SPEED AND ACCURACY: LARGE-SCALE MACHINE LEARNING ALGORITHMS AND THEIR APPLICATIONS

JIANXIONG DONG

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

OCTOBER 2003

Abstract

Speed and Accuracy: Large-scale Machine Learning Algorithms and their Applications

Jianxiong Dong, Ph.D.
Concordia University, 2003

Over the past few years, considerable progress has been made in the area of machine learning. However, when these learning machines, including support vector machines (SVMs) and neural networks, are applied to massive sets of high-dimensional data, many challenging problems emerge, such as high computational cost and the way to adapt the structure of a learning system. Therefore, it is important to develop some new methods with computational efficiency and high accuracy such that learning algorithms can be applied more widely to areas such as data mining, Optical Character Recognition (OCR) and bio-informatics.

In this thesis, we mainly focus on three problems: methodologies to adapt the structure of a neural network learning system, speeding up SVM's training and facilitating test on huge data sets. For the first problem, a local learning framework is proposed to automatically construct the ensemble of neural networks, which are trained on local subsets so that the complexity and training time of the learning system can be reduced and its generalization performance can be enhanced. With respect to SVM's training on a very large data set with thousands of classes and high-dimensional input vectors, block diagonal matrices are used to approximate the original kernel matrix such that the original SVM optimization process can be divided into hundreds of sub-problems, which can be solved efficiently. Theoretically, the run-time complexity of the proposed algorithm linearly scales to the size of the data set, the dimension of input vectors and the number of classes. For the last problem, a fast iteration algorithm is proposed to approximate the reduced set vectors simultaneously based on the general kernel type so that the number of vectors in the decision function of each class can be reduced considerably and the testing speed is increased significantly.

The main contributions of this thesis are to propose effective solutions to the above three problems. It is especially worth mentioning that the methods which are used to solve the last two problems are crucial in making support vector machines more competitive in tasks where both high accuracy and classification speed are required. The proposed SVM algorithm runs at a much higher training speed than the existing ones such as svm-light and libsvm when applied to a huge data set with thousands of classes. The total training time of SVM with the radial basis function kernel on Hanwang's handwritten Chinese database (2,144,489 training samples, 542,122 testing samples, 3755 classes and 392-dimensional input vectors) is 19 hours on P4. In addition, the proposed testing algorithm has also achieved a promising classification speed, 16,000 patterns per second, on MNIST database. Besides the efficient computation, the state-of-the-art generalization performances have also been achieved on several well-known public and commercial databases. Particularly, very low error rates of 0.38%, 0.5% and 1.0% have been reached on MNIST, Hanwang handwritten digit databases and ETL9B handwritten Chinese database.

Acknowledgments

I would like to express my extreme gratitude to Prof. Ching Y. Suen and Prof. A. Krzyżak for their supervision. I have benefited greatly from their expertise, vision and clarity of thought in conducting this research. Their constant supports, numerous suggestions and careful reading of technical reports and papers have played a crucial role in the development of this thesis. I am indebted to them for helping me through the most difficult time during my Ph.D. period. Both advisors provide me financial support and give me free research atmosphere and always encourage me to conduct the research which I am interested in. Without their guidance and advice the present work would have not been possible.

The work for my thesis is done at Centre for Pattern Recognition and Machine Intelligence (**CENPARMI**), which provides excellent facilities and stimulating research environment. I really enjoy discussing some research problems with some friends at **CENPARMI**, such as Javad Sadri and Karim Abou-Moustafa. Dr. Qizhi Xu, one of my good friends, helps me a lot during my Ph.D. period. I also enjoy the friendship of Yun Li, Wu Ding, Nenghong Fu and Yan Zhang at **CENPARMI**. These friends have made my life at **CENPARMI** an enriching experience.

I am also indebted to Abramovitz Beverley for her kindness and help in proofreading the papers. I am grateful for support from NSERC and FCAR and thank Dr. Changping Liu in Hanwang company for providing us with Hanwang handwritten digit and Chinese character databases for research.

Finally, I thank my parents for their love and support. Without them I even can not start the work in the first place.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Efforts to understand the nature of learning and intelligence, and realization of these capabilities in human minds, are among the most fundamental activities, which have brought growing interests to research communities in a wide variety of disciplines including education, cognition science, computer science, neuro-science, engineering, social science and physical science. Learning from data is one of the basic ways humans perceive the world and acquire knowledge. Nowadays, there are huge amounts of online data available at an astonishingly increasing pace on the worldwide internet and a fast searching engine is expected to find user-specific information efficiently. Medical institutions expect to create data-driven tools based on patients' historical data for diagnostic purpose; financial companies require computer-automated tools to analyze large amounts of data from customers so that they can make a good prediction of marketing behaviors and process financial transactions; the governments fund universities and companies to develop robust learning-based intelligent systems to discover the national threat and detect illegal activities. Therefore, large-scale learning algorithms and related tools must be developed to help us better understand the nature of learning and solve these industrial problems.

In this thesis, we focus on a specific problem in this domain by dealing with supervised learning algorithms for pattern classification on extremely large data sets with high dimensional real-valued feature vectors and thousands of classes. In real world applications, many problems are related to this subject. For example, it is a

challenging task to recognize handwritten Chinese characters efficiently and precisely because they have large shape variations and more than 6,000 categories; many grocery stores collect data from transactions, often generating gigabytes of data every day. The existing algorithms have difficulty in analyzing these data and categorizing them for further processing; on the internet, text-based web documents are very large, dynamic, and they have high dimensions. A large-scale adaptive learning algorithm is required to classify these documents for the development of a fast and accurate searching engine. Unfortunately, many existing algorithms exhibit poor scaling capabilities when dealing with huge high-dimensional data [115]. In order to tackle the problem, the following theoretical and computational issues have to be taken into account:

- Theoretical issues.

    1. Adapting the data-driven classifier structure. In most cases, little reliable prior information about the statistical law underlying the problem is available. Fitting a known function to data and estimating its finite number of parameters by means of the maximal likelihood method are usually insufficient. It is necessary to infer a function from the data.
    2. The curse of dimensionality [3]. For classification problems, we expect that a larger number of features may hold better discriminant power and lead to an improvement in performance. In practice, however, it has been frequently observed that adding more features degrades the classification accuracy. Bellman [3] called it "the curse of dimensionality" and found that increasing the number of factors requires an exponential increase in the number of computational resources such as training samples.
    3. Controlling the generalization performance on a finite data set. Before statistical learning theory was proposed by Vapnik [119][121], methods such as nearest neighbor classifier that have a good asymptotical solution often fail in a finite sample set and they often only minimize the empirical risk on the training set without taking into account the complexity of classifiers that have a significant effect on generalization performance. The classifiers based on structure risk minimization [121] lead to a trade-off between both factors.

- Algorithm issues.

    1. Develop a learning algorithm with low time/space computational complexity. For example, most learning algorithms minimize (or maximize) a cost function under constrained conditions. We expect that these algorithms converge quickly and their time complexities linearly depend on the number of training samples and the dimension of a feature vector.

    2. Trade-off between limited computational resources and speed. Most workstations and PCs have limited computational resources such as memory, bus bandwidth and fast hardware cache although their CPUs (Central Processor Units) are usually fast due to recent advances in technology. When a learning algorithm runs on a large data set, which is too big to be completely loaded into the memory, a huge amount of data may have to be exchanged between the harddisk and the memory. As a result, the usage rate of CPU is low. Even when a huge amount of data is loaded into the memory, frequent page operations due to high Translation Look-aside Buffers (TLBs) miss ratio will lead to a high cost [90]. This issue is related to the computer architecture and its operating system.

    3. Good algorithm structure for parallel implementation. The low price and high performance of micro-processors such as Intel Pentium IV provide us a chance to develop an algorithm with a parallel structure, which can be easily implemented on a multi-processor platform. This is a potential trend for a large-scale learning algorithm because the computational costs are distributed to different processors and each processor does not require a large number of computational resources to finish its sub-tasks. It can substantially alleviate the above problem.

    4. Online learning. In order to reduce the error rate of a classifier, a set of new training samples will be appended to the historical training set, on which the classifier has been trained before. Is it possible to adjust the parameters of this classifier based on increased training samples, rather than to re-train the classifier again on the whole set?

Among the above theoretical issues, how to control the generalization performance

on a finite set is well formulated in statistical learning theory (also called Vapnik–Chervonenkis (VC) theory). Numerous experiments [98][57][24][33] have shown that kernel-based support vector machines (SVM) [121][100][104] have solved the second theoretical problem because SVM performs well in a huge dimensional feature space. For the first theoretical issue, unfortunately, we can not find a classifier that is inherently superior to any other methods on any distribution, which has been indicated by **No Free Lunch Theorem** [41]. It is the type of problem, prior distribution and other information that determine which type of classifier should be used to get the best performance. Therefore, it is important to design the best possible classifier for the specified problem under the guidance of some general principles such as divide and conquer, structure risk minimization.

The above algorithmic issues are rarely addressed in the literature of pattern recognition and machine learning, especially on a large data set, where many difficult problems exist. In general, large-scale computational issues are discussed in the domains of numerical analysis, computer architecture, algorithm analysis and operating system. In order to design an efficient and robust learning algorithm that can be applied to solve some real-world problems, it is necessary to know them well, including not only theories but also many practical issues.

In this thesis, we mainly focus on algorithmic issues on a large data set and provide faithful answers to the following three questions:

- How to adapt the structure of a neural network to yield a good generalization performance for classification on a large data set?
- How to design an efficient training algorithm for support vector machines on a data set of huge size with millions of high-dimensional samples and thousands of classes such that the run-time complexity of the algorithm linearly scales to the size of data set, the dimension of input vectors and the number of classes?
- How to design an algorithm for support vector machines so that they can achieve both high accuracy and classification speed in the test phase?

In addition, the methods which provide the solutions to the above problems will be applied to solve real world classification problems, including handwritten digit recognition, handwritten lowercase letters of the English alphabet and Chinese character recognition. The algorithm efficiency and the state-of-the-art performance on large

data sets are our goals. At the same time, more practical principles are provided to help industrial developers and academic researchers to design an efficient and robust classifier.

1.2 Outline of Thesis

The thesis is organized in eight chapters. The main parts are Chapters 3, 5, 6 and 7, which are intended to be reasonably self-contained so that experienced readers can jump right to these chapters quickly.

- In Chapter 2, we review some main classification techniques and learning algorithms in terms of speed, accuracy and memory requirements on extremely large data sets and discuss why they do not satisfy the desirable goal. In addition, some basic optimization techniques and key issues from other disciplines such as numerical analysis, computer architecture, operating system and parallel computation are described briefly to help researchers implement a high performance learning algorithm. These techniques are also useful for industrial developers.

- In Chapter 3, we propose a general local learning framework to effectively alleviate the complexities of classifier design by means of "divide and conquer" principle and ensemble method. The learning framework consists of a quantization layer which uses generalized learning vector quantization (GLVQ)[95] and an ensemble layer which uses multi-layer perceptrons (MLP). The proposed method is tested on public handwritten character data sets, and it has obtained promising results consistently. In contrast to other methods, the proposed method has been shown to be especially suitable for large-scale real-world classification problems. Nevertheless, this method sets some parameters manually such as the number of prototypes for GLVQ and number of hidden units for MLP. How to set these parameters optimally still requires more heuristic knowledge. In the next chapter, support vector machine will have a capability to determine automatically the structure once the kernel has been selected.

- In Chapter 4, we introduce some basic concepts and ideas of support vector machines and its connection to statistical learning theory. The main goal is to help the readers get some background and understand the details of the

optimization problems in the next chapter. The expert reader may skip this chapter and move on to the next chapter where we start to describe our new fast SVM algorithm.

- In Chapter 5, training a support vector machine on a huge size of data set with thousands of classes is a challenging problem. This chapter proposes an efficient algorithm to solve this problem. The key idea is to introduce a parallel optimization step to quickly remove most non-support vectors, where block diagonal matrices are used to approximate the original kernel matrix so that the original problem can be decomposed into hundreds of sub-problems which can be solved more efficiently. In addition, some effective strategies such as kernel caching and efficient computation of kernel matrix are integrated to speed up the training process. Our analyses of the space and time complexity of the proposed algorithm show that its time complexity grows linearly with the number of classes and size of the data set. In the experiments, many appealing properties of the proposed algorithm have been investigated and the results show that the proposed algorithm has a much better scaling capability than Libsvm and SVM^{light}. Moreover, the state-of-the-art performances on several large databases have been achieved. Particularly, on MNIST and Hanwang handwritten digit databases, very low error rates of 0.38% and 0.5% have been reached, respectively.

- In Chapter 6, the fast SVM algorithm developed in Chapter 5 is applied to solve the recognition problem of handwritten Chinese characters, which have more than three thousand categories and large shape variations. Feature extraction is one of the most important and difficult elements of learning and it plays a key role in obtaining a high accuracy. We propose a new feature extraction method based on the gradient map of gray-scale images and improve the nonlinear normalization method proposed by Yamada et al.[127]. The experiments have been done on ETL9B, handwritten Chinese character database and a state-of-the-art performance has been achieved at a raw error rate of 1.0%. Moreover, we have designed an effective method to tune the parameters for support vector machines.

- In Chapter 7, a fast iteration algorithm is proposed to approximate the reduced set vectors shared by each binary SVM solution for multi-class classification simultaneously. The iteration algorithm can be applied to the general kernel types such as $k(\| \mathbf{x} - \mathbf{x}' \|^2)$ and $k(\mathbf{x}^T\mathbf{x}')$. In addition, we present a fast block algorithm in the test phase to speed up the classification further. Experimental results have shown that the classification speeds on MNIST and Hanwang handwritten digit databases on P4 1.7 Ghz were about 16,000 and 10,895 patterns per second without sacrificing the classification accuracy of the original SVM system. The speed-up factor of 110 on MNIST database has been achieved.

- Finally, we summarize this thesis in Chapter 8 with some concluding remarks.

1.3 Name conventions

In order to make notations internally consistent, we adopt some general principles as follows. Lower-case bold letters, for example, $\mathbf{x}$, are used to denote vectors, while upper-case bold letters, such as $\mathbf{A}$, denote matrices. Vectors are considered to be column vectors, with the corresponding row vector denoted by a superscript T indicating the transpose. The notation $\mathbf{A} = (A_{ij})$ is used to denote that the matrix $\mathbf{A}$ consists of elements A_{ij}. $(\mathbf{A})_{ij}$ is used to denote the ij element of the matrix $\mathbf{A}$. k is used to denote an index or kernel.

The following symbols are used frequently in the thesis and they are listed below:

$\mathbb{R}^d$	d-dimensional real space
$\mathbf{x}$	column vector
$\mathbf{A}^T, \mathbf{x}^T$	transpose
$\mathcal{C}_k$	kth class
$\in$	element in set
$\| \cdot \|$	norm of a vector
$\subseteq$	set inclusion
μ_k	mean vector for class $\mathcal{C}_k$
$\mathbf{e}$	vector of ones $(1, 1, \ldots, 1)^T$
$P(\cdot)$	probability
$p(\cdot)$	probability density function
Σ	covariance matrix
$\mathbf{w}$	normal vector to a hyperplane
∇	gradient operator

Chapter 2

Main classification methods and computational techniques

In this chapter we first review some main classification methods to show their advantages and shortcomings in terms of accuracy and computational costs when these existing methods are applied to a large data set. These methods include not only traditional ones such as discriminant functions, nearest neighbor-based method, but also modern advanced classifiers such as ensemble methods (Bagging [8] and AdaBoost [45]) and support vector machines. The survey of these methods can provide us some important clues about classifier selection and design. Then some computational tools and techniques are introduced to design an efficient and robust learning algorithm.

2.1 Main classification methods

Many classification methods can be found in the literature. We do not attempt to provide a complete review of existing methods, but rather to describe some of the most important and popular techniques such as discriminant functions, neural networks and kernel-based methods.

2.1.1 Linear and quadratic discriminant functions

Linear discriminant function [40] is a simple and basic classification method. Linear discriminant function performs well when data from different classes are linearly

separable or the covariance matrices of all the classes are identical [40]. The linear function (or hyperplane) is generally written as

$$g_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k. \tag{1}$$

where $\mathbf{w}_k$ is called the weight vector for class $\mathcal{C}_k$ and b_k the threshold. When data is linearly separable, the weight vector and threshold can be obtained by perceptron learning rules [93]. For non-separable cases, these parameters can be obtained by assuming that data distribution is gaussian-like or minimizing the mean square error [40]. For a two-class problem, weight vector can be determined by Fisher criterion [42]. Although linear discriminant function can not achieve a high accuracy in most real cases, its appealing property is low computational cost, only n multiplications and one addition where n is the dimension of input vector. The expression of dot product can be easily implemented in vector processors and a single processor with stream instruction and multiple data (SIMD) technology that is widely used in Intel Pentium series. Therefore, for a large-scale classification problem with thousands of classes, linear discriminant function is a good choice for pre-classification.

Quadratic discriminant function is one of the popular parametric methods when multivariate normal distribution is used as the class density function. It can be given as

$$g_k(\mathbf{x}) = (\mathbf{x} - \mu_k)^T \Sigma_k^{-1} (\mathbf{x} - \mu_k) + \log |\Sigma_k| - 2 \log P_k. \tag{2}$$

for class $\mathcal{C}_k$ where μ_k and Σ_k denote the mean vector and covariance matrix for $\mathbf{x}$ in class $\mathcal{C}_k$, respectively and P_k is the *a priori* probability for class $\mathcal{C}_k$. Since it employs the covariance matrix, the required computational time and storage is $O(n^2)$ for n-component feature vectors. The performance of the quadratic discriminant function is largely sensitive to the estimation errors of a covariance matrix. Kimura et al. [66] proposed a modified quadratic discriminant function (MQDF), where a pseudo-Bayesian estimate [1] of the form $\lambda\Sigma_0 + (1-\lambda)\Sigma_k$ [40] is employed where Σ_0 is a diagonal matrix and the data is projected onto d-dimensional low-rank eigen subspace which is spanned by principal components. As a result, the performance of MQDF is less sensitive to the estimation error of covariance matrix than that of QDF and the computation cost and storage is much less than QDF's, just $O(nd)$ where d is usually much smaller than n. MQDF performs very well in handwritten

[1] also called regularized estimate of covariance matrix

character recognition [67][68][27]. The possible reason of its success is that the data can support a simple decision boundary such as quadratic and estimates via Gaussian models are stable [52]. The problem of curse of dimensionality can be alleviated to some extent by controlling the dimension d of eigen subspace. Nevertheless, when a large number of training samples are available and problems involve multimodal (multi local maxima) densities, non-parametric methods such as nearest neighbor and support vector machine usually perform better than MQDF.

2.1.2 Prototype methods

Let the prototype consist of N pairs $\{(\mathbf{x}_i, y_i)\}$, $i = 1, \ldots, l$ and y_i is the label of sample $\mathbf{x}_i$. In most cases, $\mathbf{x}_i$ associated with the prototype is typically an example from the training set. The classification of an unseen pattern $\mathbf{x}$ is to assign its class to the label of the closest prototype by a distance measure[2].

1-nearest neighbor is one of the simple and popular prototype methods, where prototypes are all training samples. The good property is that classification do not need to take into account the form of data density function and its asymptotic probability of error is never greater than twice the Bayesian error [20]. In order to achieve a high accuracy, a huge number of training samples are required. Consequently its computational cost is prohibitively high.

In learning vector quantization (LVQ) due to Kohonen [69], prototypes are placed with respect to the decision boundary to reduce the classification error by attracting the prototypes of the correct class and repelling prototypes of incorrect class. The decision boundary of LVQ is piece-wise hyperplane. LVQ is defined in the form of algorithm, rather than optimization of a cost function, which makes it difficult to analyze its property. Generalized learning vector quantization (GLVQ) [95] adjusts the prototypes based on minimization of classification errors (MCE) [60] and the classification performance is improved. The performance of both methods largely depends on initial prototypes, which are usually set by some clustering methods such as k-means, self-organizing mapping [70]. Compared with 1-nearest neighbor, LVQ usually achieves a better performance and much lower computation cost.

[2]For example, Euclidean distance.

2.1.3 Feed-Forward Neural Networks

Multi-layer feed-forward neural network is widely applied in pattern recognition. Its success is most likely attributed to the efficiency of gradient-based back-propagation learning algorithm [94] and powerful nonlinear approximation capability. Networks with sigmoidal nonlinearities and two layers of weights can approximate any decision boundary to an arbitrary accuracy [49]. Thus it can be used to model the *posterior* probabilities of class membership. For multi-layer proceptron (MLP) trained by minimizing a sum-of-square cost function, the network output can be interpreted as the conditional average of the target data when training data is sufficiently large [4].

Nevertheless, many problems such as overfitting and local minima associated with MLP have not been solved yet. One of the most important problems is how to determine the network structure such as connection method and number of layers to achieve a good generalization performance on a large database. Network pruning, a process of deleting the irrelevant weights of a network before invoking inference, can be used to optimize the size of the network. Some methods eliminate the weights with the smallest magnitude [53]. Optimal brain damage [77] and optimal brain surgeon [51] use the information of all second order derivatives of the error function for network pruning. Although these methods can improve the generalization performance, the computational cost of pruning an initial large fully connected network is high. For other methods such as LeNet1 [76] and LeNet5 [79], the network structure is customized manually for the specific application. LeNet1, a small network, is built for USPS handwritten digit database while LeNet5 for MNIST digit database. Both networks contain feature extraction layers, where network weights are locally connected and shared so that the number of free weights are greatly reduced. However, to build such networks requires good prior knowledge.

2.1.4 Ensemble methods

The ensemble method is one way of combining classifiers which are constructed based on data re-sampling. When the outputs of a classifier are interpreted in terms of bias-and-variance decomposition, the ensemble methods mainly reduce the variance of these single classifiers. Bagging [8] and AdaBoost [45] are popular ensemble methods. Bagging employs the bootstrap sampling method to generate training subsets

while the creation of each subset of AdaBoost depends on previous classification results. For two methods, the final decision is made by the majority vote. Numerous experiments [8][2][107][86] have shown that Bagging and AdaBoost are just effective for weak classifiers such as classification tree, neural networks and perform well in a small dataset.

However, on the large data set, two issues for the ensemble methods need to be taken into account. If an ensemble method generate N classifiers and their outputs are combined, the cost is about N times as high as a single base classifier. In addition, if the training data set is too large to be stored in memory, the cost of re-sampling can not be ignored. The other issue is that the base classifiers for ensemble method learn the sampling subset, rather than the total set. As a result, if the number of constructed classifiers is not large enough, many training samples will not be learned. As a result, the ensemble method may not perform better than a single base classifier [28].

2.1.5 Kernel-based methods

Kernel method is usually used to estimate the data density function. Suppose we have samples $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_l$ that are generated from a probability density function $f(\mathbf{x})$. The smooth *Parzen* estimate [89] is given by

$$f_l(\mathbf{x}) = \frac{1}{lh^d} \sum_{i=1}^{l} k(\frac{\mathbf{x} - \mathbf{x}_i}{h}). \tag{3}$$

where k is kernel function and integrable on $\mathbb{R}^d$ and $\int k = 1$, $\mathbf{x} \in \mathbb{R}^d$ and h (> 0) denotes the bandwidth of the kernel. Under some general conditions f_l can prove to consistently converge [3] to $f(\mathbf{x})$ [89]. That is, with enough samples, the kernel method can be used to estimate an arbitrarily complicated unknown density, whatever unimodal and mixture of multi-modals. On the other hand, a large number of samples are required to grow exponentially with the dimensionality of feature space in order to get a good estimate. Once we get non-parametric density estimate for each class, Bayes's theorem can be applied to classification directly.

If our purpose is classification, the estimation of each class density function separately may be unnecessary and even misleading. For classification, the *posteriori*

[3] converge in mean square

probabilities are sufficient [40]. An estimate of the class density function will increase the extra learning complexity which is more likely to degrade the performance in terms of Occam's razor principle. From the geometric viewpoint, we only need to care about the decision boundary which is relevant to classification. For example, in neural network literature, the center of RBF network is determined by some density estimation method such as clustering. The strategy does not perform best in terms of the above argument.

In recent years support vector machine, one of the kernel methods, plays a key role in pattern classification and has achieved promising performances in many applications such as handwritten digit recognition [24], classification of web pages [57] and face detection [87]. SVM holds many good properties in both theoretical and practical aspects. It maps the original data by means of a Mercer kernel [85] to a high dimensional feature space, where a linear hyperplane with the minimal empirical risk and maximal margin in this feature space is found. Its non-asymptotical error which depends on empirical risk and margin can be bounded using Vapnik-Chervonenkis (VC) theory [121] while the performance of most previous methods can be analyzed only asymptotically. Many problems in pattern recognition such as curse of dimensionality and over-fitting often occur in a finite data set. However, when the suitable kernel and its parameter is selected, these problems seldom happen for SVM even in an infinite dimensional space. The other appealing property of SVM is that its classifier structure is data-driven and automatically determined by solving a constrained convex quadratic programming problem. This avoids customizing the structure manually to achieve a high performance in contrast to a neural network classifier. For example, LeNet5 [79], a variant of convolution network, is highly customized to character recognition and has shown much better performance than fully connected multi-layer perceptron (MLP). Moreover, Training a SVM can achieve a unique global minimum while many local minima exist during the training of a usual neural network.

However, there are two important problems for SVM, which limit its wide applications. One is that the computational cost of existing SVM training algorithms is prohibitively high on a very large data set with thousands of classes. The other is that many support vectors are generated after the optimization process. As a result, its testing speed is very low. Therefore, it is necessary to develop fast algorithms for

SVM's training and testing.

2.2 Computational techniques

The computational cost of a classifier depends on not only its own structure but also to a large extent on the implementation issues related to other research areas such as numerical analysis, computer architecture and operating systems. When we design a classifier for a very large data set, usually there is a large amount of data movement between the CPU and memory, memory and hard disk. Without good design, paging operations and cache misses will occur frequently, which possibly dominates the overall cost. In the communities of machine learning and pattern recognition, these issues are rarely mentioned in the literature. It is important for designers to know some techniques or issues about high performance computation and design their classification algorithm to suit the current processor architecture and operating system in order to make full use of the advanced properties of current processors and operating systems and reduce or avoid data traffic caused by a large amount of data movement. In the following, several techniques or strategies are presented to design a classifier efficiently.

2.2.1 BLAS

In statistical pattern recognition, vector or matrix operations are basic. Basic Linear Algebra Subprogram (BLAS) [74][75][37][38] is a package with the aims of providing an efficient and portable implementation of these operations on high-performance computers, especially those with hierarchical memory and parallel processing capability. The BLAS is divided into three levels: Level 1 BLAS provides a standard interface for vector-vector operations [74], Level 2 BLAS deals with matrix-vector operations, and Level 3 BLAS supports matrix-matrix operations. The performance gain from the optimized implementation strongly depends on the level of the BLAS. In the above three levels, the performance gain of Level 3 BLAS is the most obvious and its optimized implementation usually shows orders of magnitude of speed-ups, compared with naive code generated by compilers. Among the Level 3 routines, the most important one is matrix-matrix multiplication. It is often preferable to partition the matrices into blocks and to perform the operations on these blocks in order to

make full reuse of data which can be stored in cache or local memory. If the involved matrix is of order n, the efficient memory blocking can make naive $O(n^3)$ fetch cost become $O(n^2)$ due to the reuse of operands[124].

BLAS has been implemented on different platforms by hardware vendors such as IBM's ESSL[4], Intel's MKL[5], SGI's SCSL[6]. More information about BLAS can be found in BLAS FAQ[7]. These hand-tuned BLAS provides a much higher performance gain than the naive code. Among the optimized BLAS, ATLAS[124](Automatically Tuned Linear Algebra Software) is an approach for automatic generation and optimization of numerical software across different hardware platforms by generating many versions for the same routine and finding the best to perform the operation using sophisticated search techniques and robust timing mechanisms. ATLAS has been tested on different platforms and its performance is very competitive with that of vendor-supplied BLAS[124]. The software is available in public from the website[8].

2.2.2 Optimization techniques for processor architecture

In the past decade the exponential growth of computer power in close relation to Moore's law has made large-scale computing feasible at lower cost. The current processor provides new features and enables software developers to deliver a higher level of performance than previous ones in multi-medial applications ranging from 3-D graphics, video encoding/decoding to intelligent computing (speech recognition, bioinformetric computation et al.). For these applications, a large amount of data have to be processed. One way to increase the performance is to execute several computations in parallel so that multiple computations are done with a single instruction. The single-instruction, multiple-data (SIMD) is a technique to achieve this type of parallel computation. The other way is to optimize the data memory accesses to reduce cache miss and avoid performance penalties due to a large number of paging operations. In order to understand these techniques, it is necessary to know some basic characteristics of modern microprocessor. We use Intel's P4 processor to illustrate these characteristics due to its cutting-edge performance and popularity. Figure 1

[4]http://www.rs6000.ibm.com/software/Apps/essl.html
[5]http://developer.intel.com/software/products/mkl/index.htm
[6]http://www.sgi.com/software/scsl.html
[7]http://www.netlib.org/blas/faq.html
[8]http://www.netlib.org/atlas/

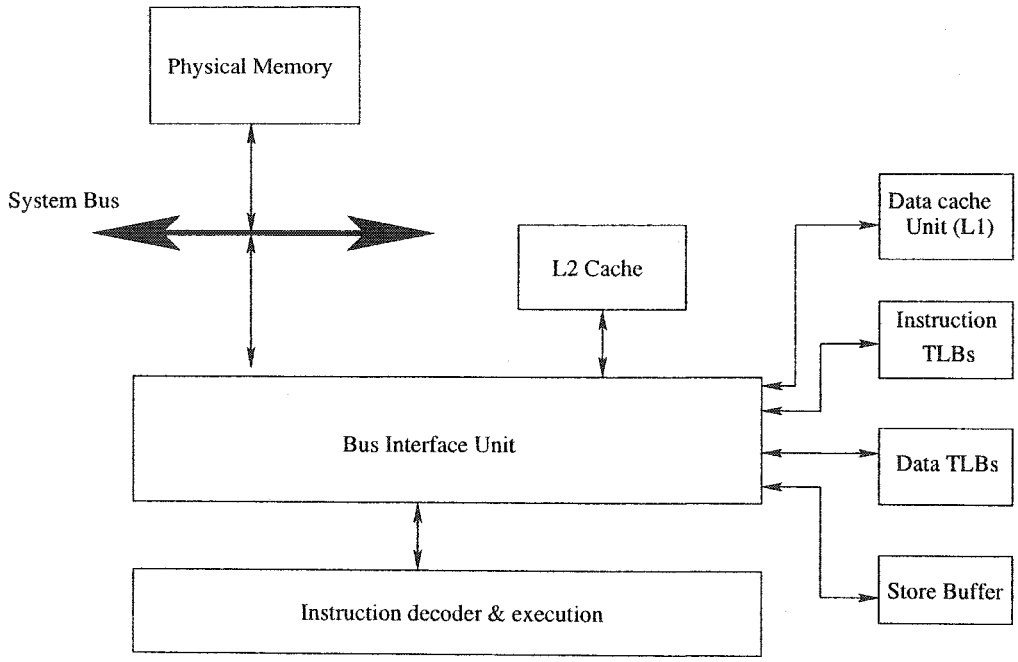

Figure 1: Cache structure of the Pentium 4 processor

shows the cache structure of the Pentium 4[54].

Pipeline and execution units

In P4, instructions are fetched and decoded into a sequence of μ ops which are stored in trace cache [55]. This helps to build a long pipeline. For example, the unit that performs the multiplication can start a new multiplication as soon as the old one finishes the first stage. Although it does not make one multiplication operation finish faster, it helps to execute several multiplications simultaneously. In addition, P4 has more execution units, including two arithmetic logical units (ALU) and one float execute units, load/store units. That is, integer addition and floating point multiplication can execute simultaneously per clock cycle. The above mechanisms enhance parallelism.

Cache and TLBs

Cache is a set of small, high-speed buffer memories that temporarily store the contents of main memories which are currently used. Accessing the data in cache takes much

less time than in main memory. The processor with a cache memory can spend far less time waiting for instructions and data to be fetched or stored. That is, cache memory can alleviate the mis-match between CPU's and main memory's speed. Data is fetched from the main memory to the cache line that is the smallest data transfer unit. For example, the line size of L1 (the first level cache) in P4 is 64 bytes. So when you only read one byte from the main memory, you will actually get 64 bytes. From Figure 1 it can be observed that P4 contains separate data and instruction caches. L2 (the second level cache) is shared by data and instructions. All caches use LRU (least recently used) replacement algorithm to update the cache. Table 1 provides the parameters for L1 and L2 caches in P4 [55].

Table 1: P4 processor cache parameters

Level	Capacity	Associativity	Line Size(bytes)	Write Update Policy
L1	8 KB	4	64	write through
L2	256 KB or 512 KB	8	64	write back

In Table 1, the concepts of some cache parameters are referred to [90][108].

TLBs (Translation Look-aside Buffers) store the most recently used page-directory and page-table entries. They speed up memory accesses when paging is enabled by reducing the number of memory accesses that are required to read the page tables in system memory. When TLBs do not contain the translation information for a requested page, extra bus cycles to page directory and page tables in memory are performed. As a result, the cost will be increased. The TLBs are usually active in protected mode with paging enabled. Table 2 shows the parameters for data and instruction TLBs in P4 processor [55].

Table 2: P4 processor TLB parameters

Category	Entries	Associativity
Instruction TLB (4-KByte Page)	128	4
Data TLB (4-KByte Page)	64	Fully set

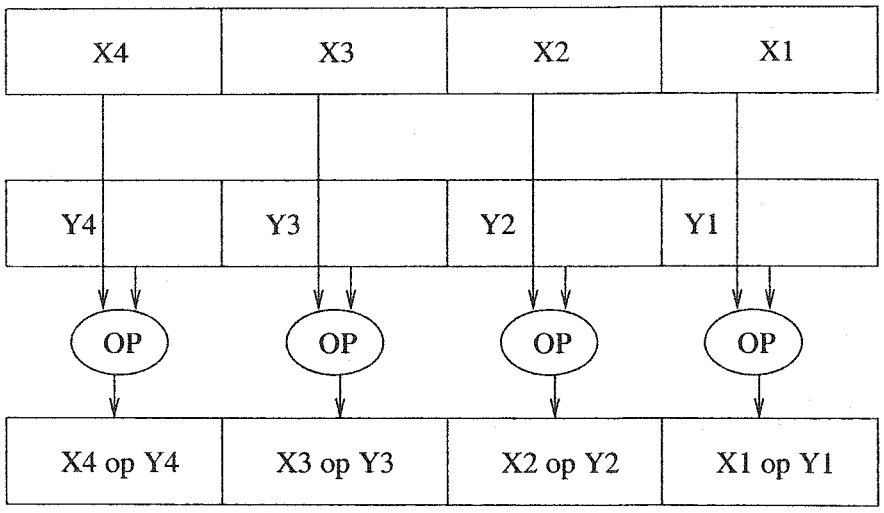

Figure 2: Typical SIMD operations

SIMD technology

SIMD (Single instruction and multi-data) is a technology to execute several computations in parallel. Figure 2 shows a typical SIMD computation [55]. where four packed data elements (X1, X2, X3, X4, and Y1, Y2, Y3, Y4) are operated in parallel. The same operation (OP) is applied to the corresponding pair of data elements (X1 and Y1, X2 and Y2, X3 and Y3, and X4 and Y4). The results of the four parallel computations are stored as a set of four packed data elements. For example, op is multiplication and data type is single-precision float point. That is to say, a single instruction can simultaneously perform four float-point multiplications. Therefore, this is a useful technique for processing vector data. Nevertheless, SIMD technology has a high demand of data caching. If an SIMD instruction starts, data is still in the main memory, instead in L1 cache. The operation stalls and waits for fetching data from the main memory. It will take more CPU cycles. In the following subsection, some empirical rules are recommended for a better data caching.

Rules for data caching

Cache blocking is an important technique that reduces the bus traffic and makes maximal reuse of L1 and L2 caches and reduces the cache miss rate and thereby improves the memory utilization performance. This technique has been successfully

applied to high performance computation in memory-hierarchy computers [125]. The basic idea is that if an application uses a large data set that can be reused multi-times the performance gain can be obtained by dividing the large data set into groups which are small enough to fit in the cache. Although many techniques in the P4 processor can be used to improve memory utilization, we just focus on some general rules that can be suitable for different platforms. The main points are summarized as follows:

- Alignment of data. Data are aligned on their nature size address boundary. For example, the base address of 32-bit data is a multiple of four. In P4, the size of a cache line is 64 bytes. Therefore, a 64-byte or greater data structure or array should be aligned so that its base address is a multiple of 64. Accesses across the cache line boundary will incur a significant performance penalty.

- Data are contiguous. When data is contiguous, the space locality is increased. As a result, the cache hit ratio will be increased when these data are sequentially accessed since most processors support automatic local data prefetch.

- Re-organize the data and enable vectorization. The purpose is to make use of SIMD technology to speed up computation. For example, we calculate the dot product of an array of 2-D vector and a fixed vector. In general, we can define the following data structure:

```
struct vector {
   float x;
   float y;
} VecArray[256];
vector a;
float dotProduct[256];

Initialize a and VecArray;
for (int i = 0; i < 256; i++)
    dotProduct[i] = a.x * VecArray[i].x + a.y * VecArray[i].y;
```

  In order to make efficient use of SIMD operations, data are re-organized as follows:

```
float xVecArray[256];
float yVecArray[256];
float aX[4];
float aY[4];
float dotProduct[256];
Initialize xVecArray, yVecArray, aX and aY, where x component of
the fixed vector is duplicated into the four slots, the same for
y component of the fixed vector.
float *p1, *p2, *p3;
p1 = xVecArray;
p2 = yVecArray;
p3 = dotProduct;
for ( int i = 0; i < 64; i++)
{
    movaps xmm0, p1;
    movaps xmm1, p2;
    mulps  xmm0, aX;
    mulps  xmm1, aY;
    addps  xmm0, xmm1;
    movaps p3, xmm0;
    p1 += 4;
    p2 += 4;
    p3 += 4;
}
```

In the implementation of SIMD, xVecArray, yVecArray, aX, aY and dotProduct are aligned to a 16-byte boundary and the four dot products in the inner loop are calculated, compared with one dot product in the traditional X87 FPU implementation. This computation is more efficient due to the intrinsic parallelism of SIMD operations.

- Block a large data set that will be re-used many times. The large data set is divided into several groups, each of which can fit into an L2 cache. An aligned contiguous workspace is allocated. These group data sets are copied to the workspace and operations are performed in the workspace. The benefit is to

increase the cache hit rate. At the same time, TLB misses will be reduced. The technique is very useful for a large-scale learning algorithm since the data set is usually passed many times and is large so that it can not be fit into the memory. Sometimes even though data can be stored in the memory, the finite TLBs can not contain all page numbers of these data set.

2.2.3 Parallel algorithm on multi-processors

The performance of uniprocessor, driven by the microprocessor, has been considerably enhanced. But its power is still limited when a large-scale computation is required. Parallel machines may play an increasingly important role in the future. The way to improve the performance of a uniprocessor is to construct a distributed-memory parallel machine which consists of many off-the-shelf microprocessors and pieces of network equipment which are in charge of communications between microprocessors. In the literature of pattern recognition and machine learning, less work has been done to design a learning algorithm to run on parallel machine although parallel computing is a high demand for a large-scale learning. Aberdeen et al. [1] designed a cluster of 196 Pentium III processors to train a neural network which consists of about two million adjustable parameters for Japanese Optical Character Recognition. The cluster runs with an average performance of 163 GFlops per second and a price/performance ratio of 92.4 cents/MFlops/s has been achieved, which is very attractive. But Aberdeen et al. did not design a parallel training algorithm which is better fit into the cluster.

In the parallel machines which consist of multi-processors, insufficient parallelism and long latency network communication are the two biggest challenges [90]. The problem of an inadequate parallism can be alleviated primarily in software with an algorithm that has a better parallel performance. The latter can be dealt with by the architecture and network hardware, communication software. When a single microprocessor becomes more powerful and network communication is a bottleneck, designing a better parallel learning algorithm becomes more important. Several empirical rules are recommended to design a good parallel learning algorithm:

- Block the original problem into sub-problems. Full attention must be paid to reduce the dependency among these sub-problems for maximal parallelism.
- The data for each sub-problem are stored locally as much as possible in order

to reduce the communication cost of the network.

- The algorithm for solving sub-problems is optimized to fit the memory-hierarchy microprocessor.

2.3 Summary

In this chapter we review some main classification techniques in the literature and point out their advantages and shortcomings in terms of accuracy and computational cost when they are applied for the classification of a large dataset. Then some useful optimization techniques related to computer architecture are presented and general optimized principles are suggested to help design a fast, robust and industry-strength classifier. These techniques may be useful for readers who are interested in an efficient implementation of a classifier to solve real-world classification problems.

Chapter 3

Local Learning Framework

3.1 Introduction

Over the last decade, neural networks have been widely applied to solve very complex classification problems in the real world. There is a growing realization that these problems can be facilitated by the development of multi-net systems [105]. Multi-net systems can provide feasible solutions to some difficult tasks that could not be solved by a single net. A single neural net often exhibits the overfitting behavior which results in a weak generalization performance when trained on a limited set of data. Some theoretical and experimental results [72][118] have shown that an ensemble of neural networks can effectively reduce the variance that is directly related to the classification error.

A number of studies have addressed the problem of the construction of a multi-net system to achieve a better performance. The ensemble ("committee") and modular combination are two basic methods used to construct multi-net systems. The two popular ensemble methods are Bagging [8] and AdaBoost [45]. Bagging employs the bootstrap sampling method to generate training subsets while the creation of each subset in AdaBoost depends on previous classification results. Unlike Bagging, AdaBoost obviously attempts to capture classification information of "hard" patterns. However, the flaw is that it can also easily fit the noise in the training data. In modular combination, the task or problem is decomposed into a number of subtasks, and a complete task solution requires the fusion of outputs of all the modules [105]. Jocobs [59] proposed a mixture-of-experts model that consists of expert networks and

a gating network. The training goal is to have the gating network learn an appropriate decomposition of the input space into different regions and switch the most suitable expert network to generate the outputs of input vectors falling within each region. In the model, it is assumed that data in local regions has a gaussian distribution, which is usually not true for a complex data distribution. Further, the model only selects the most suitable expert network to make a decision, rather than combining the decisions of different expert networks. Most experiments show that an ensemble method in a local region is more effective than the individual best neural network.

In this chapter, we present a method to construct a hierarchical local learning framework for pattern classification to systematically address the above problems. The framework consists of two layers. In the first layer, the Learning Vector Quantization (LVQ) approach is used to partition the pattern space into clusters or subspaces. In the second layer, different ensembles of local learning machines that are trained in local neighborhood regions are combined to make the final decision. LVQ which minimizes the average expected misclassification error, builds piecewise linear hyperplanes between neighboring codebook vectors to approximate Bayes decision boundary [71]. Due to the complex shape of decision boundary in real-world classification problems, approximation by piecewise linear hyperplanes results in approximation error. We have to use more powerful neural network ensembles in order to improve and fine-tune the approximation of the Bayes decision boundary.

The remainder of the chapter is organized as follows. First, the learning framework is presented, followed next by a discussion on how to choose the right models. In Section 4 we provide experimental results on several public databases of handwritten characters to illustrate the advantage of the proposed method. The conclusions are given in Section 5.

3.2 Formulation of learning framework

We start with formally defining each part of the proposed learning framework. Fig. 3 shows a basic structure of the system that consists of a vector quantization layer and an ensemble layer.

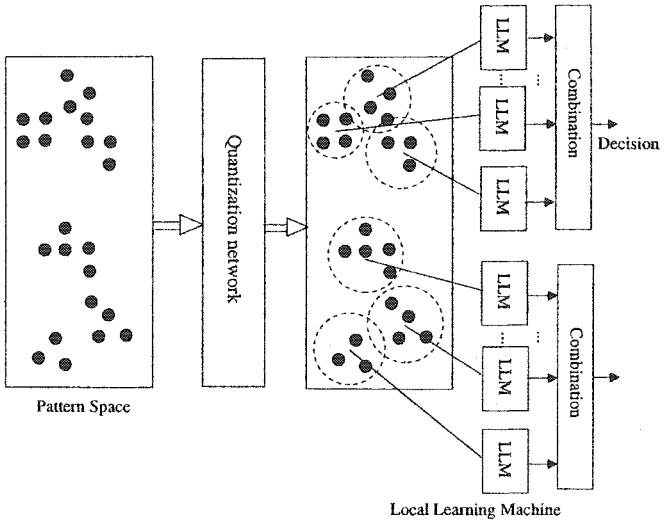

Figure 3: A general local learning framework. Here local learning machines denote classifiers that are designed in local regions.

3.2.1 Vector quantization

Vector quantization can be viewed as low cost signal compression method where most information is stored in a number of codebook vectors. The classical mean square error (MSE) criterion is often assumed as a design criterion. It has been shown that for labelled patterns an increase of the codebook size resulting in increased MSE accuracy does not necessary lead to an accurate reproduction of the Bayes rule [26]. Following Bayes decision philosophy Kohonen intuitively introduced LVQ1, which utilizes information about the class to which a pattern belongs [70]. It has been shown [25] that Kohonen's LVQ1 which does not minimize some explicitly risk function also does not minimize the Bayes risk. Juang & Katagiri [60] proposed an effective discriminant learning scheme called Minimum Classification Error (MCE) approach that minimizes the expected loss in Bayes decision theory by a gradient descent procedure. Several generalized LVQs schemes based on the MCE [61][95][96] have been proposed. In this paper we adopt the generalized learning vector quantization (GLVQ) approach [95][96] due to the fact that convergence of reference vectors is guaranteed [97].

Let $\mathbf{m}_{kr}$ be the r-th reference vector in class $\mathcal{C}_k$. Let $\Lambda_k = \{\mathbf{m}_{kr} | r = 1, \cdots, N_k\}$, $k = 1, \cdots, c$ where c is the number of classes, and $\Lambda = \bigcup_{k=1}^{c} \Lambda_k$ is the collection of all reference vectors. The feature space region for class $\mathcal{C}_k$ is $\mathcal{R}_k$. Suppose that input

vector $\mathbf{x} \in \mathcal{R}_k$ is presented to the system. Define discriminant function $g_k(\mathbf{x}; \Lambda)$ of class $\mathcal{C}_k$ as follows:

$$\begin{aligned} g_k(\mathbf{x}; \Lambda) &= -d_k \\ &= -\min_r \| \mathbf{x} - \mathbf{m}_{kr} \|^2 \\ &= -\| \mathbf{x} - \mathbf{m}_{ki} \|^2, \end{aligned} \tag{4}$$

where $\| \cdot \|$ denotes the Euclidean norm. The misclassification measure $\mu_k(x; \Lambda)$, is defined as follows:

$$\begin{aligned} \mu_k(\mathbf{x}; \Lambda) &= \frac{-g_k(\mathbf{x}; \Lambda) + g_l(\mathbf{x}; \Lambda)}{g_k(\mathbf{x}; \Lambda) + g_l(\mathbf{x}; \Lambda)} \\ &= \frac{d_k - d_l}{d_k + d_l}. \end{aligned} \tag{5}$$

where $g_l(\mathbf{x}; \Lambda) = \max_{i \neq k} g_i(\mathbf{x}; \Lambda) = -d_l = -\| \mathbf{x} - \mathbf{m}_{lj} \|^2$. If $\mu_k(\mathbf{x}; \Lambda) \geq 0$ then $\mathbf{x}$ is misclassified, and $\mu_k(\mathbf{x}; \Lambda) < 0$ corresponds to the correct decision.

In Bayes decision theory, we often minimize a risk function to obtain the optimal decision. In order to make loss function differentiable, we use a "soft" nonlinear sigmoid function instead of a "hard" zero-one threshold.

$$\begin{aligned} l_k(\mathbf{x}; \Lambda) &= l_k(\mu_k(\mathbf{x}; \Lambda)) \\ &= \frac{1}{1 + \exp(-\xi(t)\mu_k(\mathbf{x}; \Lambda))} \quad (\xi > 0). \end{aligned} \tag{6}$$

Our cost function is an empirical loss given by

$$L(\Lambda) = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{c} l_k(\mathbf{x}_i; \Lambda) 1_{(\mathbf{x}_i \in \mathcal{R}_k)} \tag{7}$$

where N is the number of training samples and $1_{(.)}$ is an indicator function. The cost function can be minimized by a gradient descent procedure

$$\Lambda_{t+1} \leftarrow \Lambda_t - \epsilon(t) \nabla L(\Lambda_t) \tag{8}$$

where Λ_t denotes the parameters set at the t-th iteration and ϵ is the learning rate. Then according to equation (8) the learning rules for the reference vectors can be written

$$\begin{aligned} \mathbf{m}_{ki} &\leftarrow \mathbf{m}_{ki} + 4\epsilon(t)\xi(t)l(\mu_k)(1 - l(\mu_k))\frac{d_l}{(d_k + d_l)^2}(\mathbf{x} - \mathbf{m}_{ki}) \\ \mathbf{m}_{lj} &\leftarrow \mathbf{m}_{lj} - 4\epsilon(t)\xi(t)l(\mu_k)(1 - l(\mu_k))\frac{d_k}{(d_k + d_l)^2}(\mathbf{x} - \mathbf{m}_{lj}) \, . \end{aligned} \tag{9}$$

3.2.2 Construction of ensembles

After performing vector quantization, each reference vector can be regarded as a cluster center. Although we can collect training subsets for each cluster by the nearest neighboring rule and train local learning machines on these subsets, we face the following limitations:

1. The numbers of training samples in some of these subsets are not large enough; as a result, classifiers designed from these subsets will exhibit weak generalization ability.

2. This method ignores information contained in the "boundary patterns" between neighboring clusters while most misclassification errors occur near these boundaries.

In order to overcome the above problems, we add neighboring samples into the training subsets. The procedure can be summarized as follows:

Collect training subsets

Input: A series of training samples $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$, reference vectors $\mathbf{m}_i$, $i = 1, \cdots, \sum_{k=1}^{c} N_k$; S_j, $j = 1, \cdots, \sum_{k=1}^{c} N_k$, where N is the number of samples, c and N_k denote the number of classes and the number of reference vectors for class C_k, respectively.

Output: training subsets S_j, $j = 1, \cdots, \sum_{k=1}^{c} N_k$.

Initialize: Set sets S_j to be empty.

for $p = 1$ to N

1. For sample $\mathbf{x}_p$ find L nearest neighboring reference vectors, i.e.,

$$\begin{aligned} i_1 &= \arg\min_i \| \mathbf{x}_p - \mathbf{m}_i \| \\ i_k &= \arg\min_{i \notin \{i_1, i_2, \cdots, i_{k-1}\}} \| \mathbf{x}_p - \mathbf{m}_i \| \end{aligned}$$

where $\| \cdot \|$ denotes the Euclidean norm and $k = 1, \cdots, L$.

2. Inject sample $\mathbf{x}_p$ into L training subsets

$$S_j = \{\mathbf{x}_p\} \bigcup S_j \quad j \in \{i_1, i_2, \cdots, i_L\}$$

end **for**

From the above procedure, we can observe that the obtained sets S_j are partially overlapping. Next, we build the ensemble of networks following the Bayesian framework. We employ neural networks to model the *posteriori* probability by a mixture of the neighboring expert nets, i.e.,

$$P(\mathcal{C}_k|\mathbf{x}) = \sum_{i=1}^{L} P(e_i)P(\mathcal{C}_k|\mathbf{x}, e_i) \;\; k = 1, \ldots, c. \tag{10}$$

Here we assume that all expert nets e_i, $i = 1, \ldots, L$ are independent. $P(e_i)$ denotes a *priori* probability of expert net e_i and $P(\mathcal{C}_k|\mathbf{x}, e_i)$ means *posteriori* probability for expert net e_i.

Finally, the classification procedure for a new pattern can be summarized as follows:

1. Calculate Euclidean distances between a new pattern $\mathbf{x}$ and reference vectors.
2. Find L closest reference vectors to $\mathbf{x}$.
3. Feed new pattern $\mathbf{x}$ into the expert nets located at L reference vectors and calculate their outputs.
4. Calculate *posteriori* probability $P(\mathcal{C}_k|\mathbf{x})$ of the class label according to Eq. (10).
5. The final decision is the label maximizing $P(\mathcal{C}_k|\mathbf{x})$, $k = 1, \ldots, c$.

3.3 Model selection

The framework introduced in section 2 can be applied to different learning models. Thus model selection plays an important role in the overall performance of the designed system. Although some clustering models such as self-organizing maps [71] are available for vector quantization, these models have a common flaw due to the fact that they use the MSE criterion, which is not directly related to the Bayes risk minimization. To avoid this problem we select GLVQ vector quantization scheme.

GLVQ assumes that initial positions and number of reference vectors of each class are known while in practice this information is usually not available. It is well

known that initial positions of reference vectors have a great impact on the final performance of some clustering algorithms such as self-organizing maps, LVQ and neural gas. LVQ typically employs k-means algorithm on the training data of each class to obtain initial positions of the reference vectors. However, the classical k-means algorithm often converges to a "bad" local minimum. Further, high computation cost often renders k-means algorithm unfeasible for large-scale clustering. Here we use an algorithm proposed by[7], which uses k-means algorithm and "smoothing" procedure to refine the initial points. That is, multiple sub-samples J are drawn and clustered independently with K centers via the k-means method. Then each of J solutions is refined among $K * J$ centers using k-means and one of the solutions having the minimal distortion over $K * J$ centers is chosen. This algorithm is especially suitable for large-scale clustering tasks since clustering occurs only on sub-samples and the solution is not easily corrupted by "outliers".

In the ensemble layer, multi-layer perceptron (MLP) [94] is used as a local learning machine due to two reasons: first, powerful nonlinear decision capability of the MLP can be used to discriminate "hard" patterns; second, combination of the component experts decisions can be motivated by the *posteriori* probabilistic interpretation of the outputs.

Finally, we use a simple averaging method to combine component expert nets because the *priori* probability of each expert net is unknown.

3.4 Experimental results

In this section, we present test results for our learning framework applied to several public data sets of handwritten characters. We thoroughly discuss the results of experiments, describe some related design parameters and discuss some practical implementation issues. In addition, we provide an extensive performance comparison with other popular classifiers.

In our experiment, linear normalization and feature extraction based on the stroke edge are applied. All character images are size-normalized to fit the 32×32 box while their aspect ratios are preserved. Also, a directional feature from the gradient of gray scale image [46] is extracted by using the Robert edge operator. That is, we first use the mean 3×3 filter to smooth the image. Subsequently the smoothed

image is divided into 9×9 blocks. In each block, the strengths of the gradients with each of 32 quantized gradient directions are accumulated, so that the 81 gradient directional spectra of the original image are generated. The spatial resolution is reduced from 9×9 to 5×5 by down sampling with a 5×5 gaussian filter. Similarly the directional resolution is reduced from 32 to 16 by down-sampling with a 1×5 one-dimensional gaussian filter. Finally a feature vector of size 400 (5 horizontal, 5 vertical and 16 directional resolutions) is produced. Since most components of the feature vector are zero, principal component analysis can be employed to compress the highly dimensional feature vector to a 160-dimensional vector, which is then used in all subsequent experiments.

Before we present the results of experiments, we discuss some related design parameters. For GLVQ, we first use Bradley's algorithm [7] to determine the initial positions of twelve reference vectors within the training data set of each class. In equation (6), $\xi(t)$ is set to $(t/T + \xi_0)$ rather than t as recommended by Sato [96], where t denotes the number of epochs in which all training samples are involved, T denotes the number of training samples, and ξ_0 is set to 0.05. The reason for that is already high ($> 90\%$) classification rate on the training set after the initialization of GLVQ, and thus too small a value of ξ_t will result in a large penalty, which causes reference vectors to be adjusted dramatically. The principle used here is consistent with that of setting initial temperature in the deterministic annealing [23]. The learning step size decreases linearly with the number of steps t, i.e, $\alpha(t) = \alpha_0 \times (1.0 - t/t_{\max})$ with $\alpha_0 = 0.01$, where $t_{\max}$ is equal to the number of epochs times the number of training samples. The number of epochs is set to 200. The ensemble layer of local learning machines consists of multi-layered perceptrons (MLP) with single hidden layers of 40 units each. The networks employ sigmoid activation functions $1.0/(1.0 + \exp(-a))$. All MLP's are trained with the gradient method using momentum factor. The momentum factor and initial learning step size are set to 0.9 and 0.25, respectively. The number of component expert nets (L) is set to 15.

3.4.1 Handwritten digit databases

The first two experiments were performed on two well-known MNIST and CENPARMI databases of handwritten digits. MNIST database consists of 60,000 training samples and 10,000 test samples. All digits have been size-normalized and centered in

a 28×28 box. CENPARMI database consists of 6000 samples. We use 4000 samples for training and 2000 samples for testing.

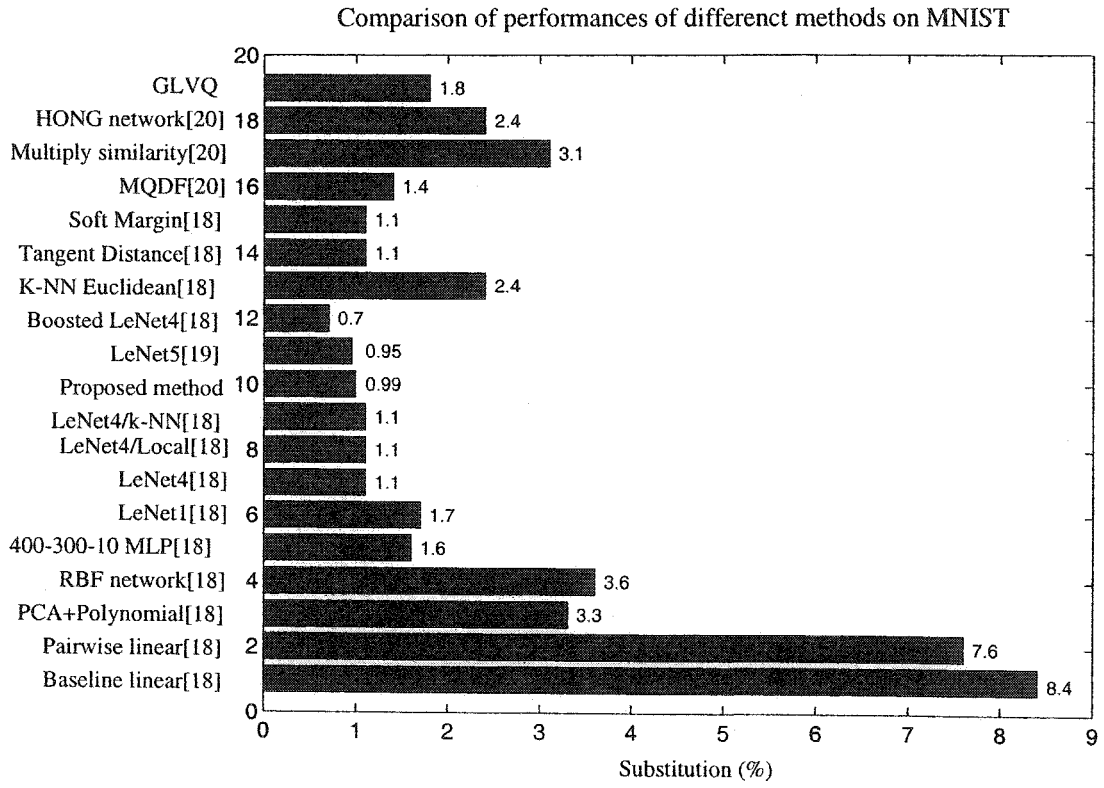

Figure 4: Error rates of different methods on the test set of MNIST database. Each bar represents a classifier.

Figure 4 compares the performances of different algorithms tested on the MNIST database. The proposed method is used on the original MNIST database and achieves a good performance with 0.99% substitution error, which is comparable to 0.95% error rate of LeNet5 [78][79], one of the best classifiers on the market. Boosted LeNet4 [78] is the state-of-art result classifier, but it was trained on a perturbed MNIST database, where the training set was augmented with artificially altered versions of the original training samples. Further, the method proposed by us clearly outperforms the best discriminant method called the Modified Quadratic Discriminant Functions (MQDF) [27]. It can be deduced from the Table 3 that the proposed method scales well to a small database of handwritten digits.

Table 3: Comparisons of different methods on CENPARMI database(%)

Methods	Recognized	Substituted	Rejected
Suen et al. [110]	93.05	0.00	6.95
Cho [17]	96.05	3.95	0.00
S.W LEE [81]	97.80	2.20	0.00
400-20-10 MLP [28]	96.20	3.80	0.00
400-20-10 MLP+AdaBoost [28]	97.20	2.80	0.00
MQDF [27]	98.00	2.00	0.00
GLVQ	96.30	3.70	0.00
proposed method	98.10	1.90	0.00

3.4.2 NIST lowercase database

NIST database of handwritten lowercase characters consists of 26,000 training samples and 12,000 test samples. In the training set, each category contains 1000 samples. The database contains some uppercase and noisy garbage patterns that do not belong to any of the 26 categories. About 6% of these patterns are highly confusing patterns such as "q" and "g", "i" and "l",which can barely be identified by humans (refer to Fig. 5 for typical examples). We thus cleaned the database and discarded test samples of three categories including "q","i" and "g". Consequently, we obtained a training set with 23,937 samples and a test set with 10,688 samples.

Automatic recognition of handwritten lowercase characters without context information is a challenging task. In the past ten years, a great progress has been made in the handwritten character recognition field, especially in the on-line and off-line digit recognition. A quick scan of the table of contents of *IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE Transactions on Neural Networks, IEEE Transactions on Systems, Man, and Cybernetics, Pattern Recognition, International Journal of Pattern Recognition and Artificial Intelligence, Pattern Recognition Letters, The International Workshop on Frontiers in Handwriting Recognition and The International Conference on Document Analysis and Recognition* since 1990's reveals that little work has been done on the recognition of handwritten lowercase characters. There is no benchmark to compare different algorithms on the same database.

Srihari [109] extracted some structural features such as 4-directional concaves,

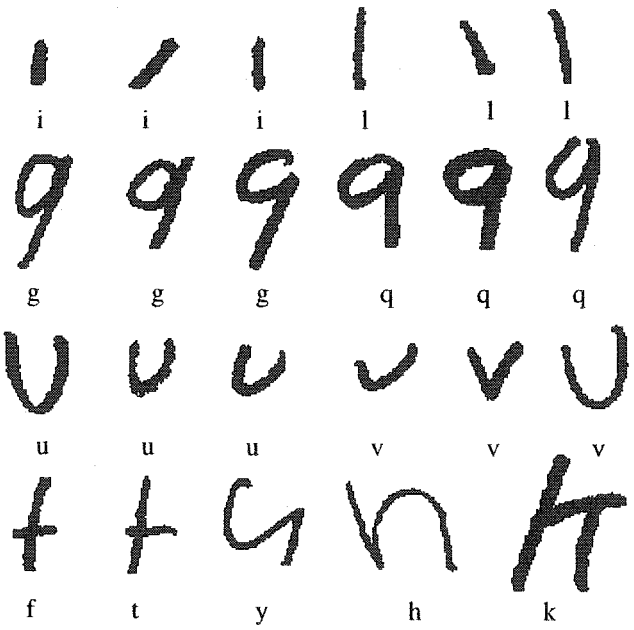

Figure 5: Confusing patterns in NIST lowercase database.

strokes (horizontal,vertical and diagonal), end-points, cross-points using morphological operators and three moment features and implemented a neural network classifier trained on NIST lowercase training subset with 10,134 samples using the above features. The recognition rate on the modified NIST lowercase test set with 877 samples was 85%. Toshihiro [84] extracted and combined three different stroke/background and contour-direction features. The proposed classifier is a three-layer MLP network trained on a NIST training subset with 10,968 samples. The recognition rate for lowercase characters on the modified NIST test subset with 8,284 samples was 89.64%. Obviously, the above researchers discarded some samples of the original test set. In summary, the above experimental results indicate that techniques of recognizing handwritten characters are far away from maturity. They differ from techniques applied in handwritten digit recognition in that there exists a greater overlap among class patterns. Similar patterns are distributed as clusters. This also motivates the usage of local ensembles to capture the discrimitive information of "boundary" patterns.

In the experiment, we pick up confusing patterns from the categories "g" and "q" and put them into a new category. Twenty-two classes are assigned to eight prototype vectors and five classes with a small quantity of data to four prototype vectors. The

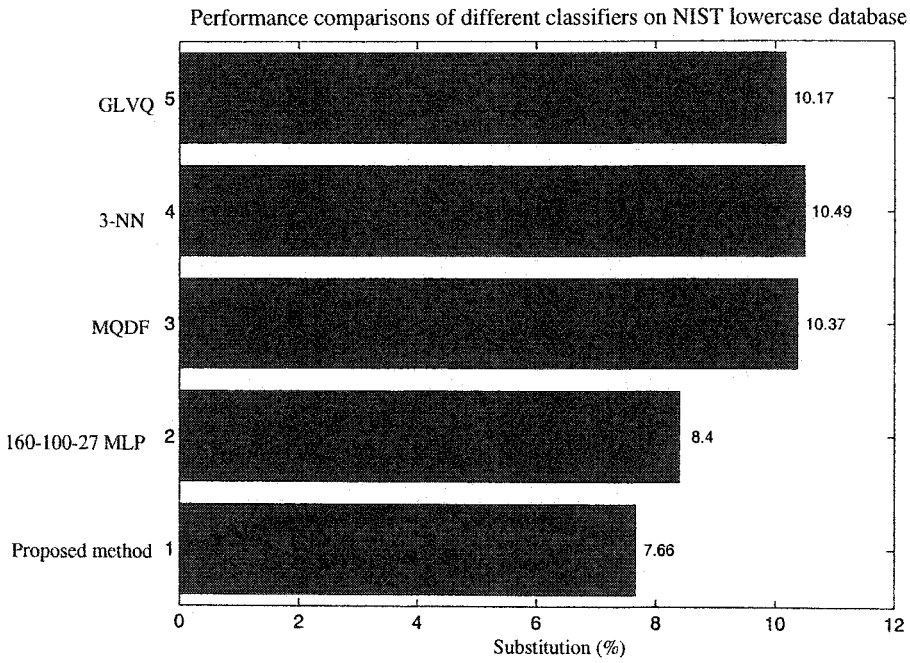

Figure 6: Error rates of different methods on the test set of NIST lowercase database, each bar represents a classifier.

Table 4: Cumulative recognition rate of the proposed method (%)

Candidate	top rank	2 ranks	3 ranks	4 ranks	5 ranks
recognition rate	92.34%	96.9%	98.09%	98.46%	98.85%

MLP in the ensemble layer contains 40 hidden units. The experimental results are illustrated in Fig. 6.

In practical handwriting recognition that integrates segmentation and classification or makes use of postprocessing, the classifier does not necessarily output a unique class. The cumulative accuracy of top ranks is also of practical importance. Table 4 shows the cumulative recognition rate of the proposed method.

3.4.3 CENPARMI lowercase database

Due to some problems with the NIST lowercase database, we collected samples from university students and built a lowercase database. All samples are clean and preserve

a complete stroke structure. Patterns within the same category have a large variety of shapes. The lowercase samples are stored in binary format with a scanning resolution of 300 DPI. The database contains samples from 195 writers. We divide the samples into training set and test set according to writer identities. The samples of randomly selected 120 writers are used as training set and the rest as test set. As a result, the training set consists of 14,810 samples and test set contains 8,580 samples. Fig. 7 shows some samples in the database.

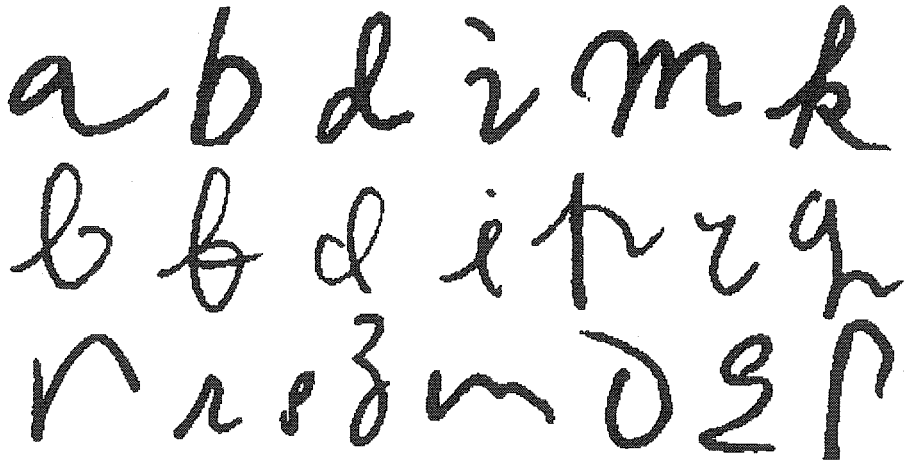

Figure 7: Samples in CENPARMI lowercase database.

In this experiment, we do not only evaluate the performance of the proposed method but also investigate several factors that have an effect on the overall performance. First, we outline the designed parameter setting. The number of reference vectors of each class is set to 8 and MLP in the ensemble layer has 40 hidden units. Other parameters are the same as those in the first experiment. In order to better evaluate the performance, several other classifiers including boosted MLP are used for a comparison with the proposed method. The results of the experiments are depicted in Fig. 8.

It can be observed that the proposed method outperforms the boosted MLP. AdaBoost is not as powerful as one might expect. It almost does not boost the MLP classifier although training error is reduced to zero by combining 15-component MLPs.

Second, we investigate the effect of the number of prototype vectors on the GLVQ

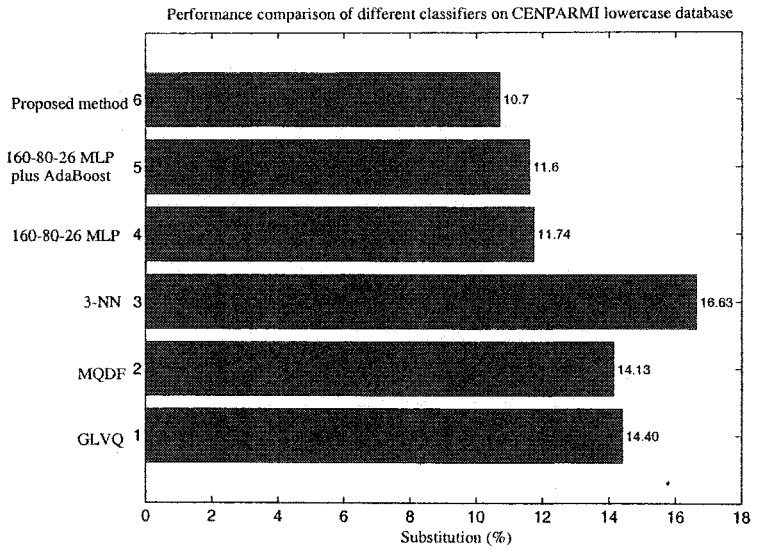

Figure 8: Error rates of different methods on the test set of CENPARMI lowercase database, each bar represents a classifier.

performance. Too many prototype vectors will result in overfitting the data while too few cannot capture the distribution of the samples within each class. Fig. 9 shows the relationship between GLVQ performance and the number of prototype vectors within each class.

Finally, we verify the claim that minimizing the MSE error does not directly result in a reduction in the error rate. Here the mean squared error is defined as

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \min_{\mathbf{m}_{kr} \in \Lambda} \| \mathbf{x}_i - \mathbf{m}_{kr} \|^2 \ . \tag{11}$$

We plot function curves of MSE, the empirical loss (see eq. (7)) and training error rate versus the number of iterations in Fig. 10.

In Fig. 10 MSE is monotonically increasing and the empirical loss and training error rate are monotonically decreasing. Thus minimization of the empirical loss and that of the training error rate are consistent. However, minimization of MSE is not necessarily linked to a reduction of the training error rate. This also indicates that most clustering algorithms such as SOM that minimize the mean square error are not suitable for classification [71].

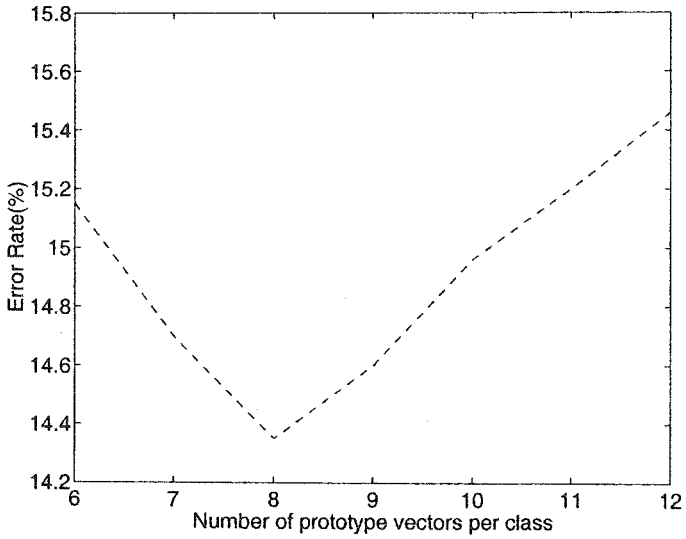

Figure 9: Relationship between GLVQ performance and the number of reference vectors per class.

3.5 Conclusions

In this work, we proposed a general local learning framework to alleviate the complexity problem of classifier design. Our method is based on a general "divide and conquer" principle and ensemble. According to this principle, a complex real-world classification problem can be divided into many sub-problems that can be easily solved. Ensemble method is used to reduce the variance and to improve the generalization ability of the neural network.

We also designed an effective method to construct a good ensemble on the varied training subset. Ensembles trained on subsets can effectively capture the information of "boundary patterns" that play an important role in classification. Our method was extensively tested on several public handwritten character databases, including databases of handwritten digits and lowercase characters. It consistently achieved a high performance.

The proposed method can be easily scaled to a small training set while still preserving a good performance. But it is especially suitable for a large-scale real-world classification tasks such as Chinese and Korean character recognition. The results are very encouraging and strongly suggest to apply the proposed method to data mining

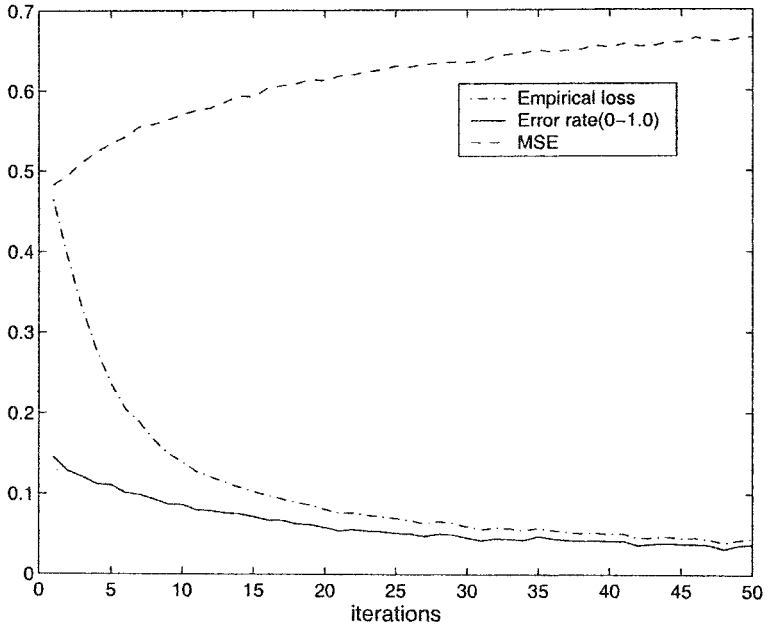

Figure 10: Function curves of MSE, the empirical loss and training error rate versus the number of iterations.

of real-world data.

Chapter 4

Support Vector Machines

This chapter gives a brief introduction to support vector machines and provides readers with a basic background for understanding the fast SVM training and testing algorithms described in Chapters 5 and 7. We try to keep the presentation in a self-contained way to ensure that these materials are easily understood and suitable for the interested readers who may not work directly in the machine learning domain and like to apply SVM to solve problems in their own domains. The maximal margin classifier on the data which are linearly separable is first presented to introduce some important concepts for SVM such as margin and dual representation. From this simple model the readers can gain some geometric intuitions on how SVM works. Then soft margin classifier is introduced to handle non-separable cases in the original space. The limited power of linear SVM in the original space motivates the introduction of nonlinear SVM which applies a kernel to implicitly map the data into a high dimensional feature space so that the separable capability is enhanced while keeping the computational efficiency. After that, the generalization theory is used to explain why SVM usually exhibits a good generalization performance. Finally, a detailed description of Sequential Minimal Optimization (SMO)[91][63] is given since it is used in a fast algorithm presented in Chapter 5. More information about SVM and kernel methods can be found in books [121][21][104]. Readers who have a good background in SVM can skip this chapter and go to the next one.

4.1 Maximal margin classifier

A hyperplane in $\mathbb{R}^d$ can be written as

$$\{\mathbf{x} \in \mathbb{R}^d | < \mathbf{w}, \mathbf{x} > +b = 0\}. \tag{12}$$

where $\mathbf{x} \in \mathbb{R}^d$ and $< . >$ denotes inner product. The $\mathbf{w}$ is a vector orthogonal to the hyperplane. The hyperplane splits the input space $\mathbb{R}^d$ into two half spaces which correspond to the inputs of two classes. Fig. 11 illustrates a hyperplane for separating data set for two classes.

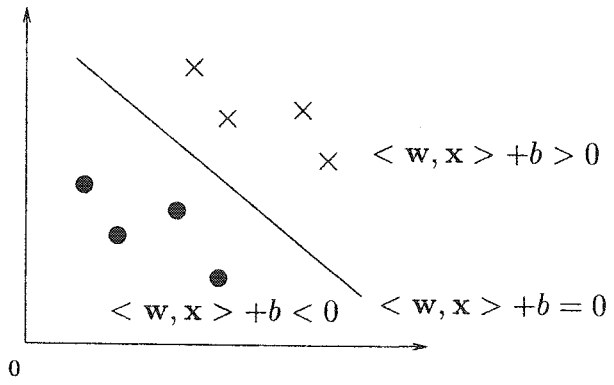

Figure 11: A hyperplane for separating data for two classes.

Then we give the definition of linear separable data set.

Definition 4.1. (Linearly separable data) Given that training samples $\{(\mathbf{x}_i, y_i)\} \subseteq \mathbf{X} \times \mathbf{Y}$, where $\mathbf{X} \subseteq \mathbb{R}^d$, $\mathbf{Y} = \{-1, 1\}$ and $i = 1, \ldots, l$. The data is linearly separable if there exists a hyperplane such that $y_i(< \mathbf{w}, \mathbf{x} > +b) > 0$.

The linearly separable hyperplane equation $< \mathbf{w}, \mathbf{x} > +b = 0$ is represented by $\mathbf{w}$ and b. By multiplying both $\mathbf{w}$ and b by the same non-zero factor, we can obtain the same hyperplane equation. In order to remove the scaling freedom, a unique scaling factor will be determined. First, we find two hyperplanes which are parallel: one H_+ is the closest to positive samples; the other H_- to negative samples. Let subset Ω_+ which corresponds to positive inputs; Ω_- which corresponds to negative inputs. Then we can obtain

$$-b_+ = \min_{\mathbf{x} \in \Omega_+} < \mathbf{w}, \mathbf{x} >, \tag{13}$$

$$-b_- = \max_{\mathbf{x}\in\Omega_-} < \mathbf{w}, \mathbf{x} >, \tag{14}$$

Then $H_+ : < \mathbf{w}, \mathbf{x} > +b_+ = 0$ and $H_- : < \mathbf{w}, \mathbf{x} > +b_- = 0$. Let $b = (b_+ + b_-)/2$ and $\delta = (b_- - b_+)/2$. Since two-class data are linearly separable, δ is non-zero. Therefore, H_+ and H_- can be rewritten as

$$H_+ : < \frac{\mathbf{w}}{\delta}, \mathbf{x} > +\frac{b}{\delta} = 1, \tag{15}$$

$$H_- : < \frac{\mathbf{w}}{\delta}, \mathbf{x} > +\frac{b}{\delta} = -1. \tag{16}$$

Then the scaled separating hyperplane becomes $< \frac{\mathbf{w}}{\delta}, \mathbf{x} > +\frac{b}{\delta} = 0$. Therefore, we can define a canonical form of a hyperplane which removes the scaling freedom.

Definition 4.2. (Canonical hyperplane [104]) The pair $(\mathbf{w}, b)$ is called a canonical form of the hyperplane with respect to $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l$, if it is scaled such that

$$\min_{i=1,\ldots,l} | < \mathbf{w}, \mathbf{x}_i > +b| = 1, \tag{17}$$

which indicates that the points closest to hyperplane have a distance of $1/\|\mathbf{w}\|$.

In Fig. 12, the margin, defined as the distance from the closest point to the hyperplane $< \mathbf{w}, \mathbf{x} > +b = 0$, is $1/\|\mathbf{w}\|$. In order to enable the separating hyperplane generalize well, the maximal margin should be sought in terms of the geometric intuition now. The reason why the hyperplane with a maximal margin performs well will be explained using the generalization theory in a later section.

Now we need to find the optimal hyperplane with the maximal margin. The problem can be formulated as a linearly constrained quadratic programming problem as follows:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to } y_i(< \mathbf{w}, \mathbf{x}_i > +b) \geq 1, \quad i = 1, \ldots, l. \tag{18}$$

This is a convex quadratic programming problem since the objective function is convex and these points which satisfy the linear constraints define a convex set. By introducing positive Lagrange multipliers α_i, $i = 1, \ldots, l$, one for each inequality constraints, we define the Lagrangian function

$$L_P(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i(y_i(< \mathbf{w}, \mathbf{x}_i > +b) - 1). \tag{19}$$

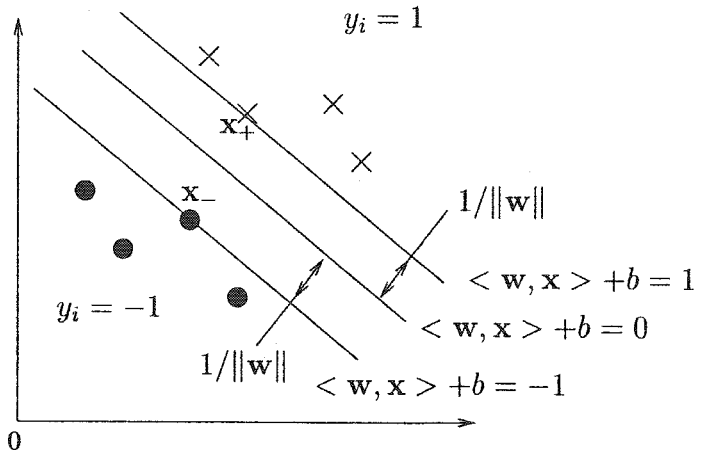

Figure 12: Scale $\mathbf{w}$ and b such that the points $\mathbf{x}_+$ and $\mathbf{x}_-$ closest to the hyperplane satisfy $< \mathbf{w}, \mathbf{x}_+ > +b = 1$ and $< \mathbf{w}, \mathbf{x}_- > +b = -1$, respectively.

We can solve the equivalent Wolf dual problem [44]: maximize L_P subject to the constraints that the gradients of L_P with respect to $\mathbf{w}$ and b vanish, and to the constraints that $\alpha_i \geq 0$.

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0, \tag{20}$$

$$\frac{\partial L_P}{\partial b} = 0, \tag{21}$$

$$\alpha \geq 0, \tag{22}$$

From equations (20) and (21), we have

$$\mathbf{w} = \sum_i^l y_i \alpha_i \mathbf{x}_i, \tag{23}$$

$$\sum_i^l \alpha_i y_i = 0. \tag{24}$$

Substitute equality constraints in Eqs. (20) and (21) into Eq. (19) to give the **dual** formulation together with the constraints of α

$$\begin{array}{ll} \text{maximize} & L_D(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j < \mathbf{x}_i, \mathbf{x}_j > \\ \text{subject to} & \alpha \geq 0 \\ & \sum_i \alpha_i y_i = 0. \end{array} \tag{25}$$

After we solve α in the dual problem, the decision function can be written as

$$\begin{aligned} f(\mathbf{x}) &= \text{sgn}(< \mathbf{w}, \mathbf{x} > +b) \\ &= \text{sgn}(\sum_{i}^{l} \alpha_i y_i < \mathbf{x}_i, \mathbf{x} > +b). \end{aligned} \tag{26}$$

where

$$\text{sgn}(u) = \begin{cases} 1 & \text{if } u > 0 \\ -1 & \text{otherwise} \end{cases} \tag{27}$$

It can be observed in the dual problem (25) and the decision function (26) that training vectors $\mathbf{x}_i$ only occur in the form of dot product.

4.2 Soft margin classifier

When data is not perfectly separable due to noises and outliers, we introduce slack variables to allow the margin inequality constraints [19] in the primal problem (19) to be violated.

$$\begin{aligned} \text{subject to} \quad & y_i(< \mathbf{w}, \mathbf{x}_i > +b) \geq 1 - \xi_i, \ i = 1, \ldots, l, \\ & \xi_i \geq 0, \ i = 1, \ldots, l. \end{aligned}$$

When an error occurs, ξ_i is greater than 1. So $\sum_i \xi_i$ can be regarded as the upper bound of training errors. It is expected to maximize the margin and minimize the training errors. The primal problem (19) can be re-defined as

$$\begin{aligned} &\text{minimize}_{\mathbf{w},\mathbf{b},\xi} \quad \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ &\text{subject to} \quad y_i(< \mathbf{w}, \mathbf{x}_i > +b) \geq 1 - \xi_i, \ i = 1, \ldots, l, \\ &\qquad\qquad \xi_i \geq 0, \ i = 1, \ldots, l. \end{aligned} \tag{28}$$

This is still a convex quadratic programming problem and the positive parameter C is chosen by the user. A larger C assigns a higher penalty to training errors. The corresponding Lagrangian of (28) is

$$\begin{aligned} L_P(\mathbf{w}, b, \xi, \alpha, \beta) &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i}^{l} \xi_i \\ &\quad - \sum_{i=1}^{l} \alpha_i [y_i(< \mathbf{w}, \mathbf{x}_i > +b) - 1 + \xi_i] - \sum_{i=1}^{l} \beta_i \xi_i \end{aligned} \tag{29}$$

with $\alpha_i \geq 0$ and $\beta_i \geq 0$. The Karush-Kuhn-Tucker (KKT) optimality conditions [73] are given by

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l} y_i \alpha_i \mathbf{x}_i = 0, \tag{30}$$

$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \beta_i = 0, \quad \forall i \tag{31}$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{l} y_i \alpha_i = 0, \tag{32}$$

$$y_i(< \mathbf{x}_i, \mathbf{w} > + b) - 1 + \xi_i \geq 0, \quad \forall i \tag{33}$$

$$\alpha_i[y_i(< \mathbf{w}, \mathbf{x}_i > + b) + \xi_i - 1] = 0, \quad \forall i \tag{34}$$

$$\beta_i \xi_i = 0, \quad \forall i \tag{35}$$

$$\alpha_i \geq 0, \quad \forall i \tag{36}$$

$$\beta_i \geq 0, \quad \forall i \tag{37}$$

$$\xi_i \geq 0. \quad \forall i \tag{38}$$

where Eqs. (34) and (35) are called KKT "complementary" conditions. Substitute Eqs. (30),(31) and (32) into Eq. (29) and obtain dual objective function:

$$L_D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j < \mathbf{x}_i, \mathbf{x}_j > . \tag{39}$$

which is the same as that in the maximal margin case. The difference is that from the constraint (47) we obtain $\alpha_i \leq C$ since $\beta_i \geq 0$. Therefore, the dual formulation in soft margin case is given by

$$\begin{array}{ll} \text{maximize}_\alpha & L_D(\alpha) = \sum_{i=1}^{l} - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j < \mathbf{x}_i, \mathbf{x}_j >, \\ \text{subject to} & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, l, \\ & \sum_i^l y_i \alpha_i = 0. \end{array} \tag{40}$$

4.3 Nonlinear support vector machine

The decision function Eq. (26) is a linear function of the data. Its power is limited and needs to be generalized to the nonlinear case. It can be observed that the data in the training problem Eq. (40) and decision function Eq. (26) is in the form of a dot product. A nonlinear function Φ [5] is introduced to map the data to a high

dimensional inner product space $\mathcal{H}$ [1] by

$$\Phi : \mathbb{R}^d \to \mathcal{H}. \tag{41}$$

The mapping Φ is implemented by a kernel function K that satisfies Mercer's conditions [85] such that $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. The kernel trick is that we never need to explicitly represent the nonlinear mapping Φ and just replace $< \mathbf{x}_i, \mathbf{x}_j >$ by $K(\mathbf{x}_i, \mathbf{x}_j)$ in the training algorithm.

4.4 The role of margin in SVM

In the design of SVM training algorithm, we expect to find a hyperplane with a large margin to separate the data. Intuitively the hyperplane with a large margin has a good generalization performance. It is necessary to know why the margin plays a crucial role in SVM from a technical viewpoint. Let us start to explain it by means of Vapnik's statistical learning theory [121].

Let data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l) \in \mathbf{X} \times \mathbf{Y}$, be generated i.i.d. (independently drawn and identically distributed) from cumulative probability distribution $P(x, y)$, where $\mathbf{X} \subseteq \mathbb{R}^d$ and $\mathbf{Y} = \{1, -1\}$. Learning is to find one from a set of functions $f(\mathbf{x}, \alpha)$[2]: $\mathbf{X} \mapsto \{1, -1\}$ such that the expected misclassification error on the test set, also drawn from $P(\mathbf{x}, y)$,

$$R(\alpha) = \int \frac{1}{2} |f(\mathbf{x}, \alpha) - y| dP(\mathbf{x}, y). \tag{42}$$

is minimal. Eq. (42) is called the expected risk (or actual risk). But since $P(\mathbf{x}, y)$ is usually unknown, we use the errors over the training samples to replace it, defined by

$$R_{\text{emp}}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |f(\mathbf{x}_i, \alpha) - y_i|. \tag{43}$$

The R_{emp} is called "empirical risk". The empirical risk can be connected with the expected risk by a probability bound [120]. That is, for any $f(\mathbf{x}, \alpha)$ and $l > h$, with a probability of at least $1 - \eta$,

$$R(\alpha) \leq R_{\text{emp}}(\alpha) + \sqrt{\frac{h(\log \frac{2l}{h} + 1) - \log(\eta/4)}{l}}. \tag{44}$$

[1] The completion of this space can be constructed so that we can obtain Hilbert space.

[2] α are adjustable parameters

holds, where h is a non-negative integer called the Vapnik Chervonenkis (VC) dimension[3], and is a measure of the capacity of the function class $f(\mathbf{x}, \alpha)$. The second term in Eq. (44) is called "confidence (capacity) term", which is an increasing function of h for the fixed η. Although the above bound can be very loose, it provides us a guideline for designing the learning algorithm: in order to reduce the expected risk, we can minimize the right side by taking into account the training errors and capacity of learning function classes together. Fig. (13) illustrates the meaning of the above bound.

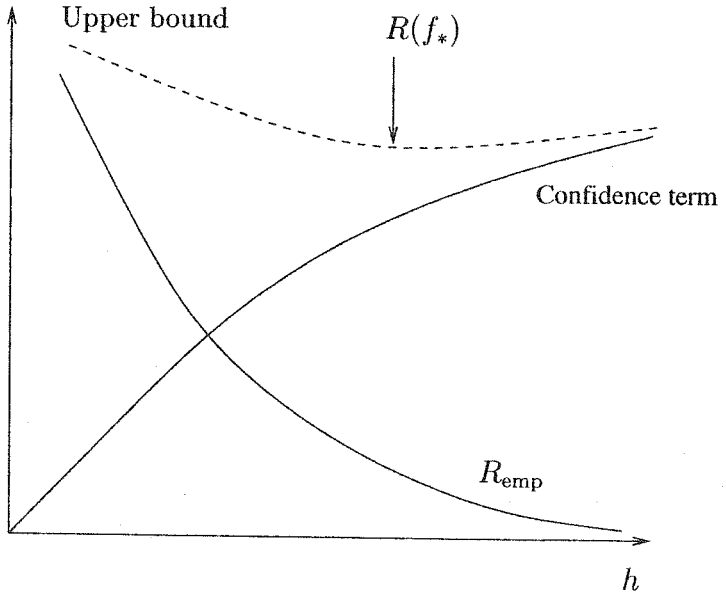

Figure 13: The picture of VC bound. For a family of functions $f(\mathbf{x}, \alpha)$, we choose f_* that gives the lowest upper bound in the expected risk.

Now we need to find a good function class with a small capacity such that a function from them, chosen by the training set, can lead to a small actual risk. It is known that separating hyperplanes in $\mathbb{R}^d$ has a VC dimension of $d + 1$ [14]. So separating hyperplanes in a high dimensional feature space will produce an extremely high VC dimension, and may not generalize well (or obtain a lower actual risk). For example, in the nonlinear SVM, the dimension of feature space $\mathcal{H}$ is infinity when the radial basis function kernel is used [14]. The VC dimension of separating hyperplanes is infinity.

[3]The simple definition for VC dimension is referred to Burges's tutorial[14].

How does SVM solve this problem since SVM finds a hyperplane in the feature space of possibly infinite dimension? The key point is that SVMs correspond to large margin hyperplanes, which can still have a small VC dimension in terms of the following theorem:

Theorem 4.4.1. (Vapnik[121]) Consider hyperplanes $< \mathbf{w}, \mathbf{x} > +b = 0$ where $\mathbf{w}$ is normalized such that they are in canonical form, w.r.t. a set of points $\mathbf{X}^* = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$, i.e., $\min_{i=1,\ldots,l} | < \mathbf{w}, \mathbf{x}_i > +b| = 1$. The set of decision functions $f_{\mathbf{w},b}(\mathbf{x}) = \mathrm{sgn}(< \mathbf{w}, \mathbf{x} > +b)$ defined on $\mathbf{X}^*$ and satisfying the constraint $\|\mathbf{w}\| \leq \Lambda$ has a VC dimension satisfying

$$h \leq R^2 \Lambda^2. \tag{45}$$

where R is the radius of the smallest sphere around the origin containing $\mathbf{X}^*$.

From the above theorem, we can make the following remarks:

- The VC dimension of separating hyperplanes can be small and independent of the dimension of the feature space by bounding the length of weight vector $\|\mathbf{w}\|$. Here the bound is chosen by a priori information. In the SVM training, $\|\mathbf{w}\|$ is minimized.

- The above theorem can be used to select the parameters of an SVM kernel. For example, when training errors are very small, we minimize $R^2\|\mathbf{w}\|^2$, an upper bound of VC dimension, and expect to obtain a lower testing error.

More technical results are referred to [21], where the margin directly appears on the error bound.

4.5 Sequential minimal optimization

There are many methods of solving SVM optimizations. Sequential Minimal Optimization (SMO), introduced by Platt [91] and improved by Keerthi et al. [63], is one of the most easily implementable algorithms and derived by optimizing the subsets of two points at each iteration. The power of SMO exists in fact that no extra optimization package is required and an analytical solution for two-point optimization problem can be explicitly provided. In the next chapter, since we will present a fast

SVM training algorithm which will use SMO, it is necessary to give a sketch of SMO here.

In this section, we will consider the general case in SVM optimization, where $\mathbf{x}$ is replaced by $\Phi(\mathbf{x})$, $< \mathbf{x}_i, \mathbf{x}_j >$ by $K(\mathbf{x}_i, \mathbf{x}_j)$ in the soft margin classifier. In order to understand SMO algorithm, we need to answer the following three questions:

- What are the optimal conditions for stopping the algorithm?
- How are two parameters chosen to be optimized such that a significant progress on the objective function can be made?
- How are the two chosen parameters jointly optimized given that all the others are fixed?

Since SVM optimization is a convex quadratic programming problem, KKT conditions are necessary and sufficient for $\mathbf{w}$,b,α to be a solution [44]. In the general case, KKT conditions for the primal problem are given by

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l} y_i \alpha_i \Phi(\mathbf{x}_i) = 0, \tag{46}$$

$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \beta_i = 0, \quad \forall i \tag{47}$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{l} y_i \alpha_i = 0, \tag{48}$$

$$y_i(< \mathbf{w}, \Phi(\mathbf{x}_i) > + b) - 1 + \xi_i \geq 0, \quad \forall i \tag{49}$$

$$\alpha_i[y_i(< \mathbf{w}, \Phi(\mathbf{x}_i) > + b) + \xi_i - 1] = 0, \quad \forall i \tag{50}$$

$$\beta_i \xi_i = 0, \quad \forall i \tag{51}$$

$$\alpha_i \geq 0, \quad \forall i \tag{52}$$

$$\beta_i \geq 0, \quad \forall i \tag{53}$$

$$\xi_i \geq 0. \quad \forall i \tag{54}$$

The above conditions can be simplified by taking into account three cases of α_i:

1. Case 1: $\alpha_i = 0$. From Eq. (47), $\beta_i = C - \alpha_i = C$. Then we can get $\xi_i = 0$ from Eq. (51). The following inequality holds from Eq. (49)

$$y_i(< \mathbf{w}, \Phi(\mathbf{x}_i) > + b) - 1 \geq 0. \tag{55}$$

2. Case 2: $0 < \alpha_i < C$. From Eq. (47), $\beta_i = C - \alpha_i > 0$. Then $\xi_i = 0$ according to Eq. (51). So we have from Eq. (50)

$$y_i(< \mathbf{w}, \Phi(\mathbf{x}_i) > +b) - 1 = 0. \tag{56}$$

3. Case 3: $\alpha_i = C$. From Eq. (47), $\beta_i = 0$. Then we have from Eq. (50) since $\xi_i \geq 0$

$$y_i(< \mathbf{w}, \Phi(\mathbf{x}_i) > +b) - 1 \leq 0. \tag{57}$$

Then we simplify KKT conditions without checking the threshold b due to Keerthi et al. [63]. Let $F_i =< \mathbf{w}, \Phi(\mathbf{x}_i) > -y_i = \sum_{j=1}^{l} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) - y_i$. Since

$$\begin{aligned} y_i(< \mathbf{w}, \Phi(\mathbf{x}_i) > +b) - 1 &= y_i(< \mathbf{w}, \Phi(\mathbf{x}_i) > +b) - {y_i}^2 \\ &= y_i(< \mathbf{w}, \Phi(\mathbf{x}_i) > -y_i + b) \\ &= y_i(F_i + b) \end{aligned}$$

Eqs. (55), (56) and (57) can be rewritten as

$$y_i(F_i + b) \geq 0 \text{ if } \alpha_i = 0 \tag{58}$$

$$y_i(F_i + b) = 0 \text{ if } 0 < \alpha_i < C \tag{59}$$

$$y_i(F_i + b) \leq 0 \text{ if } \alpha_i = C \tag{60}$$

The above KKT conditions can continue to be simplified when the following indexed sets at a given α are defined:

$$\begin{aligned} I_0 &= \{i : 0 < \alpha_i < C\}, \\ I_1 &= \{i : y_i = 1,\ \alpha_i = 0\}, \\ I_2 &= \{i : y_i = -1,\ \alpha_i = C\}, \\ I_3 &= \{i : y_i = 1,\ \alpha_i = C\}, \\ I_4 &= \{i : y_i = -1,\ \alpha_i = 0\}. \end{aligned}$$

The conditions in Eqs. (58) through (60) can be rewritten as

$$-b \leq F_i \ \ \forall i \in I_0 \cup I_1 \cup I_2; \ \ -b \geq F_i \ \ \forall i \in I_0 \cup I_3 \cup I_4. \tag{61}$$

The optimal conditions hold if and only if there exists $b' = -b$ satisfying Eq. (61). Let us define $b_{\text{up}} = \min\{F_i : i \in I_0 \cup I_1 \cup I_2\}$ and $b_{\text{low}} = \max\{F_i : i \in I_0 \cup I_3 \cup I_4\}$. Then optimal conditions hold at some α iff

$$F_{\text{i\_low}} = b_{\text{low}} \leq F_{\text{i\_up}} = b_{\text{up}}. \tag{62}$$

In numerical solution it is not easy to achieve the exact optimal conditions. We approximate it by

$$F_{\text{i\_low}} = b_{\text{low}} \leq F_{\text{i\_up}} = b_{\text{up}} + 2\tau \tag{63}$$

Eq. (63) gives the approximated KKT conditions and provides an answer to the first question mentioned above.

With respect to question 2 we need to find a pair (i_1, i_2) which violates KKT conditions. We use either of the following conditions to check whether a particular i violates KKT optimality.

$$i \in I_1 \cup I_2 \quad \text{and} \quad F_i \leq F_{\text{i\_low}} - 2\tau \tag{64}$$

$$i \in I_3 \cup I_4 \quad \text{and} \quad F_i \geq F_{\text{i\_up}} + 2\tau \tag{65}$$

$$b_{\text{up}} \quad \leq \quad b_{\text{low}} - 2\tau \tag{66}$$

If Eq. (64) holds, the violating pair is $(i, \text{i\_low})$. Similarly, the violating pairs in Eqs. (65) and (66) are $(i, \text{i\_up})$ and $(\text{i\_low}, \text{i\_up})$, respectively. More details on choosing the violating pairs are referred to Keerthi et al.'s paper [63].

After we have determined the violating pair, we need to find a method to update their corresponding α values. Without loss of generality, we assume that two parameters that are chosen are α_1 and α_2 given that other parameters are fixed. The linear constraint $\sum_{i=1}^{l} y_i \alpha_i = 0$ implies

$$y_1\alpha_1 + y_2\alpha_2 = y_1\alpha_1^{\text{old}} + y_2\alpha_2^{\text{old}}. \tag{67}$$

We first compute α_2^{new}. Then α_1^{new} can be obtained from Eq. (67)

$$\begin{aligned} \alpha_1^{\text{new}} &= y_1(y_1\alpha_1^{\text{old}} + y_2\alpha_2^{\text{old}}) - y_1y_2\alpha_2^{\text{new}} \\ &= \alpha_1^{\text{old}} + y_1y_2(\alpha_2^{\text{old}} - \alpha_2^{\text{new}}). \end{aligned} \tag{68}$$

Due to the box constraint $0 \leq \alpha_1, \alpha_2 \leq C$, we can compute a new box constraint for α_2 according to Eq. (67), given by

$$L \leq \alpha_2^{\text{new}} \leq H, \tag{69}$$

where if $y_1 \neq y_2$

$$\begin{aligned} L &= \max(0, \alpha_2^{\text{old}} - \alpha_1^{\text{old}}), \\ H &= \min(C, C + \alpha_2^{\text{old}} - \alpha_1^{\text{old}}). \end{aligned}$$

or if $y_1 = y_2$

$$\begin{aligned} L &= \max(0, \alpha_1^{\text{old}} + \alpha_2^{\text{old}} - C), \\ H &= \max(C, \alpha_1^{\text{old}} + \alpha_2^{\text{old}}). \end{aligned}$$

Before we start to compute an updating formula for α_2, the following notations [21] are defined

$$\begin{aligned} K_{ij} &= K(\mathbf{x}_i, \mathbf{x}_j) \\ v_i &= \sum_{j=3}^{l} y_j \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) = F_i^{\text{old}} + y_i - \sum_{j=1}^{2} y_1 \alpha_j^{\text{old}} K_{ij}, \quad i = 1, 2, \\ s &= y_1 y_2 \\ \alpha_1 + s\alpha_2 &= \alpha_1^{\text{old}} + s\alpha_2^{\text{old}} = \gamma \end{aligned}$$

where γ is constant. The objective as a function of α_1 and α_2 in the dual formulation is given by

$$\begin{aligned} L_D(\alpha_1, \alpha_2) &= \alpha_1 + \alpha_2 - \frac{1}{2} K_{11} \alpha_1^2 - \frac{1}{2} K_{22} \alpha_2^2 \\ &\quad - y_1 y_2 K_{12} \alpha_1 \alpha_2 - y_1 \alpha_1 v_1 - y_2 \alpha_2 v_2 + \text{constant}, \end{aligned} \tag{70}$$

Substitute $\alpha_1 = \gamma - s\alpha_2$ into Eq. (70) and remove the variable α_1. The objective function can be simplified by

$$\begin{aligned} L_D(\alpha_2) &= \gamma - s\alpha_2 + \alpha_2 - \frac{1}{2} K_{11} (\gamma - s\alpha_2)^2 - \frac{1}{2} K_{22} \alpha_2^2 \\ &\quad - s K_{12} (\gamma - s\alpha_2) \alpha_2 - y_1 (\gamma - s\alpha_2) v_1 - y_2 \alpha_2 v_2 + \text{constant} \\ &= -\frac{1}{2} (K_{11} + K_{22} - 2K_{12}) \alpha_2^2 \\ &\quad + [y_2 (F_1^{\text{old}} - F_2^{\text{old}}) + \alpha_2^{\text{old}} (K_{11} + K_{22} - 2K_{12})] \alpha_2 + \text{constant}. \end{aligned}$$

Let η be $K_{11} + K_{22} - 2K_{12}$. Then $\eta = \|\Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2)\|^2 \geq 0$. If $\eta > 0$, the stationary point satisfies

$$\frac{\partial L_D}{\alpha_2} = -\eta \alpha_2 + y_2 (F_1^{\text{old}} - F_2^{\text{old}}) + \alpha_2^{\text{old}} \eta = 0 \tag{71}$$

We can get

$$\alpha_2^{\text{new,unclipped}} = \alpha_2^{\text{old}} + \frac{y_2 (F_1^{\text{old}} - F_2^{\text{old}})}{\eta} \tag{72}$$

Then we clip $\alpha_2^{\text{new,unclipped}}$ according to Eq. (69). If $\eta = 0$, the objective function is

$$L_D(\alpha_2) = y_2 (F_1^{\text{old}} - F_2^{\text{old}}) \alpha_2 \tag{73}$$

We evaluate $L_D(\alpha_2)$ at the two endpoints L and H and set α^{new} to be the one with the larger objective function value. Finally, we summarize the above results:

1. case 1: $\eta > 0$

$$\alpha_2^{\text{new,clipped}} = \begin{cases} L & \text{if } \alpha_2^{\text{new,unclipped}} < L, \\ \alpha_2^{\text{new,unclipped}} & \text{if } L \leq \alpha_2^{\text{new,unclipped}} \leq H, \\ H & \text{if } \alpha_2^{\text{new,unclipped}} > H. \end{cases} \tag{74}$$

2. case 2: $\eta = 0$

$$\alpha_2^{\text{new,clipped}} = \begin{cases} L & \text{if } y_2(F_1^{\text{old}} - F_2^{\text{old}}) < 0, \\ H & \text{otherwise.} \end{cases} \tag{75}$$

Chapter 5

Fast SVM Training Algorithm

5.1 Introduction

Support vector machine (SVM) [121][21][104] has emerged as the state-of-the-art classification technique and has achieved excellent generalization performance in a wide variety of applications, such as handwritten digit recognition [98][24][33], categorization of web pages [57] and face detection [87]. Some basic problems, such as multi-local minima and curse of dimensionality in the neural network literature [4], do not occur in SVM. In addition, the structure of support vector classifier is data-driven and automatically determined for a chosen kernel while the structure of neural network classifier is usually customized heuristically in order to achieve a good generalization performance. However, since the training complexity grows with the size of the data set, training support vector machines on a large data set is very slow and has become a bottle-neck of SVM's application. Therefore, it is important to develop fast algorithms for training SVM for large-scale classification problems.

Although many methods for solving the optimization problem of support vector machines are available [104], we list here only the prominent ones which can be used to train SVMs on a large data set, such as Chunking [121][87], Sequential Minimal Optimization (SMO)(see [91][63]) and SVM^{light} [58]. The chunking algorithm starts with an arbitrary subset (chunk of data, working set) which can fit in the memory, and solves the optimization problem on it by the general optimizer. Support vectors (SVs) remain in the chunk while other points are discarded and replaced by a new working set with large violations of KKT (Karush-Kuhn-Tucker) conditions [73]. The

rationale of this operation is that only support vectors contribute to the final form of a decision function. In addition, the chunking algorithm is based on the sparsity of SVM's solution. That is, support vectors actually take up a small fraction of the whole data set. But one of the problems associated with the chunking algorithm is that there may be many active candidate support vectors during the optimization process than the final ones so that their size can go beyond the chunking space. The method of selecting a new working set by evaluating KKT conditions without efficient kernel caching may lead to a high computational cost [87].

SMO, introduced by Platt [91] and improved by Keerthi et al. [63], further takes the decomposition idea to an extreme and optimizes the subsets of two points at each iteration. The power of SMO reveals itself in the fact that no extra optimization package is required and an analytical solution for a two-point optimization problem can be explicitly given. Several heuristics have been suggested to select the working set. Keerthi et al. [63] further enhanced the performance of SMO by pointing out the inefficiency of updating one-thresholded parameters in Platt's algorithm and replacing it with two-thresholded parameters. The important contribution of Keerthi et al's modification is that the pair of patterns chosen for optimization is theoretically determined by two-thresholded parameters and the optimization on this subset leads to a considerable progress in the objective function. In practice, when the size of a data set grows bigger, it is still a problem to determine the optimal pair at low cost.

SVM^{light} [58] is a general decomposition algorithm, where a good working set is selected by finding the steepest feasible direction of descent with q nonzero elements. The q variables that correspond to these elements compose the working set. When q is set to 2, Lin et al. [15] pointed out that the selected working set corresponds to the optimal pair in Keerthi el al.'s modification of SMO. SVM^{light} caches q rows of kernel matrix (row caching) to avoid kernel re-evaluations and LRU (Least Recently Used) is applied to update the rows in the cache[1]. When the size of the training set is very large, the number of cached rows becomes small due to the limited memory. As a result, the number of active variables is not large enough to achieve a fast optimization.

By reviewing the behaviors of the above algorithms, we have concluded that they are inefficient on large data sets due to three key factors. The computational cost

[1]Here cache means a part of memory, not hardware cache

of training SVM primarily depends on kernel evaluations. Efficient kernel caching can reduce or avoid kernel re-evaluations. The LRU caching policy may fail because elements of kernel matrix are usually accessed irregularly [43]. In addition, evaluating kernel elements on the fly is not efficient at all because data access is not temporally local. Massive evaluation of kernel elements via blocking algorithm will reduce hardware cache misses and speed up the computation. The second problem originates from frequent access of portions of non-contiguous memories, which potentially can lead to high cache misses. Training SVM on a large data set usually requires the access of a large size of memory. In a virtual memory system, accessing this memory irregularly will cause high Translation Look-aside Buffer (TLB) misses [90]. Consequently memory access will take more time. Finally, although the existing algorithms can be used to train SVMs on a large data set with multi-classes, the computational cost is high. For example, with respect to one-against-the-others training strategy [98] for multi-classes, the training cost for m classes is about m times as high as that for two classes.

The main contribution of this chapter is to present efficient solutions to the above problems. Two steps have been designed to train support vector machines. The first step is called parallel optimization, in which the kernel matrix of support vector machine is approximated by block diagonal matrices so that the original optimization problem can be decomposed into hundreds of sub-problems, which can be easily and efficiently solved. The great advantage of this step is to remove most non-support vectors quickly and collect training sets for the next step called sequential working set algorithm. Associated with these two steps, some effective strategies such as kernel caching and good selection of working set are integrated to speed up the training process. In addition, Block Linear Algebra Subprogram (BLAS) [38][125], which is optimized on Intel P4, can be used to efficiently calculate the kernel matrix. Experiments on the large MNIST handwritten digit database have shown that the proposed method has achieved one magnitude of order speed-up, compared with existing algorithms such as SVM^{light} and LIBSVM [15]. Moreover, the state-of-the-art generalization performance has been obtained on other well-known public and commercial character databases.

This chapter is organized as follows. Support vector machines are first introduced. Then in Section 5.3, a fast training algorithm for SVM is presented. A discussion of

efficient implementation strategies is given in Section 5.4. In Section 5.5, we analyze the space and runtime complexity of the proposed algorithm. Extensive experiments in Section 5.6 have been conducted to investigate the properties of the proposed algorithm on very large databases and its generalization performance on several public handwritten character databases. Finally, we summarize this chapter with some concluding remarks.

5.2 Support Vector Machine

Let $\{\mathbf{x}_i, y_i\}$, $i = 1, \ldots, l$, $y_i \in \{-1, 1\}$ and $\mathbf{x}_i \in \mathbb{R}^n$ be the training samples where $\mathbf{x}_i$ is the training vector and y_i is its corresponding desired output. Here we use bold font to denote a column vector. Boser et al. [5] showed that training support vector machine for a pattern recognition problem can be formalized as the following quadratic optimization problem:

$$\begin{array}{ll} \text{maximize:} & \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\alpha^T \mathbf{Q}\alpha \\ \text{subject to:} & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, l \\ & \sum_{i=1}^{l} y_i \alpha_i = 0 \end{array} \tag{76}$$

where α is a vector of length l and its component α_i corresponds to a training sample $\{\mathbf{x}_i, y_i\}$, $\mathbf{Q}$ is an $l \times l$ semi-definite kernel matrix and C is a parameter chosen by the user. A larger C assigns a higher penalty to the training errors. The training vector $\mathbf{x}_i$ whose corresponding α_i is non-zero is called support vector. Support vector machine maps training vector $\mathbf{x}_i$ into high dimensional feature space by the function $\Phi(\mathbf{x})$ such that $(\mathbf{Q})_{ij} = y_i y_j (\mathbf{K})_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$ and $k(\mathbf{x}_i, \mathbf{x}_j) = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$. When the above optimization problem is solved, we can obtain an optimal hyperplane in high dimensional feature space to separate the two-class samples. The decision function is given by

$$f(\mathbf{x}) = \text{sgn}(\sum_{i=1}^{l} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) - b), \tag{77}$$

where

$$\text{sgn}(u) = \begin{cases} 1 & \text{if } u > 0, \\ -1 & \text{otherwise.} \end{cases} \tag{78}$$

In the latter algorithm, we resort to a technique proposed by Keerthi et al. [63][64] to select two variables for optimization and determine the stopping conditions. Training patterns can be split into five sets first: $I_0 = \{i : 0 < \alpha_i < C\}$, $I_1 = \{i : y_i = 1,\ \alpha_i = 0\}$, $I_2 = \{i : y_i = -1,\ \alpha_i = C\}$, $I_3 = \{i : y_i = 1, \alpha_i = C\}$ and $I_4 = \{i : y_i = -1,\ \alpha_i = 0\}$. Then we define

$$b_{\text{up}} = F_{i\_\text{up}} = \min\{F_i : i \in I_0 \cup I_1 \cup I_2\}, \tag{79}$$

$$b_{\text{low}} = F_{i\_\text{low}} = \max\{F_i : i \in I_0 \cup I_3 \cup I_4\}. \tag{80}$$

where $F_i = \sum_{j=1}^{l} y_j \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) - y_i$. Keerthi et al. [63] showed that optimal conditions for a solution to (76) hold at some α if and only if $b_{\text{low}} \leq b_{\text{up}}$. Moreover, the worst violating pair of patterns $(i\_\text{up}, i\_\text{low})$ are chosen for optimization, which can lead to a large increase in the objective function [104].

5.3 A fast algorithm for training SVM

For the optimization described in (76), the key problem is that the dense kernel matrix Q can not be stored into the memory when the number of training samples l is very large. We can observe the fact that the optimal solution to (76) still holds if any non-support vector is removed. Moreover, numerous experiments [98][33] have shown that support vectors actually constitute only a small fraction of the training samples. If most non-support vectors can be removed quickly in the first step, SVM training can be accelerated dramatically. Divide-and-conquer[2] is a general principle for solving complex problems. That is, we can divide the original problem (76) into small sub-problems which can be solved easily. Since kernel matrix Q is symmetric and semi-positive definite, its block diagonal matrices are semi-positive definite, and can be written as

$$\mathbf{Q}_{\text{diag}} = \begin{bmatrix} \mathbf{Q}_{11} & & & \\ & \mathbf{Q}_{22} & & \\ & & \ddots & \\ & & & \mathbf{Q}_{kk} \end{bmatrix} \tag{81}$$

where $l_i \times l_i$ square matrices $\mathbf{Q}_{ii}$, $i = 1, \ldots, k$, $\sum_{i=1}^{k} l_i = l$, are called block diagonal. Then we replace the kernel matrix with $\mathbf{Q}_{ii}$ in (76) so that we obtain k optimization

[2] A good example in computer science is **quick-sort**.

sub-problems as follows:

$$\begin{aligned} \text{maximize:} \quad & \textstyle\sum_{j=1}^{l_i} \alpha_j^{(i)} - \frac{1}{2}\alpha^{(i)T}\mathbf{Q}_{ii}\alpha^{(i)} \\ \text{subject to:} \quad & 0 \le \alpha_j^{(i)} \le C, \;\; j = 1, \ldots, l_i \\ & \textstyle\sum_{j=1}^{l_i} y_j^{(i)}\alpha_j^{(i)} = 0 \end{aligned} \tag{82}$$

where $i = 1, \ldots, k$ and $\alpha^{(i)}$ is a vector of length l_i. Optimization of each subproblem in (82) is equivalent to that in (76) with the constraints that l_i components of α are free and the rest are set to zero. Optimization of k subproblems can be used to remove non-support vectors quickly in (76) based on the assumption that non-support vectors in (82) is a subset of those in (76). Further, we extend it to train support vector machines with multi-classes, where the one-against-the-others classification strategy is used. The computational diagram is depicted in Fig. 14.

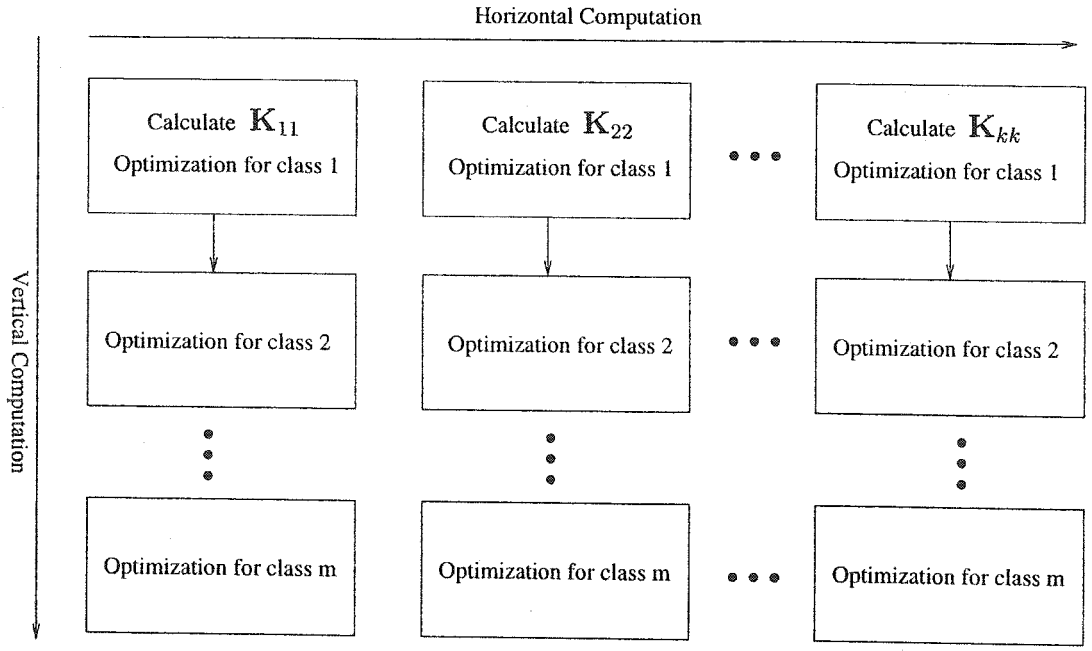

Figure 14: Parallel optimization diagram.

The computation in Fig. 14 is efficient due to three aspects. Firstly, kernel matrix can be effectively divided into block diagonal matrices such that each of them can fit into the memory. Many off-the-shelf algorithms can be used to solve this problem. Secondly, for vertical computation, all classes share the same block kernel matrix,

which needs to be calculated once. Finally, after the calculation of the block matrices in the first row, optimizations from the second class to the mth class are independent. Also, computations on different columns are independent. The computation framework is suitable for parallel optimization on the architecture of multi-processors since optimization independence and data locality can maximize parallelism and reduce the cost of long latency remote communication in using multi-processors [90].

After the above parallel optimization, most non-support vectors for each class will be removed from the training set. Then a new training set for each class can be obtained by collecting support vectors from the optimization sub-problems in the same row as shown in Fig. 14. Although the size of the new training set is much smaller than that of the original one, the memory may not be large enough to store the kernel matrix, especially when dealing with a large data set. Therefore, a fast sequential working set algorithm for training SVM is proposed and summarized as follows:

Fast Sequential Optimization for Training SVM

Input: Training set is S, and the fixed size of the working set is d, where $d \leq l$ and l is the size of the training set. Also, square kernel caching matrix with the dimension d is provided.

Output: $\alpha_i, i = 1, \ldots, l$.

Initialization: Shuffle the training set; set α_i to zero and select a working set B such that $B \subseteq S$.

Optimization:

Repeat

1. Apply modified SMO to optimize a sub-problem in working set B, in combination with some effective techniques such as kernel caching, efficient computation of kernel matrix, then update α_i.
2. Select a new working set with a queue technique.

Until the specified stopping conditions are satisfied.

The above algorithm can also be used as the optimizer for parallel optimization in Fig. 14, where the size of working set is the same as that of training subset since each block diagonal matrix can be stored into the memory, and step 2 is skipped.

5.4 Strategies of implementation

Section 5.3 provides a general computational framework for training support vector machines. For efficient computation, one needs to take into account some issues such as kernel caching, the computation of kernel matrix, selection of a new working set and stopping conditions.

5.4.1 Kernel caching

Kernel cache in this paper is defined as a part of contiguous memory that stores the $d \times d$ square kernel matrix on the working set. The size of the working set d should be large enough to contain all support vectors in the whole training set and small enough to satisfy the memory constraint. Since kernel matrix on the working set is completely cached, each element of the kernel matrix needs to be evaluated only once and must be calculated via a fast method presented later before starting the optimization so that during the optimization all kernel elements are available.

5.4.2 Optimization on the working set

In our optimizer, a worst violating pair of patterns ($i\_\text{up}, i\_\text{low}$) are always updated according to Eqs. (79)(80) , instead of choosing them just from I_0, the previous i_up and i_low in Keerthi et al. SMO [63]. The strategy of choosing the worst violating pair for optimization is also used in SVMTorch [18]. Since kernel elements are completely cached, evaluation of F_i in Eqs. (79)(80) has low computational cost. Updating F_i can be done efficiently as follows:

$$\Delta F_i = y_{i\_\text{up}} \Delta\alpha_{i\_\text{up}} k(\mathbf{x}_i, \mathbf{x}_{i\_\text{up}}) + y_{i\_\text{low}} \Delta\alpha_{i\_\text{low}} k(\mathbf{x}_i, \mathbf{x}_{i\_\text{low}}). \tag{83}$$

where $i = 1, \ldots, d$, $\Delta\alpha_{i\_\text{low}} = \alpha_{i\_\text{low}}^{\text{new}} - \alpha_{i\_\text{low}}^{\text{old}}$ and $\Delta\alpha_{i\_\text{up}} = \alpha_{i\_\text{up}}^{\text{new}} - \alpha_{i\_\text{up}}^{\text{old}}$. F_i is also cached. We use the following pseudocode to illustrate the simple optimization procedure.

Optimization on the working set

1 **Initialization:** $F_i = -y_i$, $i = 1, \ldots, d$.

2 **Loop**

2.1 Select worst violating pair ($i\_\text{up}, i\_\text{low}$) and calculate b_{up} and b_{low}.

2.2 If $b_{\text{low}} < b_{\text{up}} + 2\tau$, then goto 3.

2.3 Update α_{i_up}, α_{i_low}.

2.4 Update F_i in terms of Eq. (83) where $i = 1, \ldots, d$.

2.5 Update I_0, I_1, I_2, I_3, I_4.

3 Load a new working set, update F_i and kernel matrix.

4 If global stopping conditions are satisfied, the algorithm terminates; otherwise goto 2.

Here τ is a positive tolerance parameter. In the above loop, multiplication operations for updating F_i contribute to cost $O(d)$ in each iteration; comparison operations for selecting the worst violating pair contribute to $O(d)$.

5.4.3 Selection of a new working set

After optimization on the current working set is finished, a new data set will be loaded to replace non-support vectors by queue operations. Two operations associated with a queue data structure are defined by

1. **Enqueue**($Q_S, id(\mathbf{x})$): Append the index[3] of a sample $\mathbf{x}$ at the rear of the queue Q_S.

2. **Dequeue**(Q_S): Remove the index of a sample from the front of the queue S and return it.

Operation $id(.)$ returns the index of a sample in set S. Each operation above takes $O(1)$ time. The queue initially stores the indices of all training samples S. An important step prior to that is to shuffle the training set randomly before starting

[3]Here the index denotes the sequential number of a sample in the total training set S.

parallel optimization such that distribution of training samples from different classes is balanced. Suppose that the working set is stored in an array $\mathbf{B}[1,\ldots,i,\ldots,d], i = 1,\ldots,d$, where each array element is an $n-$dimensional vector. The index set of non-support vectors is a sub-sequence $\{i_1, i_2, \ldots, i_k, \ldots, i_{\text{nsv}}\}$, $k = 1,\ldots,\text{nsv}$, where nsv is the number of non-support vectors of the current working set. Then the algorithm of selecting a new working set is summarized as follows:

Algorithm for selecting a new working set

Initialization: $\mathbf{B}[i] \leftarrow S[\textbf{Dequeue}(Q_S)], i = 1,\ldots,d.$

Selection:

$k \leftarrow 0.$

Repeat

1. **Enqueue**$(Q_S, id(\mathbf{B}[i_k]))$.
2. $\mathbf{B}[i_k] \leftarrow S[\textbf{Dequeue}(Q_S)]$.

Until $k = \text{nsv}$.

After a new working set is loaded, the kernel matrix on the new working set must be updated via the method in subsection 5.4.4. In addition, only F_i for these new training samples need to be calculated. Moreover, for a large data set, only the working set is required to be in the memory. One of the good properties of the above method is that IO (Input/output) access is efficient since training samples in the data file are accessed sequentially when a new working set is loaded.

5.4.4 Calculation of kernel matrix

When a kernel can be represented as a function of dot product, matrix multiplication can be used to calculate kernel elements efficiently. Obviously, three kernels such as linear, polynomial kernel and radial basic function (RBF) belong to this type. Here we describe the details of computing RBF kernel matrix, which can be easily extended to other types of kernel. RBF kernel can be written as

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(- \parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2 /(2\sigma^2)) \tag{84}$$

where $\| \cdot \|$ is the Euclidean norm and

$$\| \mathbf{x}_i - \mathbf{x}_j \|^2 = \mathbf{x}_i^T \mathbf{x}_i + \mathbf{x}_j^T \mathbf{x}_j - 2\mathbf{x}_i^T \mathbf{x}_j \tag{85}$$

$i, j = 1, \ldots, d$. Terms $\mathbf{x}_i^T \mathbf{x}_i$ can be calculated by calling CBLAS function **cblas_sdot**. Let array $\mathbf{A}$ consist of vectors $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d)$. $\mathbf{x}_i^T \mathbf{x}_j$, $i, j = 1, \ldots, d$ can be rewritten as $\mathbf{A}^T\mathbf{A}$ (Gram matrix), which can be easily calculated by calling CBLAS function **cblas_ssyrk**. Due to symmetry of the kernel matrix, only elements in the upper triangle are calculated for the implementation of **cblas_ssyrk**.

The kernel matrices in Fig. 14 can be calculated via the above method, where $d = l_i$ in (81). To update the kernel matrix when a new working set is loaded, we do not need to compute the total kernel matrix since some elements can be re-used. Let the new working set be split into two sets: support vector set represented by the array $\mathbf{B}_{\text{sv}}$ and non-support vector set $\mathbf{B}_{\text{nsv}}$. Updating the total kernel matrix requires $\mathbf{B}_{\text{sv}}^T\mathbf{B}_{\text{sv}}$, $\mathbf{B}_{\text{nsv}}^T\mathbf{B}_{\text{nsv}}$ and $\mathbf{B}_{\text{sv}}^T\mathbf{B}_{\text{nsv}}$. Since kernel elements $k(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{x}_i \in \mathbf{B}_{\text{sv}}$ and $\mathbf{x}_j \in \mathbf{B}_{\text{sv}}$, can be re-used, $\mathbf{B}_{\text{sv}}^T\mathbf{B}_{\text{sv}}$ does not need to be re-calculated. $\mathbf{B}_{\text{nsv}}^T\mathbf{B}_{\text{nsv}}$ and $\mathbf{B}_{\text{sv}}^T\mathbf{B}_{\text{nsv}}$ can be evaluated by calling CBLAS function **cblas_ssyrk** and **cblas_sgemm**.

The rationale of the usage of BLAS package is its computation efficiency, portability and maintenance. The key computational kernel of BLAS package such as matrix multiplication is implemented by hardware vendors in assembly language, which makes efficient use of cache, memory and instructions such as single instruction and multi-data (SIMD) [56] on Intel Pentium series or vector instructions in vector processors [90]. Moreover, BLAS has been efficiently implemented on different platforms, which enables the proposed SVM algorithm to perform well across platforms. Furthermore, the performance of the proposed method grows with the computational power of a processor in the future when a new BLAS package is plugged in. Consequently the cost of software maintenance is reduced.

5.4.5 Reduction of cache and TLB misses

In Fig. 14, CBLAS function **cblas_ssyrk** is used to calculate kernel matrices $\mathbf{Q}_{ii}$. **cblas_ssyrk** stores results into the upper triangle of the symmetric kernel matrix. Thus we need to copy the elements in the upper triangle into the lower triangle so that SMO can access contiguous memory during the optimization process. Crude copy operations will result in high TLB misses and cache thrashing, which will lead

to a high cost. The reason is that kernel matrix takes a large portion of paged memory and the limited TLB does not contain all memory page numbers for the kernel matrix. In order to solve this problem, the upper-triangle of the kernel matrix is broken up into contiguous blocks of a fixed size $d_{\mathbf{B}}$ and a workspace with the size of $d_{\mathbf{B}}d$ is allocated[4]. Fig. 15 shows how to copy elements into the lower-triangle.

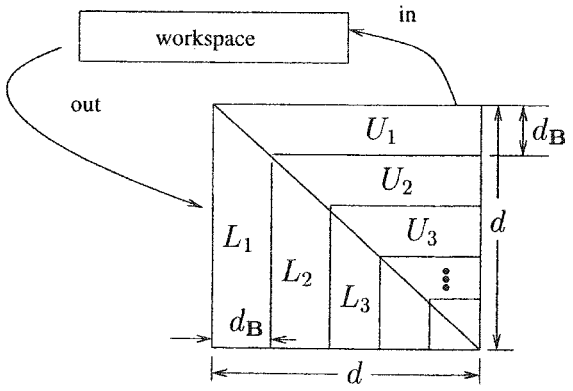

Figure 15: Copy upper-triangle elements into lower-triangle via a workspace.

The workspace is used to copy blocks of the upper-triangle into the symmetric blocks of lower-triangle. $d_{\mathbf{B}}$ can be determined by the size of the second-level cache (L2) if available. Similarly, the method can be used to copy elements of $\mathbf{B}_{\text{nsv}}^T\mathbf{B}_{\text{nsv}}$ and $\mathbf{B}_{\text{sv}}^T\mathbf{B}_{\text{nsv}}$ into the kernel matrix during the sequential working set optimization.

5.4.6 Insert positive samples into the working set

In Fig. 14, it is possible that there is no positive sample[5] on some working set since the size of negative samples is much larger than that of positive samples for one-against-the-other multi-class training strategy. As a result, SMO will lead to an incorrect solution. In order to tackle this problem, we first randomly collect one sample from each class. During the training, if no positive sample on a working set is found, we replace the first negative sample using the positive sample from the collected set.

[4]Align the starting address so that it begins at a cache-line boundary.
[5]Its corresponding desired output is 1.0.

5.4.7 Stopping conditions

With respect to the stopping conditions, most algorithms evaluate KKT conditions of each training sample. If none violates KKT, the algorithm stops. On a large data set the computational cost for this step is high. Instead, heuristic stopping rules are suggested. We track the variations of b_{up}, b_{low} and the number of support vectors on the two successive working sets. If all these variations are small and a sufficient number of training samples have been learned, the algorithm terminates. The heuristic stopping rules are given by

$$\begin{aligned} &(\, |\Delta \text{sv}| < 20 \;\text{ and }\; |\Delta b_{\text{up}}| < 2\tau \;\text{ and }\; |\Delta b_{\text{low}}| < 2\tau \\ &\text{and Number of learned samples} > l \,) \; \textit{or} \; (\, \text{sv} \geq d \,) \end{aligned} \tag{86}$$

where $|\,.\,|$ computes the absolute value and l is the size of training set. Adding the constraint on the number of learned samples is necessary. Otherwise, in an extreme case, if the loaded samples are all non-support vectors, the algorithm is likely to terminate too early. As a result, the optimization is incomplete and SVM will suffer the risk of bad generalization performance. Furthermore, although the above stopping conditions and parallel optimization may lead to a 'not-too-precise' optimal solution, they are essential to control the computational cost within a reasonable bound.

5.5 Analysis of space and runtime complexity

Before analyzing the space and runtime complexity[6] of the proposed algorithm, some notations are defined below:

- l: size of original training set
- d: size of the working set
- n: dimension of the feature vector
- m: number of classes
- k: number of working sets in Fig 14
- h_{ij}, $i = 1, \ldots, m$; $j = 1, \ldots, k$: number of support vectors on the working set at the ith row, jth column in Fig. 14

[6] Analysis is based on a single processor

where $k = \lceil l/d \rceil$. For the sake of simplicity, the size of working set d is used in both parallel and sequential optimizations. In addition, we assume that the final number of support vectors of SVM for each class is not larger than d and the dominant computational cost comes from multiplication operations without considering the cost of memory access.

5.5.1 Space complexity

The requirements of storage space in the two steps are different. In both steps, only a working set is loaded into the memory, rather than the total training set. Usually n is much smaller than d. The dominant storage comes from the kernel matrix. In Fig. 14, its size is estimated as $4d^2$ bytes[7]. In the second algorithm, since some kernel elements are re-used, we can not totally overwrite the kernel matrix and require a new matrix with the same size to calculate $\mathbf{B}_{\text{nsv}}^T\mathbf{B}_{\text{nsv}}$ and $\mathbf{B}_{\text{sv}}^T\mathbf{B}_{\text{nsv}}$. Its total storage space for sequential working set optimization is estimated as $8d^2$ bytes.

5.5.2 Analysis of runtime complexity

It is difficult to precisely analyze the runtime complexity of the algorithm. Therefore we just estimate dominant computational cost under some assumptions. In Fig. 14, computation of kernel matrices has time complexity of $g_{\text{ker}}^{(1)} = O(\frac{1}{2}knd^2) = O(\frac{1}{2}lnd)$. The factor $1/2$ appears because **cblas_ssyrk** only updates the upper-triangle of the kernel matrix. Now we estimate the cost of optimization on the working set. For each iteration, updating F_i in Eq. (83) contributes to a main cost, which is $O(d)$. In order to estimate the number of iterations on the working set, an ideal model is assumed: at least one support vector at each iteration is added. So the number of iterations is not larger than the number of support vectors. Then the time complexity of optimization can be approximated by

$$g_{\text{op}}^{(1)} = O(\sum_{i=1}^{m}\sum_{j=1}^{k} h_{ij}d) \tag{87}$$

Therefore, the total time complexity for parallel optimization is estimated by

$$g_{\text{ker}}^{(1)} + g_{\text{op}}^{(1)} = O(\frac{1}{2}lnd + \sum_{i=1}^{m}\sum_{j=1}^{k} h_{ij}d). \tag{88}$$

[7]Data type of a kernel element is float. The size of a float data type is 4 bytes.

After the parallel optimization step, the size of training set for class i is given by

$$P_i = \sum_{j=1}^{k} h_{ij}, \tag{89}$$

where $i = 1, \ldots, m$. Let r_{ij} $(r_{ij} \geq 0)$, $i = 1, \ldots, m$, $j = 1, \ldots, T_i$ be the number of support vectors on the jth working set of class i where T_i denotes the number of optimized working sets of class i when the optimization is terminated. For the sake of analysis $r_{i0} = 0$. The costs of computing $B_{\text{sv}}^T B_{\text{nsv}}$ and $B_{\text{nsv}}^T B_{\text{nsv}}$ on the jth working set of class i are $O(r_{i(j-1)}(d - r_{i(j-1)})n)$ and $O(\frac{1}{2}(d - r_{i(j-1)})^2 n)$, respectively. Thus the total cost of updating kernel matrices can be approximated by

$$g_{\text{ker}}^{(2)} = O(\sum_{i=1}^{m} \sum_{j=0}^{T_i - 1} (r_{ij}(d - r_{ij})n + \frac{1}{2}(d - r_{ij})^2 n)). \tag{90}$$

Expression (90) can be simplified to

$$g_{\text{ker}}^{(2)} = O(\sum_{i=1}^{m} \sum_{j=0}^{T_i - 1} (d - r_{ij})n \frac{d + r_{ij}}{2}). \tag{91}$$

It can be observed that $\sum_{j=0}^{T_i-1}(d - r_{ij})$ is the number of learned samples for class i. Let $\sum_{j=0}^{T_i-1}(d - r_{ij}) = \beta_i P_i$. According to the stopping conditions in (86), β_i is greater than 1.0. Since $d/2 \leq (d + r_{ij})/2 \leq d$, $g_{\text{ker}}^{(2)}$ is bounded as follows

$$O(\frac{1}{2} dn \sum_{i=1}^{m} \beta_i P_i) \leq g_{\text{ker}}^{(2)} \leq O(dn \sum_{i=1}^{m} \beta_i P_i). \tag{92}$$

The cost for optimization on the working sets is

$$g_{\text{op}}^{(2)} = O(\sum_{i=1}^{m} \sum_{j=1}^{T_i} r_{ij} d), \tag{93}$$

Substituting $\sum_{j=0}^{T_i-1} r_{ij} = dT_i - \beta_i P_i$ into (93), we can yield

$$g_{\text{op}}^{(2)} = O(d^2 \sum_{i=1}^{m} (T_i + r_{iT_i}) - d \sum_{i=1}^{m} \beta_i P_i). \tag{94}$$

By adding (88),(90) and (94) together the total cost $g_{\text{ker}}^{(1)} + g_{\text{op}}^{(1)} + g_{\text{ker}}^{(2)} + g_{\text{op}}^{(2)}$ is approximated by

$$O(\frac{1}{2} lnd + d \sum_{i=1}^{m} P_i + \sum_{i=1}^{m} \sum_{j=0}^{T_i - 1} \frac{1}{2} n(d^2 - r_{ij}^2) + d^2 \sum_{i=1}^{m} (T_i + r_{iT_i}) - d \sum_{i=1}^{m} \beta_i P_i). \tag{95}$$

When inequality (92) is applied to (95), the total cost is bounded above by

$$O(d(\frac{1}{2}ln + n\sum_{i=1}^{m}\beta_i P_i - \sum_{i=1}^{m}(\beta_i - 1)P_i + d\sum_{i=1}^{m}(T_i + r_{iT_i}))) \tag{96}$$

and bounded below by

$$O(d(\frac{1}{2}ln + \frac{1}{2}n\sum_{i=1}^{m}\beta_i P_i - \sum_{i=1}^{m}(\beta_i - 1)P_i + d\sum_{i=1}^{m}(T_i + r_{iT_i}))). \tag{97}$$

Since P_i, β_i and T_i are decreasing functions of d, there exists an optimal value of d such that the minimum of total cost is reached. From Eqs. (96)(97), the bound on the total cost is split into two terms: one for kernel computation and the other for SMO optimization. The term $O(d^2\sum_{i=1}^{m}(T_i + r_{iT_i}) - d\sum_{i=1}^{m}(\beta_i - 1)P_i)$ can be used to approximate the cost for SMO optimization. Based on the assumption that at least one support vector on each working set is added with the progress of optimization, we get $T_i < d$ [8]. When l is large, kernel computation will dominate the cost. Let $P_i = \mu_i l$. Then the upper bound of kernel computation is given by

$$O(\frac{1}{k}nl^2(\frac{1}{2} + \sum_{i=1}^{m}\mu_i\beta_i)). \tag{98}$$

Now we list the properties of (98).

- When most non-support vectors are removed at the parallel optimization step, both μ_i and β_i become small. Thus the computation cost becomes low.

- If $\mu_i \leq 0.1$ and $\beta_i \leq 2.0$ [9], the cost becomes $O(\frac{1}{k}(\frac{1}{2} + \frac{m}{5})nl^2)$. If the number of classes m is large, the cost is $O(\frac{1}{5}\frac{1}{k}mnl^2)$. With respect to the one-against-the-others training strategy for multi-classes, assume that training SVMs for different classes does not share the computation of kernel matrix since a large kernel matrix can not fit into the memory. The cost of kernel computation is about $O(\frac{1}{2}mnl^2)$, which is much higher than $O(\frac{1}{5}\frac{1}{k}mnl^2)$, especially on a large training set where k is much larger than 1. Moreover, since $O(\frac{1}{5}\frac{1}{k}mnl^2) = O(\frac{1}{5}dmnl)$, for a fixed working size d, m and n, the cost linearly scales with the size of training set l.

[8] In practice, T_i is usually much smaller than d.

[9] In practice, this has been frequently observed in experiments.

5.6 Experimental results

This section is divided into two subsections. In Subsection 5.6.1, we investigate the various properties of the proposed algorithm and compare it with the existing algorithms on large data sets. Subsection 5.6.3 focuses on its generalization performance on handwritten character data sets. The state-of-the-art results will be reported on three well-known handwritten digit databases CENPARMI, USPS, MNIST and on NIST handwritten database for lowercase characters. Excellent performances on two very large commercial data sets (Hanwang handwritten digit, Hanwang handwritten Chinese) will also be reported.

The code was compiled by Microsoft visual C++6.0. All experiments were performed on a PC with single Intel P4 1.7Ghz processor with 256K L2 (second-level) cache, SDRAM [10] of 1.5 Gigabytes and 200 G hard disk (7200 RPM). The operating system was Windows 2000 Professional.

There are several character databases used in our experiments. CENPARMI handwritten digit database [110][111] consists of 6,000 unconstrained handwritten numerals originally collected from dead letter envelopes by the U.S. postal service at different locations. The numerals in this database are stored in binary format with a resolution of approximately 166 PPI. We use 4000 numerals [11]. USPS [12] database contains 9298 handwritten digits (7291 for training, 2007 for testing) [123].

MNIST [78] handwritten digit database consists of 60,000 training samples and 10,000 testing samples [13], which originate from NIST database. The preprocessing was done by LeCun's research group and a linear transform was performed such that all patterns were centered into 28×28 while keeping the aspect ratio. The pixel values of resulting gray-scale images were scaled to fall in the range from -1.0 to 1.0.

Hanwang handwritten digit database is a large commercial database from Hanwang, an OCR company in China. It consists of 1,321,718 training samples and 300,000 testing samples. The samples are in binary format. Some samples in this database are illustrated in Fig. 16.

NIST database of handwritten lowercase characters originally consists of 26,000

[10] Single Data Rate Memory

[11] CENPARMI digit database consists of three subsets A, B and C. The subsets A and B are used for training and C for testing.

[12] Available from www.kernel-machines.org/data.html.

[13] This database is available from http://yann.lecun.com/exdb/mnist/.

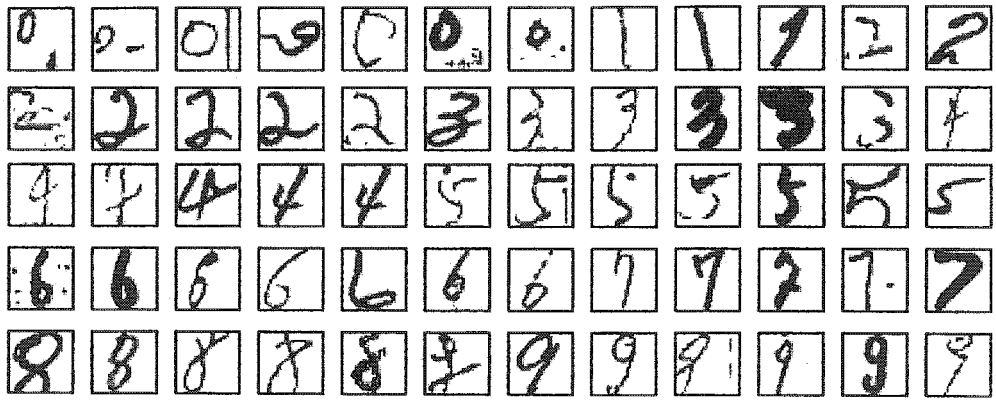

Figure 16: Some samples from Hanwang digit database.

training samples and 12,000 testing samples. In the training set, each category contains 1000 samples. This database includes some uppercase and noisy garbage patterns that do not belong to any of the 26 categories. About 6% of these patterns are highly confusing patterns such as "q" and "g", "i" and "l", which can scarcely be identified by human. So we clean the database and remove testing samples of three categories including "q", "i" and "g". Finally, we obtain a training set of 24,092 samples and a testing set of 10,688 samples.

Hanwang handwritten Chinese database is a large commercial database from Hanwang company. It consists of four subsets A, B, C, and D with different qualities. There are 3755 categories, each of which has about 800 samples. In our experiment, we randomly split the data set into two parts: 2,144,489 for training and 542,122 for testing. Some samples in this database are shown in Fig. 17.

For character recognition, discriminative feature extraction is an important step to enhance the generalization performance of a learning algorithm. In our experiments, features for handwritten digit, lowercase characters and handwritten Chinese are extracted based on the gradients of an image. For handwritten digit and lowercase characters, a 576 dimensional feature vector [33] is obtained from the normalized patterns of size 22×22 and a 400 dimensional feature vector on those of size 18×18. On Hanwang handwritten digit database, a 576 dimensional feature vector is first extracted. Then its dimension is reduced to 120 by principal component analysis [48]. For handwritten Chinese characters, a feature vector of size 1296 is generated [35]. Then its dimension is reduced to 392 by multiple discriminant analysis [41].

Figure 17: Some samples from Hanwang handwritten Chinese database.

The parameters C in (76) and τ are set to 10.0 and 0.01, respectively. ATLAS [124](Automatically Tuned Linear Algebra Software) [14] optimized on P4 is the default BLAS package for computing the kernel matrix. The value of σ^2 in (84) is set to 0.3, unless mentioned otherwise. This value is chosen using a cross-validation method on a subset [35].

5.6.1 Algorithm properties and comparisons of training performance

Computation of kernel matrix

Three methods for computing kernel matrix are tested on Hanwang handwritten digit database. One is simple C implementation of **cblas_sgemm** and **cblas_ssyrk**; the other two methods are BLAS packages optimized on P4: ATLAS and MKL6.0 [15]. The RBF kernel is used. The size of working set d is set to 8000. Training performance

[14] Available from http://math-atlas.sourceforge.net/.

[15] http://developer.intel.com/software/products/mkl/index.htm.

measures are shown in Table 5, where μ_i, β_i and T_i are defined in Section 5.5, and BSV

Table 5: Performance measures for ATLAS on Hanwang handwritten digit database

Class	0	1	2	3	4	5	6	7	8	9
SV	7208	3256	7639	8255	6541	7221	5066	5506	7245	5876
BSV	65	441	30	88	89	47	114	452	35	354
μ_i	0.107	0.040	0.112	0.114	0.099	0.108	0.073	0.076	0.091	0.055
β_i	1.00	1.96	1.00	1.00	1.00	1.00	1.08	1.09	1.00	1.59
T_i	69	18	101	102	46	67	24	28	57	34

denotes the number of bounded support vectors. Note that the size of the working set for class 3 is set to 8500 at the second step. It can be observed from Table 5 that μ_i is less than 0.1, which infers that most non-support vectors are removed at the step of parallel optimization. In addition, most β_i are close to 1.0. That is, at the step of sequential optimization, the algorithm usually goes through the training set once and stops. In order to see how the kernel computation influences the training speed, we compare the performance of the above three methods, as shown in Table 6.

Table 6: Performance comparisons of three methods for kernel computation

Methods	ATLAS	MKL6.0	C implementation
$g_{\text{Ker}}^{(1)} + g_{\text{Ker}}^{(2)}$(seconds)	1957	2265	7401
$g_{\text{op}}^{(1)} + g_{\text{op}}^{(2)}$ (seconds)	755	1169	758
Total training time (seconds)	2712	3434	8159

In Table 6, the kernel computation dominates the cost for C implementation, which is consistent with the complexity analysis in Section 5.5, while it does not seem to be true for ATLAS and MKL6.0. In fact, ATLAS and MKL6.0 are both optimized on P4 and efficiently make use of SIMD instructions, where several computations are done with a single instruction in parallel [55]. That is, parallelism exists in ATLAS and MKL6.0. When we analyze the computational complexity, kernel elements are assumed to be calculated sequentially. In addition, compared with C implementation, ATLAS achieves a speed-up factor of about 3 on P4.

The size of working set

In Section 5.5, we claim that there exists an optimal size of the working set which achieves the minimal cost. Therefore we continued to test the proposed method on Hanwang digit database and to observe how the size of the working set affects the performance. Fig. 18 shows the training time and average number of support vectors with the growing size of the working set. It can be seen from Fig. 18 that the training

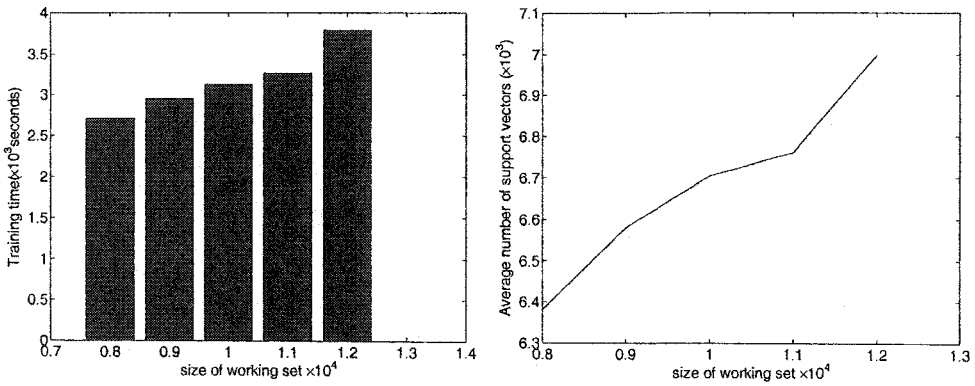

Figure 18: Training time and average number of support vectors with the size of working set.

time and average number of support vectors are growing with the size of the working set ($\geq$ 8000), but the growth rate is slow. That is, training time is not sensitive to this parameter in a large range such that the users do not need to tune this parameter obviously. Moreover, although the average number of support vectors is increasing, the substitution error rate on Hanwang testing set remains unchanged (0.5%). This fact indicates that a working set good size can remove the redundant support vectors to some extent. For the optimal training speed, it is better to set different working sizes at these two steps. Moreover, at the second step, the size of the working set is chosen to be close to the number of support vectors in each class.

Testing different kernels and comparing performance with existing algorithms

In this experiment we tested the proposed method on MNIST database with RBF and polynomial kernel and on Hanwang handwritten digit database with RBF kernel,

and compared its performance with that of existing SVM packages such as SVM^{light} [16] and LIBSVM [17]. For RBF kernel, a 576 dimensional discriminative feature vector was extracted [33]. With respect to the polynomial kernel, we directly applied SVM on 28×28 pixel images. Since patterns on the original MNIST are not truly located at the center, the preprocessing was performed by first enclosing the pattern in the bounding rectangular box, and then by translating this rectangle into the center of a 28×28 box. Then patterns were smoothed using the following mask:

$$\frac{1}{16}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{99}$$

Next, we used DeCoste's [24] idea to normalize each pattern by its Euclidean-norm scalar value such that the dot product was always within $[-1, 1]$. We used the polynomial kernel $(\mathbf{x}_1 \cdot \mathbf{x}_2)^7$. The dimension of input vectors was 784 (28×28). The size of the working set was still set to 8000. There exists a fast method for computing general polynomial kernel $(\mathbf{x}_1 \cdot \mathbf{x}_2)^q$, where q is a positive integer. Let $u_- = \lfloor \log_2 q \rfloor$ and $u_+ = \lceil \log_2 q \rceil$. Then

$$u = \begin{cases} u_- & \text{if } u_- + q - 2^{u_-} < u_+ + 2^{u_+} - q, \\ u_+ & \text{otherwise.} \end{cases} \tag{100}$$

The polynomial kernel can be calculated by

$$(\mathbf{x}_1 \cdot \mathbf{x}_2)^q = \overbrace{((((\mathbf{x}_1 \cdot \mathbf{x}_2)^2)^2)\cdots)^2}^{u}(\mathbf{x}_1 \cdot \mathbf{x}_2)^{q-2^u}. \tag{101}$$

The number of multiplication operations is $u + |q - 2^u|$. For example, $(\mathbf{x}_1 \cdot \mathbf{x}_2)^7 = (((\mathbf{x}_1 \cdot \mathbf{x}_2)^2)^2)^2(\mathbf{x}_1 \cdot \mathbf{x}_2)^{-1}$.

In order to ensure that the comparisons are fair, the following pre-conditions are assumed:

- Dense feature vector that is stored in a contiguous memory.
- Similar size of cache that stores kernel elements.
- The fixed stopping tolerance.

[16] Available from http://svmlight.joachims.org/.
[17] Available from http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

- The fixed kernel parameters and C in (76).
- The fixed experimental platform.

The post condition is that each method should achieve a similar error rate on the test set. Before we conduct the performance comparisons, some facts on Libsvm and SVM^{light} are provided. Libsvm efficiently implements Keerthi et al's SMO and uses a simple LRU row caching strategy, which is the same as SVM^{light}'s and different shrinking methods. The stopping conditions for libsvm are

$$b_{\text{low}} < b_{\text{up}} + 2\tau. \tag{102}$$

and termination criteria for SVM^{light} are given by

$$\begin{array}{rcll} y_i(F_i + b) & \geq & -\tau & \forall i \ \ \alpha_i = 0, \\ |y_i(F_i + b)| & < & \tau & \forall i \ \ 0 < \alpha_i < C, \\ y_i(F_i + b) & \leq & \tau & \forall i \ \ \alpha_i = C, \end{array} \tag{103}$$

where τ is a stopping tolerance and set to 0.01. In addition, SVM^{light} skips checking KKT conditions of inactive variables before it terminates. The cache sizes on MNIST and Hanwang handwritten digit databases are set to 250 M [18] and 400 M, respectively. The working set size for the proposed method is set to 8000. Table 7 shows comparisons of the total training time of three methods on MNIST and Hanwang handwritten digit databases. It can be seen that the proposed method performs best

Table 7: Comparisons of total training time of three methods(hours). A: one-against-others training strategy for multi-classes; B: one-against-one for multi-classes.

Database		Proposed method	SVM^{light}(v5.0)		Libsvm(v2.4)	
			A	B	A	B
MNIST	RBF	0.064	1.11	0.37	3.04	0.39
	POLY	0.075	2.03	0.54	4.52	0.54
Hanwang digit	RBF	0.75	-	7.63	-	16.21

on both databases and its training time scales well with the size of the training set. Also, the above table indicates that SVM^{light} and Libsvm perform much better based on a one-against-one training strategy for multi-classes than on a one-against-others

[18] Megabytes. $1M = 2^{20}$ bytes.

strategy. The reason is that for a one-against-others strategy the limited cache only stores a small number of rows of kernel matrix when the size of training set is large so that cache thrashing occurs frequently and LRU caching policy is prone to failure. Moreover, we can see that the computational cost of SVM^{light} and Libsvm is very high when the training strategy for multi-classes is one-against-others.

IO efficiency

With respect to the proposed method, only the working set is required to be stored in memory, rather than the whole training set. A new working set is loaded into memory from a data file by queue operations such that data are accessed sequentially most of the time. As a result, the IO buffer for file reading operations will be efficiently utilized and file access time will be considerably reduced. On Hanwang handwritten digit database, when the size of the working set is set to 8000, the total IO time is only 14.39 seconds, which is much less than the total training time of 2712 seconds.

Performance on a large database with thousands of classes

This experiment was performed on Hanwang handwritten Chinese character database. The σ^2 parameter of RBF kernel in Eq. (84) is set to 0.8, which can be obtained according to the method in [35]. For the proposed algorithm, the size of the working set at two steps are set to 8000 and 3000, respectively. The total training time is about 19 hours. With respect to Libsvm and SVM^{light}, their computational cost of Libsvm on this database is prohibitively high based on a one-against-other training strategy for multi-classes. Therefore, we only test SVM^{light} using a one-against-one strategy. Its training time is about 644 hours. Also, in our experiment, the training speed of SVM^{light} is fast on the data set of one pair of classes since the number of samples for each class is small, just about 800. However, its time complexity quadratically grows with the number of classes (m) whereas the dominant cost of the proposed algorithm proves to linearly scale with m in (98) for a sufficiently large m. This experimental result indicates that the proposed algorithm's advantage over Libsvm and SVM^{light} is more obvious on a database of huge size with a large number of classes.

5.6.2 Performance on a large artificial data set

In the previous experiments, the size of working set is large enough to contain support vectors. In some cases, the condition is not satisfied due to the memory constraint. Therefore, we have to find a method to deal with this problem. In this experiment, the proposed method is tested on an artificially generated data set called *ringnorm* [9], which has 20 dimensions and 2 classes. Class 1 is multivariate normal with a mean of zero and a covariance matrix 4 times the identity. Class 2 has unit covariance matrix and mean (a, a, ..., a), where $a = \frac{2}{\sqrt{20}}$ [19]. According to Fukunaga et al's method [47], the precise Bayes' error rate for the above two-class classification problem can be computed using numerical integration which yields 1.24%. The generated training and testing sets consist of 100 million and 3 million samples, respectively, where two classes have the same *priori* probability [20]. The σ^2 of RBF kernel in Eq. (84) and C are set to 198.0 and 20.0, respectively. At the parallel optimization step, the size of the working set is 8000. After parallel optimization ends, the size of training set for the sequential optimization is 5807025. Then we divide the set into many subsets, each of which consists of 14,000 samples. The size of working set is still 8000. At this stage, we observed that on each subset the number of support vectors is about 7900 and among these support vectors the number of bounded ones is about 7700. The total training time and IO time are 15.89 hours and 316 seconds, respectively. The final decision is made using SVMs on these subsets in a majority vote method: for each unseen pattern, the output of each SVM is added together. This strategy is similar to that in the ensemble methods such as Bagging [8]. The error rate on the test set using the above rule is 1.23%, a little lower than the Bayes' error rate. Theoretically the classification accuracy of any classifier should not be higher than that of the optimal Bayes classifier. The bias may come from the fact that the data generated by a computer does not exactly characterize the two-class multivariate Gaussian distributions. Moreover, we tested each SVM independently on the test set and observed that their error rates are close. Their mean and standard deviation are 0.0126 and 7.0×10^{-5}. This indicates that each subset is sufficient statistically for the classification.

[19] In paper [9], a is $\frac{1}{\sqrt{20}}$

[20] Data source is referred to http://www.cs.toronto.edu/~delve/data/ringnorm.

5.6.3 SVM's generalization performance for handwritten character recognition

In this subsection, we investigate the generalization performance of SVM on several handwritten character databases when it is trained by the proposed algorithm.

Recognition of handwritten digits and lowercase characters

The feature extraction [33] involves edge detection. Smoothing is a necessary prerequisite step. Since it is difficult to find an optimal filtering function to smooth the image, we follow a strategy similar to that of human vision: first recognize the global shape under the coarse resolution and then focus on the details under the fine resolution. Thus, for all handwritten digit databases mentioned above and NIST lowercase database, when training a basic SVM, the images are blurred three times using mask (99), which does not only remove noises but also reduces the number of support vectors. When training virtual SVM [102], we smooth the image only once to avoid the excessive removal of the details.

With respect to virtual SVM (VSVM), support vectors for different classes are first merged [21]. Then virtual patterns are generated by shifting them one pixel in four directions: up, down, left and right. RBF kernel is used in the following experiments. Tables 8,9, 10 show the performance comparisons on CENPARMI, USPS and handwritten NIST lowercase database, respectively. Misrecognized patterns for the proposed method on these databases are illustrated in Figs. 19,20,21.

Table 8: Error rates of different methods on CENPARMI database (%).

Methods	Recognized	Error rate
Cho's [16]	96.05	3.95
S.W. Lee's[81]	97.80	2.20
400-20-10 MLP + AdaBoost [28]	97.20	2.80
MQDF [27]	98.00	2.00
GLVQ [30]	96.30	3.70
Local Learning Framework [30]	98.10	1.90
Proposed method	98.70	1.30
Svc-rbf [83]	98.90	1.10

[21] Svms from different classes share some support vectors

Table 9: Performance comparison of some methods on the USPS database. USPS+, one variant of USPS database, contains some machine-printed digit samples.

Classifiers	training set	test error	reference
Nearest-neighbor	USPS+	5.9%	Simard et al. [106]
LeNet1	USPS+	5.0%	LeCun et al. [76]
Optimal margin classifier	USPS	4.6%	Boser et al. [5]
SVM	USPS	4.0%	Schölkopf et al. [98]
Local learning	USPS+	3.3%	Bottou et al. [6]
Virtual SVM	USPS	3.2%	Schölkopf et al. [99]
Virtual SVM, local kernel	USPS	3.0%	Schölkopf [100]
Boosted Neural Nets	USPS+	2.6%	Drucker et al. [39]
Tangent distance	USPS+	2.6%	Simard et al. [106]
Feature-based virtual SVM	USPS	2.34%	the proposed method
Human error rate	—	2.5%	Bromley et al. [10]
Human error rate	—	1.51%	our statistical results [29]

Table 10: Error rates of different methods on the test set of NIST lowercase database.

Methods	Raw error rate (%)
GLVQ [30]	10.17
MQDF [30]	10.37
160-100-27 MLP [30]	8.40
Local Learning Framework [30][31]	7.66
Feature-based SVM	7.44
VSVM	6.70

On MNIST handwritten digit database, in order to further enhance the performance of VSVM, multi-scale virtual patterns are generated. After we trained SVM, the number of merged support vectors was 10,422 and then virtual patterns were generated by shifting these support vectors by one pixel in four directions (up, down, left and right) and smoothing them once, twice and three times, respectively. When the virtual patterns are smoothed once, the trained SVM is called VSVM^a. The total size of the virtual training set was 156,330 ($= 3 \times 5 \times 10,422$). Performance comparisons of different methods on MNIST testing set are shown in Table 11.

In Table 11, VSVM^b is virtual SVM with multi-scale virtual patterns. Although VSVM^b achieved the state-of-the-art performance, the performance gap is not as

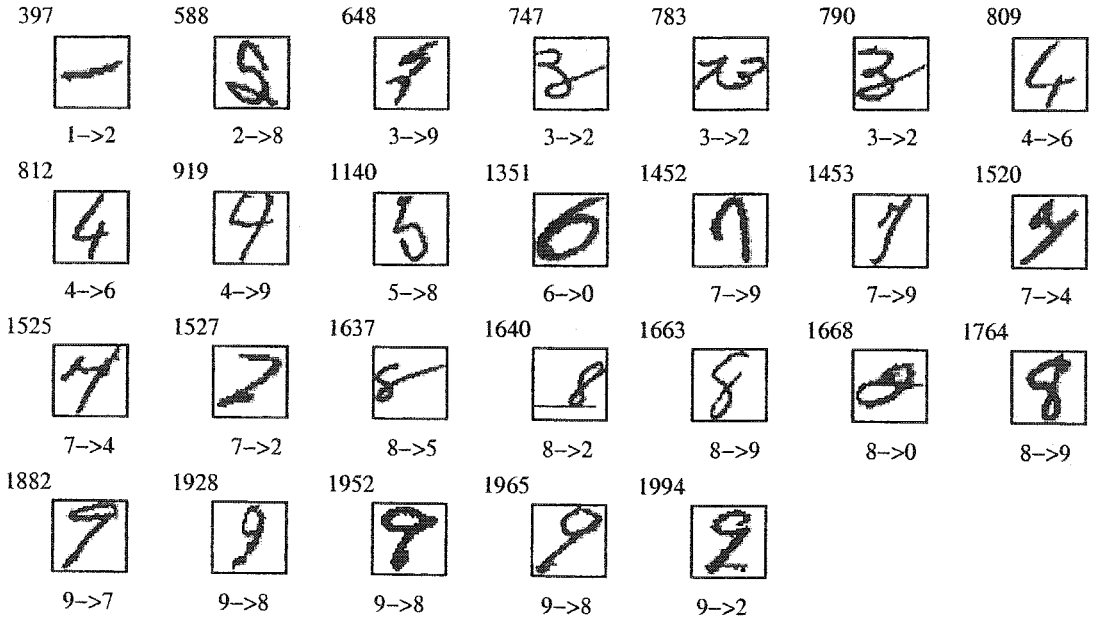

Figure 19: The 26 errors (1.3% error rate) for the CENPARMI test set. The number in the upper-left corner of each image indicates the test sample number. The first digit on the bottom specifies the true label. The second digit on the bottom is the recognized digit label.

obvious as that between VSVM^a and feature-based SVM. If the trade-off between cost and accuracy is taken into account, VSVM^a is a better choice in practice. Liu et al. [83] also obtained similar results based on different features using SVM. The performances of other methods in [78] were reported on pixel values without explicit feature extraction. The above comparisons indicate that extraction of discriminative features is still a better choice to improve the generalization performance of a classifier than other strategies when a prior knowledge for the specified classification problem is available. Fig. 22 shows the misrecognized patterns of VSVM^b on MNIST testing set.

5.6.4 Rejection performance of SVM

In some real world applications such as cheque recognition, the reliability rate is more important than the raw error rate. So it is necessary to evaluate SVM's rejection

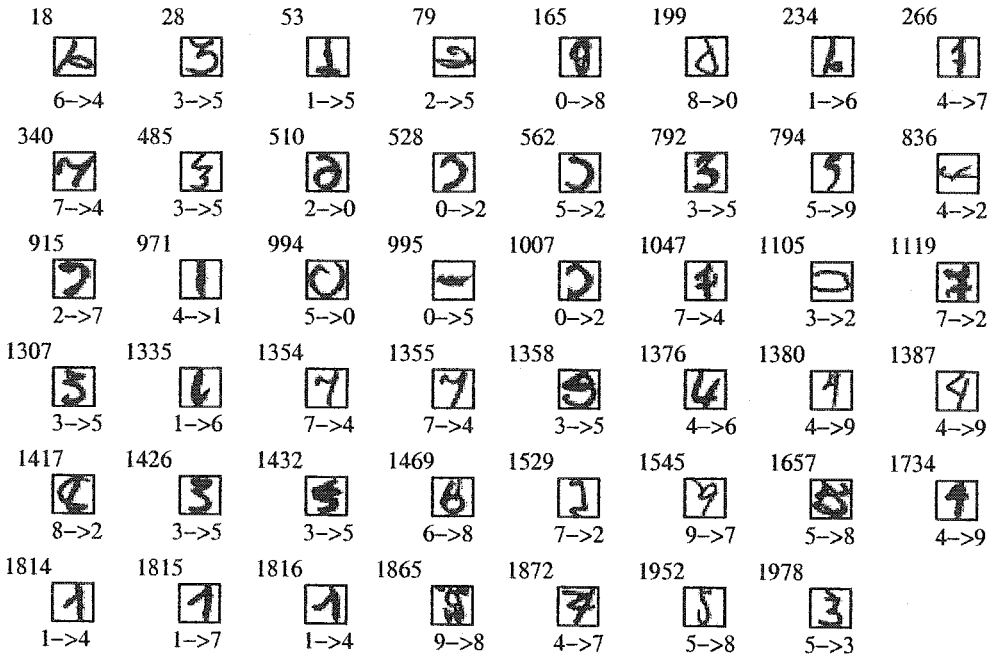

Figure 20: The 47 errors (2.34% error rate) for the USPS test set. The number in the upper-left corner of each image indicates the test sample number. The first digit on the bottom specifies the true label. The second digit on the bottom is the recognized digit label. From the above figure, it can be seen that there are obviously four errors in the original labelling (234, 971, 994, 1978).

performance. The reliability is defined by

$$\text{Reliability} = \frac{\text{Recognition rate}}{\text{100\%- Rejection rate}}.$$

Patterns are rejected when the difference between the outputs of the top two classes is smaller than a predefined threshold ξ. Experiments aimed at evaluating rejection performance were conducted on MNIST database. Its results are shown in Table 12.

5.7 Conclusions

We have presented an efficient training algorithm for support vector machines on several databases as well as a huge database with thousands of classes. This algorithm consists of two steps: parallel and sequential optimizations. At the parallel optimization step, most non-support vectors are quickly removed so that the training time

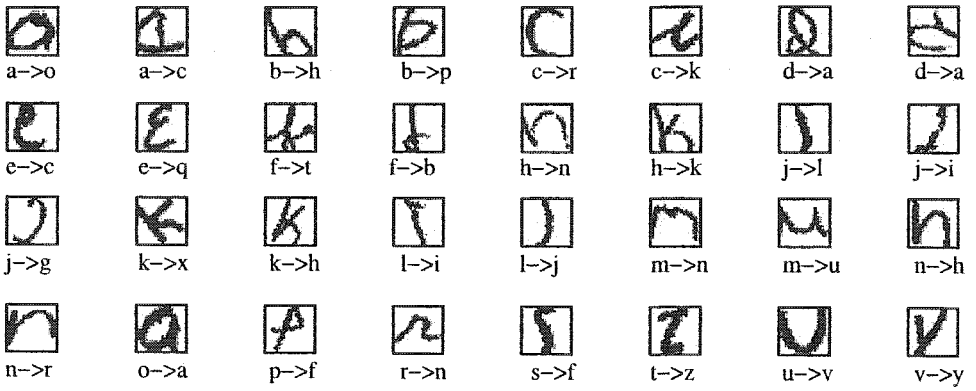

Figure 21: Some misclassified patterns on the test set of NIST database for handwritten lowercase characters. The first character on the left at the bottom specifies the true label. The character on the right is the recognized character label.

for sequential optimization can be reduced dramatically. In addition, some effective strategies, such as kernel caching and efficient computation of kernel matrix, are integrated to speed up the training process. Further, the space and runtime complexity of the proposed algorithm are analyzed and we show that its runtime complexity linearly scales with the number of classes and the size of the data set.

Extensive experiments have been conducted to study various appealing properties of the proposed algorithm. Compared with Libsvm and SVM^{light}, the proposed algorithm has a much higher training speed, especially on databases of a huge size with thousands of classes. In addition, we tested the generalization performance of feature-based SVM on several handwritten character databases trained by the proposed algorithm. The state-of-the-art performances have been achieved on these databases. Particularly on MNIST and Hanwang handwritten digit databases, very low error rates of 0.38% and 0.5% were obtained, respectively.

Table 11: Comparisons of generalization performances of different methods on MNIST database.

Methods	Raw error rate
LeNet1 [78]	1.7%
400-300-10 network [78]	1.6%
Polynomial SVM [13]	1.4%
Tangent Distance [78]	1.1%
LeNet5 [78]	0.9%
Virtual polynomial SVM [102]	0.8%
Boosted LeNet4 [78]	0.7%
SVM on vision-based feature [114]	0.59%
Virtual SVM with 2pix VSVs [24]	0.56%
Feature-based SVM	0.60%
VSVM^a	0.44%
SVC-rbf [83]	0.42%
VSVM^b	0.38%

Table 12: Rejection performances for VSVM^a under different ξ

ξ	Recognition rate	Substitution	Rejection rate	Reliability
1.1	96.69%	0.04%	3.27%	99.96%
1.4	94.41%	0.01%	5.58%	99.99%
1.7	91.51%	0	8.49%	100%
–	99.56%	0.44%	0	99.56%

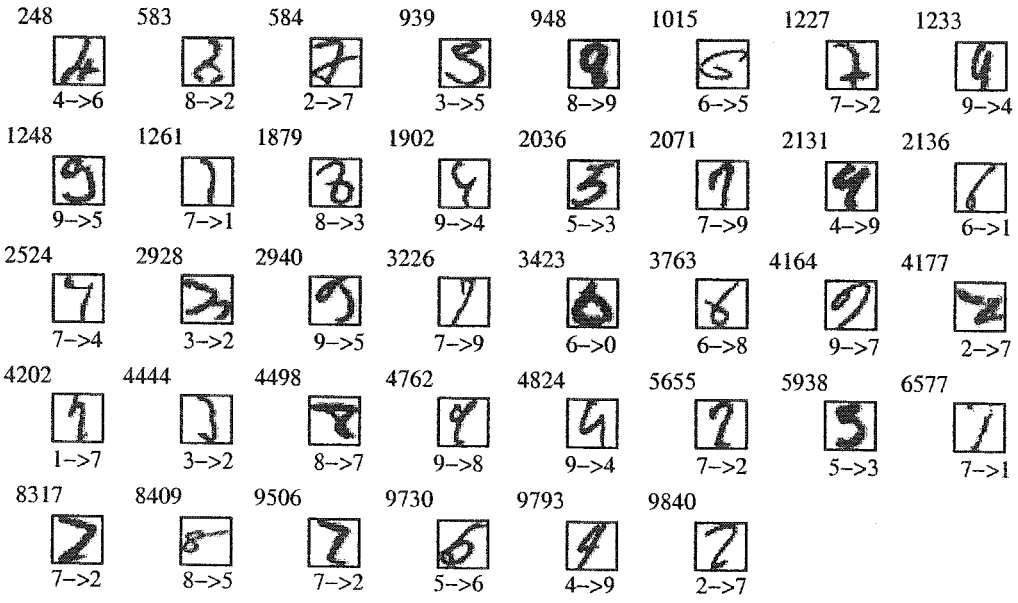

Figure 22: The 38 errors (0.38% error rate) for MNIST test set. The number in the upper-left corner of each image indicates the test sample number. The first digit on the bottom specifies the true label. The second digit on the bottom is the recognized digit label.

Chapter 6

Handwritten Chinese character recognition using SVM

6.1 Introduction

Important progress has been made in research on handwritten Chinese character recognition in the last decade [68], [62] and [11]. In recent years structural analysis approach [128] seems to have been overtaken by pattern matching methods involving extraction of statistical features in research on handwritten character recognition. The accuracy of an overall recognition system depends to a large extent on the discriminative capability of the extracted features and generalization performance of the classifier.

Handwritten Chinese characters have complex structures and large shape variations. Also, many similar patterns exist. Matching methods based on pixel values alone can not perform well, some preprocessing steps such as shape normalization and feature extraction are necessary. Nonlinear normalization [127] is a useful technique for correcting nonlinear shape variations and homogenizing the two-dimensional line density so that space can be more efficiently utilized and feature sampling points are stabilized for pattern matching methods. However, the original method operates on binary images. Nonlinear normalization may produce many jags which have a considerable impact on feature extraction. It is difficult to smooth these jags without destroying stroke information.

With respect to classifiers for the recognition of handwritten Chinese characters,

which have thousands of categories, many good classification techniques such as multi-layer perceptron can not be directly applied due to the complexity of modelling and "local minima problem". Learning vector quantization (LVQ) [122], which generates some representative reference patterns as templates, and modified quadratic discriminant function (MQDF) [66],[68] are two main classification methods for handwritten Chinese characters since both have achieved good performance. Although LVQ is a supervised learning method, its decision boundaries are pairwise hyperplanes. When the real class boundary is nonlinear, a large bias will be generated. MQDF assumes that data in each class is distributed unimodally in a gaussian-like manner. The maximal likelihood approach is applied to generate the density model for each class and then the Bayesian decision rule is used for classification. This method, which does not focus directly on the goal of classification, leads to a sub-optimal rule since the estimation of class density is usually a hard and ill-posed problem without sufficient prior knowledge.

The purpose of this chapter is to improve the accuracy of handwritten Chinese character recognition by using an enhanced nonlinear normalization method and support vector machine (SVM). Nonlinear normalization is applied to gray-scale images, and jags can be removed while preserving the edge information. Support vector machine is usually applied to classification problems with a small number of categories such as handwritten digit recognition [24] due to the lack of an efficient algorithm for training SVM on a large data set with thousands of classes. Recently this problem has been solved by our proposed algorithm [32],[34] so that we can test the performance of SVM in more real-world challenging classification applications.

This chapter is organized as follows. Section 2 describes the improved nonlinear shape normalization. Then the feature extraction method is given in Section 3. Section 4 presents a method to tune kernel parameters for support vector machine on a large data set. The experimental results are described in Section 5. Finally, we summarize the paper and draw conclusions.

6.2 Improved nonlinear normalization

Lee and Park [80] compared the performance and computational complexity of existing nonlinear normalization methods and experimental results indicated that Yamada

et al's method based on line density [127] performs better while having a higher computational cost and side effects of jags. In fact, run-length coding of images can speed up the calculation of line-density histograms and reduce costs [82]. Kim et al. [65] extended Tsukumo et al's method [116] from a binary image to a gray-scale image in order to partially alleviate the effect of jags. But Tsukumo et al's method defines the line density in one dimension while Yamada et al's method works in two dimensions. As a result, the overall performance of Tsukumo et al's is not as good as Yamada et al's. In order to effectively remove jags after nonlinear normalization, we can use modified Yamada et al's method on binary image to determine the position of sampling points and produce the gray-scale image. Noise filtering and edge smoothing are much easier on a gray-scale image than on a binary image. The diagram of the nonlinear normalization scheme is shown in Fig. 23.

In Fig. 23, median filtering is used to remove nonlinear noise such as salt-and-pepper noise, which often appears in binary character images. Smoothing operator is a linear lowpass filter for Gaussian noise. Linear normalization is executed in the form of backward mapping from normalized image to input image in order to avoid the holes and gaps in the normalized images. Also, the aspect ratio is preserved so that the global character shape is not changed. The size of linear normalization (128×128) is usually larger than the original one. The rationale is that the increased gaps between character strokes will be helpful for noise removal since most Chinese characters have complex structures and frequent two-pixel gap between strokes in the original image makes noise removal difficult. The gray-scale image is just a copy of a binary image. But it is represented in float data type.

Regarding nonlinear normalization, Yamada et al's method [127] is applied to binary image for calculating the histogram of line density and determining the position of a sampling point. Then the gray-scale image is produced. But in Yamada et al's method, the pixel value of the output image is attributed from one pixel in the original image. Due to the discrete nature of input and output images, many sampling points of the input image are mapped to the same point in the output image. It is more reasonable to consider all contributions from these input sampling points. To describe precisely the proposed method, we introduce some notations. Let $f(i, j)$ be an input image, whose size is $I \times J$, $i = 1, \ldots, I$ and $j = 1, \ldots, J$. Let $g(m, n)$ be the normalized image, whose size is $M \times N$, $m = 1, \ldots, M$ and $n = 1, \ldots, N$. Let $\rho(i, j)$ be the line

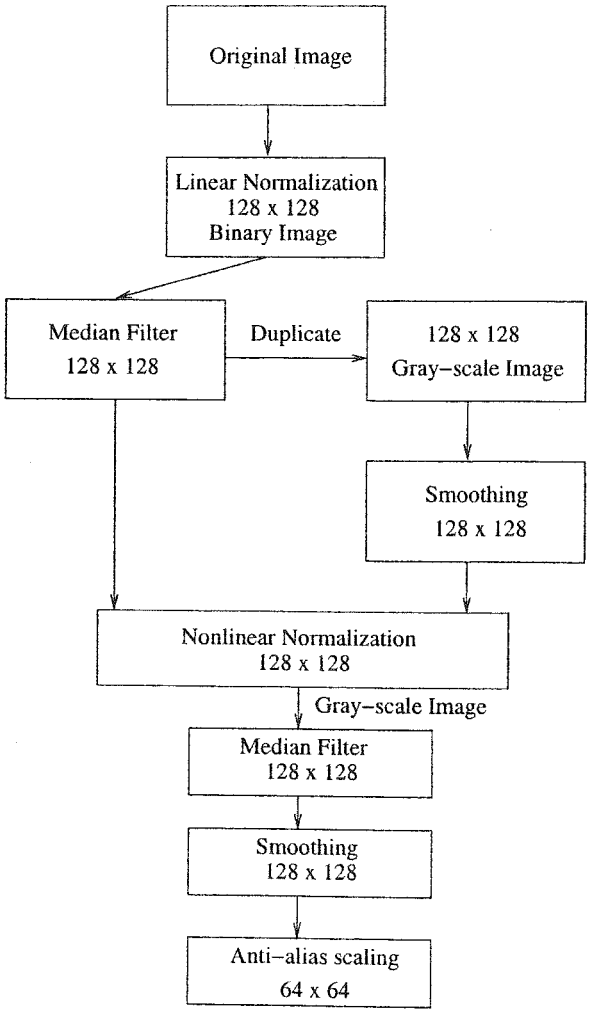

Figure 23: Diagram of nonlinear normalization

density calculated from the input binary image by Yamada et al's method. The feature projection functions on the x and y axes are defined as

$$\begin{aligned} H(i) &= \sum_{j=1}^{J} \rho(i,j), \\ V(j) &= \sum_{i=1}^{I} \rho(i,j). \end{aligned} \tag{104}$$

The forward function, which maps position (i,j) on the input image to position (m,n) on the normalized image, is given by

$$m = \phi(i)$$

$$= \lfloor \sum_{k=1}^{i} H(k) \times \frac{M}{\sum_{k=1}^{I} H(k)} + 0.5 \rfloor,$$

$$n = \psi(j)$$

$$= \lfloor \sum_{l=1}^{j} V(l) \times \frac{N}{\sum_{l=1}^{J} V(l)} + 0.5 \rfloor. \quad (105)$$

where $\lfloor A \rfloor$ is the maximal integer not larger than A (floor of A). Define sets $S_m = \{i | \phi(i) = m\}$, where $m = 1, \ldots, M$. Let n_m be the number of elements in set S_m. In order to assure the pixel value of all positions in the normalized image, a backward mapping is required. Fig. 24 shows the backward mapping for index m.

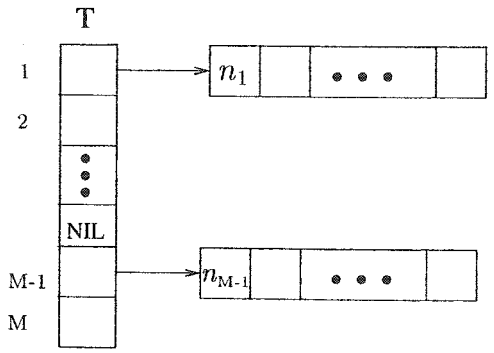

Figure 24: Backward mapping

In Fig. 24, T is an array of length M. Element $T[i]$ is a pointer to the starting address of an array that contains the size of set S_i and its elements. NIL means the empty pointer. That is, no input index i is mapped here. To calculate all pixel values in the normalized image, each element in T must be a non-empty pointer. The following procedure assigns a new value to the NIL pointer using the nearest neighbor principle.

Assign a new value to the NIL pointer

Input: $T[i]$, $i = 1, \ldots, M$, where $T[M] \neq$ NIL

Output: $T[i]$, where $T[i] \neq$ NIL, $i = 1, \ldots, M$

$k \leftarrow M$.

Repeat

IF $T[k] =$ NIL

$T[k] = T[k+1]$

END IF

$k \leftarrow k - 1$.

Until $k = 0$.

Similarly, we can determine the backward mapping for index n. After that, each pixel in the normalized image can have its corresponding pixels in the input gray-scale image. If multiple pixels in the input image are mapped to the same pixel in the normalized image, the maximal value among these source pixels is assigned to the destination pixel. In fact, this operation can also be regarded as a nonlinear filter, which reduces the effects of jags considerably. In the last operation, anti-alias scaling method [126] is applied and a gray-scale image of size 64×64 is obtained.

6.3 Feature Extraction

Most Chinese characters consist of line strokes so that information based on the edge direction is important for feature extraction. Kimura et al. [68] proposed a weighted directional code feature, which calculates the directional histogram of chain codes in a local region. Similarly, Sun et al. [113] define 12 directional patterns using 3×3 masks and determine the edge orientation according to them, rather than chain code. However, both methods are just suitable for feature extraction in a binary image. For a gray-scale image, we calculate directional histograms based on image gradients. The procedure for feature extraction is given as follows:

- The gray-scale normalized image is standardized such that its mean and maximum values are 0 and 1.0, respectively [46].
- Center a 64×64 normalized image into an 80×80 box in order to efficiently utilize the information in the four peripheral areas[1]
- Robert edge operator [92] is applied to calculate gradient strengths and directions. For example, the gradient magnitude and direction of pixel $g(m, n)$ are calculated as follows:

$$\Delta u = g(m, n) - g(m + 1, n + 1),$$

[1]Peripheral strokes for Chinese characters usually contain more stable information than the central strokes.

$$\begin{aligned} \Delta v &= g(m, n+1) - g(m+1, n), \\ \theta(m, n) &= \arctan(\frac{\Delta v}{\Delta u}), \\ s(m, n) &= \sqrt{\Delta u^2 + \Delta v^2}. \end{aligned} \tag{106}$$

where $\theta(m, n)$ and $s(m, n)$ specify the direction and gradient magnitude of pixel (m, n), respectively.

- Using a similar strategy as in [113],[62], we divide the image into 9×9 subareas of size 16×16, where each subarea overlaps with eight pixels of the adjacent subareas. Each subarea is divided into four parts: A, B, C and D (see Fig. 25). A is a 4×4 area at the center. B is a 8×8 area exclusive of area A. C is a 12×12 area exclusive of area B. D is a 16×16 area exclusive of area C.

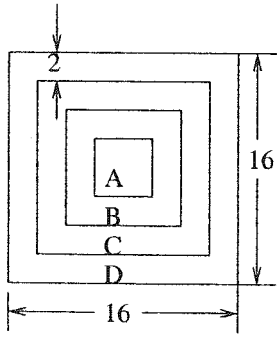

Figure 25: 16×16 sub-area.

- In each area of A,B,C,D, the strengths of gradients with each of 32 quantized gradient directions are accumulated. In order to reduce side effects of position variations, a mask (4, 3, 2, 1) is used to down-sample the accumulated gradient strengths of each direction. As a result, a 32 dimensional feature vector is obtained.

- After gradient strengths on 32 quantized directions is generated, the directional resolution is reduced from 32 to 16 by down-sampling using one-dimensional mask (1,4,6,4,1) [46].

- A feature vector of size 1296 (9 horizontal, 9 vertical and 16 directional resolutions) is generated.

- The variable transformation $x^{0.4}$ is applied to each component of the feature vector such that the distribution of each component is normal-like [48].
- Scale the feature vector by a constant factor such that values of feature components range from 0 and 1.0.

In order to save the processing time and storage cost, multiple discriminant analysis is applied to reduce the dimension [41].

6.4 Parameter selection for support vector machine

Given training vectors $\mathbf{x}_i \in \mathbb{R}^n$, $i = 1, \ldots, l$ and a vector $y \in \mathbb{R}^l$ such that $y_i \in \{-1, 1\}$, training a support vector machine is to find α, which can be obtained by solving the following convex quadratic optimization problem:

$$\begin{aligned} \text{Maximize} \quad & \mathbf{e}^T\alpha - \tfrac{1}{2}\alpha^T Q\alpha \\ \text{subject to} \quad & 0 \le \alpha_i \le C, \;\; i = 1, \ldots, l \\ & \mathbf{y}^T\alpha = 0 \end{aligned} \tag{107}$$

where $\mathbf{e} \in \mathbb{R}^l$ is a vector whose components are one, Q is an $l \times l$ semi-positive definite kernel matrix with elements Q_{ij}, where $Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, $i, j = 1, \ldots, l$, and kernel function K satisfies the Mercer conditions [85].

Good selection of C and kernel parameters plays an important role in the good generalization performance of SVM. One of the simple and efficient methods is to use a cross validation set to tune these parameters. For a large data set with thousands of classes, it is not effective to train SVM on the training set with all classes and evaluate its performance on the validation set due to high computational cost. Thus, a good selection of a training subset and a validation set with a small number of classes is required. Our basic idea is to select these classes such that the cost for selection procedure is not high and patterns in these classes are similar or easily misclassified. The proposed method is given below:

1. Group the training data in terms of classes.
2. Divide training data of each class into two parts: 75 percent for training and the rest for validation.

3. Use discriminant function with a low computation cost for each class. For instance, we can use a Euclidean discriminant function

$$d_i(\mathbf{x}) = \| \mathbf{x} - \mu_i \|^2, \quad i = 1, \ldots, c. \tag{108}$$

where μ_i is the mean vector of class i and c is the number of classes.

4. Evaluate the performance of discriminant functions on the validation set and calculate the confusion matrix defined by

$$\text{CF} = \begin{pmatrix} n_{11} & n_{12} & \cdots & n_{1c} \\ n_{21} & n_{22} & \cdots & n_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ n_{c1} & n_{c2} & \cdots & n_{cc} \end{pmatrix} \tag{109}$$

where each row i corresponds to class $\mathcal{C}_i$ and each column j means the number of patterns classified to class $\mathcal{C}_j$.

5. Calculate the number of misclassified patterns for each class according to

$$er_i = \sum_{j=1, j \neq i}^{c} n_{ij}. \tag{110}$$

6. Sort er_i, $i = 1, \ldots, c$ in a decreasing order and obtain the subscripts $i_1, \ldots, i_I$ of the top $I (\ll c)$ choices.

7. Find the classes whose patterns can be classified to the class set $\{\mathcal{C}_{i_1}, \mathcal{C}_{i_2}, \ldots, \mathcal{C}_{i_I}\}$ according to the confusion matrix

$$\mathcal{S} = \bigcup_{k=1}^{I} \{\mathcal{C}_j | n_{i_k j} \neq 0\}. \tag{111}$$

8. Choose the training set and cross validation set from class set $\mathcal{S}$ for support vector machine to tune C and kernel parameters.

After C and kernel parameters are obtained, we train the support vector machine on the whole training set with all classes. The training algorithm is described in [32],[34].

6.5 Experiments

Experiments were performed on P4 with Windows 2000 Professional system equipped with 1.5 Gigabytes RAM. The source code for SVM algorithm [34] was written in C++ and compiled by Microsoft visual C++ 6.0 [2].

In our experiments we used ETL9B database created by Electrotechnical Laboratory of Japan. It contains 200 samples for each of 3036 categories, 2965 Chinese and 71 Hiragana characters. All samples are binary images of size 64 (width) by 63 (height). The samples on this database are divided into 5 sets (A to E), each with 40 characters per category. In our experiment, sets A through D are used for training and set E for testing. As a result, the sizes of training and testing set are 485760 and 121440, respectively.

For support vector machine, the one-against-others method was used to construct 3036 classifiers. That is, each classifier was constructed by separating one class from the rest. The classification decision was made by choosing the class with the largest classifier output value.

In the following, several experiments have been done to investigate the performance of improved nonlinear normalized method, feature extraction and support vector machines.

In our experiments, some parameters are set manually. In Fig. 23, the mask for smoothing operation is given by

$$\frac{1}{16}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{112}$$

The constant factor in the feature extraction is set to 0.05 and the size of 1296-dimensional feature vector is reduced to 392 when discriminant analysis is applied.

6.5.1 Nonlinear Normalization

In this part, we visually compare the performance of Yamada et al's nonlinear normalization method and our method. Fig. 26 shows the normalized images. From this figure, it can be observed that the proposed method successfully fixed the jag problem. Moreover, some strokes in peripheral regions are distorted in Yamada et al.'s

[2] http://www.cenparmi.concordia.ca/~people/jdong/HeroSvm.html.

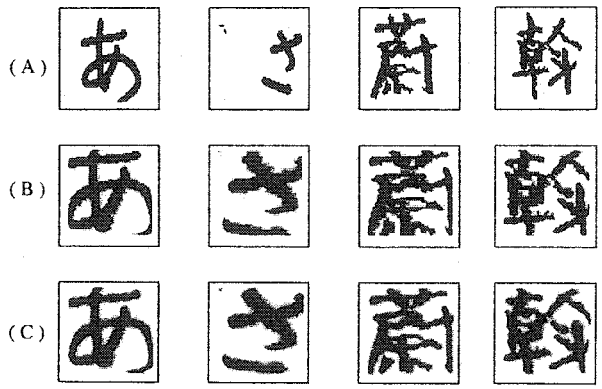

Figure 26: (A) Original images, (B) Normalized images using Yamada et al's, (C) Normalized images using the proposed method.

method. This side-effect was also observed by Liu et al.[82]. They tried to fix this problem by increasing the line density in the peripheral regions by means of a hybrid definition of Tsukumo et al.'s [116] and Yamada et al's. But this modified definition did not perform well in our experiments. Fig. 26.C shows that the improved method removes this side-effect completely.

6.5.2 Coarse classification

There are 3036 classes for ETL9B. In order to reduce the computational cost and speed up the classification, a pre-classifier is required. The goal of a pre-classifier is to obtain a high cumulative recognition rate so that a small number of selected candidates include the true class label. The performance of the following three pre-classifiers are investigated

- Discriminant function based on city-block distance

$$d_i(\mathbf{x}) = -\sum_{j=1}^{n} |\mathbf{x}_j - \mu_{ij}|. \tag{113}$$

- Discriminant function based on Euclidean distance

$$d_i(\mathbf{x}) = -\parallel \mathbf{x} - \mu_i \parallel^2 . \tag{114}$$

- Discriminant function for the normal density[3]

$$d_i(\mathbf{x}) = (\mathbf{S}_w^{-1}\mu_i)^T\mathbf{x} - \frac{1}{2}\mu_i^T\mathbf{S}_w^{-1}\mu_i \tag{115}$$

  where $\mathbf{S}_w$ is the within-class scatter matrix.

- Support vector machine with a linear kernel

$$d_i(\mathbf{x}) = \mathbf{w}_i^T x + b_i \tag{116}$$

where $i = 1, \ldots, c$ and μ_i is the mean vector of class C_i. Figure 27 shows the cumulative recognition rates of these pre-classifiers.

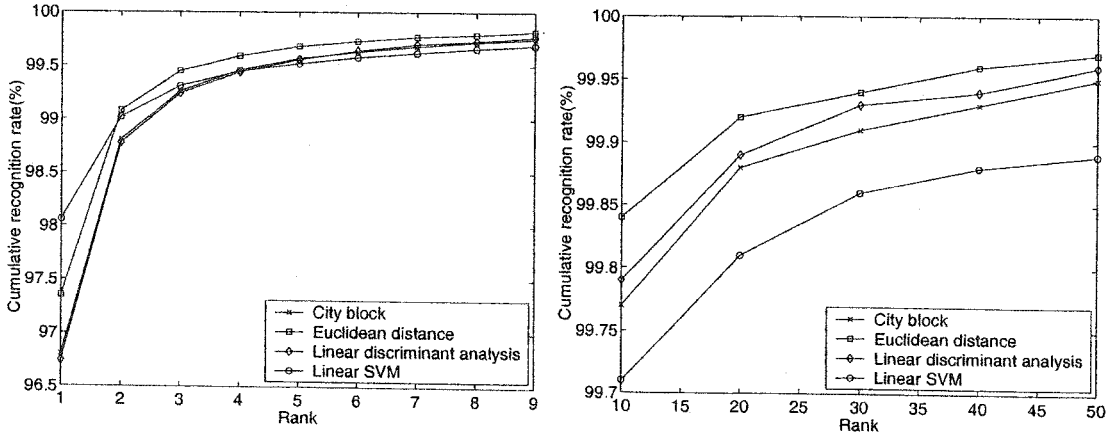

Figure 27: Comparisons of cumulative recognition rates

Although linear SVM performs best on the rank-1 recognition accuracy, its cumulative recognition rate degrades quickly, compared with other classifiers. It is surprising that linear SVM performs worse than linear discriminant analysis. One possible reason is that linear SVM is trained using one-against-the-rest strategy and the same desired output (-1.0) is imposed on the sample which does not belong to the current class. Therefore, under this strategy SVM only focused on the decision boundary without differentiating the patterns in other classes. Linear discriminant analysis is derived from the assumption of normal density for each class and takes into account the difference. This result indicates that linear SVM is not suitable for

[3]Assume that the covariance matrices and prior probability of each class are close to each other.

pre-classification and the method by which SVM generates outputs of *posterior* probability needs to be developed. Among the above pre-classifiers, discriminant function based on Euclidean distance is the best choice and its cumulative recognition rate at rank 20 is 99.92%.

6.5.3 Performance of support vector machine

After pre-classification, support vector machine with RBF kernel is used as the main classifier to make the final decision from the 20 candidates obtained by Euclidean discriminant function. The RBF kernel is given by

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{\| \mathbf{x} - \mathbf{x}' \|^2}{2\sigma^2}). \tag{117}$$

We use the proposed method in Section 4 to determine an optimal parameter for σ^2 and C in eq. (107). In the method, we use Euclidean discriminant function and set the parameter I to 20. After the method is applied, the cardinal number of set S is 144. As a result, the sizes of training and validation sets for tuning parameters C and σ^2 are 21600 (=144 × 200 × 75%) and 7200 (=144 × 200 × 25%), respectively. The parameter σ^2 is set to 0.8 and C to 10.0 when the highest performance on validation set has been achieved.

On the training set, each class consists of only 160 samples, which is not sufficient to achieve high performance when SVM with RBF kernel is used. In order to increase the number of training samples, we shift each training image in four directions (left, right, down, up) by one pixel. The size of the shifted training set is 2,428,000 (= 485760 × 5). Table 13 shows the recognition accuracy of different methods on ETL9B.

Table 13: Recognition accuracy of different methods on ETL9B

Classifiers	Size of training set	Substitution error (%)
Nearest neighbor	485760	2.9%
SVM[a]	485760	1.1%
SVM[b]	2,428,000	1.0%
MQDF [68]	576840	0.95%
Improved MQDF [117]	546480	0.59%
Asymmetric Mahalanobis distance [62]	546480	0.58%

It can be seen from Table 13 that SVM^b on the shifted training set just slightly improves the performance while sacrificing the computational speed since the average number of support vectors for SVM^b is 587, which is larger than that for SVM^a (312). For other groups' results, the error rate was averaged on several testing sets using the rotation method. In addition, we can observe that there is a gap between SVM's performance and the best one. But SVM has potential to fill this gap in the future. Some misclassified patterns for the proposed method are shown in Fig. 28. We can see that most patterns are misclassified to those with the similar shape.

Figure 28: A part of misclassified patterns on ETL test set. The first character on the bottom specifies the original label. The second character is the recognized label.

Although a high recognition rate on ETL9B has been reported, techniques for off-line handwritten Chinese character recognition are far from mature. In a recent survey, Suen et al. [112] showed that on larger real databases the recognition accuracy was still low. Therefore, we tested SVM on Hanwang handwritten digit database mentioned in Chapter 5 by following the same procedures except for adding virtual patterns. The recognition rate on its testing set was just 96.97%.

6.6 Conclusion

In this chapter we propose a method to improve Yamada et al's [127] nonlinear normalization scheme so that the jags effects and stroke distortion in the peripheral region in the normalized image can be removed. In addition, a method is presented to generate the good subsets with a small number of classes for tuning SVM's kernel parameters when SVM is applied to solve a large classification problem on a large data set with thousands of classes. Several experiments were conducted on ETL9B handwritten Chinese database and the recognition accuracy of 99% has been achieved.

Chapter 7

Fast SVM Testing Algorithms

7.1 Introduction

Support vector machines (SVMs) have achieved excellent performance in many applications such as handwritten digit recognition [98] and face detection [87]. One drawback of SVM is that its classification speed is very slow during the testing stage because it is proportional to the number of support vectors. On a large data set many support vectors will be generated since Vapnik [121] has shown that the recognition error rate is bounded by the ratio between the number of support vectors and the size of the training set. The low classification speed makes SVM less competitive in tasks where both accuracy and speed are required. Hence, it is important to develop a fast algorithm to speed up SVM's classification.

Much research has been done to speed up SVM's classification process. Basically these works can be divided into two categories. One is to reformulate the SVM training problem and yield a good accuracy with a small number of support vectors. Osuna et al. [88] added some terms into the cost function in order to enhance the sparsity of SVM's solution. The reduction of run-time is in the 50-95% range. The other is the reduced set method [12][13][101][103] which approximates the decision function with a much smaller number of reduced set vectors. Experiments on MNIST handwritten digit database have shown a factor of about fifty improvement in speed without sacrificing the accuracy of the original virtual SVM solution [13]. The current unconstrained conjugate gradient method for finding the reduced set vectors is very expensive computationally, because it has to find a solution in a space of a

huge number of variables [13]. In [101] an iteration procedure for computing fast approximations of the Gaussian kernel expansions is presented. Although the iteration method is attractive, numerical instabilities often occur and the computational cost is still high. In addition, due to the prohibitive cost, the experiment was just conducted to approximate the SVM solution of each class separately.

In this chapter we extended the iteration method in [101] to the general kernel and designed a fast algorithm to approximate the reduced set vectors shared by each binary SVM solution for multi-class classification. Experimental results on MNIST database and Hanwang handwritten digit database with RBF kernel were very promising, *i.e.* about 16,000 and 10,895 patterns per second on MNIST and Hanwang databases, respectively.

This chapter is organized as follows. The mathematical formula is first derived to approximate the reduced set vector simultaneously. In Section 3, an efficient algorithm is designed to find these reduced set vectors. Section 4 presents a fast block algorithm in the test phase. Experimental results are given in Section 5. Finally we summarize this chapter with some concluding remarks.

7.2 Simultaneous approximation of reduced set vectors

In support vector machines the training data $\{\mathbf{x}_i\}$, $i = 1, \ldots, l$, $\mathbf{x}_i \in \mathcal{X} \subset \mathbb{R}^n$ are mapped into a dot product space $\mathcal{H}$ with a map $\Phi : \mathcal{X} \mapsto \mathcal{H}$ such that the mercer kernel [85] $k(\mathbf{x}, \mathbf{x}') =< \Phi(\mathbf{x}), \Phi(\mathbf{x}') >$. The bold font is used to denote a column vector. SVM [5] finds a linear hyperplane to separate the data in the space $\mathcal{H}$, whose solution is expressed in terms of kernel expansion, given below

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w}^T\Phi(\mathbf{x}) + b \\ &= \sum_{i=1}^{l} \beta_i k(\mathbf{x}, \mathbf{x}_i) + b \end{aligned} \tag{118}$$

where $\mathbf{x}_i$ whose corresponding β_i is nonzero is called support vector and $\mathbf{w} = \sum_{i=1}^{l} \beta_i \Phi(\mathbf{x}_i)$.

For multi-class classification, the one-against-others strategy is used to train SVM so that there is a corresponding binary SVM for each class. Suppose that their decision functions are $f_m(\mathbf{x}) = \Phi(\mathbf{x})^T\mathbf{w}^m + b_m = \sum_{i=1}^{N^m} \beta_i^m k(\mathbf{x}, \mathbf{x}_i^m) + b_m$, where M is

the number of classes, $\mathbf{x}_i^m$ and N^m is the support vector and the number of support vectors for class m, respectively. The final decision rule is given by

$$m^* = arg\max_m f_m(\mathbf{x}) \tag{119}$$

Let $U = \bigcup_{m=1}^{M}\{\mathbf{x}_i^m | i = 1, \ldots, N^m\}$ be a set of collection of support vectors from M classes and $|U|$ be the cardinal number of set U. Then $\mathbf{F} = \{\Phi(\mathbf{x})|\mathbf{x} \in U\}$ live in a subspace span$(\mathbf{F}) \subseteq \mathcal{H}$. If the dimension of the subspace span$(\mathbf{F})$ is much smaller than $|U|$, SVM solution for each class can be re-written equivalently in terms of kernel expansion which consists of dim(span$(\mathbf{F})$) vectors. Unfortunately, since the matrix $k(\mathbf{x}, \mathbf{x}')$, $\mathbf{x} \in U$ and $\mathbf{x}' \in U$, is positive definitive, the dimension of the subspace span$(\mathbf{F})$ is equal to $|U|$ [104]. Therefore, instead of the equivalent representation, we can seek a subspace with a much lower dimension and project $\mathbf{w}^m$ onto the subspace. Let us suppose that $\{\Phi(\mathbf{z}_i)|\mathbf{z}_i \in \mathbb{R}^n\}$, $i = 1, \ldots, L$ and $L < |U|$, span this subspace. In order to find the basis vectors and projections, we can minimize the following unconstrained optimization problem.

$$\text{Minimize } E = \sum_{m=1}^{M} \| \sum_{i=1}^{L} \gamma_i^m \Phi(\mathbf{z}_i) - \sum_{i=1}^{N^m} \beta_i^m \Phi(\mathbf{x}_i^m) \|^2 \tag{120}$$

Let vector $\beta^{\mathbf{m}} = (\beta_1^m, \ldots, \beta_{N^m}^m)^T$, matrix $\mathbf{A} = (\Phi(\mathbf{z}_1), \ldots, \Phi(\mathbf{z}_L))$, matrix $\mathbf{B}^m = (\Phi(\mathbf{x}_1^m), \ldots, \Phi(\mathbf{x}_{N^m}^m))$ and vector $\gamma^m = (\gamma_1^m, \ldots, \gamma_L^m)^T$. The optimization problem (120) can be rewritten as

$$\text{Minimize } E = \sum_{m=1}^{M} \| \mathbf{A}\gamma^m - \mathbf{B}^m\beta^m \|^2 \tag{121}$$

For the fixed basis vectors $\Phi(\mathbf{z}_i)$, $i = 1, \ldots, L$, the optimal coefficients γ^m can be computed using the stationary condition:

$$\nabla_{\gamma^m} E = 0 \tag{122}$$

So we can yield $2A^T(\mathbf{A}\gamma^m - \mathbf{B}^m\beta^m) = 0$, then $\gamma^m = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{B}^m\beta^m$. We can see that $(\mathbf{A}^T\mathbf{A})_{ij} = k(\mathbf{z}_i, \mathbf{z}_j)$, where $i, j = 1, \ldots, L$ and $(\mathbf{A}^T\mathbf{B}^m)_{ij} = k(\mathbf{z}_i, \mathbf{x}_j^m)$, $i = 1, \ldots, L$ and $j = 1, \ldots, N^m$.

In order to find the optimal $\mathbf{z}_i$, $i = 1, \ldots, L$, we use the strategy in [101]: sequentially optimize one vector $\mathbf{z}$ while fixing other vectors. Without loss of generality,

$\mathbf{z}_1$ is chosen for the optimization. Let $\Psi^m = \sum_{i=1}^{N^m} \beta_i^m \Phi(\mathbf{x}_i^m) - \sum_{i=2}^{L} \gamma_i^m \Phi(\mathbf{z}_i)$. The optimal $\gamma_1^m, m = 1, \ldots, M$ can be computed by

$$\frac{\partial E}{\partial \gamma_1^m} = 0 \tag{123}$$

Then $\gamma_1^m = \frac{<\Psi^m, \Phi(\mathbf{z}_1>}{k(\mathbf{z}_1, \mathbf{z}_1)}$, $m = 1, \ldots, M$. Substitute γ_1^m into (120) and we can obtain

$$\text{Minimize } E' = \sum_{m=1}^{M} (\| \Psi^m \|^2 - \frac{< \Psi^m, \Phi(\mathbf{z}_1) >^2}{k(\mathbf{z}_1, \mathbf{z}_1)}) \tag{124}$$

which is equivalent to maximizing $\sum_{m=1}^{M} \frac{<\Psi^m, \Phi(\mathbf{z}_1)>^2}{k(\mathbf{z}_1, \mathbf{z}_1)}$. The stationary condition is given by

$$\nabla_{\mathbf{z}_1} \sum_{m=1}^{M} \frac{< \Psi^m, \Phi(\mathbf{z}_1) >^2}{k(\mathbf{z}_1, \mathbf{z}_1)} = 0 \tag{125}$$

For a kernel type $k(\mathbf{x}, \mathbf{x}') = k(\| \mathbf{x} - \mathbf{x}' \|^2)$, we thus obtain

$$\begin{aligned}
g_1(\mathbf{z}_1) &= \sum_{m=1}^{M} ((\sum_{i=1}^{N^m} \beta_i^m k(\| \mathbf{x}_i^m - \mathbf{z}_1 \|^2) \\
&\quad - \sum_{i=2}^{L} \gamma_i^m k(\| \mathbf{z}_i - \mathbf{z}_1 \|^2)) \\
&\quad \times (\sum_{j=1}^{N^m} \beta_j^m k'(\| \mathbf{x}_j^m - \mathbf{z}_1 \|^2) \mathbf{x}_j^m \\
&\quad - \sum_{j=2}^{L} \gamma_j^m k'(\| \mathbf{z}_j - \mathbf{z}_1 \|^2) \mathbf{z}_j)), \\
g_2(\mathbf{z}_1) &= \sum_{m=1}^{M} ((\sum_{i=1}^{N^m} \beta_i^m k(\| \mathbf{x}_i^m - \mathbf{z}_1 \|^2) \\
&\quad - \sum_{i=2}^{L} \gamma_i^m k(\| \mathbf{z}_i - \mathbf{z}_1 \|^2)) \\
&\quad \times (\sum_{j=1}^{N^m} \beta_j^m k'(\| \mathbf{x}_j^m - \mathbf{z}_1 \|^2) \\
&\quad - \sum_{j=2}^{L} \gamma_j^m k'(\| \mathbf{z}_j - \mathbf{z}_1 \|^2))), \\
\mathbf{z}_1 &= \frac{g_1(\mathbf{z}_1)}{g_2(\mathbf{z}_1)}.
\end{aligned} \tag{126}$$

For a kernel type $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}^T\mathbf{x}')$, we can obtain

$$\begin{aligned}
g_1(\mathbf{z}_1) &= k(\| \mathbf{z}_1 \|^2) \sum_{m=1}^{M}((\sum_{i=1}^{N^m} \beta_i^m k(\mathbf{z}_1^T \mathbf{x}_i^m) \\
&\quad - \sum_{i=2}^{L} \gamma_i^m k(\mathbf{z}_1^T \mathbf{z}_i)) \\
&\quad \times(\sum_{j=1}^{N^m} \beta_j^m k'(\mathbf{z}_1^T \mathbf{x}_j^m)\mathbf{x}_j^m \\
&\quad - \sum_{j=2}^{L} \gamma_j^m k'(\mathbf{z}_1^T \mathbf{z}_j)\mathbf{z}_j)), \\
g_2(\mathbf{z}_1) &= k'(\| \mathbf{z}_1 \|^2) \sum_{m=1}^{M}(\sum_{i=1}^{N^m} \beta_i^m k(\mathbf{z}_1^T \mathbf{x}_i^m) \\
&\quad - \sum_{i=2}^{L} \gamma_i^m k(\mathbf{z}_1^T \mathbf{z}_i))^2, \\
\mathbf{z}_1 &= \frac{g_1(\mathbf{z}_1)}{g_2(\mathbf{z}_1)}.
\end{aligned} \tag{127}$$

After that, we use an iteration to update $\mathbf{z}_1$

$$\mathbf{z}_1^{t+1} = \frac{g_1(\mathbf{z}_1^t)}{g_2(\mathbf{z}_1^t)} \tag{128}$$

where t denotes the iteration step. Similarly, we can update $\mathbf{z}_i$, $i = 2, \ldots, L$. For example, for radial basis function (RBF) kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2})$, the iteration formula for $\mathbf{z}_p$, $p = 1, \ldots, L$ is given by

$$\begin{aligned}
g_1(\mathbf{z}_p^t) &= \sum_{m=1}^{M}((\sum_{i=1}^{N^m} \beta_i^m \exp(-\frac{\| \mathbf{x}_i^m - \mathbf{z}_p^t \|^2}{2\sigma^2}) \\
&\quad - \sum_{i=1}^{L} \gamma_i^m \exp(-\frac{\| \mathbf{z}_i - \mathbf{z}_p^t \|^2}{2\sigma^2}) + \gamma_p^m) \\
&\quad \times(\sum_{j=1}^{N^m} \beta_j^m \exp(-\frac{\| \mathbf{x}_j^m - \mathbf{z}_p^t \|^2}{2\sigma^2})\mathbf{x}_j^m \\
&\quad - \sum_{j=1}^{L} \gamma_j^m \exp(-\frac{\| \mathbf{z}_j - \mathbf{z}_p^t \|^2}{2\sigma^2})\mathbf{z}_j + \gamma_p^m \mathbf{z}_p^t)), \\
g_2(\mathbf{z}_p^t) &= \sum_{m=1}^{M}((\sum_{i=1}^{N^m} \beta_i^m \exp(-\frac{\| \mathbf{x}_i^m - \mathbf{z}_p^t \|^2}{2\sigma^2})
\end{aligned}$$

$$
\begin{aligned}
& -\sum_{i=1}^{L} \gamma_i^m \exp(-\frac{\| \mathbf{z}_i - \mathbf{z}_p^t \|^2}{2\sigma^2}) + \gamma_p^m) \\
& \times (\sum_{j=1}^{N^m} \beta_j^m \exp(-\frac{\| \mathbf{x}_j^m - \mathbf{z}_p^t \|^2}{2\sigma^2}) \\
& -\sum_{j=1}^{L} \gamma_j^m \exp(-\frac{\| \mathbf{z}_j - \mathbf{z}_p^t \|^2}{2\sigma^2}) + \gamma_p^m)) \\
= & \sum_{m=1}^{M} (\sum_{i=1}^{N^m} \beta_i^m \exp(-\frac{\| \mathbf{x}_i^m - \mathbf{z}_p^t \|^2}{2\sigma^2}) \\
& -\sum_{i=1}^{L} \gamma_i^m \exp(-\frac{\| \mathbf{z}_i - \mathbf{z}_p^t \|^2}{2\sigma^2}) \\
& +\gamma_p^m)^2, \\
\mathbf{z}_p^{t+1} = & \frac{g_1(\mathbf{z}_p^t)}{g_2(\mathbf{z}_p^t)}.
\end{aligned} \tag{129}
$$

7.3 Fast algorithm for simultaneous approximation

For the sake of simplicity, we just use RBF kernel to illustrate the algorithm. Since the iteration formulae (126) and (127) show similar structures, the following can be easily adapted to other kernel types such as polynomial kernel. Before the algorithm is presented, some notations are defined below:

- $\mathbf{K}^{\mathbf{z}}$: an $L \times L$ square matrix and $(\mathbf{K}^{\mathbf{z}})_{ij} = k(\mathbf{z}_i, \mathbf{z}_j)$
- $\mathbf{K}^{\mathbf{zx}}$: an $L \times |U|$ matrix and $(\mathbf{K}^{\mathbf{zx}})_{ij} = k(\mathbf{z}_i, \mathbf{x}_j)$, $\mathbf{x}_j \in U$, $i = 1, \ldots, L$ and $j = 1, \ldots, |U|$
- $\mathbf{K}^{\mathbf{zx}^m}$: $L \times N^m$ matrix and $(\mathbf{K}^{\mathbf{zx}^m})_{ij} = k(\mathbf{z}_i, \mathbf{x}_j^m)$, $i = 1, \ldots, L$ and $j = 1, \ldots, N^m$
- $\mathbf{v}_1$: a workspace vector of size MaxSv $= \max N^m$, $m = 1, \ldots, M$
- $\mathbf{v}_2$: a workspace vector of size n
- $\mathbf{v}_3$: a workspace vector of size M
- $\mathbf{v}_4$: a workspace vector of size n
- $\mathbf{R}$: a workspace of size $L \times$ MaxSv

- $\mathbf{D}$: $n \times M$ workspace matrix
- $\mathbf{Q}$: $L \times M$ workspace matrix
- e^m: approximation error of class m and $e^m = \| \sum_{i=1}^{L} \gamma_i^m \Phi(\mathbf{z}_i) - \sum_{i=1}^{N^m} \beta_i^m \Phi(\mathbf{x}_i^m) \|^2$
- $\mathbf{Z}$: $L \times n$ matrix and its ith row vector $(\mathbf{Z}^T)_i = \mathbf{z}_i$
- $\mathbf{X}$: $|U| \times n$ matrix and its ith row vector $(X^T)_i = \mathbf{x}_i$, $\mathbf{x}_i \in U$
- $\mathbf{X}^m$: $N^m \times n$ matrix and $(\mathbf{X}^T)_i = \mathbf{x}_i^m$
- $\mathbf{idx}^m$: a vector of size N^m that stores index numbers for support vectors of class m. That is, the corresponding rows in the matrix $\mathbf{X}$ where support vectors of class m are located.
- γ^m: vectors of length L and $\gamma^m = (\gamma_1^m, \ldots, \gamma_L^m)^T$
- β^m: $\beta^m = (\beta_1^m, \ldots, \beta_{N^m}^m)^T$

In the above notations, $(\mathbf{K})_{ij}$ denotes an element located in the ith row and jth column of matrix $\mathbf{K}$. It can also be written as $\mathbf{K}[i][j]$. $(\mathbf{K})_i$ denotes the ith column vector of matrix $\mathbf{K}$. Then a fast algorithm for the simultaneous optimization of reduced set vectors is proposed and summarized below:

Fast simultaneous optimization of reduced set vectors

Input: $\mathbf{X}$ and $\mathbf{idx}^m, \beta^m$, $m = 1, \ldots, M$.

Output: $\mathbf{z}_i$, $i = 1, \ldots, L$ and γ^m, $m = 1, \ldots, M$.

Initialization:

1.1 Select row vectors from $\mathbf{X}$ randomly and initialize $\mathbf{Z}$.

1.2 Compute matrices $\mathbf{K^z}$ and $\mathbf{K^{zx}}$.

1.3 Initialize γ^m, $\gamma^m = (\mathbf{K}^z)^{-1}(\mathbf{K^{zx^m}})\beta^m$,
$m = 1, \ldots, M$.

Optimization:

2.1 $t \leftarrow 0$.

Repeat

For $p = 1$ to L

2.2.1 $\mathbf{v}_2 \leftarrow 0$, TotalSum $\leftarrow 0$.

2.2.2 $\mathbf{Q}[j][m] \leftarrow \gamma_j^m \mathbf{K}^{\mathbf{z}}[j][p]$,
$j = 1, \ldots, L$, $m = 1, \ldots, M$.
$\mathbf{v}_3[m] \leftarrow \sum_{j=1}^{L} \mathbf{Q}[j][m]$,
$\mathbf{D} = \mathbf{Z}^T\mathbf{Q}$.

2.2.3

For $m = 1$ to M

2.2.3.1 Sum $\leftarrow \gamma_p^m - \mathbf{v}_3[m]$.

2.2.3.2 $\mathbf{v}_1[i] \leftarrow \beta_i^m \mathbf{K}^{\mathbf{zx}}[p][\mathbf{idx}^m[i]]$,
$i = 1, \ldots, N^m$;
Sum $\leftarrow$ Sum $+ \sum_{i=1}^{N^m} \mathbf{v}_1[i]$,
TotalSum $\leftarrow$ TotalSum + Sum$\times$Sum.

2.2.3.3 $\mathbf{v}_4 = (\mathbf{X}^m)^T\mathbf{v}_1$.

2.2.3.4 $\mathbf{v}_4 \leftarrow$ Sum $\times (\mathbf{v}_4 - (D)_m + \gamma_p^m \mathbf{z}_p^t)$.

2.2.3.5 $\mathbf{v}_2 \leftarrow \mathbf{v}_2 + \mathbf{v}_4$.

End

2.2.4 Update $\mathbf{z}_p$: $\mathbf{z}_p \leftarrow \frac{1}{\text{TotalSum}} \times \mathbf{v}_2$.

End

2.3 Update $\mathbf{K}^{\mathbf{z}}$ and $\mathbf{K}^{\mathbf{zx}}$.

2.4 Compute the inverse matrix of $\mathbf{K}^{\mathbf{z}}$ using Cholesky factorization [50].

2.5 ApproxError $\leftarrow 0$.

2.6

For $m = 1$ to M

2.6.1 $R[i \times N^m + j] \leftarrow \mathbf{K}^{\mathbf{zx}}[i][\mathbf{idx}^m[j]]$, $i = 1, \ldots, L$ and $j = 1, \ldots, N^m$.

2.6.2 $\mathbf{v}_1 = \mathbf{R}\beta^m$, where $\mathbf{R}$ is an $L \times N^m$ matrix.

2.6.3 Update γ^m: $\gamma^m = (\mathbf{K}^{\mathbf{z}})^{-1}\mathbf{v}_1$.

2.6.4 Compute e^m.

2.6.5 ApproxError $\leftarrow$ ApproxError $+ e^m$.

End

2.7 $t \leftarrow t+1$.

Until $t > T$ or ApproxError $< \epsilon$.

In the above algorithm, T specifies the maximal iteration number and ϵ is a threshold for the approximation error. In [34], a fast method can be used to calculate kernel matrices $\mathbf{K^z}$ and $\mathbf{K^{zx}}$ by means of Block Linear Algebra Subprogram (BLAS) [38]. In the following, we describe briefly the method to compute $\mathbf{K^z}$. For kernel element $\exp(-\frac{\|\mathbf{z}_i-\mathbf{z}_j\|^2}{2\sigma^2})$, $\| \mathbf{z}_i - \mathbf{z}_j \|^2$ can be rewritten as

$$\| \mathbf{z}_i - \mathbf{z}_j \|^2 = \mathbf{z}_i^T\mathbf{z}_i + \mathbf{z}_j^T\mathbf{z}_j - 2\mathbf{z}_i^T\mathbf{z}_j \tag{130}$$

$i, j = 1, \ldots, L$. Terms $\mathbf{z}_i^T\mathbf{z}_i$ can be calculated by calling CBLAS function **cblas_sdot**. $\mathbf{z}_i^T\mathbf{z}_j$, $i, j = 1, \ldots, L$ can be rewritten as $\mathbf{ZZ}^T$ (Gram matrix), which can be easily calculated by calling CBLAS function **cblas_ssyrk**. Due to symmetry of the kernel matrix, only elements in the upper triangle are calculated for the implementation of **cblas_ssyrk**. Similarly, CBLAS function **cblas_sgemm** can be applied to calculate $K^{\mathbf{zx}}$. Further, **cblas_sgemm** and **cblas_sgemv** are used to compute the multiplication between matrix and matrix and between matrix and vector in the above algorithm, respectively.

7.3.1 Analysis of run-time complexity

For the sake of complexity analysis, we assume that the computational cost of $\exp(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2})$ is linearly scale to that of $\mathbf{x}^T\mathbf{x}'$. In addition, we just consider the cost of multiplication operations. In addition, the specified iteration step T is used to control the stopping condition. So the cost of **2.6.4** in the above algorithm is not taken into account. There are three loops in the algorithm. First the cost for each step is estimated.

1.2 : $O(\frac{1}{2}L^2n + Ln|U|)$

1.3 : The inverse matrix $\mathbf{K^z}$ is computed using Cholesky factorization. The cost is $O(L^3)$ [22]. The total cost for this step is $O(L^3 + \sum_{m=1}^{M}(LN^m + L^2))$

2.2.2 : $O(LM + nLM)$

2.2.3.2 : $O(N^m + 1)$

2.2.3.3 : $O(nN^m)$

2.2.3.4 : $O(2n)$

2.2.4 : $O(n)$

2.3 : $O(\frac{1}{2}L^2 + Ln|U|)$

2.4 : $O(L^3)$

2.6.2: $O(LN^m)$

2.6.3: $O(L^2)$

Based on the estimated cost of each step, the total cost can be computed as

$$\begin{aligned} O(\frac{1}{2}L^2n + Ln|U| + L^3 + \sum_{m=1}^{M}(LN^m + L^2) + T(L^2M \\ +nL^2M + L\sum_{m=1}^{M}(N^m + 1) + L\sum_{m=1}^{M}nN^m + nL \\ +2LMn + \frac{1}{2}L^2n + Ln|U| + L^3 \\ +\sum_{m=1}^{M}LN^m + ML^2)) \end{aligned} \tag{131}$$

The expression (131) can be simplified as

$$\begin{aligned} O(\frac{1}{2}(T+1)L^2n + (T+1)Ln|U| + (T+1)L^3 \\ +(2T+1)L\sum_{m=1}^{M}N^m + (T(n+2)+1)ML^2 \\ +T(M+n)L + 2TLMn + TLn\sum_{m=1}^{M}N^m) \end{aligned} \tag{132}$$

In order to show which steps dominate the computational cost, we ignore some items with light cost and yield

$$\begin{aligned} O(\tfrac{1}{2}TL^2n + TLn|U| + TL^3 \\ +TnML^2 + TLn\textstyle\sum_{m=1}^{M}N^m) \end{aligned} \tag{133}$$

(133) shows that the dominant cost comes from steps **2.2.2**, **2.2.3.3**, **2.3**, **2.4**. The computations at these steps mainly depend on **BLAS** functions. Since BLAS package such as matrix multiplication is implemented by hardware vendors in assembly language, which makes efficient use of cache, memory and instructions such as single instruction and multi-data (SIMD) [56] on Intel Pentium series or vector instructions in vector processors [90]. Moreover, BLAS has been efficiently implemented on different platforms, which enables the proposed algorithm to perform well across platforms. Therefore, the proposed algorithm can be expected to run efficiently.

7.4 Block algorithm in the test phase

After the reduced vector sets $\mathbf{z}_i$, $i = 1, \ldots, L$ and weighted vectors γ^m, $m = 1, \ldots, M$ are generated, the decision function for classifying an unknown pattern $\mathbf{x}$ is given by

$$m^* = arg\max_m \sum_{i=1}^{L} \gamma_i^m k(\mathbf{z}_i, \mathbf{x}) \tag{134}$$

Usually we classify an unknown pattern one by one. But the computation done in this way is not efficient since the reduced set vectors $\mathbf{z}_i$ is loaded into the level 1 and level 2 caches from the main memory again. The cache hit ratio is low. When the dimension n of vectors is high and L is large, it is likely to lead to Translation Lookaside Buffer (TLB) misses [90]. In a computer system with hierarchical memory, a large number of memory accesses will waste a lot of CPU cycles and the cost is high. Therefore, a block algorithm is designed to classify a group of unknown patterns once. The memory can be efficiently accessed since the reduced set vectors are shared by the group of unknown patterns. First we define some notions in the block algorithm below:

- Z: $L \times n$ matrix and its ith row vector $(\mathbf{Z}^T)_i = \mathbf{z}_i$
- P: the number of pattern vectors in a group
- X: a $P \times n$ matrix. Each row vector denotes a test vector.
- $\mathbf{K^{zx}}$: an $L \times P$ matrix and $(\mathbf{K^{zx}})_{ij} = k(\mathbf{z}_i, (\mathbf{X}^T)_j)$, $i = 1, \ldots, L$, $j = 1, \ldots, P$.
- Q: a $P \times M$ matrix

- **E**: a $M \times L$ matrix and $(\mathbf{E}^T)_m = \gamma^m$
- **v**: a vector of length M, which stores thresholds for M binary SVMs in (118)
- **o**: a vector of length P, which stores the classified results.

The the block algorithm is presented as follows:

Fast Block Algorithm for classification

Input: Z, X, E, v.

Output: o.

1. Compute $\mathbf{K}^{\mathbf{zx}}$.

2. $\mathbf{Q} = (\mathbf{K}^{\mathbf{zx}})^T \mathbf{E}^T$.

3.

For $i = 1$ to P

3.1. $o[i] \leftarrow arg\max_j(\mathbf{Q}[i][j] - \mathbf{v}[j])$,
$j = 1, \ldots, M$.

End

In the above block algorithm, $\mathbf{K}^{\mathbf{zx}}$ can be efficiently computed using the method in Section 3. The BLAS function **cblas_sgemm** is used to compute $\mathbf{Q}$ at step **2**.

7.5 Experimental results

To study speed and accuracy of th proposed methods, two handwritten digit databases were used in our experiments: MNIST and Hanwang. MNIST [78] handwritten digit database consists of 60,000 training samples and 10,000 testing samples [1], which originate from NIST database. A linear transform was performed such that all patterns were centered into 28×28 while keeping the aspect ratio. The pixel values of resulting gray-scale images were scaled to fall in the range from -1.0 to 1.0.

[1] This database is available from http://yann.lecun.com/exdb/mnist/.

Hanwang handwritten digit database [36] is a large commercial database from Hanwang, an OCR company in China. It consists of 1,321,718 training samples and 300,000 testing samples.

In these experiments, a 576 dimensional feature vector [33] is obtained based on the gradients of an image from the normalized patterns of size 22×22. On Hanwang handwritten digit database, a 576 dimensional feature vector is first extracted. Then its dimension is reduced to 120 by principal component analysis [48]. The above feature vectors are used to train SVMs with RBF kernel. For a polynomial kernel, a 784-dimensional vector based on the image pixels is generated [33].

The code was compiled by Microsoft visual C++6.0. All experiments were performed on a PC with single Intel P4 1.7 Ghz processor with 256K L2 (second-level) cache, SDRAM [2] of 1.5 Gigabytes and 200 G hard disk (7200 RPM). The operating system was Windows 2000 Professional.

Support vector machines in our experiments are trained by the fast algorithm in [36]. The strategy for training SVMs with multi-classes is one-against-others. The value of σ^2 for RBF kernel is set to 0.3. In the following subsections some algorithm properties are investigated.

7.5.1 Convergence speed

Support vector machines are trained on MNIST with a 576-dimensional feature vector and RBF kernel. The numbers of support vectors for each class are shown in Table 14. Some support vectors from different classes are shared. Therefore, when they are

Table 14: Number of support vectors of the original SVM

digit	0	1	2	3	4	5	6	7	8	9
N^m	2069	1286	2950	2788	2506	2789	1942	2508	3081	2552

merged, the total number $|U|$ of different support vectors is just 10799. L in the optimization problem (120) is set to 1000. Thus the number of free parameters is 586,000 ($576 \times 1000 + 1000 \times 10$). The approximation errors E in (120) in different steps are depicted in Fig. 29. It can be observed that the proposed algorithm in

[2]Single Data Rate Memory

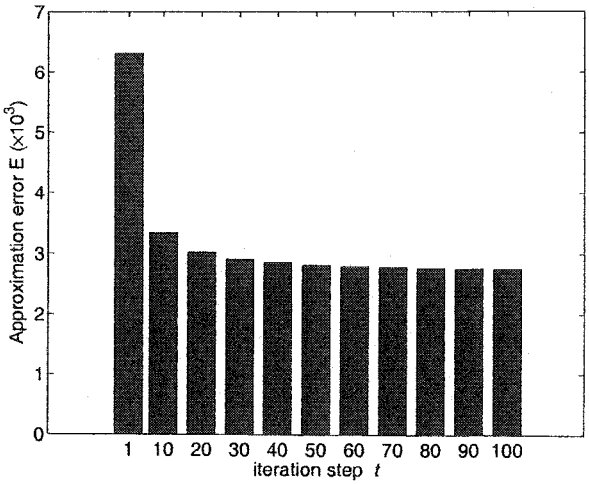

Figure 29: Approximation Error E every 10 steps.

Section 3 reaches the minimum at a fast rate. In addition, since the computational time for each iteration is about 116 seconds, the optimization takes about 1.93 hours (116×60 seconds).

7.5.2 Accuracy vs number of reduced vectors L

The approximation error E usually grows with the number of reduced vectors L. When L is sufficiently large, the approximation error could be very small so that the classification accuracy determined by Eq. (134) can be the same as the original one. However, in order to speed up the classification in the test phase, L must be small without sacrificing the accuracy. Table 15 illustrates the classification accuracy vs. L. The error rate of the original SVM system is 0.61%. From the above table we

Table 15: Classification accuracy vs. number of reduced vectors

L	400	600	800	1000
Error rate	0.72%	0.63%	0.62%	0.61%

can see that in the reduced set method L could be much smaller than the size of the original merged support vectors $|U|$ while the accuracy is maintained. Also, the above results indicate that the approximation error does not have a direction connection to

the classification accuracy.

7.5.3 Block algorithm

In order to further reduce the computational cost in the test phase, a 576-dimensional feature vector is reduced to 120 dimensions by principal component analysis [33]. After the dimension reduction, the SVM's error rate on the test set is 0.61%. $|U|$ in the SVM system is 9739. When L is set to 600, the reduced set method has achieved an error rate of 0.64%. The classification speeds for the original SVM system and one based on the reduced set method on MNIST test set are 149 and 1923 patterns per second, respectively when test patterns are classified one by one. When the block algorithm is applied, the classification in the test phase can be sped up further. Table 16 shows how the classification speed grows with the number of group patterns. When P is set to 2000, the current recognition system achieved a speed-up factor of

Table 16: Classification speed (patterns per second) vs. number of group patterns P

P	20	100	500	1000	2000
test speed	6,807	11,904	14,556	15,244	16,393

110, compared with the original SVM system. The speed-up mainly comes from the fact that BLAS package makes efficient use of memory and SIMD instructions in P4.

7.5.4 Large handwritten digit database

Hanwang handwritten digit database is much larger than MNIST. After SVMs are trained on it using RBF kernel and 120-dimensional feature vectors, the total number $|U|$ of the merged support vectors is 41,417. When L is set to 1000, the recognition error rate on Hanwang test set is 0.53%, which is very close to that (0.50%) of the original SVM system [36]. The classification speed is about 10,895 patterns per second.

7.5.5 Polynomial kernel

When the algorithm in Section 3 is slightly modified to adapt to a polynomial kernel, we use the modified algorithm to find the reduced set vectors for SVMs trained on

MNIST with the polynomial kernel and 784-dimensional vectors. The polynomial kernel is $(\mathbf{x}^T\mathbf{x}')^7$. After the training, $|U|$ is 13767. When L is set to 800, The classification accuracy of the reduced set system on the test set is 98.6%, slightly inferior to the original one (98.72%) [33].

7.6 Conclusions

The chapter presents a fast iteration algorithm to approximate the reduced set vectors shared by each binary SVM solution for multi-class classification simultaneously. The algorithm can be applied to SVMs with the general kernel types such as $k(\| \mathbf{x}-\mathbf{x}' \|^2)$ and $k(\mathbf{x}^T\mathbf{x}')$. In addition, a fast block algorithm in the test phase is proposed to speed up the classification further. Experimental results on MNIST and Hanwang handwritten digit databases look very promising. The achieved classification speeds in the test phase on the two databases on P4 are 16,000 and 10,895 patterns per second without sacrificing the accuracy of the original SVM system. The proposed algorithms lead to a significant increase in speed, which enables SVM to be very competitive in tasks where both accuracy and speed are major concerns.

Chapter 8

Conclusions

8.1 Summary

This thesis presents some fruitful solutions to several existing problems of neural networks and support vector machines when they are applied to very large-scale data sets. The three questions raised in Chapter 1 "Introduction" can be answered faithfully. Particularly, we believe that the proposed methods for support vector machines described in Chapters 5 and 7 are crucial to real applications where learning is performed on a data set of huge size and both high accuracy and speed are required. Now we add some remarks to the answers to these three questions.

Regarding the first question "*How to adapt the structure of a neural network to yield a good generalization performance for classification on a large data set?*", we present chapter 3 to answer it. The main issue is to deal with the trade-off between accuracy and model complexity (or bias and variance). We assume that real data exists in the low-dimensional manifold. Learning is performed in a local space to obtain a good accuracy regardless of the complexity of the true classification boundary. Ensembles of neural networks are used to reduce the model variance. Experimental results on the medium data set such as MNIST have shown that the proposed local learning framework achieved a good performance.

For the second question "*How to design an efficient training algorithm for support vector machines on a data set of huge size with millions of high-dimensional samples and thousands of classes such that the run-time complexity of the algorithm linearly scales to the size of data set, the dimension of input vectors and the number*

of classes?", we present chapters 5 and 6 to describe the algorithm and analyzing its space and run-time complexity. Various appealing properties of the algorithm have been studied and experimental results on several large databases, including public and commercial ones, have shown a very promising performance. We can say fairly that essentially this problem has been solved although the solution may not be the best one. In addition, we tackle the above two problems successfully based on the general principle of **divide and conquer**[1]. The principle is widely applied to algorithm design [2]. In Chapter 3, a classification task in the global space is decomposed into sub-tasks in local spaces, which effectively deal with the dilemma of bias and variance. In Chapter 5, the complex optimization problem for SVM training on a large data set is split into hundreds of sub-problems, which can be solved efficiently. Our success in solving these problems indicates that this principle could be beneficially applied to design an efficient learning algorithm especially on a large set. Moreover, for the design of a high-performance learning algorithm, some issues from other areas such as computer architecture, operating system and numerical analysis still have to be considered.

We answer the third question in Chapter 7 "*How to make support vector machines achieve both high accuracy and classification speed in the test phase?*". Although usually there is a trade-off between accuracy and speed in scientific computation, the algorithms presented in Chapter 7 have shown that it is possible for support vector machines to achieve both high accuracy and classification speed in the test phase. Experimental results on very large handwritten digit database have been promising.

Finally, we hope that techniques and methods in this thesis can be applied to solve more complex learning problems in industry and academic research, which have not been tried in this thesis.

8.2 Future work

During the development of this thesis some important problems have not been addressed yet due to the time constraint. I believe that the performance could be further improved when the following problems can be solved:

[1] Given a complex problem, split it into several smaller sub-problems and independently solve each sub-problem and combine solutions to sub-problems to yield a solution for the original one.

[2] For example, quick-sort.

- For local learning framework, one of the problems is how to determine the optimal number of prototypes for the partition of the global pattern space. The traditional methods such as k-means clustering usually set the value heuristically. Since the parameter has a significant effect on the performance, users have to spend much time tuning this parameter by trial and error, which is not effective. A more effective way is to apply a data-driven strategy. That is to say, the optimal number of prototypes could be set automatically by solving an optimization problem.

- In the proposed fast SVM training algorithm, the convergence has not be addressed theoretically yet. Moreover, the proposed method gives only an approximate solution to the original problem. Although its performance on numerous problems looks promising, it is still important to close the gap between two solutions theoretically so that we can find some necessary conditions to apply the proposed algorithm to real-world problems.

- For the proposed fast SVM testing algorithm, the computational cost is still high on large data sets with thousands of classes. More research work has to be done.

Bibliography

[1] D. Aberdeen, J. Baxter, and R. Edwards, "92 c/mflops/s, ultra-large-scale neural-network training on a PIII cluster," *Proceedings of Super Computing*, Dallas, TX, November 2000.

[2] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: bagging, boosting and variants," *Machine Learning*, vol. 36, pp. 105–139, 1999.

[3] R. Bellman, *Adaptive Control Processes: A Guided Tour*, Princeton University Press, New Jersey, 1961.

[4] C.M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, 1995.

[5] B.E. Boser, I.M. Guyon, and V.N. Vapnik, "A training algorithm for optimal margin classifiers," *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, D. Haussler, Ed., pp. 144–152. ACM Press, Pittsburgh, PA, 1992.

[6] L. Bottou and V. Vapnik, "Local learning algorithm," *Neural Computation*, vol. 4, no. 6, pp. 888–901, 1992.

[7] P.S. Bradley and U.M. Fayyad, "Refining initial points for k-means clustering," *Proceedings of the Fifteenth International Conference on Machine Learning(ICML'98)*, pp. 91–99, Morgan Kaufmann, San Francisco, CA, 1998.

[8] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[9] L. Breiman, "Bias, variance and arcing classifiers," Tech. Rep. 460, UC-Berkeley, Berkeley, CA, 1996.

[10] J. Bromley and E. Säckinger, "Neural-network and k-nearest-neighbor classifiers," Tech. Rep. 11359-910819-16TM, AT&T, 1991.

[11] H. Bunke and P. Wang, editors. *Handbook of Character Recognition and Document Image Analysis.* World Scientific, 1997.

[12] C.J.C. Burges, "Simplified support vector decision rules," *Proceedings of the 13th International Conference on Machine Learning*, pp. 71–77, Morgan Kaufmann, San Mateo, CA, 1996.

[13] C.J.C. Burges and B. Schölkopf, "Improving the accuracy and speed of support vector learning machines," *Advances in Neural Information Processing Systems*, M. Mozer, M. Jordan, and T. Petsche, Eds., vol. 9, pp. 375–381. MIT Press, Cambridge, MA, 1997.

[14] C.J.C. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and Knowledge Discovery*, pp. 121–167. 1998.

[15] C.C. Chang and Chih-Jen Lin, "Libsvm: a library for support vector machines," Tech. Rep., Dept. of Computer Science and Information Engineering, National Taiwan University, 2003.

[16] S.B. Cho and J.H. Kim, "Combining multiple neural networks by fuzzy integral for robust classification," *IEEE Trans. Systems, Man and Cybernetics*, vol. 25, no. 2, pp. 380–384, 1995.

[17] S.B. Cho, "Neural-network classifiers for recognizing totally unconstrained handwritten numerals," *IEEE Trans. Neural Networks*, vol. 8, no. 1, pp. 43–53, January 1997.

[18] R. Collobert and S. Bengio, "SVMTorch: Support vector machines for large-scale regression problems," *Journal of Machine Learning Research*, vol. 1, pp. 143–160, 2001.

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

[20] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Information Theory*, vol. 13, pp. 21–27, January 1967.

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

[22] J.Du Croz and N.J. Higham, "Stability of methods for matrix inversion," *IMA Journal of Numerical Analysis*, vol. 12, pp. 1–19, 1992.

[23] C. Decaestecker, "Finding prototypes for nearest neighbour classification by means of gradient descent and deterministic annealing," *Pattern Recognition*, vol. 30, no. 2, pp. 281–288, 1997.

[24] D. DeCoste and B. Schölkopf, "Training invariant support vector machines," *Machine Learning*, vol. 46, no. 1-3, pp. 161–190, 2002.

[25] C. Diamantini and A. Spalvieri, "Certain facts about Kohonen's LVQ1 algorithm," *IEEE Trans. Circuits Syst. I*, vol. 47, pp. 425–427, May 1996.

[26] C. Diamantini and A. Spalvieri, "Quantizing for minimum average misclassification risk," *IEEE Trans. Neural Networks*, vol. 9, no. 1, pp. 174–182, January 1998.

[27] J.X. Dong, C.Y. Suen, and A. Krzyżak, "Comparison of algorithms for handwritten numeral recognition," Tech. Rep., CENPARMI, Concordia University, Montréal, Canada, 1999.

[28] J.X. Dong, C.Y. Suen, and A. Krzyżak, "An empirical study of boosting algorithm," Tech. Rep., CENPARMI, Concordia University, Montréal, Canada, 2000.

[29] J.X. Dong, Adam Krzyżak, and C.Y. Suen, "Statistical results of human performance on USPS database," Tech. Rep., CENPARMI, Concordia University, Montréal, Canada, 2001.

[30] J.X. Dong, A. Krzyżak, and C.Y. Suen, "Local learning framework for handwritten character recognition," *Engineering Applications of Artificial Intelligence*, vol. 15, no. 2, pp. 151–159, 2002.

[31] J.X. Dong, A. Krzyżak, and C.Y. Suen, "Local learning framework for recognition of lowercase handwritten characters," *Proceedings of International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pp. 226–238. Springer Lecture Notes in Computer Science, Leipzig Germany, July 2001.

[32] J.X. Dong, C.Y. Suen, and A. Krzyżak, "A fast SVM training algorithm," *Pattern Recognition with Support Vector Machines*, S.-W. Lee and A. Verri, Eds., pp. 53–67. Springer Lecture Notes in Computer Science LNCS 2388, Niagara Falls, Canada, August 2002.

[33] J.X. Dong, C.Y. Suen, and A. Krzyżak, "A fast SVM training algorithm," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 17, no. 3, pp. 367–384, 2003.

[34] J.X. Dong, C.Y. Suen, and A. Krzyżak, "A fast parallel optimization for training support vector machine," *Proceedings of 3rd International Conference on Machine Learning and Data Mining*, P. Perner and A. Rosenfeld, Eds., pp. 96–105. Springer Lecture Notes in Artificial Intelligence (LNAI 2734), Leipzig, Germany, July 2003.

[35] J.X. Dong, A. Krzyżak, and C.Y. Suen, "High accuracy handwritten Chinese character recognition using support vector machine," *Proceedings of International Workshop on Artificial Neural Networks*, Florence, Italy, September 2003.

[36] J. X. Dong, A. Krzyżak, and C.Y. Suen, "A fast parallel SVM algorithm and its applications to very large datasets," submitted to *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2003.

[37] J.J. Dongarra, J.D. Croz, S. Hammarling, and R.J. Hanson, "An extended set of fortran basic linear algebra subprograms," *ACM Transactions on Mathematical Software*, vol. 14, no. 1, pp. 1–17, 1988.

[38] J.J. Dongarra, J.Du Croz, I.S. Duff, and S. Hammarling, "A set of level 3 basic linear algebra subprograms," *ACM Trans. Math. Soft.*, vol. 16, pp. 1–17, 1990.

[39] H. Drucker, R. Schapire, and P. Simard, "Boosting performance in neural networks," *International Journal of Pattern Recognitiona and Artificial Intelligence*, vol. 7, no. 4, pp. 705–719, 1993.

[40] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons. Inc, New York, 1973.

[41] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification*, John Wiley & Sons. Inc, New York, second edition, 2001.

[42] R.A. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugenics*, vol. 7, pp. 179–188, 1936.

[43] G.W. Flake and S. Lawrence, "Efficient SVM regression training with SMO," *Machine Learning*, vol. 46, no. 1-3, pp. 271–290, March 2002.

[44] R. Fletcher, *Practical Methods of Optimization*, John Wiely and Sons, Inc., 2nd edition, 1987.

[45] Y. Freund and R. Schapire, "Experiments with a new boosting algorithm," *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148–156, Bari, Italy, 1996.

[46] Y. Fujisawa, M. Shi, T. Wakabayashi, and F. Kimura, "Handwritten numeral recognition using gradient and curvature of gray scale image," *Proceedings of International Conference on Document Analysis and Recognition*, pp. 277–280, India, August 1999.

[47] K. Fukunaga and T.F. Krile, "Calculation of Bayes' recognition error for two multivariate gaussian distributions," *IEEE Trans. Computers*, vol. C-18, no. 3, pp. 220–229, July 1969.

[48] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, second edition, 1990.

[49] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, no. 3, pp. 183–192, 1989.

[50] G.H. Golub and C.F.Van Loan, *Matrix computations*, The Johns Hopkins University Press, Baltimore, Maryland, 3nd edition, 1996.

[51] B. Hassibi and D.G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," *Advances in Neural Information Processing Systems*,

S. Hanson, J. Cowan, and L. Giles, Eds., vol. 5, pp. 164–171. Morgan Kaufmann, San Mateo, CA, 1992.

[52] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Series in Statistics. Springer-Verlag, New York, 2001.

[53] J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, New York, 1991.

[54] Intel, *The IA-32 Intel Architecture Software Developer's Manual.* Volume 3: System Programmer's Guide. Order number 245472, 2002.

[55] Intel, *Intel Pentium 4 and Intel Xeon Processor Optimization Reference Manual.* Order Number: 248966, 2002.

[56] Intel, *The IA-32 Intel Architecture Software Developer's Manual.* Volume 1: Basic Architecture. Order number 245470, 2002.

[57] T. Joachims, "Text categorization with support vector machine: learning with many relevant features," *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pp. 137–142, Springer Verlag, Heidelberg, Chemnitz, DE, 1998.

[58] T. Joachims, "Making large-scale support vector machine learning practical," *Advances in kernel methods: Support Vector Machines*, B. Schölkopf, C. Burges, and A. Smola, Eds., pp. 169–184. MIT Press, Cambridge, MA, December 1998.

[59] R.A. Jocobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.

[60] B.H. Juang and S. Katagiri, "Discriminative learning for minimum error classification," *IEEE Trans. Signal Processing*, vol. 40, no. 12, pp. 3043–3054, December 1992.

[61] S. Katagiri, C.H. Lee, and B.H. Juang, "Discriminative multilayer feedforward networks," *Proc. IEEE Workshop Neural Networks for Signal Processing*, pp. 11–20, Piscataway, NJ, August 1991.

[62] N. Kato, M. Suzuki, S. Omachi, H. Aso, and Y. Nemoto, "A handwritten character recognition system using directional element feature and asymmetric Mahalanobis distance," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 21, no. 3, pp. 258–262, 1999.

[63] S.S. Keerthi, S.K. Shevade, C. Bhattachayya, and K.R.K. Murth, "Improvements to Platt's SMO algorithm for SVM classifier design," *Neural Computation*, vol. 13, pp. 637–649, March 2001.

[64] S.S. Keerthi and E.G. Gilbert, "Convergence of a generalized SMO algorithm for SVM classifier design," *Machine Learning*, vol. 46, no. 3, pp. 351–360, March 2002.

[65] S.Y. Kim and S.-W. Lee, "Gray-scale nonlinear shape normalization method for handwritten oriental character recognition," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 12, no. 1, pp. 81–95, February 1998.

[66] F. Kimura, K. Takashina, S. Tsuruoka, and Y. Miyake, "Modified quadratic discriminant functions and the application to Chinese character recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 9, no. 1, pp. 149–153, 1987.

[67] F. Kimura and M. Shridhar, "Handwritten numeral recognition based on multiple algorithms," *Pattern Recognition*, vol. 24, no. 10, pp. 969–983, 1991.

[68] F. Kimura, T. Wakabayashi, S. Tsuruoka, and Y. Miyake, "Improvement of handwritten Japanese character recognition using weighted direction code histogram," *Pattern Recognition*, vol. 30, no. 8, pp. 1329–1337, 1997.

[69] T. Kohonen, *Self-Organization and Associate Memory*, Springer-Verlag, Berlin, Germany, 3nd edition, 1989.

[70] T. Kohonen, "The self organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, Sept. 1990.

[71] T. Kohonen, *Self-Organizing Maps*, Springer-Verlag, Berlin, Germany, 2nd edition, 1997.

[72] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," *Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretzky, and T. Leen, Eds., vol. 7, pp. 231–238. MIT Press, Cambridge, MA, 1995.

[73] H. Kuhn and A. Tucker, "Nonlinear programming," *Proceedings of 2nd Berkeley Symposium on Mathematical Statistics and Probabilistics.* pp. 481–492, University of California Press, 1951.

[74] C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh, "Basic linear algebra subprograms for Fortran usage," *ACM Transactions on Mathematical Software*, vol. 5, no. 3, pp. 308–323, 1979.

[75] C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh, "Algorithm 539 basic linear algebra subprograms for Fortran usage [f1]," *ACM Transactions on Mathematical Software*, vol. 5, no. 3, pp. 324–325, 1979.

[76] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L.J. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[77] Y. LeCun, J.S. Denker, and S.A. Solla, "Optimal brain damage," *Advances in Neural Information Processing Systems*, D.S. Touretzky, Ed., vol. 2, pp. 598–605. Morgan Kaufmann, San Mateo, CA, 1990.

[78] Y. LeCun, L.D. Jackel, L. Bottou, J.S. Denker, H. Drucker, I. Guyon, U.A. Müller, E. Sackinger, P. Simard, and V.N. Vapnik, "Comparison of learning algorithms for handwritten digit recognition," *Proc. Int. Conf. Artificial Neural Networks*, pp. 53–60, Paris, 1995.

[79] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 80, no. 11, pp. 2278–2324, November 1998.

[80] S.-W. Lee and J.-S. Park, "Nonlinear shape normalization methods for the recognition of large-set handwritten characters," *Pattern Recognition*, vol. 27, no. 7, pp. 895–902, 1994.

[81] S.W. Lee, "Multilayer cluster neural networks for totally unconstrained handwritten numeral recognition," *Neural Networks*, vol. 8, no. 5, pp. 783–792, 1995.

[82] C.L. Liu, I.J. Kim, and J.H. Kim, "High accuracy handwritten Chinese character recognition by improved feature matching method," *Proceedings of the 4th International Conference on Document Analysis and Recognition*, pp. 1033–1037, Ulm, Germany, August 1997.

[83] C-L. Liu, K. Nakashima, H. Sako, and H. Fujisawa, "Handwritten digit recognition using state-of-the-art techniques," *Proceedings of the eighth International Workshop on Frontiers in Handwriting Recognition*, Niagara on the lake, Canada, August 2002.

[84] T. Matsui, T. Tsutsumida, and S.N. Srihari, "Combination of stroke/background structure and contour-direction features and handprinted alphanumeric recognition," *Proc. Int. Workshop on Frontiers in Handwriting Recognition*, pp. 87–96, Taipei, Taiwan, Republic of China, 1994.

[85] J. Mercer, "Functions of positive and negative type and their connection with the theory of integral equations," *Philos. Trans. Roy. Soc. London*, vol. A(209), pp. 415–446, 1909.

[86] D. Opitz and R. Maclin, "Popular ensemble methods: an empirical study," *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, August 1999.

[87] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: an application to face detection," *Proc. IEEE. Conf. on Computer Vision and Pattern Recognition*, pp. 130–136, Puerto Rico, June 17-19 1997.

[88] E. Osuna and F. Girosi, "Reducing the run-time complexity of support vector machines," *Advances in kernel methods: Support Vector Machines*, B. Schölkopf, C. Burges, and A. Smola, Eds., pp. 271–283. MIT Press, Cambridge, MA, December 1999.

[89] E. Parzen, "On the estimation of a probability density function and the mode," *Annals of Mathematical Statistics*, vol. 33, pp. 1065–1076, 1962.

[90] D.A. Patterson and J.L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Francisco, California, second edition, 1996.

[91] J.C. Platt, "Fast training of support vector machines using sequential minimal optimization," *Advances in kernel methods: Support Vector Machines*, B. Schölkopf, C. Burges, and A. Smola, Eds., pp. 185–208. MIT Press, Cambridge, MA, December 1998.

[92] L.G. Roberts, "Machine perception of three-dimensional solids," *Optical and Electro-Optical Information Processing*, J.T. Tippet, Ed. MIT Press, Cambridge, MA, 1965.

[93] F. Rosenblatt, *Principles of Neurodynamics: Perceptron and Theory of Brain Mechanisms*, Spartan Books, Washington, DC, 1962.

[94] D.E. Rumelhart, J.L. McClelland, and PDP Research Group, *Parallel Distributed Processing*, vol. 1: Fundations, MIT Press, 1986.

[95] A. Sato and K. Yamada, "Generalized learning vector quantization," *Advances in Neural Information Processing Systems*, vol. 8, pp. 423–429. MIT Press, 1996.

[96] A. Sato and K. Yamada, "A formulation of learning vector quantization using a new misclassification measure," *Proc. IEEE Conf. on Pattern Recognition*, pp. 322–325, 1998.

[97] A. Sato and K. Yamada, "An analysis of convergence in generalized LVQ," *Proc. of International Conference on Artificial Neural Networks*, pp. 171–176, 1998.

[98] B. Schölkopf, C.J.C. Burges, and V. Vapnik, "Extracting support data for a given task," *Proceedings, First International Conference on Knowledge Discovery and Data Mining*, U. M. Fayyad and R. Uthurusamy, Eds., pp. 252–257. AAAI Press, Menlo Park, CA, 1995.

[99] B. Schölkopf, C.J.C. Burges, and V. Vapnik, "Incorporating invariances in support vector learning machines," *Artificial Neural Networks–ICANN'96*, C. Malsburg, W. Seelen, J.C. Vorbrüggen, and B. Sendhoff, Eds., vol. 1112, pp. 47–52. Springer Lecture Notes in Computer Science, Berlin, 1996.

[100] B. Schölkopf, *Support vector Learning*, Ph.D. thesis, R. Oldenbourg Verlag, München., Technical University of Berlin, 1997.

[101] B. Schölkopf, P. Knirsch, A.J. Smola, and C. Burges, "Fast approximation of support vector kernel expansions, and an interpretation of clustering as approximation in feature spaces," *Mustererkennung, DAGM-Symposium*, pp. 124–132, Springer, Berlin, 1998.

[102] B. Schölkopf, P. Simard, A. Smola, and V. Vapnik, "Prior knowledge in support vector kernels," *Advances in Neural Information Processing Systems*, M. Jordan, M. Kearns, and S. Solla, Eds., vol. 10, pp. 640–646. MIT Press, Cambridge, MA, 1998.

[103] B. Schölkopf, S. Mika, C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A.J. Smola, "Input space vs. feature space in kernel-based methods," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1000–1017, 1999.

[104] B. Schölkopf and A.J. Smola, *Learning with Kernels*, MIT Press, Cambridge, Massachusetts, 2002.

[105] A.J.C. Sharkey, "Multi-net systems," *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*, Amanda J.C. Sharkey, Ed., pp. 1–30. Springer-Verlag, London, 1999.

[106] P. Simard, Y. LeCun, and J. Denker, "Tangent prop – a formalism for specifying selected invariances in an adaptive network," *Advances in Neural Information Processing Systems*, J.E. Moody, S.J. Hanson, and R.P. Lippmann, Eds., vol. 4. Morgan Kaufmann, San Mateo, CA, 1993.

[107] M. Skurichina and R.P.W. Duin, "Bagging for linear classifiers," *Pattern Recognition*, vol. 31, no. 7, pp. 909–930, 1998.

[108] A.J. Smith, "Cache memories," *Computing Surveys*, vol. 14, no. 3, pp. 473–530, September 1982.

[109] S.N. Srihari, "Recognition of handwritten and machine-printed text for postal address interpretations," *Pattern Recognition Letters*, vol. 14, pp. 291–302, 1993.

[110] C.Y. Suen, C. Nadal, T.A. Mai, R. Legault, and L. Lam, "Recognition of totally unconstrained handwritten numeral based on the concepts of multiple experts," *Proc. Int. Workshop on Frontiers in Handwriting Recognition*, pp. 131–143, Montreal, Canada, 1990.

[111] C.Y. Suen, C. Nadal, R. Legault, T.A. Mai, and L. Lam, "Computer recognition of unconstrained handwritten numerals," *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1162–1180, July 1992.

[112] C.Y. Suen, S. Mori, S.H. Kim, and C.H. Leung, "Analysis and recognition of Asian scripts-the state of the art," *Proceedings of the 7th International Conference on Document Analysis and Recognition*, pp. 866–878, Edinburgh, Scotland, August 2003.

[113] N. Sun, M. Abe, and Y. Nemoto, "A handwritten character recognition system by using improved directional element feature and subspace method," *IEICE Trans. System & Information*, vol. J78-D-II, no. 6, pp. 922–930, June 1995.

[114] L.N. Teow and K.F. Loe, "Robust vision-based features and classification schemes for off-line handwritten digit recognition," *Pattern Recognition*, vol. 35, no. 11, pp. 2355–2364, 2002.

[115] S. Thrun, C. Faloutsos, T. Mitchell, and L. Wassermanand, "Automated learning and discovery: state-of-the-art and research topics in a rapidly growing field," Tech. Rep., Center for Automated Learning and Discovery, Carnegie Mellon University, 1998.

[116] J. Tsukumo and H. Tanaka, "Classification of handprinted Chinese characters using nonlinear normalization methods," *Proc. 9th. Int. Conf. Pattern Recognition*, pp. 168–171, Rome, Italy, November 1988.

[117] N. Tsumura, T. Wakabayashi, F. Kimura, and Y. Miyake, "High accuracy in similar characters using compound function," *IEICE Trans. System & Information*, vol. J83-D-II, no. 2, pp. 623–633, 2000.

[118] K. Tumer and J. Ghosh, "Linear and order statistics combiners for pattern classification," *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*, Amanda J.C. Sharkey, Ed., pp. 127–161. Springer-Verlag, London, 1999.

[119] V. Vapnik, "Principles of risk minimization for learning theory," *Advances in Neural Information Processing Systems*, J.E. Moody, S.J. Hanson, and R.P. Lippmann, Eds., vol. 4, pp. 831–838. Morgan Kaufman, San Mateo, CA, 1992.

[120] V. Vapnik, *The nature of statistical learning theory*, Springer Verlag, Berlin, 1995.

[121] V.N. Vapnik, *Statistical Learning Theory*, Wiley, New York, 1998.

[122] Y. Waizumu, N. Kato, K. Saruta, and Y. Nemoto, "High speed and high accuracy rough classification for handwritten characters using hierarchical learning vector quantization," *IEICE Trans. System & Information*, vol. E83-D, no. 6, pp. 1282–1290, 2000.

[123] C.H. Wang and S.N. Srihari, "A framework for object recognition in a visually complex environment and its application to locating address blocks on mail pieces," *International Journal of Computer Vision*, vol. 2, pp. 125–151, 1989.

[124] R.C. Whaley and J.J. Dongarra, "Automatically tuned linear algebra software," *Proceedings of High Performance Networking and Computing(SC'98)*, 1998.

[125] R.C. Whaley, A. Petitet, and J.J. Dongarra, "Automated empirical optimization of software and the ATLAS project," Tech. Rep., Dept. of Computer Science, Univ. of Tennessee, 2000.

[126] G. Wolberg, H.M. Sueyllam, M.A. Ismail, and K.M. Ahmed, "One-dimensional resampling with inverse and forward mapping functions," *Journal of Graphics Tools*, vol. 5, no. 3, pp. 11–33, 2001.

[127] H. Yamada, K. Yamamoto, and T. Saito, "A nonlinear normalization method for handprinted Kanji character recognition-line density equalization," *Pattern Recognition*, vol. 23, no. 9, pp. 1023–1029, 1990.

[128] K. Yamamoto and A. Rosenfeld, "Recognition of handwritten Kanji characters by a relaxation method," *Proc. of the 6th Int. Conf. on Pattern Recognition*, pp. 395–398, 1982.