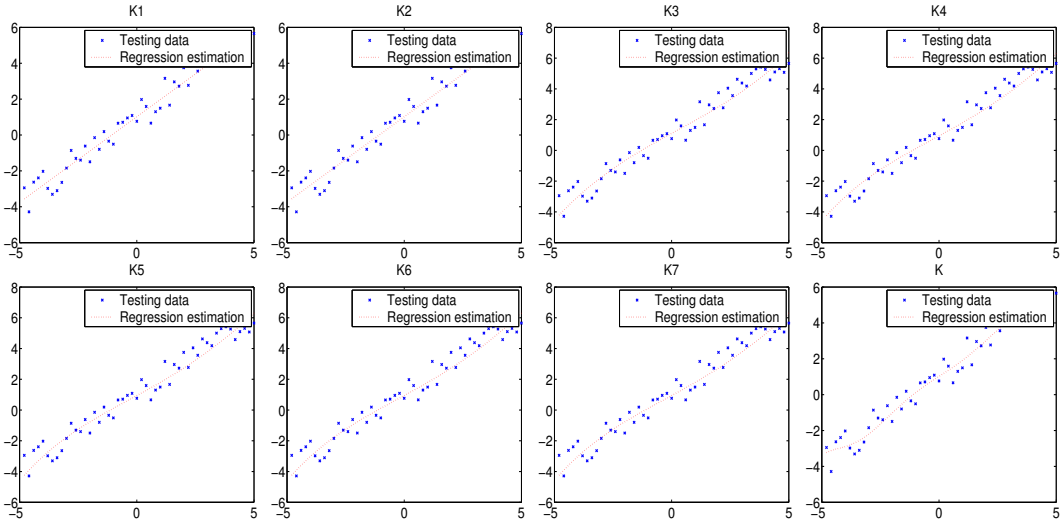

Figure 3.1: The testing result of problem 1. For the linear model, all kernels work very well.

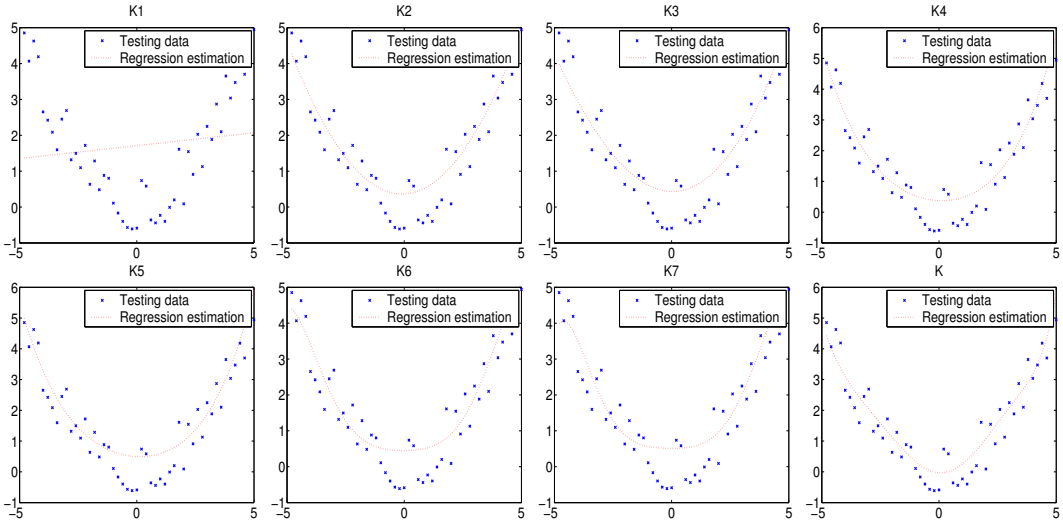

Figure 3.2: The testing result of problem 2. For the quadratic model, except for the linear kernel, all kernels work very well.

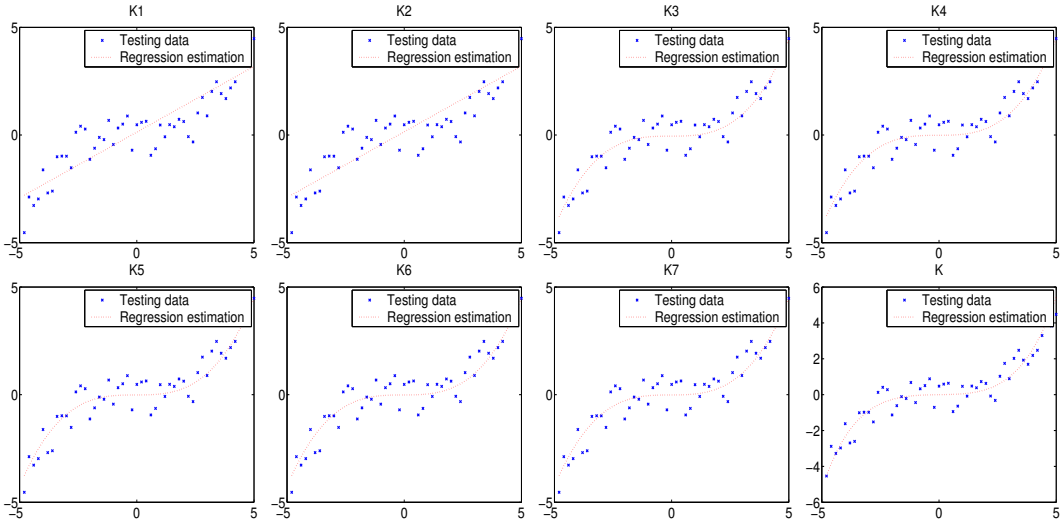

Figure 3.3: The testing result of problem 3. For the cubic model, except for the linear and quadratic regressor, all kernels work very well.

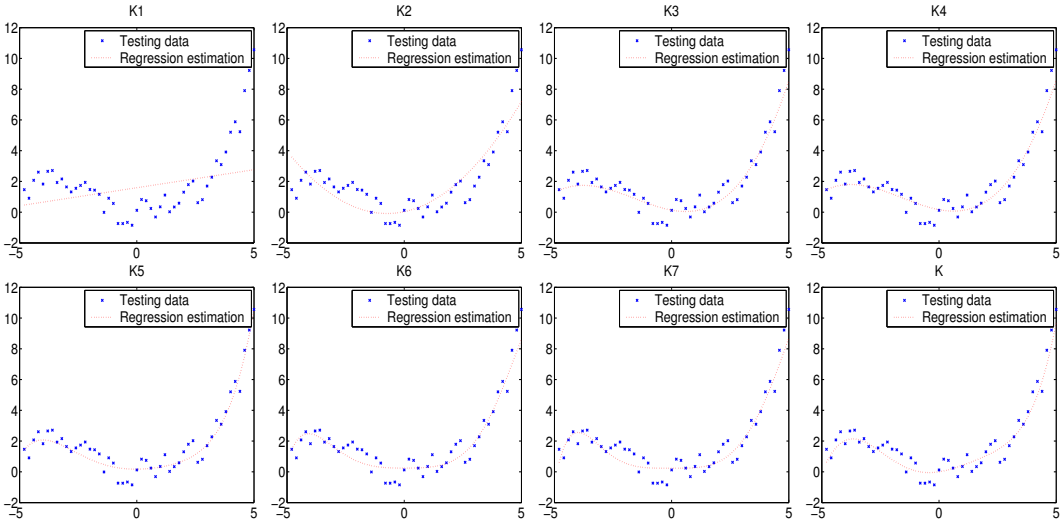

Figure 3.4: The testing result of problem 4. For a given polynomial model, the optimal kernel works much better than fixed kernels.

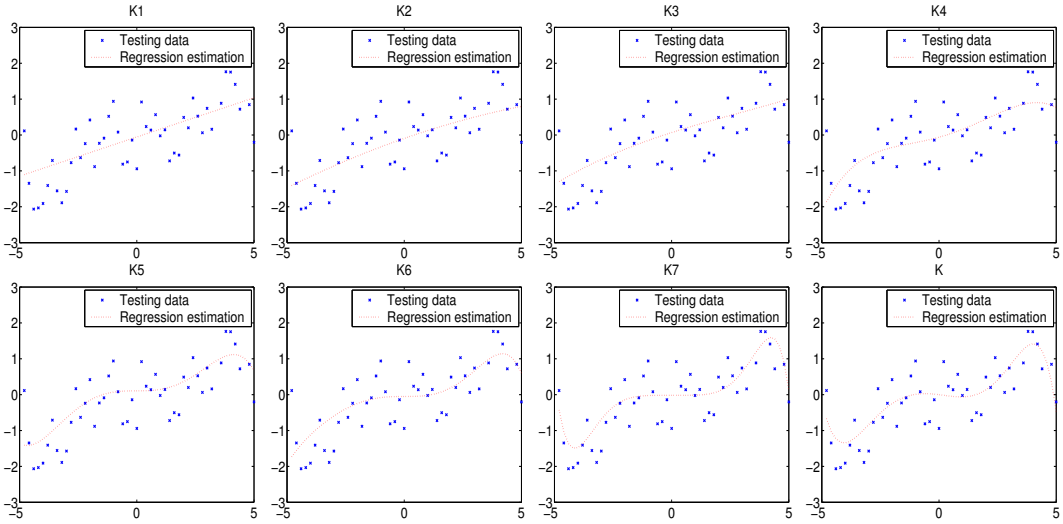

Figure 3.5: The testing result of problem 5. For the given polynomial model, the optimal kernel has the best performance.

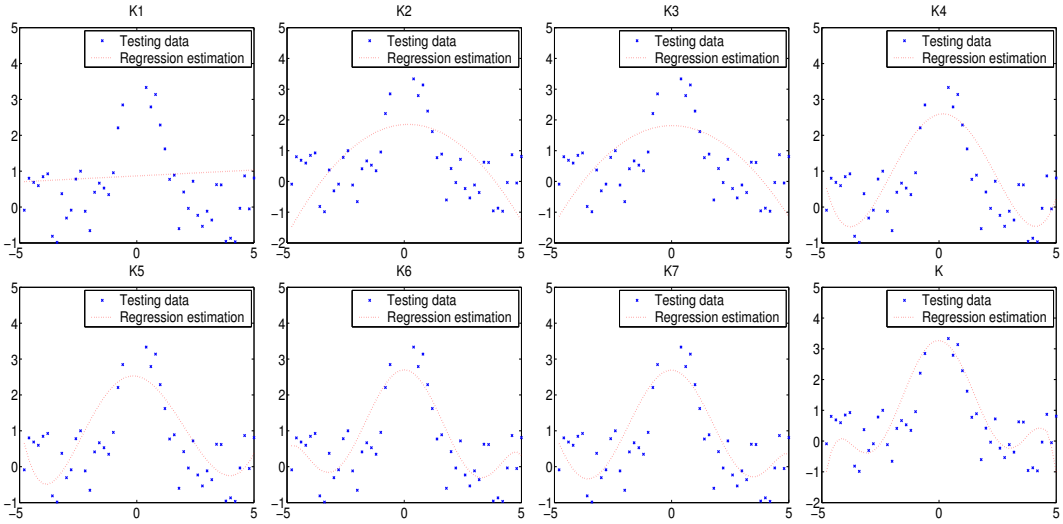

Figure 3.6: The testing result of problem 6. For the Gaussian model, the combined kernel works very well.

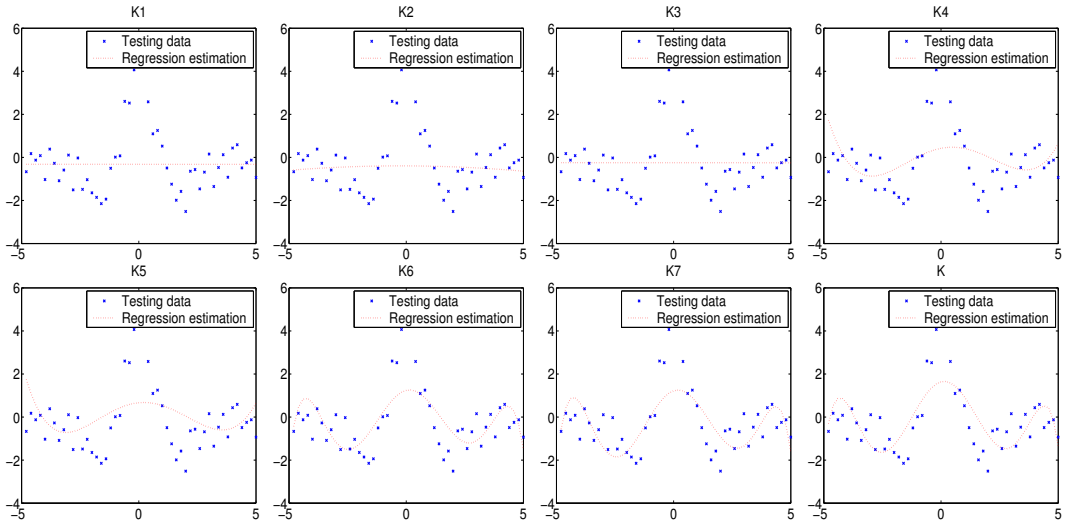

Figure 3.7: The testing result of problem 7. For the difference of Gaussian model, the combined kernel works very well.

- When none of the fixed kernels work well, their optimal linear combination may dramatically decrease the testing error (see problem 4) or work at least well as the best fixed candidate kernels (see problem 5, 6 and 7).

- Lower order models can be accurately estimated by regressors of the same or higher order. But if the regressor has too high order, it also causes larger testing error. The optimal kernel achieves a good balance of different kernels.

3.4 Application: Retina Ganglion Cell Signal Encoding

The data set we analyze in this section is the same as that in Section 2.4.

The problem is to estimate the response from the stimulus, i.e., an encoding problem. We first quantize the response spike train by counting the number

of spikes within two adjacent stimuli, that is within 100 ms. The number of spikes at time i after the i-th stimulus is the response $y_i \in \{0, 1, \ldots, 39\}$. The stimulus is recorded by a moving window with length q. For example, a stream $[0, 0, 5, 4, 3, 2, 1]$ will be recorded by a $q = 4$ moving window into $x_1 = [0, 0, 5, 4]^T$, $x_2 = [0, 5, 4, 3]^T$, $x_3 = [5, 4, 3, 2]^T$ and $x_4 = [4, 3, 2, 1]^T$. Longer window length means a more general model, since in general any response is a response of all previous stimuli. However, longer window length means higher dimension and therefore suffers from the "curse of the dimensionality". We fix $q = 20$ in this application.

We then use the proposed SVM regression kernel optimization technique to encode the stimuli.

3.4.1 Linear Model Verification

We are interested in verifying a model

$$y = w^T x + b,$$

where $y \in \mathbf{R}^n$, $x \in \mathbf{R}^{nq}$, $w \in \mathbf{R}^q$ and $b \in \mathbf{R}$. We use the proposed kernel optimization technique to learn an optimal linear combination of the polynomial kernels of order one to seven, and compare the testing error of the traditional SVM regression estimation using a linear kernel. For both methods, the parameter γ is fine tuned. We randomly pick 500 frames as the training data and the next 100 frames as the testing data. The experiments are repeated 10 times and we present the average performance here.

The comparison of testing error of the optimal kernel and the linear kernel are summarized in Table 3.2.

Neuron	1	2	3	4	5	6	7	8	9
Er_K	2.00	2.06	0.31	1.19	2.93	1.04	0.52	1.80	0.75
Er_{K_1}	2.37	2.09	0.71	0.80	3.48	1.16	0.49	2.00	0.72
Neuron	10	11	12	13	14	15	16	17	
Er_K	0.75	0.82	1.28	0.96	0.62	0.30	1.16	0.93	
Er_{K_1}	0.64	0.66	1.18	0.95	0.71	0.33	1.80	0.97	

Table 3.2: Comparison of testing errors of a linear kernel and the optimal kernel. Er_{K_1} and Er_K denote the average estimation error with a linear kernel and the optimal kernel respectively. The linear kernel works pretty well except for neuron 3 and 16, where the optimal kernel works much better. This shows that most neurons can be encoded by linear filters while neuron 3 and 16 can not be accurately encoded by linear filters.

3.4.2 Encoding via Optimal Linear Filter

The encoding problem is to estimate the retina cell's response to the stimuli. If we assume the time dependency is only significant in 20 frames, we are estimating

$$\hat{y}_t = f(x_t, x_{t-1}, ..., x_{t-20}).$$

Since we know that the linear kernel works very well in this application, at least as well as the optimal linear combination of the polynomial kernels from order one to seven. We then solve the traditional SVM regression problem to learn the underlying linear filter coefficients, w. Since we are not sure if the process is stationary, we solve the problem in the following fashion. We pick up 500 frames and solve for w, use the next 100 frames to test it and record the test error. We then pick up the next 500 frames and train them and pick up the next up the next 100 frames to test.

The averaged w over time is presented in Figure 3.8. The results match the previous results obtained using information theory by Zhong et al. [ZBJ05] very well. But one may note these are only averaged results.

3.4.3 Non-Stationary Behavior

We present the non-stationary behavior of every component of the filter coefficient vector w for each neuron. Noting that usually the last four components, w_{20}, w_{19} ,w_{18} and w_{17} (corresponding to epoch 0, -1, -2, and -3), are significant, we only plot these four components in Figure 3.9 to Figure 3.12 to save space.

The plotted results are very interesting. Actually, most of neurons' behavior is not stationary. We summarize the shift pattern here. The response of neuron 1, 3, 4, 9 is getting stronger to the stimulus at epoch -1 (corresponding to w_{19}). The response of neuron 15 and 16 is getting stronger to the stimulus at epoch -2 or -3 (corresponding to w_{18} and w_{17}). The response of Neuron 5, 7, 8, 10, 11, 12, 13 is getting weaker for the stimulus at epoch -1 (corresponding to w_{19}).

That is to say, neurons have in general two models of getting "tired". One model is "dizzy". That is, neurons are more likely to response to the previous stimuli instead of the current one. Neuron 1, 3, 4, 9, 15, and 16 belong to such model. The other model is "memoryless". That is, neurons tend to forget the previous stimulus when getting "tired". Neurons 5, 7, 8, 10, 11, 12, and 13 belong to this category.

3.5 Conclusion

I applied a recently developed SDP kernel learning method to support vector regression problems. The regression problem turns out to be more interesting

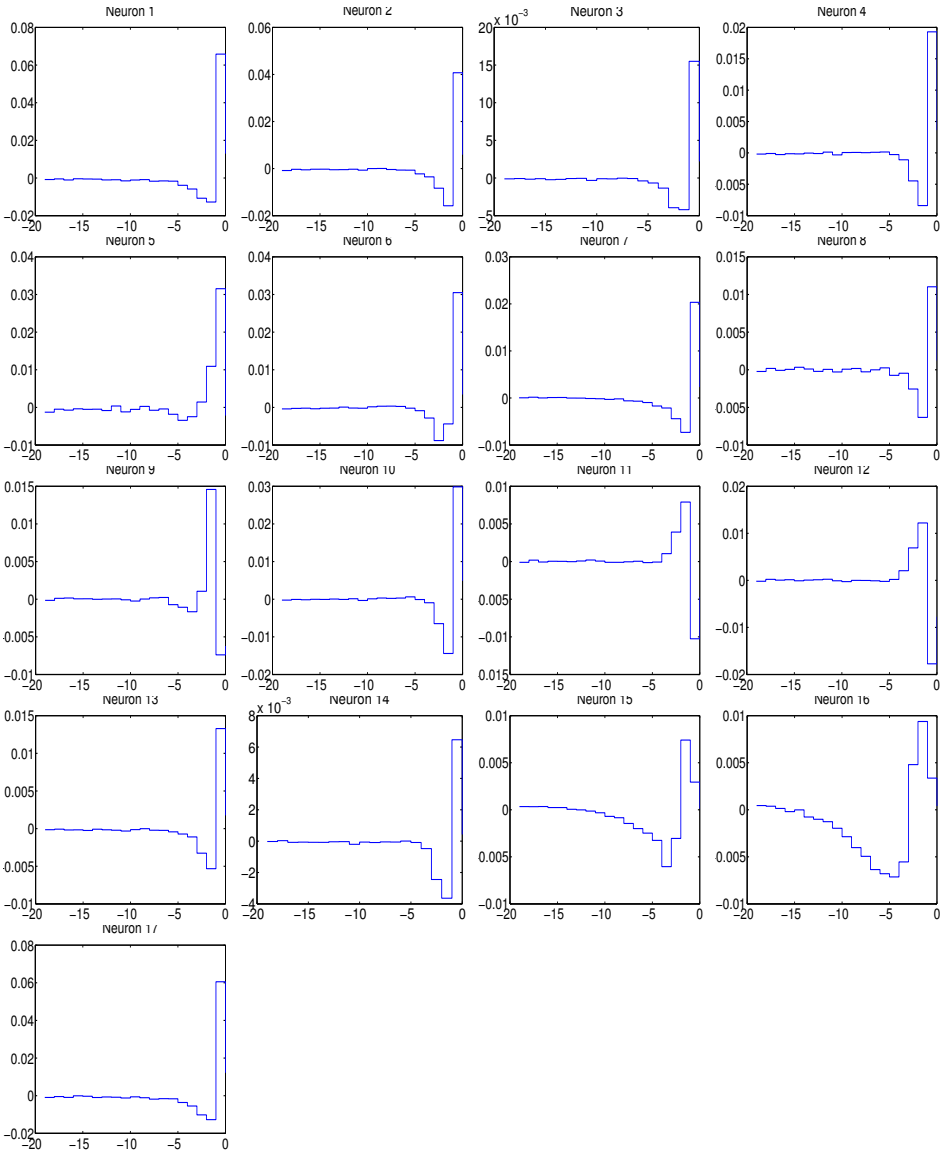

Figure 3.8: Average linear filter for neurons. The results match the previous obtained results using information theory by Zhong et al. [ZBJ05]

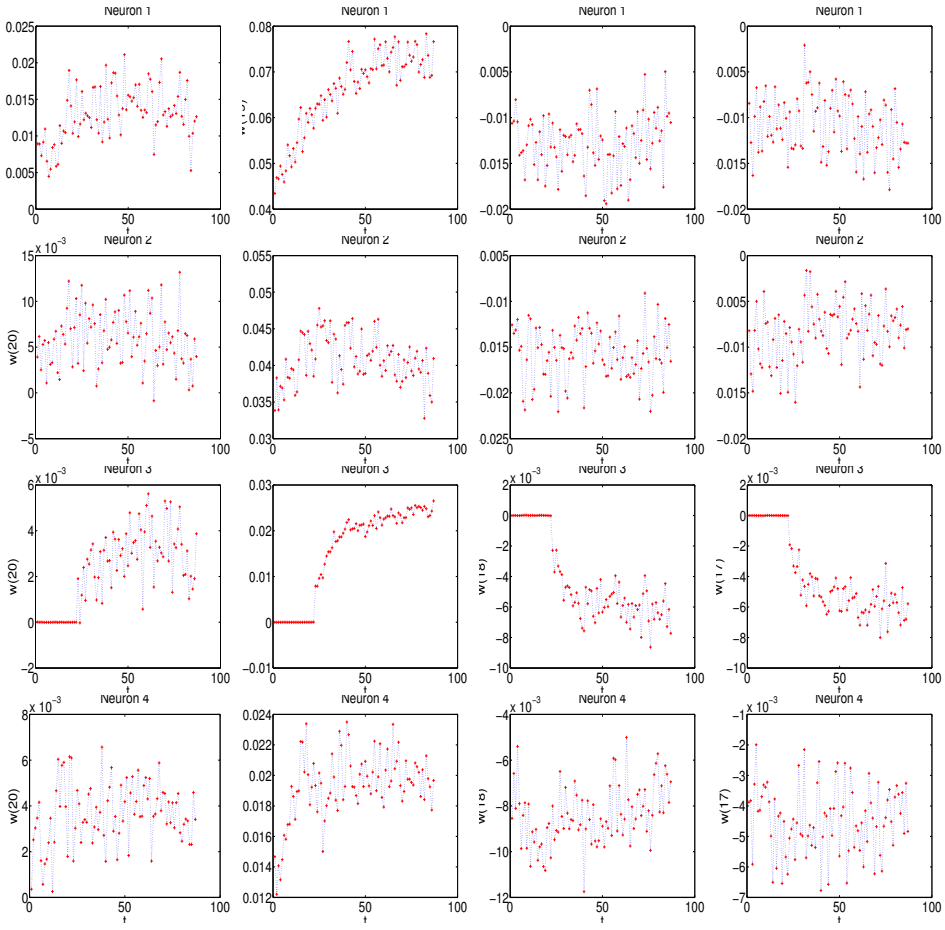

Figure 3.9: Dynamics of the encoding filter coefficients for neuron 1, 2, 3 and 4. From the left to the right, the w_{20}, w_{19}, w_{18} and w_{17} of neurons. are plotted against time. From the top to bottom are neuron 1, 2, 3, and 4. This figure demonstrates the non-stationary behavior of neurons.

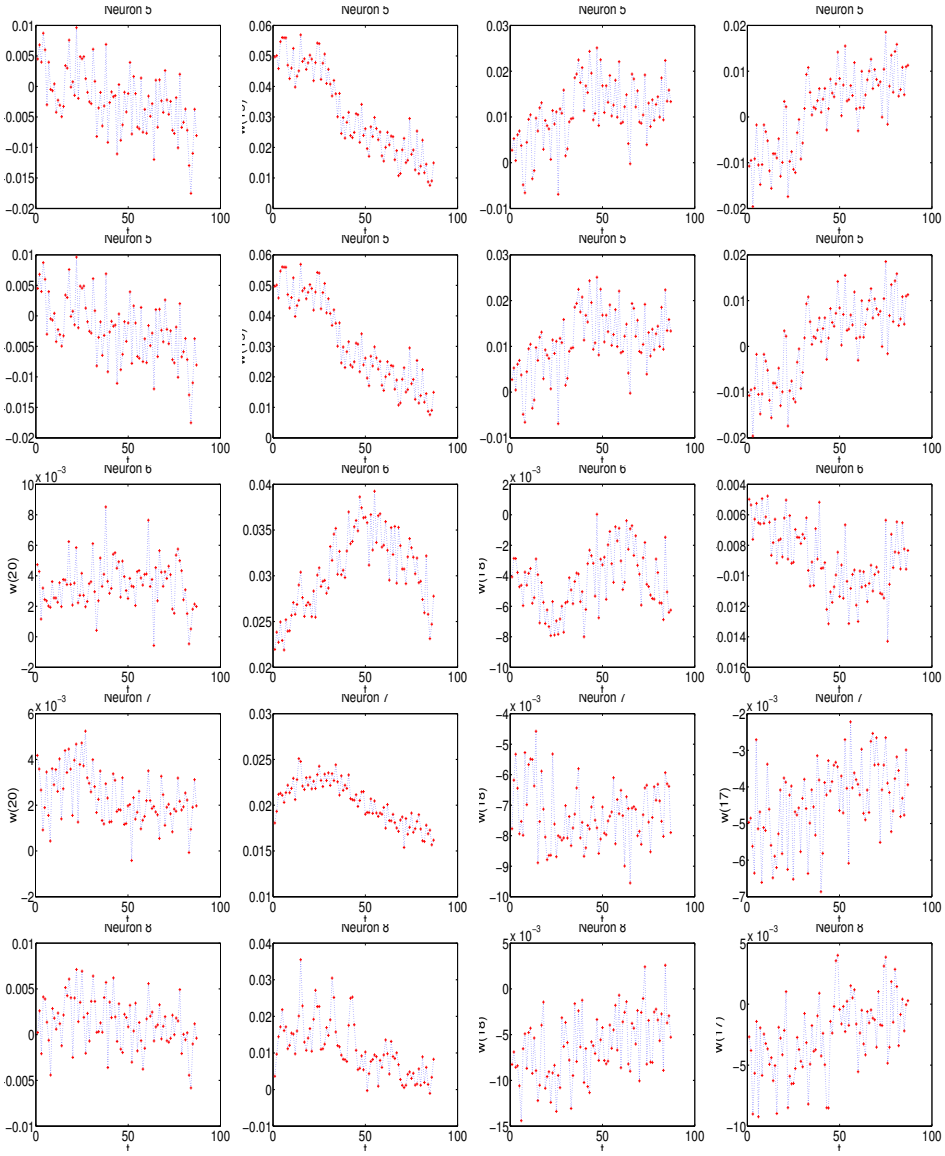

Figure 3.10: Dynamics of the encoding filter coefficients for of neuron 5, 6, 7 and 8. From the left to the right, the w_{20}, w_{19}, w_{18} and w_{17} of neurons. are plotted against time. From the top to bottom are neuron 5, 6, 7, and 8. This figure demonstrates the non-stationary behavior of the neurons.

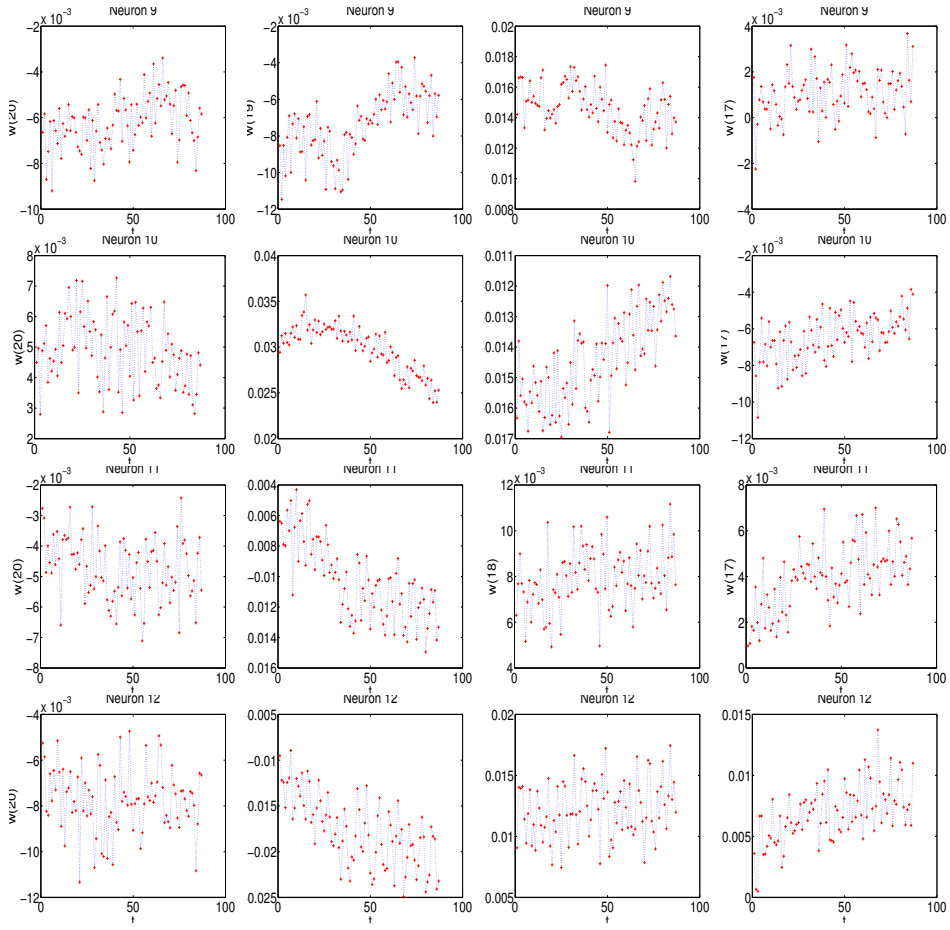

Figure 3.11: Dynamics of the encoding filter coefficients for of neuron 9, 10, 11 and 12. From the left to the right, the w_{20}, w_{19}, w_{18} and w_{17} of neurons. are plotted against time. From the top to bottom are neuron 9, 10, 11, and 12. This figure demonstrates the non-stationary behavior of the neurons.

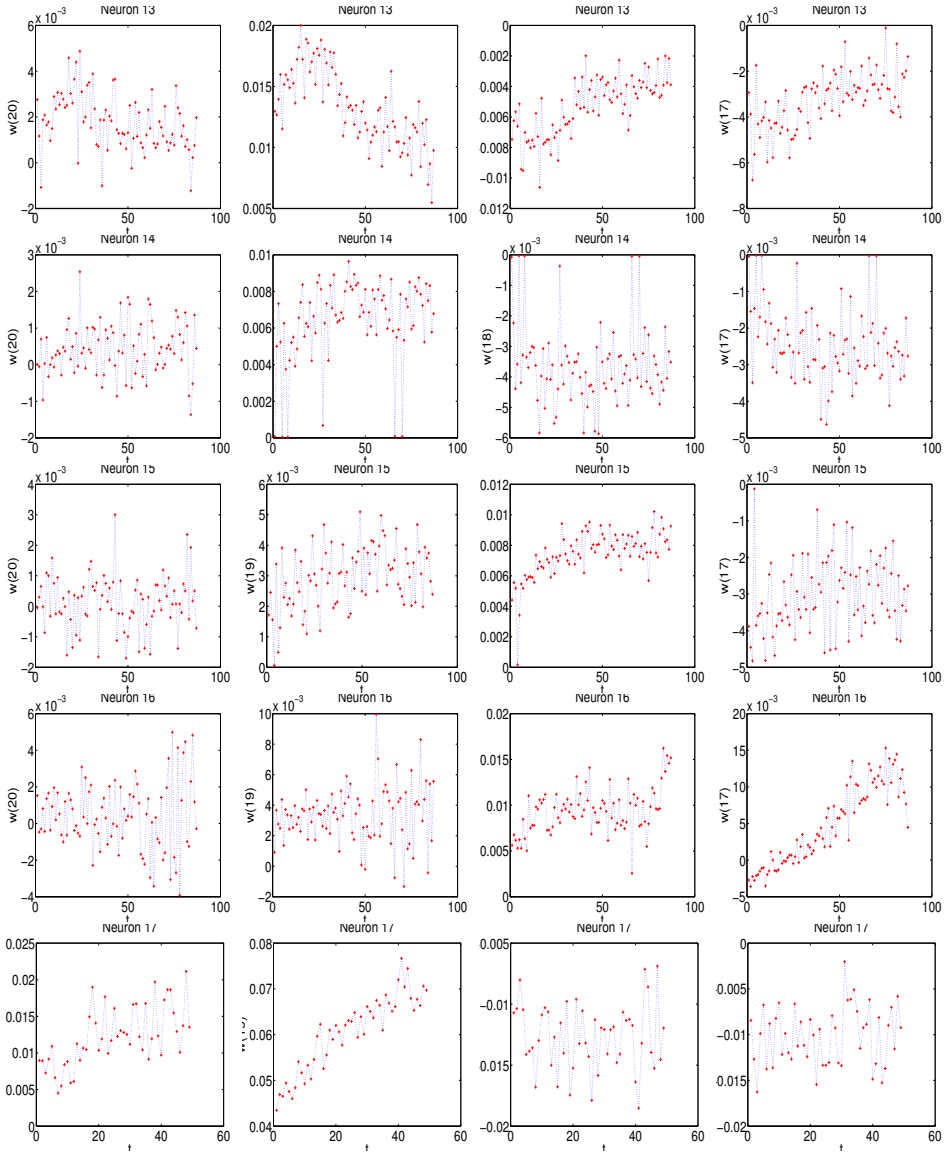

Figure 3.12: Dynamics of the encoding filter coefficients for of neuron 13, 14, 15, 16 and 17. From the left to the right, the w_{20}, w_{19}, w_{18} and w_{17} of neurons. are plotted against time. From the top to bottom are neuron 13, 14, 15, 16 and 17. This figure demonstrates the non-stationary behavior of the neurons.

than classification problem. The reason is that the kernel matrix for regression estimation is directly related to the underlying model of the regressor.

The SDP kernel learning method provides a unique way of model-free regression. It is able to algorithmically determine the kernel matrix among a family of candidate kernel and their linear combination. This is the first time that the *optimum* kernel can be found for the SVR.

In the application of the retina neuron encoding problem, two different non-stationary firing patterns are discovered after verifying the linear filter.

CHAPTER 4

Distributed Support Vector Machine

4.1 Introduction

Distributed learning is necessary if a centralized system is infeasible because of geographical, physical or computational reasons. The objectives of distributed data mining usually include robustness to changes in the network topology [ZL03], efficient representation for high-dimensional and massive data sets, reduced synchronization and communication, reduced duplication, better load balancing and good decision precision and certainty.

In distributed classification, we classify distributed data sets according to distributed training samples. The support vector machine (SVM), which is able to minimize the structural error risk function [Vap95, CV95], is one of the most popular algorithms in classification. Current parallel methods include matrix multi-coloring successive overrelaxation (SOR) method [AJ86, BFM90] and variable projection method (VPM) in sequential minimal optimization (SMO) [ZZ03]. Those methods are excellent in terms of speed. However, they all need centralized access to the training data and therefore cannot be used in distributed classification applications.

In this chapter, we consider the following problem. The training vectors are distributed over C sites. Each site is a node within a strongly connected network, i.e., a directed network in which it is possible to reach any node starting from

any other node by traversing edges in the direction(s) in which they point.

There are N_l training vectors in site l and N training vectors in all sites such that $\sum_{l=1}^{C} N_l = N$. Each training vector is denoted by x_i $i = 1, ..., N$ where $x_i \in \mathbf{R}^n$,and $y_i \in \{+1, -1\}$ is its label.

Note that the support vectors (SVs) are a natural representation of the discriminant information of the database of the underlying classification problem. We propose a distributed support vector machine (DSVM) for distributed data classification. The basic idea of the DSVM is to exchange SVs over a strongly connected network and update the local solutions for each site iteratively. Our algorithm is based on the observation that the number of support vectors (SVs) may be very limited for the local classification problems. Huge amount of SVs are usually an evidence that the problem is ill defined [Dom02]. We prove that our algorithm converges to a global optimal classifier for an arbitrarily distributed database across a strongly connected network.

4.2 Algorithm

The distributed support vector machine (DSVM) training algorithm works as follows. Each site within a strongly connected network classifies the local data set via a standard SVM algorithm, passes the found SVs to its descendant sites and receives SVs from its parent sites at each iteration. The algorithm of DSVM consists of the following steps.

Initialization

We initialize the algorithm with $t = 1$, $l = 1$ and $V^{0,l} = \emptyset$, $\forall l$. The training set at iteration t in site l is denoted by $S^{t,l}$. $S^{0,l}$ is initialized arbitrarily such that

$\cup_{l=1}^{C} S^{0,l} = S$, where S denotes the total sample space.

Iteration

Each iteration consists of the following steps.

1. For any site l, $l = 1, ..., C$, once it receives SVs from its parent sites, we repeat the following steps.

   (a) Merge the support vectors from the parent sites that are not included in the current site. That is,

   $$S^{t,l} := S^{t-1,l} \cup \{x_i : x_i \in V^{t,m}, \forall m \in \text{UPS}_l, x_i \notin S^{t-1,l}\},$$

   where UPS_l is the set of all parent sites of site l.

   (b) Solve the SVM training problem at site l:

   $$\begin{aligned} \text{maximize} \quad & -(1/2) \textstyle\sum_{i,j:x_i,x_j \in S^{t,l}} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{i:x_i \in S^{t,l}} \alpha_i \\ \text{subject to} \quad & 0 \leq \alpha_i \leq \gamma, \ \forall i : x_i \in S^{t,l} \\ & \textstyle\sum_{i:x_i \in S^{t,l}} y_i \alpha_i = 0. \end{aligned}$$

   Record the optimal objective value $h^{t_l,l}$ and the solution $\alpha^{t,l}$.

   (c) Find the set $V^{t,l} = \{x_i : \alpha_i^{t,l} > 0\}$ and pass them to all immediate descendant sites.

2. If $h^{t,l} = h^{t-1,l}$ for all l, stop; otherwise, $t := t + 1$ and go to step 1.

Every site starts to work once it receives SVs from its parent sites. In step 1(b), we start solving the newly formed problem from the best solution currently available and update this solution locally. We use $\text{SVM}^{\text{light}}$ [Joa98] as the local solver. The numerical results show that the computing speed can be dramatically

increased by applying the available best solution α^{t-1} as the initial starting point in Step 1(b). We are going to discuss this issue later in this chapter.

Now, we use a small example to demonstrate the convergence property. We randomly generate 200 independent two-dimensional data sampled from two independent Gaussian distributions. We randomly distribute all the data over 5 sites. We assume that the 5 sites form a Round-Robin network. The DSVM converges in 4 iterations and the local results are shown in the Figure 4.1. One may observe the decreasing of the local margins and their convergence to an optimal classifier.

We give the proof of the global convergence in the next section.

4.3 Proof of Convergence

In this section we prove that the DSVM method on a strongly connected network converges to the optimal classifier in a finite number of steps.

The proof is sketched as follows. The DSVM converges in a finite number of steps since the number of training vectors is finite. The objective value for any site, say site l, at iteration t is always greater or equal to the maximum of the objective of the same site in the last iteration and the maximal objective values of its parent sites due to accumulation of local SVs. Therefore, the stopping criterion, $h^{t,l} = h^{t-1,l}$ for all l and $t > T$, can be satisfied in finite number of iterations.

After the convergence, the support vectors of adjacent sites are identical, by uniqueness of the optimal solution for a strictly convex problem.

Since the network is strongly connected, the support vectors of all sites over the network are identical. Let $V^\star$ denotes the converged support vector set. The

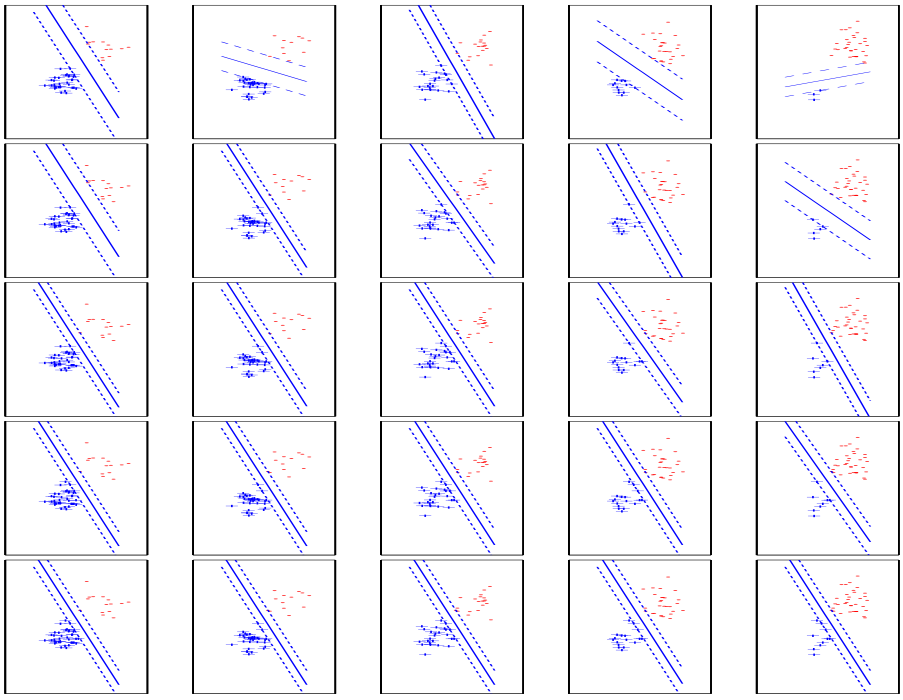

Figure 4.1: Demonstration of data distributions in iterations of the DSVM algorithm. Distributions of training vectors in 5 sites before iteration begins (the first row) and in iteration 1 to 4 (the 2nd to the 5th rows) are plotted with the local optimal classifiers. These figures show how all local classifiers converge to the global optimal classifier.

solution defined by

$$\alpha_i^\star = \begin{cases} \alpha^{t,l}, & \text{if } x_i \in V^\star \\ 0, & \text{otherwise} \end{cases}$$

is always a feasible solution for the global SVM problem (1.5). By KKT optimality conditions, $V^\star$ is also the support vector set for the union of the training samples from one site and is parent sites. By induction, $V^\star$ is the support vector set for the union of the training samples from all sites. Therefore, the solution $\alpha^\star$ is the global optimum.

4.4 Performance Studies

We first present a toy example to demonstrate the performance of the distributed support vector machine training algorithm (DSVM) over different configurations of topology, size of network, and sequential or parallel implementation, which is followed by a thorough analysis of a handwritten digits classification database using DSVM. The $\text{SVM}^{\text{light}}$ is used as our local SVM solver since $\text{SVM}^{\text{light}}$ has reported fastest performance in selected applications [Joa98]. We first introduce our terminology used in this section.

N: total number of training samples;

C: total number of distributed sites;

N^{s}: total number of support vectors for the entire problem;

T: total number of DSVM iterations;

δ: average number of transmitted training vector per iteration at one site;

Δ: total number of transmitted vectors;

$N_{t,l}$: number of training samples in site l at iteration t;

$N^{\max}$: maximum number of training sample among all sites among all iterations;

e: elapsed CPU seconds of running DSVM.

Through out this section, the machine we use to solve local SVM problem is Pentium 4 2.26GHz with 512KB RAM.

	$N_{0,1}$	$N_{0,2}$	$N_{0,3}$	$N_{0,4}$	$N_{0,5}$
class 1	338	116	485	197	264
class 2	106	168	58	144	124

Table 4.1: A randomized initial distribution of training data of the toy example.

4.4.1 A Toy Example

To demonstrate the properties of our algorithm, we generate two classes of 2D data from two independent distributions.The partitioning result is shown in Table 4.1.

We use an SVM Gaussian kernel with unit variance in each site and select the parameter γ to be 10. Since we already proved that the DSVM always converges to a regular SVM training problem solution, we do not need to worry about the classification quality in this chapter. Therefore, we did not fine tune the parameters.

Size of Network and Scalability

We use fully connected networks and compare the performance for three different sizes, $C = 5$, $C = 10$ and $C = 20$. The performance is presented in Table 4.2. The results show that our algorithm scales very well with C when the network size is not too large. That is, larger networks achieve less training time since a larger network means that more servers and more computations can be done parallel. The results also suggest that the size of network has limited effect on the communication overhead per exchange.

C	T	δ	Δ	$N^{\max}$	e
5	6	8.5	1017	635	0.80
10	4	13.0	4680	690	0.43
20	4	10.2	15504	888	0.29

Table 4.2: Algorithm performance in networks of different size. Experiments show that larger fully connected networks allow the advantage of shorter processing time but has the disadvantage of more training data accumulation. The result also suggests that the size of network has a limited effect on the communication overhead per exchange, measured by δ

Network Topology

We first test a fully connected network shown is Figure 4.2. The DSVM algorithm finds support vectors pretty fast (in four iterations).

Then we test sparser networks. The Round-Robin network, the sparsest strongly connected network shown in Figure 4.3 and a randomly generated strongly connected network shown in Figure 4.4 are used. The results are summarized in Table 4.3, where R-R means the Round-Robin network. The results show that the DSVM in denser (but not too dense) networks has significantly better performance in terms of the number of iterations and the elapsed CPU seconds than a very sparse network. However, if the data accumulation and communication cost are the main concern, a sparser network should be used.

Sequential Implementation

In order to further decrease the communication cost and data accumulation, the DSVM can be implemented sequentially in a Round-Robin network. That

N	C	Kernel	N^{s}	Network	T	δ	Δ	$N^{\max}$	e
2000	5	Gaussian	78	R-R	11	13.4	734	697	1.30
				Full	6	8.5	1017	778	0.80
				Random	6	4.7	1156	742	0.40
				Seq. R-R	15	33.0	495	648	0.55

Table 4.3: Algorithm performance for different network topologies and type of implementations. Experiments show that denser network has the disadvantage of more data accumulation. Sequential implementation achieves much less data accumulation than parallel implementation. The best computing time is achieved by a randomly strong connected network.

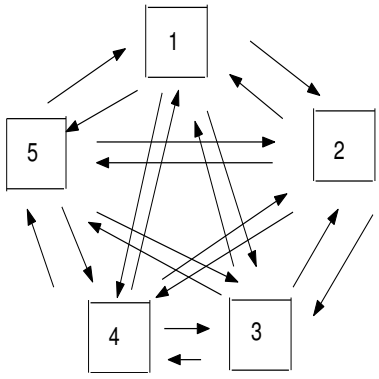

Figure 4.2: Diagram of a fully connected network with 5 sites. This is the densest network.

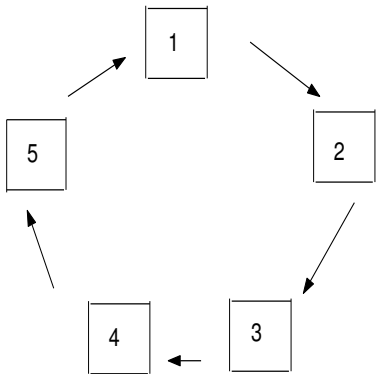

Figure 4.3: Diagram of a Round-Robin network with 5 sites. This the sparsest strongly connected network.

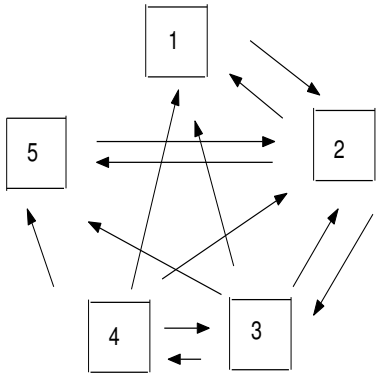

Figure 4.4: Diagram of a random strongly connected network.

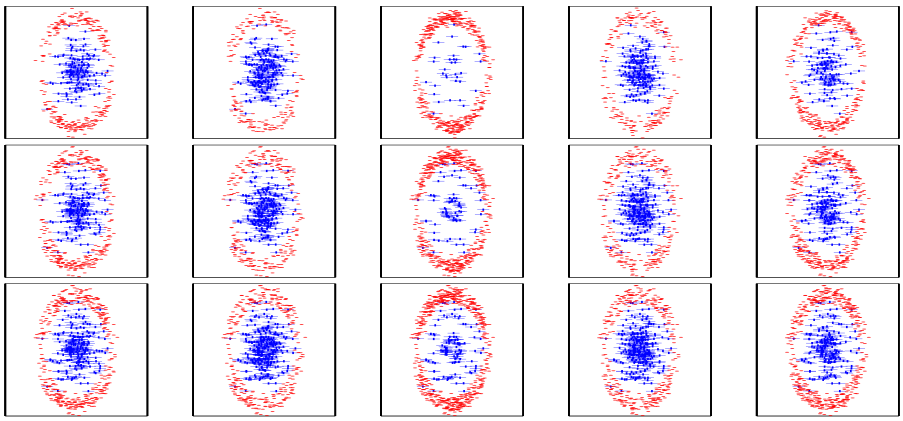

Figure 4.5: Comparison of data accumulation of the parallel and distributed implementations. The first row shows the initial distribution of the training data in 5 sites. The 2nd row shows the data distribution in the last iteration (10th iteration) after a sequential implementation. The 3rd row shows the data distribution in the last iteration (8th iteration) after a parallel implementation. A unit variance Gaussian kernel is used in the DSVM algorithm. This figure shows that the algorithm converges without merging all the data and the sequential implementation achieves less data accumulation.

is, each site begins to calculate its local solution only after receiving SVs from its parent site. A different trick for setting the initial solution is applied here. Instead of using one site's own α, we use the incoming α as the initial solution since the incoming solution is C iterations better than one site's own solution of the last iteration in a Round-Robin network. The results are also listed in Table 4.3. We observe that sequential implementation might be even faster than parallel implementation in some given examples. The data distribution in each site during the sequential and parallel iterations of the DSVM is presented in Figure 4.5. Comparing the second row (the last iteration of a sequential implementation) and the third row (the last iteration of a parallel implementation), one may clearly see the improvement of the sequential implementation in terms of data accumulation. This explains why sequential implementation of the DSVM might be even faster than the parallel version.

4.4.2 Handwritten Digits Classification

We test our algorithm on a large scale database: MNIST database of handwritten digits [LBB98]. The MNIST database of handwritten digits has a training set of 60,000 vectors. The digits have been size-normalized and centered in a fixed-size image. This database is a standard database online for people to compare their training methods.

All the digits images are centered in a 28x28 image. We vectorize each image to a 784x1 vector. The problem is to classify digit 0 from the rest. For simplicity, we use linear kernel and set $\gamma = 10$. This problem has 60000 samples of dimension 784-by-1 and 1235 support vectors.

σ	T	δ	Δ	$N^{\max}$	e
0	5	25.6	8015	2686	75.82
196.3	5	29.8	8330	3111	79.48
620.1	6	23.4	7875	3804	99.18
1204.3	5	27.1	7588	4403	83.11
1763.4	5	25.6	7175	5302	90.26

Table 4.4: Algorithm performance over the initial distribution of training data. In this table, σ denotes the standard deviation of initial training data distribution. This table shows that the unbalanced initial training vector distribution has very limited effect on the data accumulation (in terms of the total number of transferred vectors Δ) and the computing time e.

Effect of Initial Data Distribution

Before we present our main result: the scalability, we first test the effect of the initial data distribution on the DSVM.

We use 8 sites and 15000 randomly selected training samples. We randomly distribute the training data several times and record the standard deviation of the data distribution. When the data are distributed evenly, the standard deviation of the initial data distribution, σ is 0. The results presented in Table 4.4 show that the effect of the initial distribution on training time and communication cost is very limited. We therefore may use a randomly distributed example to demonstrate the scalability.

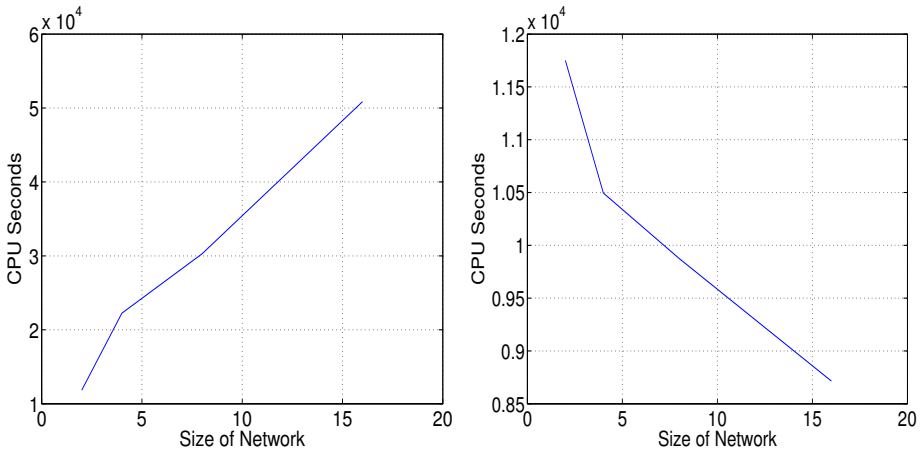

Figure 4.6: Computing time use the size of networks. *Left.* Elapsed computing time for a sequential implementation in a Round-Robin network. *Right.* Elapsed computing time for a parallel implementation in a fully connected network. Parallel implementation takes the advantage of the multiple servers and scales very well.

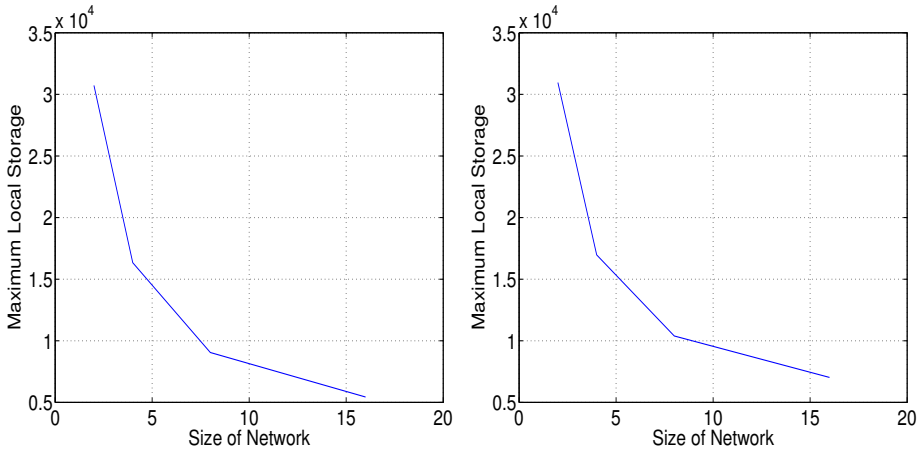

Figure 4.7: Data accumulation versus the size of networks. *Left.* Maximum number of training samples in a site after the convergence in a sequential implementation over a Round-Robin network. *Right.* Maximum number of training samples in a site after convergence in a parallel implementation over a fully connected network. Sequential implementation achieves less data accumulation.

Scalability

We test how the DSVM scales with the network size. We choose the number of nodes to be 2, 4, 8, and 16. We do these experiments two times: one using parallel implementation with a fully connected network and the other using a sequential implementation with a Round-Robin network. Figure 4.6 and 4.7 provides the performance measurements in term of CPU seconds and maximum local data accumulation $N^{\max}$ respectively. The results show that the sequential implementation in the Round-Robin network has less data accumulation, while the parallel implementation in the fully connected network achieves approximately linear scalability over the network size when the size is limited.

Online Implementation

Since the DSVM constructs local problem by adding critical data, the local SVM training problem is, therefore, essentially a problem adjustment which appends variables and constraints based on the problem of the last iteration. A naive online implementation is to input the best available solution as the initial solution so that each problem has a warm start. To show the effect of online implementation, we plot the computing time in each iteration is presented in Figure 4.8. The initial sharp increasing of CPU seconds is due to the sharp increase of the number of local SVs. The later decrease of CPU seconds is due to the advantage of the online implementation.

4.5 Conclusions and Future Research

The proposed distributed support vector machine (DSVM) training algorithm exploits a simple idea of training distributed support vector machine by exchang-

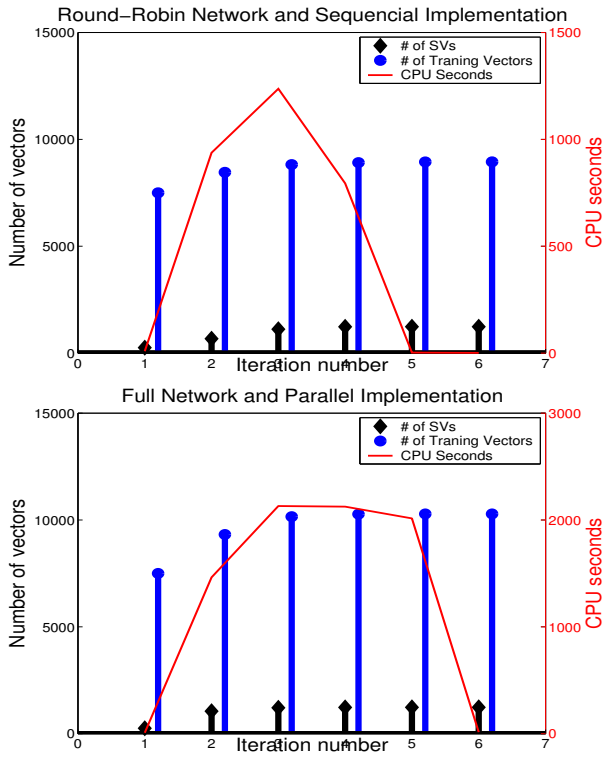

Figure 4.8: Effect of the online implementation. *Top.* The number of training vectors, support vectors and the computing time of site 1 per iteration of the DSVM, implemented sequentially over a Round-Robin network. Here one iteration means that the solution in site 1 has been updated once. *Bottom.* The number of training vectors, support vectors and computing time of site 1 per iteration of the DSVM implemented parallel over a fully connected network. These two figures show that the initial increase of the computing time is mainly caused by the increasing of the number of support vectors. The later reduction in the computing time is the result of the online implementation. Comparing these two figures, one may observe that sequential implementation gain more than the parallel one from the online implementation. The reason is that in sequential implementation each site may get the most current α value as the input.

ing support vectors. The algorithm has been proved to converge to the global optimal classifier in finite steps. Simulations and real-world database tests show that this algorithm is fast and robust. The properties of this algorithm can be summarized as follows.

The DSVM algorithm is able to work on multiple arbitrarily distributed working sets and achieves close to linear scalability if the size of network is not too large.

Data accumulation during SVs exchanging is limited if the overall number of SVs are limited. The communication cost is proportional to the number of SVs. Sequential implementation over a sparse network achieves the minimum data accumulation.

The DSVM algorithm is robust in terms of computing time and communication overhead to the initial distribution of the database. It is suitable for classification over arbitrary distributed databases.

In general, denser networks achieve less computing time while sparser networks achieve less data accumulation.

On-line implementation is much faster. When a better on-line solver for DSVM becomes available, the DSVM problem may be solved much more efficient. This is also one of our future research direction.

We believe that certain randomization techniques will enable the algorithm to avoid the possible worst case scenario: the algorithm keeps accumulating data until some site contains most of the data, though we did not encounter this problem in applications. Regarding the future research, We are interested in

developing theoretically provable convergence rates and/or bounds for the DSVM or a slightly modified version.

CHAPTER 5

Parallel Randomized Support Vector Machine

5.1 Introduction

Sampling theory has a long successful history in optimization [Cla88, AS93]. The application to the SVM training problem was first proposed by Balcazar et al. in 2001 [BDT01]. However, Balcazar et al. assume that the SVM training problem is a separable problem or a problem that can be transformed to an equivalent separable problem by assuming an arbitrary small regularization factor γ (D and $1/k$ in [BDT01] and [BDT02]). They also stated that there were a number of implementation difficulties so that no relevant numerical results could be provided [BDT02].

We propose a novel parallel randomized SVM (PRSVM) in which multiple working sets can be worked on simultaneously. The basic idea of the PRSVM is to randomly shuffle the training vectors among a network based on a carefully designed priority and weighting mechanism and to solve multiple local problems simultaneously. We prove that our algorithm, on average, converges to the global optimum classifier/regressor in less than $(6\delta/C)\ln(N + 6r(C-1)\delta)$ iterations, where δ denotes the underlying combinatorial dimension, N denotes the total number of training vectors, C denotes the number of working sites; and r denotes the size for a working set. Since the RSVM is a special case of the PRSVM, our proof naturally works for the RSVM. Note that, when $C = 1$, our result reduces

to the bound in [BDT02].

This chapter is organized as follows. We first define basic concepts and terminologies for the general support vector machine in Section 5.2, which is followed by the presentation of the algorithm in Section 5.3. We then derive the theoretical global convergence rate in Section 5.4, and provide an application in Section 5.5. We conclude our result in Section 5.6.

5.2 Support Vector Machine and the Sampling Lemma

In this section, we show that a general support vector machine may be modeled as an abstract problem that is called LP-type abstract problem [Cla88, AS93, BDT01, GW00]. The violators and extremes of the LP-type problem are defined by using the KKT optimality conditions. The concepts and definitions are used in the later proposed algorithm.

SVM Training Problems

The linear nonseparable SVM training problem has the primal form

$$\begin{array}{ll} \text{minimize} & (1/2)w^T w + \gamma \mathbf{1}^T \xi \\ \text{subject to} & y_i(w^T x_i + b) \geq 1 - \xi_i,\ i = 1, ..., N \\ & \xi \geq 0 \end{array} \tag{5.1}$$

where variables $w \in \mathbf{R}^m$, $b \in \mathbf{R}$ and slacks $\xi \in \mathbf{R}^N$. The predefined parameter γ is a regularization parameter and is usually empirically selected.

The corresponding dual of the problem (5.1) is shown as follows:

$$\begin{array}{ll} \text{maximize} & -(1/2)\alpha^T Q \alpha + \mathbf{1}^T \alpha \\ \text{subject to} & 0 \leq \alpha \leq \gamma \mathbf{1} \\ & y^T \alpha = 0, \end{array} \tag{5.2}$$

where variables $\alpha \in \mathbf{R}^N$, sample labels $y \in \mathbf{R}^N$, the Gram matrix $Q \in \mathbf{R}^{N \times N}$ and $Q_{ij} = y_i y_j x_i^T x_j$.

The LP-type Problem

The set of training vectors is denoted by $\mathcal{X}$. That is, each element of $\mathcal{X}$ is a row vector of a matrix X. Throughout this paper, we use $\mathcal{CALLIGRAPHIC}$ style letters to denote sets of the row vectors of a matrix denoted by the same letter with *italic* style. One may note for each training set $\mathcal{X}$, there is a unique constraint set that involves training vectors of rows of X. For simplicity, we also use $\mathcal{X}$ to denotes the corresponding constraint set in problem (5.1) and (5.2).

Let $\phi(\mathcal{R})$ be the optimal value of the problem (5.1) with training set $\mathcal{R}, \mathcal{R} \subseteq \mathcal{X}$. The mapping $\mathcal{R} \rightarrow \phi(\mathcal{R})$ satisfies locality and monotonicity conditions [GW00]. For support vector machines, the monotonicity property can be expressed as

$$\phi(\mathcal{X}_1) \leq \phi(\mathcal{X}_2), \text{ for all } \mathcal{X}_1 \subset \mathcal{X}_2 \subset \mathcal{X},$$

as the optimal value $\phi(\mathcal{X}_2)$ may be reduced by removing constraints.

The locality property can be expressed as follows.

If $\phi(\mathcal{X}_1) = \phi(\mathcal{X}_2)$ and $\phi(\mathcal{X}_2) < \phi(\mathcal{X}_2 \cup \{x\})$, then $\phi(\mathcal{X}_1) < \phi(\mathcal{X}_1 \cup \{x\})$,
for all $\mathcal{X}_1 \subset \mathcal{X}_2 \subset \mathcal{X}, x \in \mathcal{X}$.

The *basis* of an LP-problem $(\mathcal{X}, \phi)$ is a inclusion-minimal subset $\mathcal{B} \subseteq \mathcal{X}$ such that $\phi(\mathcal{B}) = \phi(\mathcal{X})$. Therefore, for an SVM training problem, a basis is a minimal subset of active constraints $\mathcal{X}$ and corresponding training vectors such that the optimal value will not change by adding more constraints from $\mathcal{X} \backslash \mathcal{B}$. In this chapter, a basis is called a set of *necessary support support vectors*. That is, if any constraint corresponding to a necessary support vector is removed, the

optimal value will increase (for a minimization problem).

Violators, Extremes and the Combinatorial Dimension

Let $(\mathcal{X}_p,\ \phi)$ be a partial problem of the SVM problem $(\mathcal{X},\ \phi)$, where $\mathcal{X}_p \subseteq \mathcal{X}$:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}||w||^2 + \gamma \sum_{i:x_i \in \mathcal{X}_p} \xi_i \\ \text{subject to} & y_i(w^T x_i + b) \geq 1 - \xi_i,\ \forall i : x_i \in \mathcal{X}_p \\ & \xi_i \geq 0,\ \forall i : x_i \in \mathcal{X}_p. \end{array} \tag{5.3}$$

For the optimal solution (w^p, b^p) or α^p of problem $(\mathcal{X}_p,\ \phi)$, the basis is the necessary support vector set, $\mathcal{SV}_p$. The violators of a partial problem are the vectors that violate the Karush-Kuhn-Tucker (KKT) necessary and sufficient optimality conditions. The KKT conditions for the entire problem (5.1) and (5.2) are listed as follows:

$$y_i(w^{\star T} x_i + b^\star) \geq 1 - \xi_i^\star, \quad \xi_i^\star \geq 0, \quad (\gamma - \alpha_i^\star)\xi_i^\star = 0,\ \forall i,$$

$$0 \leq \alpha^\star \leq \gamma \mathbf{1}, \quad y^T \alpha = 0, \quad w = \sum_{i=1}^{N} (\alpha_i^\star y_i x_i),$$

where $w^\star$, $b^\star$, $\xi^\star$ and $\alpha^\star$ are the optimal solution of the problem (5.1) and (5.2) respectively.

If we define ξ_i and α_i for the training vector x_i to be 0 for $x_i \in \mathcal{X} \backslash \mathcal{X}_p$, the only condition needed to be tested is

$$y_i(w^{\star T} x_i + b^\star) \geq 1,\ x_i \in \mathcal{X} \backslash \mathcal{X}_p.$$

Any training vector that violates the above conditions is called a violator to the partial problem $(\mathcal{X}_p,\ \phi)$. Let $\mathcal{V}(\mathcal{X}_p)$ denote violators of $\mathcal{X}_p$ and $\mathcal{E}(\mathcal{X}_p)$ denote extremes of $\mathcal{X}_p$. Violators and extremes satisfy the following properties:

$$\begin{aligned} \mathcal{V}(\mathcal{X}_p) &:= \{x \in \mathcal{X} \backslash \mathcal{X}_p \mid \phi(\mathcal{X}_p \cup \{x\}) \neq \phi(\mathcal{X}_p)\}, \\ \mathcal{E}(\mathcal{X}_p) &:= \{x \in \mathcal{X}_p \mid \phi(\mathcal{X}_p \backslash \{x\}) \neq \phi(\mathcal{X}_p)\}. \end{aligned}$$

Therefore we have

$$x \text{ violates } \mathcal{X}_p \;\Leftrightarrow\; x \text{ is extreme in } \mathcal{X}_p \cup \{x\}.$$

For a set of random samples $\mathcal{R}$ of size r that can be uniformly sampled, we consider the expected values

$$\begin{aligned} v_r &:= E_{|\mathcal{R}|=r}(|\mathcal{V}_\mathcal{R}|) \\ e_r &:= E_{|\mathcal{R}|=r}(|\mathcal{E}_\mathcal{R}|) \end{aligned}$$

The *combinatorial dimension* of ($\mathcal{X}$,ϕ), denoted by δ, is the size of largest basis of $\mathcal{X}$. The size of the largest basis, δ is naturally the largest number of support vectors for all subproblems ($\mathcal{X}_p$, ϕ), $\mathcal{X}_p \subseteq \mathcal{X}$. For separable problems, δ is bounded by one plus the lifted dimension, i.e., $\delta \leq n + 1$. For general nonseparable problems, we do not know a bound for δ before we actually solve the problem. What we can do is to set a sufficiently large number to bound δ from above.

The Sampling Lemma

Gartner proved the following sampling lemma [GW00]:

Lemma 1 *(Sampling Lemma). For* $0 \leq r < N$,

$$\frac{v_r}{N-r} = \frac{e_{r+1}}{r+1}.$$

Proof. By definition, we have

$$\begin{aligned}\begin{pmatrix} N \\ r \end{pmatrix} v_r &= \sum_{\mathcal{R}} \sum_{x \in \mathcal{X} \setminus \mathcal{R}} [x \text{ violates } \mathcal{R}] \\ &= \sum_{\mathcal{R}} \sum_{x \in \mathcal{X} \setminus \mathcal{R}} [x \text{ is extreme in } \mathcal{R} \cup \{x\}] \\ &= \sum_{\mathcal{Q}} \sum_{x \in \mathcal{Q}} [x \text{ is extreme in } \mathcal{Q}] \\ &= \begin{pmatrix} N \\ r+1 \end{pmatrix} e_{r+1},\end{aligned}$$

where [.] is the indicator variable for the event in brackets and the last row follows the fact that the set $\mathcal{Q}$ has $r+1$ elements. The Lemma immediately follows. □

For an LP-type problem ($\mathcal{X}$,ϕ) with combinatorial dimension δ, the sampling lemma yields

$$v_r \leq \delta \frac{N-r}{r+1}. \tag{5.4}$$

It follows that $|\mathcal{E}(\mathcal{R})| \leq \delta$.

5.3 Algorithm

We consider the following problem: the training data are distributed in $C+1$ sites, where there are C working sets and 1 nonworking set. Each working site is assigned a priority number $p = 1, 2, ..., C$. We also assume that each working site contains r training vectors, where $r \geq 6\delta^2$ and δ denotes the combinatorial dimension of the SVM problem.

Define a function $u(.)$ to record the number of copies of elements of a training set. For training set $\mathcal{X}$, we define a set $\mathcal{W}$ such that $\mathcal{W}$ contains the virtually duplicated copies of the training vectors. We have $|W| = u(\mathcal{X})$. We also define the virtual set $\mathcal{W}_p$ corresponding to training set $\mathcal{X}_p$ at site p.

Our parallel randomized support vector machine (PRSVM) works as follows.

Initialization

Training vectors $\mathcal{X}$ are randomly distributed to $C+1$ sites. Assign priorities to all sites such that each site gets a unique priority number. Set $u(\{x_i\}) = 1,\ \forall i$. Hence, $u(\mathcal{X}) = N$. We have $|\mathcal{X}_p| = |\mathcal{W}_p|$ for all p. Set $t = 0$.

Iteration

Each iteration consists of the following steps.

Repeat for $t = 1, 2, ...$

1. Randomly distribute the training vectors over the working sites according to $u(\mathcal{X})$ as follows. Let $\mathcal{S}^1 = \mathcal{W}$.

   For $p = 1 : C$

   Choose r training vectors, $\mathcal{W}_p$ from $\mathcal{S}^p$ uniformly (and make sure $r \geq 6\delta^2$);

   $\mathcal{S}^{p+1} := \mathcal{S}^p \backslash \mathcal{W}_p$;

   End For

2. Each site with priority p, $p \leq C$ solves the partial problem (5.3). Record the solution (w^p, b^p) for site p. Send this solution to all other sites q, $q \neq p$.

3. Each site with priority q, $q = 1, ..., C+1$, checks the solution (w^p, b^p) from site with higher priority $p, p < q$. Define $\mathcal{V}_{q,p}$ to be the training vectors in the site with priority q that violate the KKT condition corresponding to solution (w^p, b^p), $q \neq p$. That is,

$$\mathcal{V}_{q,p} := \{x_i | y_i(w^{pT} x_i + b^p) < 1, x_i \in \mathcal{X}_q, x_i \notin \mathcal{X}_p\}$$

   .

4. **If** $\sum_{q=p+1}^{C+1} u(\mathcal{V}_{q,p}) \leq |\mathcal{S}^p|/(3\delta)$ **then** $u(\{x_i\}) = 2u(\{x_i\})$, for all $x_i \in \mathcal{V}_{q,p}$, $\forall q \neq p$, $\forall p$;

until $\cup_{q \neq p} \mathcal{V}_{q,p} = \emptyset$ for some p.

Return the solution (w^p, b^p).

The priority setting of working sets actually defines the order of sampling. The highest priority server gets the first sampled batch of data, lower one gets the second batch and so on. This kind of sequential behavior is designed to help define violators and extremes clearly under a multiple working site configuration.

Step 2 involves a merging procedure. If $u(\{x_i\})$ copies of vector x_i are sampled to a working set $\mathcal{W}_p$, only one copy of x_i is included in the optimization problem $(\mathcal{X}_p, \phi)$ that we are solving, while we record this number of copies as a weight of this training vector.

The merging procedure has two properties:

Property 1 *A training vector that is not in working set $\mathcal{X}_p$ must not be a violator of the problem $(\mathcal{X}_p, \phi)$ if one or more copies of this vector are included in the working set $\mathcal{X}_p$. That is, $x_i \notin \mathcal{V}(\mathcal{X}_p)$, if $x_i \in \mathcal{X}_p$.*

Property 2 *If multiple copies of a vector x_i are sampled to a working set $\mathcal{X}_p$, none of those of vectors can be the extreme of the problem $(\mathcal{X}_p, \phi)$. That is, $x_i \notin \mathcal{E}(\mathcal{X}_p)$ if $u(\{x_i\}) > 1$ at site p.*

The above two properties follow immediately by definitions of violators and extremes.

One may note that the merging procedure actually constructs an abstract problem $(\mathcal{W}_p, \phi')$ such that $\phi'(\mathcal{W}_p) = \phi(\mathcal{X}_p)$. By definition, $(\mathcal{W}_p, \phi')$ is a LP-type

problem and has the same combinatorial dimension, δ, as the problem $(\mathcal{X}_p, \phi)$. If the set of violators of $(\mathcal{X}_p, \phi)$ is $\mathcal{V}_p$, the number of violators of $(\mathcal{W}_p, \phi')$ is $u(\mathcal{V}_p)$.

Step 4 plays the key role in this algorithm. It says that if the number of violators of the LP-type problem $(\mathcal{W}_p, \phi')$ is not too large, we double the weights of the violators of $(\mathcal{W}_p, \phi')$ in all sites. Otherwise, we keep the weights untouched since the violators already have enough weights to be sampled to a working site.

One may note when $C = 1$, the PRSVM is reduced to the RSVM. However, our RSVM is different from the randomized support vector machine training algorithm in [BDT01] in several ways. First, our RSVM is capable of solving general nonseparable problems, while Balcazar's method has to transfer nonseparable problems to an equivalent separable problems by assuming an arbitrarily small γ. Second, our RSVM merges examples after sampling them. Duplicated examples are not allowed in the optimization steps. Third, we test the KKT conditions to identify a violator instead of identifying a misclassified point. In our RSVM, a correctly classified example may also be a violator if this example violates the KKT condition.

5.4 Proof of the Average Convergence Rate

We prove the average number of iterations executed in our algorithm, PRSVM, is bounded by $(6\delta/C)\ln(N + 6r(C-1)\delta)$ in this section. This proof is a generalization of the one given in [BDT01]. The result in [BDT01] becomes a special case of our PRSVM.

Theorem 4 *For a general SVM training problem the average number of iterations executed in the PRSVM algorithm is bounded by* $(6\delta/C)\ln(N+6r(C-1)\delta)$.

Proof. We consider an update to be successful if the if-condition in the step 4 holds in an iteration. One iteration has C updates, successful or not.

We first show the bound of the number of successful updates. Let $\mathcal{V}_p$ denote the set of violators from sites with priority $q \geq p$ for the solution (w^p, b^p), namely

$$\mathcal{V}_p := \bigcup_{q \geq p} \mathcal{V}_{q,p}.$$

By this definition, we have

$$u(\mathcal{V}_p) = \sum_{q=p+1}^{C+1} u(\mathcal{V}_{q,p}). \tag{5.5}$$

Since the if-condition holds, we have

$$\sum_{q=p+1}^{C+1} u(\mathcal{V}_{q,p}) \leq |\mathcal{S}^p|/(3\delta) \leq u(\mathcal{X})/(3\delta). \tag{5.6}$$

By noting that the total number of training vectors including duplicated ones in each working site is always r, we have

$$\sum_{q=1}^{p-1} u(\mathcal{V}_{q,p}) \leq r(p-1) \leq r(C-1) \tag{5.7}$$

and

$$\begin{array}{rcl} \sum_{q \neq p} u(\mathcal{V}_{q,p}) & = & \sum_{q=p+1}^{C+1} u(\mathcal{V}_{q,p}) + \sum_{q=1}^{p-1} u(\mathcal{V}_{q,p}) \\ & = & u(\mathcal{V}_p) + \sum_{q=1}^{p-1} u(\mathcal{V}_{q,p}) \end{array} \tag{5.8}$$

Let $u_k(\mathcal{X})$ denote the total weights $u(\mathcal{X})$ after k successful updates. At each successful update, we have

$$u_k(\mathcal{X}) \leq u_{k-1}(\mathcal{X})(1 + \frac{1}{3\delta}) + 2r(C-1).$$

It follows by doubling (5.8) and substituting (5.5), (5.6) and (5.7) into (5.8). Since $u_0(\mathcal{X}) = N$, after k successful updates, we have

$$\begin{array}{rcl} u_k(\mathcal{X}) & \leq & N(1 + \frac{1}{3\delta})^k + 2r(C-1)3\delta[(1 + \frac{1}{3\delta})^k - 1] \\ & < & (N + 6r(C-1)\delta)(1 + \frac{1}{3\delta})^k. \end{array}$$

Let $\mathcal{X}_0$ be the set of support vectors of the original problem (5.1) or (5.2). At each successful iteration, some x_i of $\mathcal{X}_0$ must not be in $\mathcal{X}_p$. Hence, $u(\{x_i\})$ gets doubled. Since, $|\mathcal{X}_0| \leq \delta$, there is some x_i in $\mathcal{X}_0$ that gets doubled at least once every δ successful updates. That is, after k successful updates, $u(\{x_i\}) \geq 2^{k/\delta}$.

Therefore, we have

$$2^{\frac{k}{\delta}} \leq u(\mathcal{X}) \leq (N + 6r(C-1)\delta)(1 + \frac{1}{3\delta})^k.$$

By simple algebra, we have

$$k \leq 3\delta \ln(N + 6r(C-1)\delta).$$

That is, the algorithm terminates within less than $3\delta \ln(N+6r(C-1)\delta)$ successful updates.

The rest is to prove that the probability of a successful update is higher than one half. By sampling lemma, the bound (5.4), we have

$$\begin{aligned} E(u(\mathcal{V}_p)) &\leq \frac{(|\mathcal{S}^p|-r)\delta}{r+1} \\ &< \frac{|\mathcal{S}^p|}{6\delta}. \end{aligned}$$

By the Markov equality, we have

$$\begin{aligned} & \Pr\{u(\mathcal{V}_p) \leq \tfrac{|\mathcal{S}^p|}{3\delta}\} \\ \geq\ & \Pr\{u(\mathcal{V}_p) \leq 2E(u(\mathcal{V}_p))\} \\ \geq\ & \tfrac{1}{2}. \end{aligned}$$

This implies that the expected number of updates is at most twice as large as the number of successful updates, i.e., $k \leq 6\delta \ln(N + 6r(C-1)\delta)$, where k denotes the total number of updates. Note that, at the end of each iteration, we have

$$k = Ct.$$

Therefore, the PRSVM algorithm guarantees, on average, within $(6\delta/C)\ln(N + 6r(C-1)\delta)$ steps, that all the support vectors are contained by one of the C working sites. For separable problems, we have $\delta \leq n + 1$. For general nonseparable problems, δ is bounded by the number of support vectors. $\square$

The bound of average convergence rate $(6\delta/C)\ln(N + 6r(C-1)\delta)$ clearly shows the linear scalability if $N \gg \delta$. This can be true if the number of support vectors is very limited.

5.5 Simulations and Applications

We analyze the PRSVM by using synthesized data and a real-world geographic information system (GIS) database.

Through out this section, the machine we use has a Pentium IV 2.26G CPU and 512M RAM. The operation system is Windows XP. The SVM$^{\text{light}}$ [Joa98] version 6.01 was used as the local SVM solver. Parallel computing is virtually simulated in a single machine. We ignore any communication overhead.

5.5.1 Synthesized Demonstration

We demonstrate our RSVM (reduced PRSVM when $C = 1$) training procedure by using a synthesized two-dimensional training data set. This data set consists of 1000 data points: 500 positive and 500 negative. Each class of data are generated from an independent Gaussian distribution. Random noise is added.

We set the sample size r to be 100 and the regularization factor γ to be 0.2. The RSVM converges in 13 iteration. In order to demonstrate the weighting procedure, we choose three iterations (iteration 1, iteration 6 and iteration 13) and plot the weights of the training vectors in Figure 5.1. The darker a point

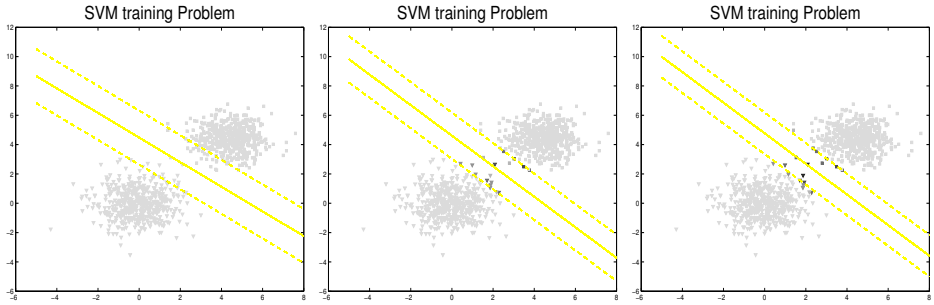

Figure 5.1: Weights of training vectors in iterations. Darker points denote higher weights. *Left.* The first iteration. *Middle.* The sixth iterations. *Right.* The last iteration. The figures demonstrate how those support vectors get higher and higher weights during iterations.

appears, the higher weight the training sample has. Figure 5.1 shows that how those "important" points stand out and get higher and higher probability to be sampled.

5.5.2 Application in a Geographic Information System Database

We select covtype, a geographic information system database, from the UCI Repository of machine learning databases as our PRSVM applications [BM98]. The covtype database consists of 581,012 instances. There are 12 measurements but 54 columns of data: 10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables [BD99]. There are totally 7 classes. We scale all quantitative variables to [0,1] and keep binary variable unchanged. We select 287831 training vectors and use our PRSVM to classify class 4 against the rest. This is a very suitable database for testing PRSVM since the database has huge number of training data and the number of SVs is limited.

We set the size of working size r to be 60000, the regularization factor γ to be

Algorithm	C	Number of Iterations	Learning Time (CPU Seconds)
SVM$^{\text{light}}$	1	-	11.7
RSVM	1	27	47.32
PRSVM	2	10	20.81
	4	7	15.52

Table 5.1: Algorithm performance comparison of SVM$^{\text{light}}$, RSVM and PRSVM. This table shows that the PRSVM really takes the advantage of the multiple servers. When the number of servers are limited, the salability of the PRSVM is good. Though the PRSVM is still not as good as the SVM$^{\text{light}}$, lack of a provable convergence rate makes the later one not always preferable.

0.2. We try three cases with $C = 1$, $C = 2$ and $C = 4$ and compare the computing time with the SVM$^{\text{light}}$ in Table 5.1. The results show that our implementation of RSVM and PRSVM achieves comparable result with the reported fastest algorithm SVM$^{\text{light}}$, though they cannot beat SVM$^{\text{light}}$ in terms of computing speed in this application. However, the lack of a theoretical convergence bound makes SVM$^{\text{light}}$ not always preferable.

We plot the number of violators and support vectors (extremes) in each iterations in Figure 5.2 to compare the performance of different number of working sites. The results show the scalability of our method. The numerical results match the theoretical result very well.

5.6 Conclusions

The proposed PRSVM has the following advantages over the previous works. It is able to solve general nonseparable SVM training problems. This is achieved by using KKT condition as the criterion of identifying violators and extremes. Second, our algorithm supports multiple working sets that may work parallel. Multiple working sets have more freedom than normal gradient based parallel algorithm since no synchronization and no special solver is required. Our PRSVM also has a provably scalable and fast average convergence bound. Finally, our numerical results show that multiple working sets have scalable computing advantage. The provable convergence bound and scalable results make our algorithm more preferable in some real time applications.

Further research is going to be conducted to accelerate the performance of the PRSVM. Intuitively, the weighting mechanism may be able to be improved so that the initial iteration plays more determinant role.

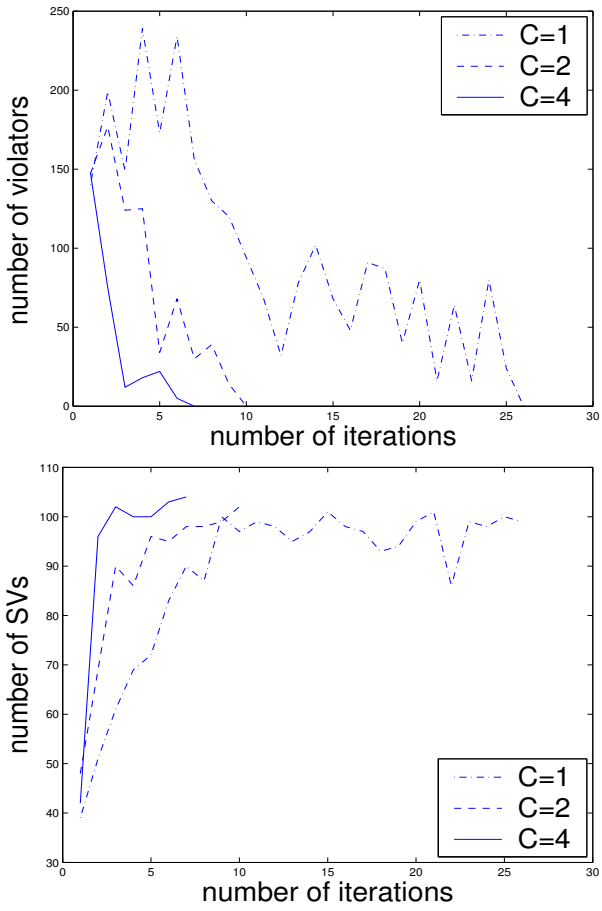

Figure 5.2: Number of violators and SVs found in each iteration of PRSVM. These figures demostrate the effect of using more servers. The system with more servers will find the violator and support vectors much faster than that with less servers.

CHAPTER 6

Maximum Likelihood Estimation of Gaussian Mixture Models

6.1 Introduction

We consider the problem of estimating the parameters in mixture distributions of the form

$$p(y) = \sum_{j=1}^{n} w_j f_j(y, \mu).$$

where

$$f_j(y, \mu) = \frac{1}{\sqrt{2\pi s_j}} e^{-\frac{(y-\mu_j)^2}{2s_j}} \tag{6.1}$$

with parameters μ, which are constrained to lie in some convex set S. The weights w are also unknown and must satisfy $w \succeq 0$ and $\mathbf{1}^T w = 1$. The variance vector s is known. Therefore, the density function f_j is a log concave function in μ.

Our goal is to estimate w and μ, based on an observed values $y_1, \ldots, y_N$ using the maximum likelihood principle. The ML estimation problem is

$$\begin{array}{ll} \text{maximize} & L(w, \mu) \\ \text{subject to} & \mathbf{1}^T w = 1, \quad w \succeq 0, \quad \mu \in S, \end{array} \tag{6.2}$$

where the *log-likelihood function* L is defined as

$$L(w, \mu) = \sum_{k=1}^{N} \log \sum_{j=1}^{n} w_j f_j(y_k, \mu).$$

In general, L is a complicated non-concave function. It is obviously concave in w for fixed μ. However, it is not concave in μ, not even when f_j is log-concave in μ because a sum of log-concave functions is not necessarily log-concave.

Dempster, Laird, and Rubin (1977) published their fundamental paper and introduced the Expectation-Maximization (EM) algorithm for maximum likelihood estimation from incomplete data [DLR97]. Thereafter, the EM algorithm for maximum likelihood estimation has been widely used and extensively studied [MK96, RWD84]. The EM algorithm estimates the parameters iteratively. Each iteration consists of an Expectation step, which computes the distribution for the set of unobserved variables, given the current estimation of the parameters, and a Maximization step, which estimates the parameters, given the estimation of the unobserved variables. Although the EM algorithm has been successfully applied in variety of contexts, it has two major limitations. One is that it can be very slow to converge in certain situations [MK96]; the other is that EM is often trapped into a local optimum.

Several authors have focused on accelerating convergence of the EM algorithm. Meng and Rubin (1993) introduced the ECM algorithm, which replaces the Maximization step by a number of simpler Conditional Maximization steps to achieve faster overall computing time [MR93]]. This algorithm was further extended to the ECME algorithm proposed by Liu and Rubin (1994) [LR94]. The ECME improves the speed of convergence by conditionally maximizing on the CM-steps the actual log likelihood instead of the Q-function with EM and ECM algorithm. The SAGE algorithm proposed by Fessler and Hero (1994) updates the parameter sequentially by alternating between several small hidden-data spaces [FH94]. Neal and Hinton [NH98] proposed an incremental EM algorithm, which updates only a subset of unobservable variables in each Expectation so that the

rate of convergence can be much greater than standard algorithm. Their algorithm already has some successful application [Now03]. Neal and Hinton (1998) also proposed a sparse variant of the EM algorithm which fixes the probabilities of the implausible values for many iterations, when we have prior bias for the unobserved variable [NH98]. Csiszar and Tusnady (1984), Hathaway (1986), and Neal and Hinton (1998) view the EM algorithm as maximizing a joint function of the parameters and of the distribution over the unobserved variables that is analogous to the Kullback-Leibler divergence [CT84, Hat86, NH98]. Matsuyama proposed the α-EM algorithm to achieve faster rate of convergence by replacing the log function in the KL-divergence by a function with parameter α [Mat00]. The speedup is due to the parameter α's effect on the eigenvalues of the Hessian matrix.

To the best of our knowledge, however, there is very limited literature trying to search the global maximum likelihood. Vempala and Wang (2002) proposed a spectral algorithm to avoid local optimum searching for learning mixtures of distributions [VW02]. The basic idea of this algorithm is to project the data to a lower dimensional space without losing much discriminant information, then infer the parameters using projected data. This algorithm, however, assumes a strong condition for the mixture and mean and variance vector and cannot guarantee to converge to a global optimum. Liu and Mahmassani (2000) used the genetic algorithm to globally search maximum likelihood [LM00]. Unfortunately, their method cannot guarantee global optimum either. Slump and Honeners (1985) search the global maximum by determining the number of stationary points in a certain region with the Kronecker-Picard (KP) integral [SH85]. Their method may become prohibitively complicated for numerical evaluation in case of many parameters.

6.2 ML Estimation of Gaussian Mixture Distributions

We first introduce the basics of Kullback-Leibler divergence and normalized entropy as a preparation for our ML estimation reformulation.

6.2.1 Kullback-Leibler divergence and normalized entropy

Definitions

The *Kullback-Leibler divergence* of two vectors $x, y \in \mathbf{R}^N$ is defined as

$$D_{\mathrm{kl}}(x,y) = \sum_{i=1}^{N} x_j \log(x_j/y_j) - \mathbf{1}^T x + \mathbf{1}^T y,$$
$$\mathbf{dom}\, D_{\mathrm{kl}} = \mathbf{R}^N_+ \times \mathbf{R}^N_{++}.$$

It can be shown that D_{kl} is convex, jointly in x and y. Moreover $D_{\mathrm{kl}}(x,y) \geq 0$ with $D_{\mathrm{kl}}(x,y) = 0$ if and only if $x = y$. (This is known as the information inequality.)

If X and Y are two matrices in $\mathbf{R}^{N\times n}$, we define $D_{\mathrm{kl}}(X,Y)$ as

$$D_{\mathrm{kl}}(X,Y) = \sum_{i=1}^{N}\sum_{j=1}^{n} X_{ij} \log(X_{ij}/Y_{ij}) - \mathbf{1}^T X \mathbf{1} + \mathbf{1}^T Y \mathbf{1},$$
$$\mathbf{dom}\, D_{\mathrm{kl}} = \mathbf{R}^{N\times n}_+ \times \mathbf{R}^{N\times n}_{++},$$

i.e., the sum of the pairwise KL-divergences of the columns of X and Y. We have $D_{\mathrm{kl}}(X,Y) \geq 0$ with equality if $X = Y$.

The *normalized entropy* of a nonzero vector $x \in \mathbf{R}^N_+$ is defined as

$$\begin{aligned} E_{\mathrm{n}}(x) &= \sum_{i=1}^{N} x_i \log(\mathbf{1}^T x/x_i), \qquad \mathbf{dom}\, E_{\mathrm{n}} \\ &= \mathbf{R}^N_+ \setminus \{0\}. \end{aligned}$$

This is a concave function of x, because

$$E_{\mathrm{n}}(x) = -D_{\mathrm{kl}}(x, (\mathbf{1}^T x)\mathbf{1}) + (N-1)\mathbf{1}^T x.$$

The normalized entropy of a matrix $X \in \mathbf{R}^{N\times n}$ is defined as the sum of the normalized entropies of its columns X_k:

$$\begin{aligned} E_{\mathrm{n}}(X) &= \sum_{k=1}^{n} E_{\mathrm{n}}(X_k) \\ &= -\sum_{k=1}^{n}\sum_{i=1}^{N} x_{ik}\log x_{ik} + \sum_{k=1}^{n}(\sum_{i=1}^{N} x_{ik})\log(\sum_{i=1}^{N} x_{ik}). \end{aligned}$$

The conjugate of $-E_{\mathrm{n}}$

The conjugate of $-E_{\mathrm{n}}$ on $\mathbf{R}^N$ is

$$(-E_{\mathrm{n}})^*(y) = \sup_x(y^Tx + E_{\mathrm{n}}(x)) = \begin{cases} 0 & \sum_{i=1}^{N} e^{y_i} \leq 1 \\ -\infty & \text{otherwise.} \end{cases}$$

This can be seen as follows. Suppose $\sum_{i=1}^{N} e^{y_i} > 1$. Define $x_i = te^{y_i}$, $i = 1,\ldots,N$. We have

$$y^Tx + E_{\mathrm{n}}(x) = \sum_{i=1}^{N} x_i\log(e^{y_i}\mathbf{1}^Tx/x_i) = t\sum_{i=1}^{N} e^{y_i}\log(\sum_{j=1}^{N} e^{y_j})$$

which increase without bound as $t \to \infty$. Next, suppose $\sum_{i=1}^{N} e^{y_i} \leq 1$. We first show that this implies that $y^Tx + E_{\mathrm{n}}(x) \leq 0$ for all x. The function $y^Tx + E_{\mathrm{n}}(x)$ is homogeneous in x, so it is sufficient to show that it is nonnegative if $\mathbf{1}^Tx = 1$:

$$\begin{aligned} y^Tx + E_{\mathrm{n}}(x) &= y^Tx - \sum_{i=1}^{N} x_i\log x_i \\ &= \sum_{i=1}^{N} x_i\log(z_i/x_i) \\ &= -D_{\mathrm{kl}}(x,z) - 1 + \mathbf{1}^Tz \\ &\leq -D_{\mathrm{kl}}(x,z) \\ &\leq 0, \end{aligned}$$

where $z_i = \exp(y_i)$. This follows from the information inequality and $\mathbf{1}^Tz \leq 1$. Since $y^Tx + E_{\mathrm{n}}(x)$ is homogeneous, the inequality holds for all nonzero $x \in \mathbf{R}^n_+$. Moreover, equality holds if $x = z$, i.e., $x_i = \exp(y_i)$ for $i = 1,\ldots,N$.

The conjugate of $-E_{\rm n}$ on $\mathbf{R}^{N\times n}$ is

$$(-E_{\rm n})^*(Y) = \begin{cases} 0 & \sum_{j=1}^N \exp(y_{jk}) \leq 1, \forall k \\ -\infty & \text{otherwise.} \end{cases}$$

6.2.2 Reformulation as a bi-convex optimization problem

We can reformulate the ML estimation problem as a bi-concave maximization problem as follows. We introduce a matrix variable $P \in \mathbf{R}^{N\times n}$, and define a function $F : \mathbf{R}^{N\times n} \times \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$, as

$$F(P, w, \mu) = L(w, \mu) - D_{\rm kl}(P, \hat{P}(w, \mu)) \tag{6.3}$$

where $D_{\rm kl}$ is the Kullback-Leibler divergence, and $\hat{P}(w, \mu) \in \mathbf{R}^{N\times n}$ with

$$\hat{P}_{kj}(w, \mu) = \frac{w_j f_j(y_k, \mu)}{\sum_{j=1}^n w_j f_j(y_k, \mu)}, \forall k, j. \tag{6.4}$$

We take as domain of F the set

$$\begin{aligned} \mathbf{dom}\, F = \{(P, w, \mu) \in \mathbf{R}^{N\times n} \times \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \mid \\ P \succeq 0, P\mathbf{1} = \mathbf{1}, w \succ 0, \mathbf{1}^T w = 1\}. \end{aligned}$$

This function can be simplified as follows:

$$\begin{aligned} F(P, w, \mu) &= L(w, \mu) - D_{\rm kl}(P, \hat{P}(w, \mu)) \\ &= \sum_{k=1}^N \log \sum_{j=1}^n w_j f_j(y_k, \mu) - \sum_{k=1}^N \sum_{j=1}^n P_{kj} \log P_{kj} \\ &\quad + \sum_{k=1}^N \sum_{j=1}^n P_{kj} \log \frac{w_j f_j(y_k, \mu)}{\sum_{j=1}^n w_j f_j(y_k, \mu)} \\ &= -\sum_{k=1}^N \sum_{j=1}^n P_{kj} \log P_{kj} \\ &\quad + \sum_{k=1}^N \sum_{j=1}^n P_{kj} \left(\log w_j + \log f_j(y_k, \mu)\right). \end{aligned} \tag{6.5}$$

(On line 3 we assume that $P\mathbf{1} = \mathbf{1}$.) This shows that F is bi-concave: if we fix w and μ, it is concave in P; if we fix P, then F is concave jointly in μ and w.

From the information inequality and the definition (6.3) it is clear that

$$\sup_{P\mathbf{1}=\mathbf{1}} F(P, w, \mu) = L(w, \mu),$$

and that the maximum is achieved by $P = \hat{P}(w, \mu)$. We can conclude that the ML estimation problem (6.2) is equivalent to the bi-concave maximization problem

$$\begin{array}{ll} \text{maximize} & F(P, w, \mu) \\ \text{subject to} & \mathbf{1}^T w = 1, \quad P\mathbf{1} = \mathbf{1}, \quad \mu \in S, \end{array} \tag{6.6}$$

with variables $P \in \mathbf{R}^{N\times n}$ and $w, \mu \in \mathbf{R}^n$.

We can also simplify (6.5) further and eliminate w. For fixed P the maximum of F over w is attained at

$$w_j = \frac{\sum_{k=1}^N P_{kj}}{\mathbf{1}^T P \mathbf{1}} = \frac{1}{N}\sum_{k=1}^N P_{kj},$$

which gives

$$\begin{aligned} &\sup_{\mathbf{1}^T w=1} F(P, w, \mu) \\ = \quad & -N\log N + \textstyle\sum_{k=1}^N \sum_{j=1}^n P_{kj} \log \frac{\sum_{k=1}^N P_{kj}}{P_{kj}} \\ & + \textstyle\sum_{k=1}^N \sum_{j=1}^n P_{kj} \log f_j(y_k, \mu) \\ = \quad & E_{\mathrm{n}}(P) + \textstyle\sum_{k=1}^N \sum_{j=1}^n P_{kj} \log f_j(y_k, \mu) - N\log N, \end{aligned}$$

where E_{n} is the normalized entropy, i.e., a concave function of P, that can be solved efficiently by a GP proved in Lemma 2.

Lemma 2 Normalized Entropy Maximization

The optimization problems of the form

$$\begin{array}{ll} \textit{maximize} & E_{\mathrm{n}}(X) + \min_{i=1,\ldots,l} \mathbf{Tr}(C_i^T X + d_i) \\ \textit{subject to} & \mathbf{Tr}(A_i^T X) \leq b_i, \quad i = 1, \ldots, m \\ & X\mathbf{1} = \mathbf{1}, \end{array} \tag{6.7}$$

with variable $X \in \mathbf{R}^{N \times n}$, has the corresponding dual formula:

$$\begin{array}{ll} \textit{minimize} & b^T\lambda + d^T\mu + \mathbf{1}^T\nu \\ \textit{subject to} & \log\left(\sum_{j=1}^N \exp(A(\lambda,\mu,\nu)_{jk})\right) \leq 0, \quad k = 1, \ldots, n \\ & \lambda \succeq 0, \quad \mu \succeq 0 \\ & \mathbf{1}^T\mu = 1, \end{array} \tag{6.8}$$

where the variables are $\lambda \in \mathbf{R}^m$, $\mu \in \mathbf{R}^l$, $\nu \in \mathbf{R}^N$, and

$$A(\lambda,\mu,\nu) = \sum_{i=1}^{l} \mu_i C_i - \sum_{i=1}^{m} \lambda_i A_i - \nu\mathbf{1}^T.$$

Proof. To derive this dual we first reformulate (6.7) as

$$\begin{array}{ll} \text{minimize} & -E_{\mathrm{n}}(X) - t \\ \text{subject to} & \mathbf{Tr}(A_i^T X) \leq b_i, \quad i = 1, \ldots, m \\ & \mathbf{Tr}\, C_i(X) + d_i \geq t, \quad i = 1, \ldots, l \\ & X\mathbf{1} = \mathbf{1}, \end{array}$$

with an auxiliary variable $t \in \mathbf{R}$. The Lagrangian is

$$\begin{array}{rl} & L(X,t,\lambda,\mu,\nu) \\ = & -E_{\mathrm{n}}(X) - t + \sum_{i=1}^{m} \lambda_i(\mathbf{Tr}(A_i^T X) - b_i) \\ & -\sum_{i=1}^{l} \mu_i(\mathbf{Tr}\, C_i(X) + d_i - t) + \nu^T(X\mathbf{1} - \mathbf{1}) \\ = & -E_{\mathrm{n}}(X) - \mathbf{Tr}(A(\lambda,\mu,\nu)^T X) - t(1 - \mathbf{1}^T\mu) \\ & -b^T\lambda - d^T\mu - \mathbf{1}^T\nu. \end{array}$$

Minimizing over X and t gives the following expression for the dual function:

$$\begin{aligned} & g(\lambda, \mu, \nu) \\ = & \begin{cases} -(-E_n)^*(A(\lambda, \mu, \nu) - b^T\lambda - d^T\mu - \mathbf{1}^T\nu & \mathbf{1}^T\mu = 1 \\ -\infty & \text{otherwise.} \end{cases} \end{aligned}$$

The dual of problem (6.7) is therefore

$$\begin{array}{ll} \text{minimize} & (-E_{\mathrm{n}})^*(A(\lambda, \mu, \nu)) + b^T\lambda + d^T\mu + \mathbf{1}^T\nu \\ \text{subject to} & \lambda \succeq 0, \quad \mu \succeq 0 \\ & \mathbf{1}^T\mu = 1. \end{array}$$

More explicitly,

$$\begin{array}{ll} \text{minimize} & b^T\lambda + d^T\mu + \mathbf{1}^T\nu \\ \text{subject to} & \sum_{j=1}^N \exp(A(\lambda, \mu, \nu)_{jk}) \leq 1, \quad k = 1, \ldots, n \\ & \lambda \succeq 0, \quad \mu \succeq 0 \\ & \mathbf{1}^T\mu = 1, \end{array}$$

which further simplifies to (6.8). Note that the dual problem is always strictly feasible. □

In conclusion, the ML problem (6.2) is equivalent to

$$\begin{array}{ll} \text{maximize} & E_{\mathrm{n}}(P) + \sum_{k=1}^N \sum_{j=1}^n P_{kj} \log f_j(y_k, \mu) \\ \text{subject to} & P\mathbf{1} = \mathbf{1}, \quad \mu \in S. \end{array} \tag{6.9}$$

6.2.3 The EM algorithm

The EM algorithm attempts to solve (6.6) by alternating maximization. It is also a greedy bi-level optimization algorithm of solving (6.6).

- In the *E-step*, the variables w and μ are fixed, and we maximize over P. This is a concave maximization problem with a simple analytical solution, given by (6.4). This follows directly from (6.3).

- In the *M-step*, the variable P is fixed and we maximize over w and μ. The maximum over w can be determined analytically, by solving

$$\begin{array}{ll} \text{maximize} & \sum_{j=1}^{n}(\sum_{k=1}^{N} P_{kj}) \log w_j \\ \text{subject to} & \mathbf{1}^T w = 1. \end{array}$$

  As already mentioned, the solution is

$$w_j = \frac{\sum_{k=1}^{N} P_{kj}}{\mathbf{1}^T P \mathbf{1}} = \frac{1}{N} \sum_{k=1}^{N} P_{kj}.$$

  The optimal μ can be determined numerically (or in some simple cases, analytically), by solving a convex optimization problem.

6.2.4 A Bi-Convex Formulation for Gaussian Mixture Models with known variances

We work out the details for Gaussian distributions on $\mathbf{R}$,

$$f_j(y) = \frac{1}{\sqrt{2\pi s_j}} e^{-(y-\mu_j)^2/(2s_j)}, \quad j = 1, \ldots, n. \tag{6.10}$$

We assume the variances s_j are nonzero and known, and that μ is constrained to lie in the simplex

$$S = \{\mu \in \mathbf{R}^n \mid 0 \leq \mu_1 \leq \cdots \leq \mu_n \leq 1\}.$$

The maximum likelihood problem is

$$\begin{array}{ll} \text{maximize} & \sum_{k=1}^{N} \log \sum_{j=1}^{n} w_j f_j(y_k) \\ \text{subject to} & \mathbf{1}^T w = 1, \quad w \succeq 0 \\ & 0 \leq \mu_1 \leq \cdots \leq \mu_n \leq 1, \end{array} \tag{6.11}$$

where f_j is given by (6.10). We have

$$\log f_j(y, \mu_j, s_j) = -\frac{1}{2}\log(s_j) - \frac{(y-\mu_j)^2}{2s_j} - \frac{1}{2}\log(2\pi),$$

so the biconvex formulation (6.9) reduces to

$$\begin{array}{ll} \text{maximize} & E_{\mathrm{n}}(P) - (1/2)\sum_{j=1}^{n}\sum_{k=1}^{N} P_{kj}\left(\log s_j + (y_k-\mu_j)^2/s_j\right) \\ \text{subject to} & P\mathbf{1} = \mathbf{1} \\ & 0 \le \mu_1 \le \mu_2 \le \cdots \le \mu_n \le 1. \end{array} \tag{6.12}$$

The terms in the sum over i in the objective can also be written as

$$\begin{aligned} &\sum_{k=1}^{N} P_{kj}\left(\log s_j + \frac{(y_k-\mu_j)^2}{s_j}\right) \\ &= \frac{1}{s_j}((\sum_{k=1}^{N} P_{kj})(s_j \log s_j + \mu_j^2) \\ &\quad -2(\sum_{k=1}^{N} y_k P_{kj})\mu_j + (\sum_{k=1}^{N} y_k^2 P_{kj})) \\ &= \frac{1}{s_j}(\sum_{k=1}^{N} P_{kj})\left(s_j \log s_j + (\mu_j - \hat{\mu}_j(P))^2\right) \\ &\quad +\frac{1}{s_j}(\sum_{k=1}^{N} y_k^2 P_{kj} - \frac{(\sum_{k=1}^{N} y_k P_{kj})^2}{\sum_{k=1}^{N} P_{kj}}) \end{aligned} \tag{6.13}$$

where

$$\hat{\mu}_j(P) = \frac{\sum_{k=1}^{N} P_{kj} y_k}{\sum_{k=1}^{N} P_{kj}}.$$

This last expression shows that for fixed P, $\mu_j = \hat{\mu}_j(P)$ is obviously the unconstrained minimizer of the cost function. With the constraints $\mu \in S$ added, the solution is not as obvious, but can be easily found by solving the quadratic program

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{n}(\alpha_j(P)\mu_j^2 - 2\beta_j(P)\mu_j + \gamma_j(P)) \\ \text{subject to} & 0 \le \mu_1 \le \mu_2 \le \cdots \le \mu_n \le 1. \end{array} \tag{6.14}$$

where

$$\begin{aligned} \alpha_j(P) &= \frac{\sum_{k=1}^{N} P_{kj}}{2s_j}, \\ \beta_j(P) &= \frac{\sum_{k=1}^{N} P_{kj} y_k}{2s_j}, \\ \gamma_j(P) &= \frac{\sum_{k=1}^{N} P_{kj}(s_j \log s_j + y_k^2)}{2s_j}. \end{aligned}$$

6.3 Generalized Benders decomposition

Problem (6.12) can be written as

$$\begin{array}{ll} \text{maximize} & E_{\mathrm{n}}(P) - V(P) \\ \text{subject to} & P\mathbf{1} = \mathbf{1}, \end{array} \tag{6.15}$$

where $V(P)$ is the optimal value of the QP (6.14), as a function of P. This problem is not convex (although E_{n} is a concave function), because V is a concave function.

6.3.1 Piecewise-linear lower bounds on V

We first note that $\alpha_j(P) \geq 0$ for all P, so

$$\alpha_j(P)\mu_j^2 \geq 2\alpha_j(P)\hat{\mu}_j\mu_j - \alpha_j(P)\hat{\mu}_j^2$$

for all μ_j, $\hat{\mu}_j$. If we choose some $\hat{\mu}$, we have

$$\begin{aligned} V(P) &= \inf_{\mu\in C} \sum_{j=1}^{N} \left(\alpha_j(P)\mu_j^2 - 2\beta_j(P)\mu_j + \gamma_j(P)\right) \\ &\geq \sum_{j=1}^{n} \left(\gamma_j(P) - \alpha_j(P)\hat{\mu}_j^2\right) + 2\inf_{\mu\in C} \sum_{j=1}^{n} (\alpha_j(P)\hat{\mu}_j - \beta_j(P))\mu_j \\ &= \sum_{j=1}^{n} \left(\gamma_j(P) - \alpha_j(P)\hat{\mu}_j^2\right) + 2\min\{0, \min_{j=1,\ldots,n} \sum_{j=i}^{n} (\alpha_j(P)\hat{\mu}_j - \beta_j(P))\}. \end{aligned}$$

The third line is true because $\sum_{j=1}^{n}(\alpha_j(P)\hat{\mu}_j - \beta_j(P))\mu_j$ is linear in μ_j and the domain C for μ is a polyhedron with $n+1$ vertices μ^{B_i} where

$$\mu^{B_i} \in \{[0,0,...,0]^T, [0,0,...,0,1]^T, ..., [0,1,1,...,1]^T, [1,1,...,1]^T\}$$

Moreover we have equality if $\hat{\mu}$ is the solution of the QP (6.14), because then

$$\inf_{\mu\in C} \sum_{j=1}^{n} (\alpha_j(P)\hat{\mu}_j - \beta_j(P))\mu_j = \sum_{j=1}^{n} (\alpha_j(P)\hat{\mu}_j - \beta_j(P))\hat{\mu}_j.$$

Since $\alpha(P)$, $\beta(P)$, $\gamma(P)$ are linear, the lower bound (6.16) is a piecewise-linear concave function of P.

6.3.2 Algorithm

We will solve problem (6.15) by computing and updating a piecewise-linear lower bound for V. We partition the set

$$\mathcal{P} = \{P \in \mathbf{R}^{N\times n} \mid P_{kj} \geq 0, \quad P\mathbf{1} = \mathbf{1}\}$$

in K polyhedral regions $\mathcal{P}_k$, $k = 1, \ldots, K$. On each region $\mathcal{P}_k$, we have a (piecewise-linear) convex function $V_{\mathrm{lb}}^{(k)}$, which gives a lower bound for V on $\mathcal{P}_k$. For each region, we define $P_{\max}^{(k)}$ and U_k as the maximizer, resp. the optimal value, of

$$\begin{array}{ll} \text{maximize} & E_{\mathrm{n}}(P) - V_{\mathrm{lb}}^{(k)}(P) \\ \text{subject to} & P \in \mathcal{P}_k. \end{array} \tag{6.16}$$

We also store a lower bound L on the optimal value of (6.15) and (if L is finite) a matrix P_{best} with $E_{\mathrm{n}}(P_{\mathrm{best}}) - V(P_{\mathrm{best}}) = L$.

The proof of the convergence of this algorithm is in Appendix 6.4.

Initialization

We initialize the algorithm with $K = 1$, $\mathcal{P}_1 = \mathcal{P}$. We take

$$V_{\mathrm{lb}}^{(1)}(P) = -\infty, \qquad P_{\max}^{(1)} = (1/n)\mathbf{1}\mathbf{1}^T, \qquad U_1 = \infty, \qquad L = -\infty.$$

Iteration

For simplicity, we will assume that the regions are sorted so that $U_1 \leq U_2 \leq \cdots \leq U_K$. Each iteration consists of the following steps.

1. Take $P := P_{\max}^{(K)}$, and evaluate $V(P)$ by solving the QP (6.14). If $E_{\mathrm{n}}(P) - V(P) > L$, take

$$L := E_{\mathrm{n}}(P) - V(P), \qquad P_{\mathrm{best}} := P.$$

Let $\hat{\mu}$ be the solution of the QP.

2. For $j = 1, \ldots, n$, add a new region

$$\mathcal{P}_{K+i} := \mathcal{P}_K \cap \left\{ P \;\middle|\; \sum_{j=i}^{n} (\alpha_j(P)\hat{\mu}_j - \beta_j(P)) \;=\; \min\{0, \min_{k=1,\ldots,n} \sum_{j=k}^{n} (\alpha_j(P)\hat{\mu}_j - \beta_j(P))\} \right\}$$

with lower bound function

$$V_{\mathrm{lb}}^{(K+i)}(P) := \max \left\{ V_{\mathrm{lb}}^{(K)}(P) \;\middle|\; \sum_{j=1}^{n} (\gamma_j(P) \; - \alpha_j(P)\hat{\mu}_j^2) + 2 \sum_{j=i}^{n} (\alpha_j(P)\hat{\mu}_j - \beta_j(P)) \right\}.$$

Replace region K with

$$\mathcal{P}_K := \mathcal{P}_K \cap \left\{ P \;\middle|\; 0 = \min\{0, \min_{j=1,\ldots,n} \sum_{j=i}^{n} (\alpha_j(P)\hat{\mu}_j - \beta_j(P)) \right\}$$

and update the lower bound

$$V_{\mathrm{lb}}^{(K)}(P) := \max \left\{ V_{\mathrm{lb}}^{(K)}(P), \sum_{j=1}^{n} (\gamma_j(P) - \alpha_j(P)\hat{\mu}_j^2) \right\}.$$

3. Solve (6.16) for $k = K, K+1, \ldots, K+n$. For each k, let $P_{\max}^{(k)}$ be the maximizer, and let $U^{(k)}$ be the optimal value.

4. Set $K := K + n$. Reorder the regions so that

$$U_1 \leq U_2 \leq \cdots \leq U_K.$$

We terminate the iteration and return P_{best} if $U_K - L$ is sufficiently small.

6.4 Proof of Convergence

In this section, we prove that the global optimization algorithm will converge to a global optimal point in a finite number of steps by proving the following two theorems.

Theorem 5 *(Finite ϵ-Convergence).*

Proof. Fix $\epsilon > 0$ arbitrarily. Let $\langle P^{(t)}, \mathrm{UBD}^{(t)} \rangle$ be the sequence of optimal solutions to the Problem (6.16) at iteration t . At each iteration, there is an accumulation of constraints from previous iterations. This implies that $\langle \mathrm{UBD}^{(t)} \rangle$ is a non-decreasing sequence which is bounded by the optimal value of the original problem. At each iteration, $P^{(t)} \in X$ where $X = P : P\mathbf{1} = \mathbf{1}$ and $P \succeq 0$. So, X is a compact set. By taking a subsequence, $\langle P^{(t)}, \mathrm{UBD}^{(t)} \rangle$ will converge to $\langle \hat{P}, \hat{\mathrm{UBD}} \rangle$ in finite steps with arbitrary tolerance $\epsilon' > 0$ such that $\hat{P} \in X$, i.e. $|\hat{P} - P^{(t)}| \leq \epsilon'$, where t is sufficient large but finite. The solution for the corresponding evaluation problem QP (6.14), therefore, also converges to $\hat{\mu}$. By convexity of $E_{\mathrm{n}}(\hat{P}) - V(\hat{P}, \mu)$ in the domain defined by $S = \{\mu : 0 \leq \mu_1 \leq \mu_2 \leq \cdots \leq \mu_n \leq 1\}$,

$$\begin{aligned} & E_{\mathrm{n}}(\hat{P}) - V(\hat{P}, \mu) \\ \leq & \sup_{\mu \in S} E_{\mathrm{n}}(\hat{P}) - V(\hat{P}, \mu) \\ \leq & \sup_{P \in X, \mu \in S} E_{\mathrm{n}}(P) - V(P, \mu) \\ \leq & \sup_{P \in X} \{E_{\mathrm{n}}(P) - V(P, \hat{\mu}) - \inf_{\mu \in S} \{DV(\hat{\mu})^T (\mu - \hat{\mu})\}\}, \end{aligned}$$

where $DV(\hat{\mu}) \in \mathbf{R}^n$ with each component $DV(\hat{\mu})_j = \frac{\partial V(P,\mu)}{\mu_j} \mid_{\mu_j = \hat{\mu}_j}$.

Let

$$\tilde{P} = \arg\max_{P \in X} \{E_{\mathrm{n}}(P) - V(P, \hat{\mu}) - \inf_{\mu \in S} DV(\hat{\mu})^T (\mu - \hat{\mu})\}.$$

By sequence convergence, we have $\tilde{P} = \hat{P}$.

Note that

$$\hat{\mu} = \arg\min_{\mu \in S} -V(\tilde{P}, \mu) \tag{6.17}$$

$$= \arg\min_{\mu \in S} -V(\hat{P}, \mu) \tag{6.18}$$

Therefore,

$$DV(\hat{\mu})^T (\mu - \hat{\mu}) \geq 0, \forall \mu \in S$$

Hence,

$$E_{\mathrm{n}}(\hat{P}) - V(\hat{P}, \hat{\mu}) - \inf_{\mu \in S} DV(\hat{\mu})^T(\mu - \hat{\mu}) \leq E_{\mathrm{n}}(\hat{P}) - V(\hat{P}, \hat{\mu})$$

for any fixed $\hat{P}$. Since we keep adding constraints, we have

$$\hat{\mathrm{UBD}} \leq E_{\mathrm{n}}(\hat{P}) - V(\hat{P}, \hat{\mu}) - \inf_{\mu \in S} DV(\hat{\mu})^T(\mu - \hat{\mu}).$$

So,

$$\hat{\mathrm{UBD}} \leq E_{\mathrm{n}}(\hat{P}) - V(\hat{P}, \hat{\mu}).$$

Since $E_{\mathrm{n}} - V$ is continuous in μ and $E_{\mathrm{n}} - V$ is upper semicontinuous at $\hat{P}$, we have $\mathrm{UBD}^{(t)} \leq (E_{\mathrm{n}} - V)^{(t+1)} + \epsilon$ for t sufficiently large. That is the algorithm's termination condition can be satisfied in finite steps.

Theorem 6 *(Global Optimality) If the conditions in Theorem 5 hold, then the algorithm will terminate at the global optimum of the original problem.*

Proof. From the proof of Theorem 1, we know that by taking subsequence, $\langle P^{(t)}, \mu^{(t)}, \mathrm{LBD}^{(t)} \rangle$ will converge to $\hat{P}, \hat{\mu}, \hat{\mathrm{LBD}}$. We prove that $(\hat{P}, \hat{\mu})$ is the global optimal solution by showing that the $\hat{\mathrm{UBD}}$ is the global overestimator of $E_{\mathrm{n}} - V$, $P \in X$, $\mu \in S$, since in the proof of finite ϵ-convergence, we already showed that $\hat{\mathrm{UBD}} \leq E_{\mathrm{n}}(\hat{P}) - V(\hat{P}, \hat{\mu})$. In each iteration, we know that $\mathrm{UBD}^{(T)} = \max_{P \in \mathcal{P}_K} E_{\mathrm{n}}(P) - V(P, \mu) \geq \max_{k=1,\ldots,K} E_{\mathrm{n}}(P) - V_{lb}^k(P, \mu)$. Since $E_{\mathrm{n}}(P) - V(P, \mu) \leq E_{\mathrm{n}}(P) - V_{lb}(P, \mu)\ \forall P \in X$ and $\hat{\mathrm{UBD}} \leq E_{\mathrm{n}}(\hat{P}) - V(\hat{P}, \hat{\mu})$, we have

$$\hat{\mathrm{UBD}} = \max_{P \in X, \mu \in S} E_{\mathrm{n}}(P) - V(P, \mu)$$

with the corresponding solution $(\hat{P}, \hat{\mu})$.

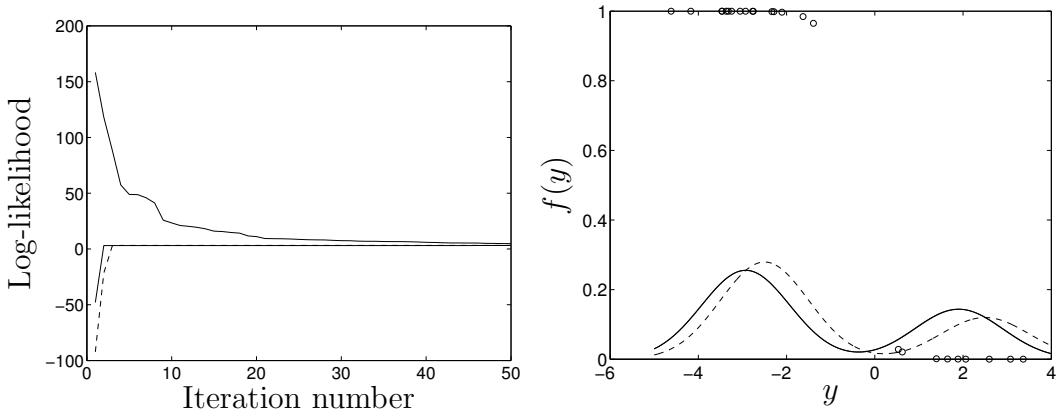

Figure 6.1: Convergence of EM Algorithm and Benders Method (Example 1). *Left.* The dashed line shows the iterates for the EM algorithm. The solid lines are the upper and lower bounds in Benders method. *Right.* The solid curve is the globally optimal distribution. The dashed line shows the distribution used to generate the data. The 25 circles show (y_k, P_{k1}) where y_k is the kth data point, and P is the optimal P in the biconvex formulation.

6.5 Demonstration

We compare the EM algorithm and the generalized Benders decomposition in this section by presenting two numerical results: one is a maximum likelihood estimation problem of 2 Gaussian mixtures with 25 data points; the other is that of 3 mixtures with 20 data points. The data are generated from a given Gaussian mixture density. Both Benders method and the EM algorithm are implemented in MATLAB with MOSEK toolbox.

Figures 6.1 and 6.2 show an example with $n = 2$, $N = 25$, and $n = 3$, $N = 20$, respectively.

These results show that for the simple 2 or 3 model cases with known variances, the EM algorithm works pretty well. Both the EM algorithm and the

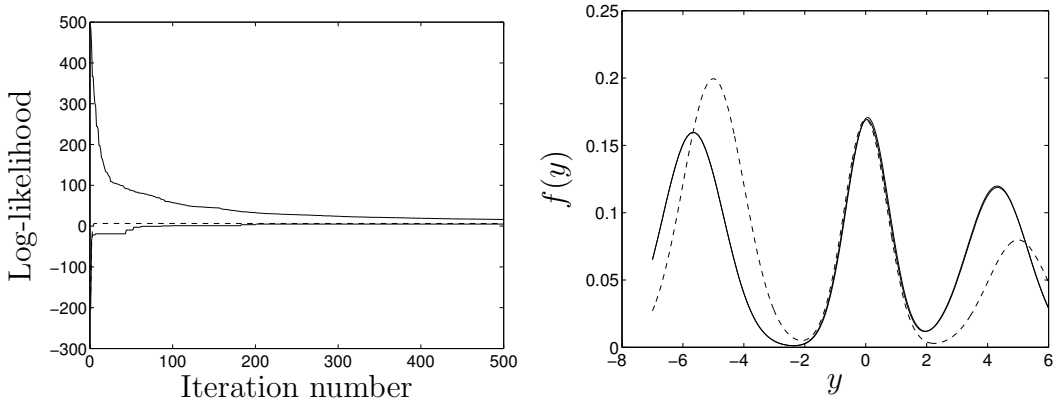

Figure 6.2: Convergence of EM Algorithm and Benders Method (Example 2). *Left.* The dashed line shows the iterates for the EM algorithm. The solid lines are the upper and lower bounds in Benders method. *Right.* The solid curve is the globally optimal distribution. The dashed line shows the distribution used to generate the data.

Benders method converge to the global optimum, though people never know how good the solution of EM is without the Benders method. To be interesting, Benders method finds the global optimum very soon (at least far before it converges) by looking at the lower bound. However, it takes long time to "prove" its global optimum. It is as expected since to prove a solution is a global optimum solution for a general non-convex problem is a NP-hard problem. The best of all, whenever Benders method halts, we have the currently best sub-optimal solution and a pair of lower and upper bound to see how good or bad this solution is.

6.6 Conclusion and Future Research

We identified that the maximum likelihood estimation problem of mixture family (if the component density function is log concave) is a biconcave problem. Bicon-

vex modeling of maximum likelihood estimation has a few implications. First, we can solve local optimum by using alternating optimization technique. The EM algorithm is one of the most famous alternating optimization algorithms. For some biconvex problems, if the gradient of the Lagrange function over one set of variables are linear in the other set of variables, we can solve such problems by using generalized Benders decomposition. It turns out that most mixture of exponential family learning problem with discrete or exponential observation model and many other learning models can be reformulated as a biconvex optimization problem.

We worked out the details of Benders decomposition for Gaussian mixture model. In this modeling, we add additional constraints to order the mean vector without losing generality. By doing this, we reduce the number of vertices of domain $\mu \in S$ from 2^n to $n + 1$ so that the number of subregions generated in each iteration is no longer an exponential number of parameters. Moreover, this algorithm generalized Floudas's method [FV93] by allowing a polyhedral domain of parameters instead of a set of artificial lower and upper bounds.

CHAPTER 7

Conclusion

Traditional support vector machines assume that the kernel matrices are fixed and the feature vectors can be locally accessed. We proposed several algorithms that work when the above assumptions do not hold.

7.1 Kernel Optimization

We proposed a kernel learning technique for support vector multi-class classification problems. We construct a function ω of the kernel matrix K as the optimal value of the dual formulation of Crammer and Singer's one-against-others method. The function $\omega(K)$ can be maximized via a semi-definite program (SDP). We successfully decomposed the linear matrix equality (LMI) constraints in the SDP into a set of smaller LMI constraints so that the formulated problem can be more efficiently solved. A similar technique is applied to the support vector regression (SVR) problem. We worked out the details of the SDP kernel optimization formulation for SVR.

When features are heterogeneous, we can not simply vectorize the features. The proposed kernel optimization technique allows us to linearly combine kernel matrices formed from heterogeneous features. An application of retina ganglion cell signal multi-class classification problem demonstrated that the proposed heterogenous feature combining method is superior to a widely used voting algo-

rithm.

The kernel selection problem may be considered a special case of the heterogeneous feature combination problem, since features are no longer homogeneous after projected by different kernels. Experiments with synthesized data for functional estimations and an application in handwritten digits multi-class classification both show that the optimal kernel in general has as good as, if not better, performance than the fixed kernels.

There is, however, no best known systematic way to select candidate kernels. We tried linear combination of different type kernels, i.e., polynomial kernels, Gaussian kernel, and etc. The idea is to algorithmically select the best feature projection. We also tried linear combination of the polynomial kernels. The idea is to optimally combine the features from spaces of different dimensions. The optimal kernels perform well in both methods. However, as long as those candidate kernels are formed from the same set of features, the optimal combination may not have significantly better performance than the best fixed kernel. To find a systematic way to combine candidate kernel matrices such that the testing error can be significantly improved is still an open problem.

7.2 Distributed Learning

We proposed a distributed support vector machine for distributed classification problems. The idea is to iteratively exchange support vectors (SVs) in a strongly connected network. We proved that this algorithm converges, on a strongly connected network, to the global optimal classifier in a finite number of steps.

We evaluated the performance of this algorithm in networks of different size, sparse and dense networks, various initial data distributions, sequential and paral-

lel implementations and off-line and online algorithms, in terms of the computing speed, communication cost, and data duplication and accumulation. Simulations and applications show that the proposed algorithm is robust to the initial distributions of training vectors. Denser networks and parallel implementation usually cause more data accumulation but less computing time. Sparse networks and sequential implementations in general achieve less data accumulation but longer computing time. On-line implementations may significantly reduce the computing time.

Our online algorithm implementation is, however, naive. We only use the best available solution as the initial solution of a local problem. If there are more advanced online SVM algorithms available, our DSVM may become much more efficient since each local problem is essentially a problem adjustment which appends variables and constraints based on the problem of the last iteration.

In order to take advantage of the randomized sampling technique, we proposed a parallel randomized support vector machine (PRSVM). This algorithm is able to train a support vector machine concurrently in multiple working sets. Each working set is sampled via a carefully designed sampling and weighting mechanism. We proved that our PRSVM algorithm converges, on average, in less than $(6\delta/C)\ln(N + 6r(C-1)\delta)$ iterations, where δ denotes the underlying combinatorial dimension, N denotes the total number of training vectors, C denotes the number of working sites; and r denotes the size for a working set. The average convergence rate is faster than the randomized SVM algorithm of [BDT01] by a factor of C.

References

[AA00] E. D. Andersen and K. D. Andersen. "The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm." In H. Frenk, K. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, pp. 197–232. Kluwer Academic Publishers, 2000.

[AJ86] Loyce M. Adams and Harry F. Jordan. "Is SOR Color-blind?" *SIAM Journal on Scientific and Statistical Computing*, 1986.

[AS93] Ilan Adler and Ron Shamir. "A Randomized Scheme for Speeding Up Algorithms for Linear and Convex Programming with High Constraints-to-Variable Ratio." *Mathematical Programming*, 1993.

[Bas98] A. Bastiere. "Methods for Multisensor Classification of Aiborne Targets Integrating Evidence Theory." *Aerospace Science and Technology*, **2**:401–411, 1998.

[BB99] O. Barzilay and V. L. Brailovsky. "On Domain Knowledge and Feature Selection Using a Support Vector Machine." *Pattern Recognition Letters*, **20**:475–484, 1999.

[BD99] Jock A. Blackard and Denis J. Dean. "Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Type from Cartographic Variables." *Computer and Electronics in Agriculture*, **24**, 1999.

[BDT01] Jose Balcazar, Yang Dai, Junichi Tanaka, and Osamu Watanabe. "Provably Fast Training Algorithm for Support Vector Machines." *Proceedings of First IEEE International Conference on Data Mining (ICDM01)*, 2001.

[BDT02] Jose Balcazar, Yang Dai, Junichi Tanaka, and Osamu Watanabe. "Provably Fast Training Algorithm for Support Vector Machines." *Technical Reports on Machematical and Computing Sciences: TR-C160*, 2002.

[BF95] Yoshua Bengio and Paolo Frasconi. "An Input Output HMM Architecture." In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pp. 427–434. The MIT Press, 1995.

[BFM90] U. Block, A. Frommer, and G. Mayer. "Block Coloring Schemes for the SOR method on Local Memory Parallel Computers." *parallel Computing*, 1990.

[BGL00] M. P. S. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. S. Furey, M. Ares, and D. Haussler. "Knowledge-based Analysis of Microarray Gene Expression Data Using Support Vector Machines." *Proceedings of the National Academy of Science*, **97**:262–267, 2000.

[BGV92] B.E. Boser, I.M. Guyon, and V. Vapnik. "A Training Algorithm for Optimal Margin Classificafiers." In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Thoery.* ACM Press, Pittsburgh, PA, 1992.

[BM98] C.L. Blake and C.J. Merz. "UCI Repository of machine learning databases.", 1998.

[Bra00] Jerome J. Braun. "Dempster-Shafer Theory and Bayesian Reasoning in Multisensor Data Fusion." *In Sensor Fusion: Architectures, Algorithms and Applications IV, Proceeding of SPIE*, **4051**:255–266, 2000.

[Bur98] C.J.C. Burges. "A Tutorial on Support Vector Machine for Pattern Recognition." *Data Mining and Knowledge Discovery*, **2**:121–167, 1998.

[CB01] Ronan Collobert and Samy Bengio. "SVMTorch: Support Vector Machines for Large-Scale Regress Problems." *Journal of Machine Learing Resarch*, **1**:276–285, 2001.

[CHH04] Sheng Chen, Xia Hong, and Chris J. Harris. "Sparse Kernel Density Contruction Using Orthogonal Forward Regression with Leave-One-Out Test Score and Local Regularization." *IEEE Transaction on Systems, Man and Cybernetics - Part B, Cybernetics*, **34**:1708–1717, 2004.

[Cla88] Kenneth L. Clarkson. "Las Vegas Algorithms for Linear and Integer Programming When the Dimension Is Small." *Proceeding of 29th IEEE Symposium on Foundations of Computer Science (FOCS'88)*, 1988.

[CM04] Vladimir Cherkassky and Yunqian Ma. "Practical Selection of SVM Parameters and Noise Estimation for SVM Regression." *Neural Networks*, **17**:113–126, 2004.

[CS01] Koby Crammer and Yoram Singer. "On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines." *Journal of Marchine Learning Research*, **2**:265–292, December 2001.

[CSB04] Carlos, Soars, and Pavel B. Brazdil. "A Meta-Learning Method to Select the Kernel Width in Support Vector Regression." *Machine Learning*, **54**:195–209, 2004.

[CT84] I. Csiszar and G. Tusnady. "Information Geometry and Alternating Minimization Procedures." In D.J. Dudewicz, editor, *Recent Results in Estimation Theory and Related Topics, Statistics and Decisions, Supplement Issue No. 1*. Oldenburg Verlag, Munich, 1984.

[CT03] L.J. Cao and Francis E.H. Tay. "Support Vector Machine with Adaptive Parameters in Financial Time Series Forcasting." *IEEE Transactions on Neural Networks*, **14**:1506–1518, 2003.

[CV95] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks." *Machine Learning*, **20**:273–297, 1995.

[CVB02] O. Chapelle, V. Vapnik, O Bousquet, and S Mukherjee. "Choosing Multiple Parameters for Support Vector Machines." *Machine Learning*, **46**:131–159, 2002.

[CZB97] R. Chellappa, Q. Zheng, P. Burlina, C. Shekhar, and K. B. Eom. "On the Positioning of Multisensor Imagery for Exploitation and Target Recognition." *Proceddings of the IEEE*, **85**:120–138, 1997.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley, 2000.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum-Likelihood from Incomplete data via the EM algorithm." *Journal of Royal Statistic Society*, **39**:1–38, 1977.

[DLR97] A.P. Dempster, N.M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data via EM algorithm." *Journal of the Royal Statistic Society*, **39**:1–38, 1997.

[Dom02] Pedro Domingos. *Handbook of Data Mining and Knowledge Discovery*. Oxford University Press, New York, 2002.

[FH94] J.A. Fessler and A.O. Hero. "Space-alternating Generalized Expectation-Maximization algorithm." *IEEE Transaction on Signal Processing*, **42**:2664–2677, 1994.

[FHL03] Dieter Fox, Jeffrey Hightower, Lin Liao, and Gaetano Borriello. "Bayesian Filtering for Location Estimation." *Pervasive Computing, IEEE Computer Society*, pp. 24–33, July 2003.

[FL02] Gary W. Flake and Steve Lawrence. "Efficient SVM Regression Training with SMO." *Machine Learning*, 2002.

[Fri96] J. Friedman. "Another Approach to Polychotomous Classification." Technical report, Dept. Statistics, Stanford University, Stanford, CA, 1996.

[FV93] C.A. Floudas and V. Visweswaran. "Primal-Relaxed Dual Global Optimization Approach." *Journal of Optimization Theory and Applications*, **78**:187–225, 1993.

[GBV93] I. Guyon, B. Boser, and V. Vapnik. "Automatic Capacity Tuning of Very Large VC-Dimension Classifier." In S.J. Hanson, J.D. Cowan, and C. LeeGiles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan Kaufmann, San Mateo, CA, 1993.

[Geo72] A. M. Geoffrion. "Generalized Benders Decomposition." *Journal of Optimization Theory and Applications*, **10**:237–254, 1972.

[GW00] Bernd Gartner and Emo Welzl. "A Simple Sampling Lemma: Analysis and Applications in Geometric Optimization." *Proceeding of the 16th Annual ACM Symposium on Computational Geometry (SCG)*, 2000.

[Hat86] R.J. Hathaway. "Another Interpretation of the EM Algorithm for Mixture Distributions." *Statistics and Probability Letters*, **4**:53–56, 1986.

[HKC04] Winston H. Hsu, Lyndon S. Kennedy, Shih-Fu Chang, Martin Franc, and John R. Smith. "Columbia-IBM news Video Story Segmentation in TRECVID 2004." *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP 2004)*, May 2004.

[HL02] Chih-Wei Hsu and Chih-Jen Lin. "A Comparison of Methods for Multiclass Support Vector Machines." *IEEE Transaction on Neural Networks*, **13**:415–425, March 2002.

[JL99] B. Jeon and D. A. Landgrebe. "Decision Fusion Approach for Multitemporal Classification." *IEEE Transaction on Geoscience and Remote Sensing*, **37**:1227–1233, 1999.

[Joa98] Thorsten Joachims. "Making Large-Scale SVM Learning Practical." *Advances in Kernel Methods - Support Vector Learning*, pp. 169–184, 1998.

[KBS02] P. Kuba, P. Brazdil, C. Soares, and A. Woznica. "Expoilting Sampling and Meta-Learning for Parameter Setting Support Vector Machines." *Proceedings fo the Workshop de Mineria de Datos Y Aprendizaje of (IBERAMIA 2002)*, pp. 217–225, 2002.

[Kre99] U. Krebel. "Pairwise Classification and Support Vector Machines." In B. Scholkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernels Methods - Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

[KW71] G.S. Kimeldorf and G. Wahba. "Some Results on Tchebycheffian Spline Function." *Journal of Mathematical Analysis and Applications*, **33**:82–95, 1971.

[Kwo01] J.T. Kwok. "Linear Dependency Between ϵ and the Input Noise in ϵ-Support Vector Regression." *Proceedings of International Conference on Artificial Neural Networks, ICANN 2001*, pp. 405–410, 2001.

[LBB98] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-Based Learning Applied to Document Recoginition." *Proceedings of the IEEE*, **86**(11):2278–2324, November 1998.

[LCB04] Gert R.G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I. Jordan. "Learning the Kernel Matrix with Semidefinite Programming." *Journal of Marchine Learning Research*, **5**:27–72, January 2004.

[LCK04] Pei-Kai Liao, Min-Kuan Chang, and C.-C. Jay Kuo. "Distributed Edge Detection with Composite Hypothesis Test in Wireless Sensor Networks." *IEEE Communications Society Globecom 2004*, pp. 129–133, 2004.

[LKT98] Shutao Li, James Tin-Yau Kwok, Ivor Wai-Hung Tsang, and Yaonan Wang. "Fusion Images with Different Focuses Using Support Vector Machines." *IEEE Transactions on Neural Networks*, **15**:1555–1560, November 1998.

[LM00] Y. Liu and H.S. Mahmassani. "Global Maximum Likelihood Estimation Procedure for Multinomial Profit (MNP) Model Parameters." *Transportation Research, Part B*, pp. 419–449, 2000.

[LR94] C. Liu and D.B. Rubin. "The ECME algorithm: a Simple Extention of EM and ECM with Faster Monotone Convergence." *Biometrika*, **81**:663–648, 1994.

[Mat00] Y. Matsuyama. "The α-EM Algorithm and its Basic Properties." *System and Computers in Japan*, **31**:12–23, 2000.

[Mer09] J. Mercer. "Functions of Positive and Negative Type and Their Connection with the Theory of Integral Equations." *Philosophical Transactions of the Royal Society*, **A**:415–446, 1909.

[MH99] D. Mattera and S. Haykin. "Support Vector Machines for Dynamic Reconstruction of a Chaotic System." In *Advances in Kernel Method: Support Vector Machine*. MIT Press, Cambridge, MA, 1999.

[MK96] G.J. Mclachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, New York, 1996.

[MM99] Olvi Mangasarian and David Musicant. "Successive Overrelaxation for Support Vector Machines." *IEEE Transactions on Neural Networks*, 1999.

[MR93] X.L. Meng and D.B. Rubin. "Maximum Likelihood Eestimation via the ECM Algorithm: a General Framework." *Biometrika*, **80**:267–278, 1993.

[NH98] R.M. Neal and G.E. Hinton. "A View of the EM Algorithm that Justifies Incrementals, Sparse, and other Variants." In M. I. Jordan and Ed. Dordrecht, editors, *Learning in Graphical Models*, pp. 355–368. Kluwer, The Netherlands, 1998.

[NH01] Milind Ramesh Naphade and Thomas S. Huang. "A Probabilistic Framework for Semantic Video Indexing, Filtering and Retrieval." *IEEE Transaction Multimedia*, **3**:141–151, March 2001.

[Now03] R.D. Nowak. "Distributed EM Algorithm for Density Estimation and Clustering in Sensor Networks." *IEEE Transaction on Signal Processing*, **51**:2245–2253, 2003.

[OFG97a] Edgar Osuna, Robert Freund, and Federico Girosi. "An Improved Training Algorithm for Support Vector Machines." *In Proceedings, 1997 IEEE Workshop on Neural Networks for Signal Processing*, pp. 276–285, 1997.

[OFG97b] Edgar Osuna, Robert Freund, and Federico Girosi. "Support Vector Machines: Training and Applications." *Massachusetts Institute of Technology Artificial Intelligence Laboratory Memo No. 1602*, 1997.

[ONR95] C.J.C. Burges and B. Scholkopf. "A New Method for Constructing Artificial Neural Networks." Interim technical report, ATT Bell Laboratories, 1995.

[PCS99] J.C. Platt, N. Cristianini, and J. Shawe-Taylor. "Large Margin DAG's for Multiclass Classification." *Advances in Neural Information Processing Systems*, **12**:547–553, 1999.

[PG98] C. Pohl and J. L. Van Genderen. "Mutisensor Image Fusion in Remote Sensing: Concepts, Methods and Applications." *International Journal of Remote Sensing*, **19**:823–854, 1998.

[RWD84] R.A. Redner, H.F. Walker, and Mixture Density. "Maximum Likelihood and the EM Algorithm." *SIAM Review*, **26**:195–238, 1984.

[Sch96] Michael S. Schmidt. "Identity Speaks With Support Vector Networks." *In Proceedings of the 28th Symposium on the Interface (INTERFACE-96)*, 1996.

[SGV99] M. Stitson, A. Gammerman, V. Vapnik, V. Volk, C. Watkins, and J. Weston. "Support Vector Regression with ANOVA Decomposition Kernels." In B. Scholkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernels Methods - Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

[SH85] C.H. Slump and B.J. Hoenders. "The Determination of the Location of the Global Maximum of a Function in the Presence of Several Local Extrema." *IEEE Transaction on Information Thoery*, **31**:490–497, 1985.

[SH01] Pero Subasic and Alison Huettner. "Affect Analysis of Text Using Fuzzy Semantic Typing." *IEEE Transactions on Fuzzy Systems*, **9**:483–496, August 2001.

[SMS99] A. Smola, N. Murata, B. Scholkopf, and K. Muller. "Asymptotically Optimal Choice of ϵ-Loss for Support Vector Machines." *Proceedings of International Conference on Artificial Neural Networks, ICANN 1998*, 1999.

[SS98] A.J. Smola and B. Scholkopf. "On a Kernel Based Method for Pattern Recognition, Regression, Approximation and Operator Inversion." *Algorithmica*, **22**:211–231, 1998.

[SS02] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels - Support Vector Machines, Regularization, Optimization and Beyond*. MIT, 2002.

[SSM98] B. Scholkopf, A.J. Smola, and K.R. Muller. "Kernel Principal Component Analysis." In B. Scholkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pp. 327–352. MIT Press, Cambridge, MA, 1998.

[SSS98] B. Scholkopf, P. Simard, A. J. Smola, and V. Vapnik. "Prior Knowledge in Support Vector Kernels." *Advances in Neural Information Processing Systems*, **10**:640–646, 1998.

[Stu99] J.F. Sturm. "A MATLAB Toolbox for Optimization over Symmetric Cones." *Optimization Methods and Software, Special issue on Interior Point Methods*, pp. 625–653, 1999.

[TBW03] U. Thissen, R. van Brakel, A.P. de Weijer, W.J. Melssen, and L.M.C. Buydens. "Using Support Vector Machines for Time Series Predication." *Chemometrics and Intelligent Laboratory Systems*, **69**:35–49, 2003.

[UCS99] D. Haussler. "Convolutional Kernels on Discrete Structures." Technical report, University of California, Santa Cruz, 1999.

[Val04] Ventzeslav Valev. "Supervised Pattern Recognition with Heterogenous Features." *LNCS*, **3287**:693–700, 2004.

[Vap95] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

[VGS97] V. Vapnik, S. Golowich, and A.J. Smola. "Support Vector Method for function approximation Regression Estimation and Signal Processing." In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pp. 281–287. MIT Press, Cambridge, MA, 1997.

[VL63] V. Vapnik and A. Lerner. "Pattern Recognition Using Generalized Portrait Method." *Automation and Remote Control*, **24**:774–780, 1963.

[VMB01] L. Valet, G. Mauris, and Ph. Bolon. "A Statistical Overview of Recent Literature in Information Fusion." *IEEE AESS System Magazine*, pp. 7–14, March 2001.

[VV97] Ramanarayanan Viswanathan and Pramod K. Varshney. "Distributed Detection with Multiple Sensors I. Fundamentals." *Proceedings of the IEEE*, **85**:54–63, January 1997.

[VW96] A.W. van der Vaart and J.A. Wellner. *Weak Convergence and Empirical Process*. Springer, 1996.

[VW02] S. Vempala and G. Wang. "A Spectral Algorithm for Learning Mixture of Distribution." *Proceeding of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002.

[Wah90] G. Wahba. "Spline Modes for Observational Data." *CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM*, **59**, 1990.

[XLB02] Youshen Xia, Henry Leung, and Eloi Bosse. "Neural Data Fusion Algorithms Based on a Linearly Contrained Least Square Method." *IEEE Transactions on Neural Networks*, **13**:320–329, March 2002.

[Xu05] Jinbo Xu. "Fold Recognition by Predicted Alignment Accuracy." *IEEE Transaction on Computational Biology and Bioinformatics*, **2**:157–165, 2005.

[ZBJ05] Q. Zhong, P.O. Boykin, A. Jacobs, S. Nirenberg, Nirenberg, and V. Roychowdhury. "A Filter Based Encoding Model for Mouse Retinal Ganglion Cells." *The 27th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS)*, 2005.

[ZL03] Yumin Zhu and X. Rong Li. "Unified Fusion Rules for Multisensor Multihypothesis Network Decision Systems." *IEEE Transactions on Neural Networks*, **33**:502–513, July 2003.

[ZZ03] G. Zanghirati and L. Zanni. "A parallel Solver for Large Quadratic Programs in Training Support Vector Machines." *Parallel Computing*, **29**:535–551, November 2003.