Studies on Improving the Efficiency of Support Vector Machines

by

NGUYEN DUNG DUC

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Professor Ho Bao Tu

School of Knowledge Science
Japan Advanced Institute of Science and Technology
March, 2006

Abstract

Motivation and Objective: In recent years support vector machine (SVM) has emerged as a powerful learning approach and successfully be applied in a wide variety of applications. The high generalization ability of SVMs is guaranteed by special properties of the optimal hyperplane and the use of kernel. However, SVM is considered slower than other learning approaches in both testing and training phases. In testing phase SVMs have to compare the test pattern with every support vectors included in their solutions. When the number of support vectors increases, the speed of testing phase decreases proportionally. To reduce this computational expense, reduced set methods try to replace original SVM solution by a simplified one which consists of much fewer number of vectors, called reduced vectors. However, the main drawback of former reduced set methods lies in the construction of each new reduced vector: it is required to minimize a multivariate function with local minima. Thus, in order to achieve a good simplified solution the construction must be repeated many times with different initial values. Our first objective was aiming at building a better reduced set method which overcomes the mentioned local minima problem. The second objective was to find a simple and effective way to reduce the training time in a model selection process. This objective was motivated by the fact that the selection of a good SVM for a specific application is a very time consuming task. It generally demands a series of SVM training with different parameter settings; and each SVM training solves a very expensive optimization problem.

Methodology: Starting from a mechanical point of view, we proposed to simplify support vector solutions by iteratively replacing two support vectors with a newly created vector; or to substitute two member forces in an equilibrium system by an equivalent force. This approach also faces the difficulties caused by the so called pre-image problem of kernel-based methods where generally there is no exact substitution of two support vectors in a kernel-induced feature space by image of a vector in input space. However this bottom-up approach possess a big advantage that the computation of the new vector involves only two support vectors being replaced, not to involve all vectors as in the former top-down approach. The extra task of the bottom-up method is to find a heuristic to select a good pair of support vectors to substitute in each iteration. This heuristic aims at minimizing the difference between the original solution and the simplified one.

Also, it is necessary to design a stopping condition to terminate the simplification process before it makes the simplified solution too different from the original one, thus the possible loss in generalization performance can get out of control. For the second problem, our intensive investigation reconfirmed that different SVMs trained by different parameter settings share a big portion of common support vectors. This observation suggests a simple technique to use the results of previously trained SVMs to initialize the search in training a new machine. In a general decomposition framework for SVM training, this initialization makes the initial local optimized solution closer to the global optimized solution; hence the optimization process for SVM training converges more quickly.

Finding and Conclusion: The bottom-up approach leads to a conceptually simpler and computationally less expensive method for simplifying SVM solutions. We found that it is reasonable to select a close support vector pair to replace with a newly constructed vector, and this construction only requires finding the unique maximum point of a univariate function. The uniqueness of solution does not only make the algorithm run faster, but it also makes the reduce set method easier to use in practice. Users do not have to run many trials and wonder about different results returned in different runs. Experimental results on real life datasets shown that our proposed method can reduce a large number of support vectors and keeps generalization performance unchanged. Comparing with former methods, the proposed one produced slightly better results, and more importantly it is much more efficient. For the second problem, experiments on various real life datasets showed that by initializing the first working set using the result of trained SVMs, the training time for each subsequent SVM can be reduced by 22.8-85.5%. This reduction is significant in speeding up the whole model selection process.

Acknowledgments

This work was carried out at Knowledge Creating Methodology Lab, School of Knowledge Science, Japan Advanced Institute of Science and Technology. I wish to express my gratitude to the many people who have supported me during my work.

I am most grateful to my supervisor, Prof. Ho Tu Bao, for providing me with his help, supervision and motivation throughout the course of this work. His insight and breadth of knowledge have been invaluable to me. Without his care, supervision and friendship I would not be able to complete this work. I want to thank Prof. Kenji Satou, who has kindly accepted me to do a minor theme research under his supervision.

I wish to express my gratefulness to the official referees of the dissertation, Prof. Kenji Satou, Prof. Yoshiteru Nakamori, Prof. Tsutomu Fujinami, and Prof. Hiroshi Motoda, for their valuable comments and suggestions on this dissertation.

I would like to express my appreciation to the Ministry of Education, Culture, Sports, Science, and Technology of Japan, and the International Information Science Foundation for providing me the scholarship and the financial support for attending international conferences.

My special thank goes to the members of the Knowledge Creating Laboratory, and the many friends of mine in JAIST for providing their helps, a friendly and enjoyable environment.

Finally, I am indebted to my parents for their forever affection, patience, and constant encouragement, to my wife for sharing me difficulties and happiness.

To my son, the greatest source of inspiration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we firstly review the many efforts currently being made to improve the efficiency of the support vector learning approach. After that we mention some limitations of the previous methods and briefly introduce our solutions in simplifying support vector solutions and in speeding-up support vector training in a model selection process. Outline of this thesis will be given in the last section of this chapter.

1.1 Efforts in Improving the Efficiency of Support Vector Learning

The support vector learning [1, 2, 3, 4] implements the following idea: it finds an optimal hyperplane in feature space according to some optimization criterion, e.g. it is the optimal hyperplane that maximizes the distance to training examples in a two-class classification task, or maximizes the flatness of a function in regression. Thus, training a support vector machine (SVM) is equivalent to solving an optimization problem in which the number of variables to be optimized is l, and the number of parameter is l^2, where l is the size of training data. This is apparently an expensive task in both memory requirement and computational power. Moreover, the optimal hyperplane lies in feature space which is constructed based on the choice of kernel. Selecting a suitable kernel for a specific application is still an open problem and SVM users have to do intensive trials of training and testing with different types of kernel and different values of parameters. Also, since the feature space does not exist explicitly the hyperplane, e.g. a classifier or a regressor, is characterized by a set of training examples called *support vectors*. To test a new pattern SVMs have to compare it with all these support vectors and this becomes a very time consuming work when the number of support vectors is large. In short, support vector is a rather computationally demanding learning approach, and in return, it can produce

high generalization ability machines in many practical applications.

There have been different directions to deal with the high resource-demanding property of support vector training. The algorithmic approach tries to find intelligent solutions for a quick convergence to the optimal solution with a limited memory available. From the observation that the SVM solutions are sparse, or many of training examples do not play any role in the forming of SVM solutions, chunking and decomposition methods [1, 5] decompose the original quadratic programming (QP) problem into a series of smaller QP problems. These method has been shown to be able to handle problems with size exceeding the capacity of the computer, e.g. RAM memory. The sequential minimal optimization (SMO) [6] can be viewed as the extreme case of decomposition methods. In each iteration SMO solves a QP problem of size two using an analytical solution, thus no optimizer is required. The remaining problem of SMO is to choose a good pair of variable to optimize. The original heuristics presented in [6] are based on the level of violating the optimal condition. There have been several works, e.g. [7, 8], trying to improve these heuristics. The general decomposition framework and some other implementation techniques like shrinking, kernel caching have been implemented in most currently available SVM softwares, e.g. SVM^{light} [9], LIBSVM [10], SVMTorch [11], HeroSvm [12]. The main obstacle for this approach is still the huge memory required for storing kernel elements when the number of training example exceeds a few hundreds of thousands. The second approach to solving large scale SVM is to parallelize the optimization. The main idea is to split training data into subsets and perform optimization on these subsets separately. The partial results are then combined and filtered again into a "cascade" of SVMs [13, 14], or a mixture of SVMs [15]. However, the price we must pay is the possibility of losing predictive accuracy because the combination of partial SVMs does not guarantee an optimal hyperplane, thus we might get a machine with lower performance than those trained by other learning approaches [16]. The third approach is to properly remove "unnecessary" examples from training data, thus simultaneously reducing the memory requirement as well as the training time. The reduced support vector machines method [17, 18] reduce the size of the original kernel matrix from $l \times l$ to $l \times m$, where m is the size of a randomly selected subset of training data considered as candidates of support vectors. The smaller matrix of size $l \times m$ (with m is much smaller than l) can be stored in memory, so optimization algorithms such as Newton method can be applied. Instead of random sampling, different techniques have been used to intelligently sample a small number of training examples from training data, e.g. using cross-training [19], boosting [19], clustering [20, 21], active learning [22, 23, 24], on-line and active learning [22]. Another way to reduce the size of the optimization problem is applying different techniques to obtain low-rank approxima-

tions on the kernel matrix using Nyström method [25], greedy approximation [26], matrix sampling [26] or matrix decomposition [27]. The drawback of this approach still is that the resulted machines can only achieve a "similar" or a comparable performance with the machines trained on the original training data. There have been also many other efficient implementation techniques to achieve approximate support vector solutions with a low cost. The core support machines in [28] reformulates the optimization in SVM training as a minimum enclosing ball (MEB) problem in computational geometry, and then adopt an efficient approximate MEB algorithm to obtain approximately optimal solution. In [29] the authors consider the application of quantum computing to solve the problem of effective SVM training.

Though training SVMs is computationally very expensive, SVM users have to spend most time for choosing a suitable kernel and appropriate parameter setting for their applications, or to deal with the model selection problem. In order to achieve a good machine, model selection has to solve two main tasks: to conduct a search in model space (a space of all available SVMs), and to evaluate the goodness of a model. Different search strategies have been proposed to improve the search, including grid search with different grid size [30], pattern search [31], and all common search strategies when applicable like gradient descent [32, 32, 33], genetic algorithms [34]. The difficulty in conducting the search in model space is that there have been no theories to suggest this type of kernel will work better than the other on a given domain, or to determine the region of parameter values where we can find the best one. Another way to speed-up model selection process is to efficiently evaluate each model in our consideration. In [35], the author proposed $\xi\alpha$-estimator specially designed for support vector machines. The $\xi\alpha$-estimator is based on the general leave-one-out method, but it is much more efficient because it does not require to perform re-sampling and retraining. The open question for model evaluation is that there is no dominated method in estimating the goodness of a model. In practice, SVM users estimate error rate of a machine mainly based on cross validation techniques like k-fold cross validation, which is very time consuming.

One common property between support vector learning and instance-based learning is that they have to compare all instances included in their solution with the new pattern in testing phase (these instances are support vectors in SVMs and all training examples in nearest neighbor machines). Except for linear SVMs where the norm vector of the optimal hyperplane can be represented by a vector in input space, the solution of a nonlinear SVM is characterized by a linear combination of support vectors in feature space. Thus to classify a new pattern, SVMs have to compare it with every support vectors via kernel calculations. This computation becomes very expensive when the number of support

vector is large. The reduced set methods, e.g. [36, 37, 38], try to replace the original SVM by a simplified SVM which consists of fewer number of support vectors, called reduced SVs. The support vectors in the simplified solution can be newly created, or selected from the set of original support vectors. The limitation of this approach lies in the construction/selection of reduced SVs that faces local minimum problem. Another approach to speed-up SVMs is to approximate the comparison in the testing phase. In [39, 40], the authors proposed to treat kernel machines as a special form of k-nearest neighbor machines. The result of testing phase is based on comparisons with nearest support vectors, where these SVs are determined in a pre-query analysis. These methods have been shown to produce very promising speed-up rate, but they require an extensive pre-query analysis and depend much on very sensitive parameters, thus cause practical difficulties for real life applications.

In summary, support vector learning is a resource demanding learning approach. There have been a huge number of works trying to make support vector machines run faster in all training, model selection, and in testing phases. Our effort described in this dissertation is two folds: making SVMs run faster in testing phase and speeding-up the support vector training in a model selection process.

1.2 Problem and Contribution

In comparing with making support vector training and model selection run faster, speeding-up SVMs in testing phase is practically important, especially for real-time or on-line applications like detection of objects in streaming video or in image [41, 42, 14, 43, 44, 45, 46], abnormal events detection [46, 47], real-time business intelligence systems [20]. In these applications, it is possible to train the machines in hours, or days, but the respond time must be limited in a restrictive period. The reduced set methods briefly introduced above have been successfully used for reducing the complexity of SVMs in many applications like handwritten character recognition [48, 49], face detection in a large collection of images [14]. However, the main difficulty still lies in the fact that it is impossible to exactly replace a complicated linear combination of many vectors in feature space by a simple one, except for linear SVMs. For linear SVMs we can represent the optimal hyperplane by only two parameters: the norm vector which is also a vector in the input space, and the bias. For nonlinear SVMs, because the feature space is constructed implicitly then the normal vector must be represented by a linear combination of images of input support vectors. The reduced set approach has no way but approximates the original combination by a fewer number of SVs, called the *reduced SVs*. In previous methods, constructing

each new support vector requires to minimize a multivariate function with local minima. Because we cannot know the global minimum has been reached or not, the construction has to repeat the search many times with different initial guesses. This repetition must be applied for every reduced SV in order to arrive at the final reduced solution, and there is also no way but to determine the goodness of the reduced solution experimentally. Our attempt in this research direction is to propose a conceptually simpler and computationally less expensive method to simplify support vector solutions. Starting from a mechanical point of view in which if each SV exerts a force on the optimal hyperplane then support vector solutions satisfy the conditions of mechanical equilibrium [50], and in an equilibrium system if we replace two member forces by an equivalent one, the stable state will not change. Thus, instead of constructing reduced vectors set incrementally like in the previous reduced set methods, two nearest SVs will be iteratively considered and replaced by a newly constructed vector. This approach leads to the construction of each new vector only requiring to find the unique maximum point of a one-variable function on (0,1), and the final reduced set is unique for each running time. Experimental results showed that this method is effective in reducing the number of support vectors and preserving generalization performance. To control the possible lost in generalization performance, we propose a quantity called *maximal marginal difference* to estimate the difference between the original SVM solution and the simplified one. The simplification process will stop before it makes the estimated difference exceed a given threshold.

Our second contribution is devoted for speeding-up the support vector training in a model selection process. By conducting intensive experiments we reconfirm that two different machines trained by two different parameter settings, or even two different choices of kernel, share a big number of support vectors. This observation suggests an inheritance mechanism in which training a new SVM in a model selection process can benefit from the results of previously trained machines. In the general decomposition framework, we propose to initialize each new working set by a set of all SVs found in previously trained machines. Moreover, if two machines use the same kernel function then one's solution can be adjusted and used as the initial point in searching for the the other's solution. This initialization makes the first local solution closer to the global solution, and the decomposition algorithm converges more quickly. Experimental results indicated that we can reduce 22.8-85.5% the training time without any impact on the result of model selection.

1.3 Thesis Outline

Chapter 2 introduces basic concepts in support vector learning. Especially it emphasizes critical properties of the optimal hyperplane and the use of kernel in classical classification and regression tasks. We intended to discuss in more detail the two most commonly used kernels: Gaussian RBF and polynomial, and the decomposition algorithm for SVM training. These fundamentals will be used in other chapters.

Chapter 3 describes attempts in making SVMs run faster in testing phase. Firstly it reviews existing methods for reducing the complexity of SVMs by reducing the number of necessary SVs included in SVM solutions. It then describes our proposed bottom-up method for replacing two SVs by a new one and the whole iterative simplification process, including a selection heuristic and a stopping condition. Experiments will be reported next, and this chapter ends with conclusions.

Chapter 4 introduces the model selection problem for support vector machines and the many efforts in making this process more efficient. It then describes a technique to speed-up SVM training in a model selection process by inheriting the result among different SVMs under consideration. Experiments on various benchmark datasets are described next for illustrating the effectiveness of the proposed method.

Chapter 5 concludes this dissertation with summarization of methodology, contribution, as well as limitation of the proposed methods. It also figures out open problems for a further research in future.

Chapter 2

Preliminaries on Support Vector Machines

2.1 Introduction

In this chapter we describe the essences of the support vector learning approach, especially we emphasize special properties of the optimal hyperplane and the use of kernel as a media for support vector algorithms to work indirectly in feature space. We also discuss different implementation techniques for an efficient support vector training.

2.2 Linear Support Vector Classification

We begin to introduce support vector machines (SVMs) by starting from the simplest task: to build a linear machine on separable data. Suppose we are given a binary supervised data $S = \left\{(\boldsymbol{x}_i, y_i), \boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}, i = 1, ..., l\right\}$. Saying that this data is separable means there exist a linear discriminant function $f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$, where (vector) $\boldsymbol{w}$ is normal of the hyperplane, b is bias/offset, that correctly separates all the positive from the negative examples. There have been many learning algorithms that can solve this task, including Rosenblatt's Perceptron [51], Fisher's Linear Discriminant [52], and support vector machines [1, 3]. In this section we will discuss the solution given by support vector machine, emphasizing its special properties concerning to the concept of maximal margin hyperplane.

2.2.1 The Maximal Margin Hyperplane

Firstly, we would like to mention two important notions of *functional margin* and *geometric margin* of a training example with respect to a hyperplane as follows

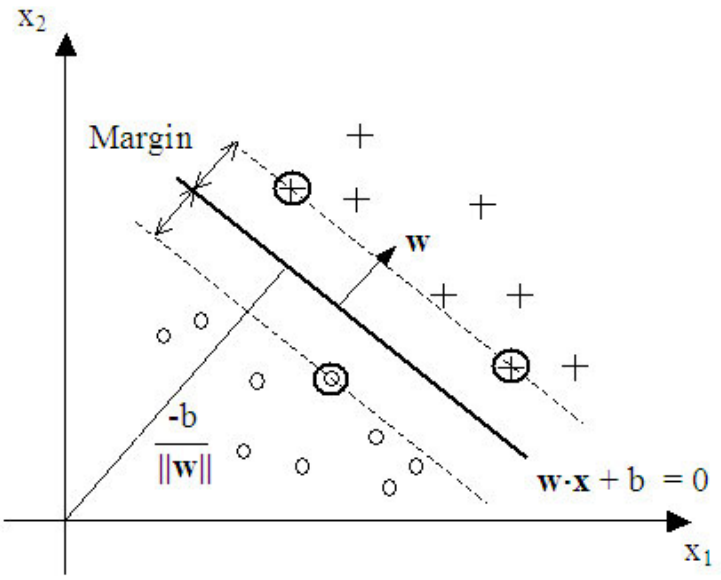


Figure 2.1: Margin of a set of examples with respect to a hyperplane. The origin has $\frac{-b}{\|\mathbf{w}\|}$ perpendicular Euclidian distance to the hyperplane.

Definition 1 *The functional margin of a training example* $(\boldsymbol{x}_i, y_i)$ *with respect to a hyperplane* $\left\{\boldsymbol{x} \in \mathbb{R}^d | f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b = 0\right\}$ *is*

$$\hat{\rho}_f(\boldsymbol{x}_i, y_i) = y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \tag{2.1}$$

In our task of binary classification, $\hat{\rho}_f(\boldsymbol{x}_i, y_i) > 0$ implies a correct classification. If $y_i = 1$, then for the functional margin to be large (i.e., for our prediction to be confident and correct) we need $\boldsymbol{w} \cdot \boldsymbol{x} + b$ to be a large positive number. Conversely, if $y_i = -1$, then for the functional margin to be large we need $\boldsymbol{w} \cdot \boldsymbol{x} + b$ to be a large negative number. More generally, if $y_i(\boldsymbol{w} \cdot \boldsymbol{x} + b) > 0$, then our prediction on this example is correct. Hence, a large functional margin represents a confident and a correct prediction. For linear classifier, however, there is one property of functional margin that makes it not a very good measure of confidence. For example, if we replace $\boldsymbol{w}$ with $2\boldsymbol{w}$ and b with $2b$, then the decision function does not change since $sign(2\boldsymbol{w} \cdot \boldsymbol{x} + 2b) = sign(\boldsymbol{w} \cdot \boldsymbol{x} + b)$. However, replacing $(\boldsymbol{w}, b)$ with $(2\boldsymbol{w}, 2b)$ multiples functional margin by a factor of 2. In other words by scaling $\boldsymbol{w}$ and b, we can make the functional margin arbitrarily large without really changing anything meaningful. Thus, it seems to be reasonable to impose some sort of normalization such that $\|\boldsymbol{w}\| = 1$, e.g. replacing $(\boldsymbol{w}, b)$ with $(\boldsymbol{w}/\|\boldsymbol{w}\|, b/\|\boldsymbol{w}\|)$. In this case, functional margin becomes geometric margin, that is actually the Euclidean distance from a point to the hyperplane.

Definition 2 *The geometric margin of a training example* $(\boldsymbol{x}_i, y_i)$ *with respect to a hyperplane* $\left\{\boldsymbol{x} \in \mathbb{R}^d | f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b = 0\right\}$ *is*

$$\begin{aligned} \rho_f(\boldsymbol{x}_i, y_i) &= y_i(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \cdot \boldsymbol{x}_i + \frac{b}{\|\boldsymbol{w}\|}) \\ &= \frac{\hat{\rho}_f(\boldsymbol{x}_i, y_i)}{\|\boldsymbol{w}\|} \end{aligned} \tag{2.2}$$

Definition 3 *The margin of a training set* $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1...l}$ *with respect to a hyperplane* $\left\{\boldsymbol{x} \in \mathbb{R}^d | f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b = 0\right\}$ *is the minimum value of the (geometric) margin over all training examples*

$$\rho_f = \min_{i=1...l} \rho_f(\boldsymbol{x}_i, y_i) \tag{2.3}$$

The hyperplane which support vector machine is looking for is the one with maximum margin over all hyperplanes, called the maximal margin hyperplane

$$f^* = \arg\max_f \rho_f \tag{2.4}$$

There are several reasons why this hyperplane possesses a high generalization ability, or will work well on testing data. Firstly, it is intuitively a good hyperplane. In Figure 2.2, the linear machine with larger margin on the right-hand side is intuitively preferable because it is likely that the larger margin classifier will classify better an unseen test point. In Figure 2.3, assuming that all test points are generated by adding bounded pattern noise to the training patterns. For example, given a training point $(\boldsymbol{x}, y)$, we generate test points of the form $(\boldsymbol{x} + \Delta\boldsymbol{x}, y)$, where $\Delta\boldsymbol{x}$ is bounded in norm by some $r > 0$. Clearly, if we manage to separate the training set with a margin $\rho > r$, we will correctly classify all test points since all training points have a distance of at least ρ to the hyperplane, the test patterns will still be on the correct side.

The second reason, which is more technical one, is based on the follows one of the bounds on generalization, the *margin percentile* bound. If we order the values in the functional margin distribution

$$M_S(f) = \{\hat{\rho}_i = y_i f(\boldsymbol{x}_i)\}_{i=1,...,l} \tag{2.5}$$

so that $\hat{\rho}_1 \leq \hat{\rho}_2 \leq ... \leq \hat{\rho}_l$ and fix a number of $k < l$, the k/l percentile $M_{S,k}(f)$ of $M_S(f)$ is $\hat{\rho}_k$. The following theorem provides a bound on the generalization error in term of k/l and $M_{S,k}(f)$.

Theorem 1 *(theorem 4.19, page 64 in [54]) Consider thresholding real-valued linear function* $\mathcal{L}$ *with unit weight vectors on an inner product space* X *and fix* $\rho \in \mathbb{R}^+$*. There is a constant* c*, such that for any probability distribution* $\mathcal{D}$ *on* $X \times \{-1, 1\}$ *with support in a*

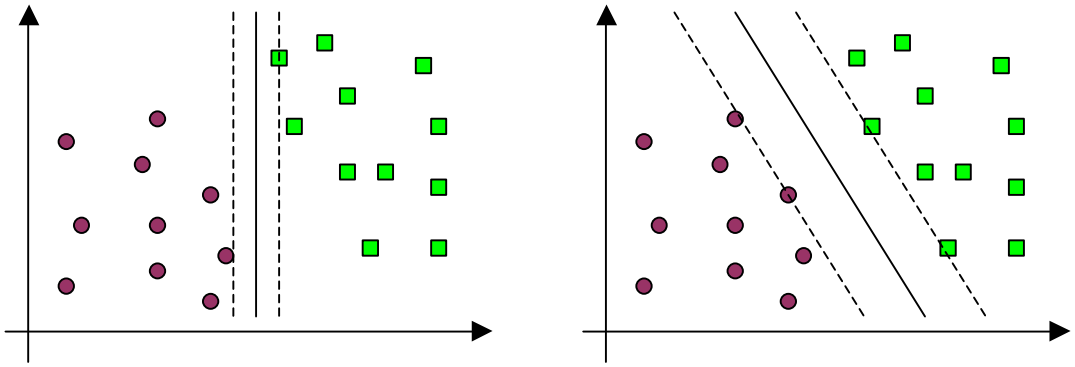


Figure 2.2: Among liner machines, the maximal margin classifier is intuitively preferable.

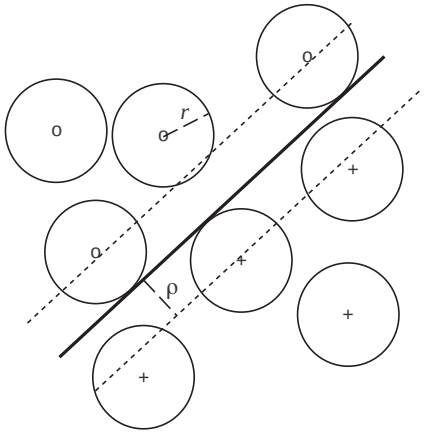


Figure 2.3: Two-dimensional example of a classification problem: separate 'o' from '+' using a straight line. Suppose that we add bounded noise to each pattern. If the optimal margin hyperplane has margin ρ, and the noise is bounded by $r < \rho$, then the line will correctly separates even the noisy patterns.[53]

ball of radius R around this origin, with probability $1-\delta$ over l random examples S, any hypothesis $f \in \mathcal{L}$ has error no more than

$$err_{\mathcal{D}}(f) \leq \frac{k}{l} + \sqrt{\frac{c}{l}\left(\frac{R^2}{M_{S,k}(f)^2}\log^2 l + \log\frac{1}{\delta}\right)} \tag{2.6}$$

for all $k < l$.

The above theorem is equivalent to the following one.

Theorem 2 *(theorem 7.3, page 194 in [53]) Consider the set of decision functions $f(\boldsymbol{x}) = sign(\boldsymbol{w}\cdot\boldsymbol{x})$ (offset b is assumed to be zero for simplicity) with $\|\boldsymbol{w}\| \leq \Lambda$ and $\|\boldsymbol{x}\| \leq R$, for some $R, \Lambda > 0$. Moreover, let $\rho > 0$, and ν denote the fraction of training examples with margin smaller than $\rho/\|\boldsymbol{w}\|$, referred to as the margin error.*

For all distributions P generating the data, with probability at least $1-\delta$ over the drawing of the l training patterns, and for all $\rho > 0$ and $\delta \in (0,1)$, the probability that a test pattern drawn from P will be misclassified is bounded from above, by

$$\nu + \sqrt{\frac{c}{l}\left(\frac{R^2\Lambda^2}{\rho^2}\ln^2 l + \ln(1/\delta)\right)} \tag{2.7}$$

where c is a universal constant.

The above two theorems say that the probability of an error occurred is bounded by a sum of the margin error ν, and a capacity term (the $\sqrt{\ldots}$ term in (2.7)), with the latter tending to zero as the number of examples l tends to infinity. The capacity term can be kept small by keeping R and Λ small, and making ρ large. If we assume that R and Λ are fixed a priori (e.g. by normalizing training examples to be bound in a ball of radius 1, and normalizing $\boldsymbol{w}$), the main influence is ρ. As can be seen from (2.7), large ρ leads to a small capacity term, but the margin error ν gets larger (because ν is the fraction of training examples with margin smaller than $\rho/\|\boldsymbol{w}\|$). A small ρ, on the other hand, will usually cause fewer points to have margins smaller than $\rho/\|\boldsymbol{w}\|$, leading to a smaller margin error; but the capacity penalty will increase correspondingly. The overall message: try to find a hyperplane f which is aligned such that even for a large ρ, there are few margin errors. By assuming the data is separable, our maximal margin hyperplane has largest margin, thus seems to be the best among hyperplanes having zero margin errors (for handling noisy data, the soft margin classifiers will be discuss in the next section).

Another preference of the maximal margin hyperplane is that, in practice, it works very well in many applications like text categorization [55], image recognition [42], handwritten digit recognition [56], Bioinformatics [57, 58].

In summary, we have a fair enough of confidence to say that a linear classifier with large margin has high generalization ability. The confidence comes from intuition, solid theoretical background, and the success in many practical applications.

2.2.2 Finding the Maximal Margin Classifier

For linearly separable training data, the support vector algorithm simply looks for the linear function $f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$ with a margin as large as possible. Without any loss in generality, we can force all training examples to have functional margin greater than a constant $\rho = 1$

$$\boldsymbol{w} \cdot \boldsymbol{x_i} + b \geq +1 \quad for \quad y_i = +1 \tag{2.8}$$

$$\boldsymbol{w} \cdot \boldsymbol{x_i} + b \leq -1 \quad for \quad y_i = -1 \tag{2.9}$$

$$\tag{2.10}$$

or equivalently

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1 \geq 0, i = 1, ...l \tag{2.11}$$

If we call hyperplane $H^+ : \boldsymbol{w} \cdot \boldsymbol{x} + b = +1$, and $H^- : \boldsymbol{w} \cdot \boldsymbol{x} + b = -1$, then the perpendicular distances from the origin to these hyperplanes are $(1 - b)/||\mathbf{w}||$ and $(-1 - b)/||\mathbf{w}||$, or the distance between H_1 and H_2 is $2/||\mathbf{w}||$. Thus we can find the hyperplane which gives maximum margin by minimize $\|\mathbf{w}\|^2$, subject to constraint (2.11)

$$minimize \qquad \frac{1}{2}\|\mathbf{w}\|^2, \tag{2.12}$$

$$subject\ to \qquad y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1, i = 1, ..., l \tag{2.13}$$

2.2.3 Soft Margin Classifiers

The above maximal (hard) margin classifier cannot be used in many real world applications due to a very restrictive requirement: the data is separable. If the data is noisy, there will be no linear separation, even if we transform the data into a high dimensional feature space. The main problem with the maximal margin classifier is that it always produces a consistent hypothesis, that is hypothesis with no training error (the functional margin is greater than 1). In real life applications where noise can always be present, this can result in a brittle estimator. In order to find the optimal margin classifier that can tolerate noises and outliers, we need a "softer" constraint (2.14) than the "hard" one in

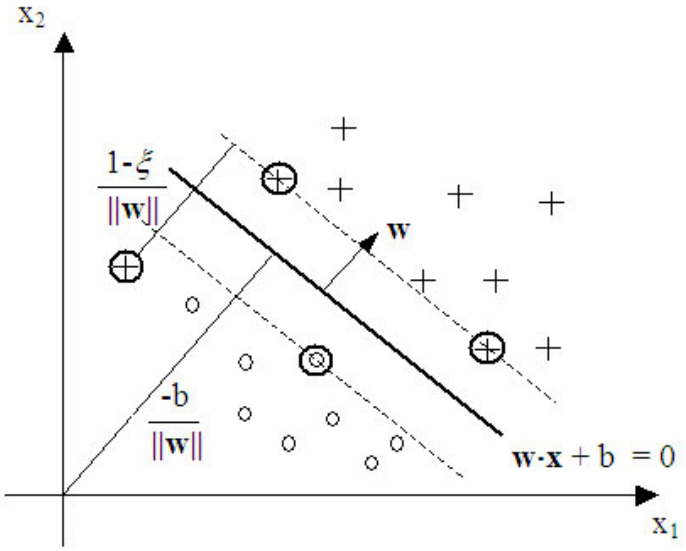


Figure 2.4: Noisy pattern will be treated softly by permitting constraint violation (e.g. having functional margin $\xi < 1$), but the objective function will be penalize a cost $C(1-\xi)$, where ξ is functional margin.

(2.11)

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, i = 1, ..., l \tag{2.14}$$

$$\xi_i \geq 0 \tag{2.15}$$

The ξ_i in (2.14) are called *slack variables*. They permit training examples to have a functional margin of $1 - \xi_i$, but those with margin less than 1 (or violating the original "hard" condition) should pay a price of $C\xi_i$ (or $C\xi_i^2$ for 2-norm soft margin) in the objective function.

$$\min_{\boldsymbol{w},b} \frac{1}{2}\boldsymbol{w}^2 + C\sum_{i=1}^{l}\xi_i \tag{2.16}$$

The parameter C will balance relative weighting between training error penalty/hard margin violation and margin largeness. Setting this parameter to be zero is equivalent to not permitting any margin error, or returning to the hard margin problem.

2.2.4 Optimization

For a consistent description, the (soft) margin optimization problem is rewritten as follows

$$minimize_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\boldsymbol{w}^2 + C\sum_{i=1}^{l}\xi_i \tag{2.17}$$

$$subject\ to \quad -(y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) - 1 + \xi_i) \leq 0, i = 1, ..., l \tag{2.18}$$

$$-\xi_i \leq 0, i = 1, ..., l \tag{2.19}$$

The Lagrangian theory is often used to solve quadratic optimization problem like this, but only with equality constraints. Since our above optimization contains inequality constraints, we need to transform this primal problem into an alternative description which is easier to solve than the primal. The transformation and method to find the optimal hyper plane is briefly described as follows.

Definition 1 *Given an optimization problem with convex domain* $\Omega \subseteq \mathbb{R}^d$

$$minimize \quad f(\boldsymbol{w}), \boldsymbol{w} \in \Omega \tag{2.20}$$

$$subject\ to \quad g_i(\boldsymbol{w}) \leq 0, i = 1, ..., k \tag{2.21}$$

$$h_i(\boldsymbol{w}) = 0, i = 1, ..., m \tag{2.22}$$

The generalized Lagrangian function is defined as

$$L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\boldsymbol{w}) + \sum_{i=1}^{k}\alpha_i g_i(\boldsymbol{w}) + \sum_{i=1}^{m}\beta_i h_i(\boldsymbol{w}) \tag{2.23}$$

Definition 2 *The Lagrangian dual problem of the primal problem is the following problem*

$$maximize \quad \theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{2.24}$$

$$subject\ to \quad \boldsymbol{\alpha} \geq \mathbf{0} \tag{2.25}$$

where $\theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \inf_{\boldsymbol{w}\in\Omega} L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$

The relation between the primal and dual problem is that the value of the dual (the value of the objective function at the optimal solution) is upper bound by the value of the primal

$$\sup\{\theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) : \boldsymbol{\alpha} \geq \mathbf{0}\} \leq \inf\{f(\boldsymbol{w}) : \boldsymbol{g}(\boldsymbol{w}) \leq 0, \boldsymbol{h}(\boldsymbol{w}) = 0\} \tag{2.26}$$

Moreover, if $\boldsymbol{w}^*$ and $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ are feasible solutions of the primal and the dual respectively, and $f(\boldsymbol{w}^*) = \theta(\boldsymbol{\alpha}, \boldsymbol{\beta})$, then $\boldsymbol{w}^*, (\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ solve the primal and the dual problems respectively. The Kuhn-Tucker theorem says that when the primal objective function f is convex, and g_i, h_i are affine functions, then the existence of $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ is the necessary and sufficient condition for the existence of $\boldsymbol{w}^*$.

Theorem 3 *(Kuhn-Tucker) Given an optimization problem with convex domain* $\Omega \subseteq \mathbb{R}^d$

$$\begin{aligned} &\textit{minimize} && f(\boldsymbol{w}), \boldsymbol{w} \in \Omega && (2.27)\\ &\textit{subject to} && g_i(\boldsymbol{w}) \leq 0, i = 1, ..., k && (2.28)\\ & && h_i(\boldsymbol{w}) = 0, i = 1, ..., m && (2.29) \end{aligned}$$

with $f \in C^1$ *convex and* g_i, h_i *affine, necessary and efficient conditions for a normal point* $\boldsymbol{w}^*$ *to be optimum are the existence of* $\boldsymbol{\alpha}^*$ *and* $\boldsymbol{\beta}^*$ *such that*

$$\begin{aligned} \frac{\partial L(\boldsymbol{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{w}} &= 0 && (2.30)\\ \frac{\partial L(\boldsymbol{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} &= 0 && (2.31)\\ \alpha_i^* g_i(\boldsymbol{w}^*) &= 0, i = 1, ..., k && (2.32)\\ g_i(\boldsymbol{w}^*) &\leq 0, i = 1, ..., k && (2.33)\\ \alpha_i &\geq 0, i = 1, ..., k && (2.34) \end{aligned}$$

The primal problem can be transformed into a simpler dual problem by setting to zero the derivatives of the Lagrangian with respect to primal variables (because this is necessary condition), and substituting the obtained relations back into the Lagrangian, hence the dependence on primal variables is removed.

Coming back to our maximal margin optimization problem, the generalized Lagrangian function is

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{w}^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i(y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^{l}\beta_i\xi_i \qquad (2.35)$$

Setting to zero the derivatives of the Lagrangian with respect to primal variables $\boldsymbol{w}, \xi_i$ and b, we have the following relations

$$\begin{aligned} \frac{\partial L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \boldsymbol{w}} &= \boldsymbol{w} - \sum_{i=1}^{l} y_i\alpha_i\boldsymbol{x}_i = \boldsymbol{0} && (2.36)\\ \frac{\partial L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \xi_i} &= C - \alpha_i - \beta_i = 0 && (2.37)\\ \frac{\partial L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial b} &= \sum_{i=1}^{l} y_i\alpha_i = 0 && (2.38) \end{aligned}$$

Replacing above relations into the Lagrangian we obtain the dual objective function

$$
\begin{aligned}
L(\boldsymbol{\alpha}, \boldsymbol{\beta}) &= \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i} \cdot \boldsymbol{x_j} \\
&\quad + C \sum_{i=1}^{l} \xi_i \\
&\quad + \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i} \cdot \boldsymbol{x_j} - b \underbrace{\sum_{i=1}^{l} \alpha_i y_i}_{=0} - \sum_{i=1}^{l} \alpha_i \xi_i + \sum_{i=1}^{l} \alpha_i \\
&\quad - \sum_{i=1}^{l} \beta_i \xi_i \\
&= -\frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i} \cdot \boldsymbol{x_j} \\
&\quad + \sum_{i=1}^{l} \underbrace{\xi_i \left(C - \alpha_i - \beta_i \right)}_{=0} \\
&\quad + \sum_{i=1}^{l} \alpha_i \\
&= \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i} \cdot \boldsymbol{x_j} \qquad (2.39)
\end{aligned}
$$

The conditions $C - \alpha_i - \beta_i = 0$ and $\beta_i \geq 0$ enforce $\alpha_i \leq C$. We arrive at the dual optimization which is simpler and easier to solve

$$
\begin{aligned}
maximize_{\boldsymbol{\alpha}} \quad & L(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i} \boldsymbol{x_j} && (2.40) \\
subject\ to \quad & \sum_{i=1}^{l} y_i \alpha_i = 0 && (2.41) \\
& 0 \leq \alpha_i \leq C, i = 1, ..., l && (2.42)
\end{aligned}
$$

The Karush-Kuhn-Tucker (KKT) complementary conditions (the third condition in the Kuhn-Tucker theorem) are

$$
\begin{aligned}
& \alpha_i (y_i (\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1 + \xi_i) = 0, i = 1, ..., l && (2.43) \\
& \xi_i (\alpha_i - C) = 0, i = 1, ..., l && (2.44)
\end{aligned}
$$

These conditions imply that non-zero slack variables $\xi_i \neq 0$ can only occur when $\alpha_i = C$, and points for which $0 < \alpha_i < C$ have functional margin of 1 (because $\xi_i = 0$).

In other words, only active constraints will have non-zero dual variables, and the solution for the primal depends only on these training points. In support vector learning, the term *support vectors* refers to those examples for which the dual variables are non-zero. Because the value of bias b does not appear in the dual optimization problem, b^* is chosen so that $y_i f(\boldsymbol{x}_i) = 1$ for any i with $0 < \alpha_i^* < C$.

The above optimization is a convex programming problem (maximizes a convex function on a convex domain), thus it has unique optimized solution $(\boldsymbol{w}^*, b^*)$ [59]. Solving this problem we will arrive at our decision function

$$y = sign\left(\boldsymbol{w}^* \cdot \boldsymbol{x} + b^*\right) \tag{2.45}$$

$$= sign\left(\sum_{\alpha_i \neq 0} y_i \alpha_i^* \boldsymbol{x_i} \cdot \boldsymbol{x} + b^*\right) \tag{2.46}$$

For hard margin classifiers, the box constraints $0 \leq \alpha_i \leq C$ is simply replaced by $0 \leq \alpha_i$. Readers are suggested to refer Chapter 4 and Chapter 5 in [54] for detail.

2.3 Nonlinear Support Vector Classification

2.3.1 Learning in Feature Space

The limitation of the above machines is that complex real-world applications require more expressive hypothesis space than linear functions. In other words, the target concept cannot be expressed as a simple linear combination of the given attribute, but in general requires that more abstract features of the data be exploited. Multiple layers of thresholded linear functions were proposed as a solution to this problem, and this approach led to the development of multi-layer neural networks and learning algorithms such as back-propagation for training such system.

Kernel representations offer an alternative solution by projecting the data into a high dimensional feature space to increase the computational power of the linear machines described in previous section. In the above optimization problems, training examples appear only in the form of dot product between pairs of individuals. By replacing the dot product with an appropriately chosen kernel function, we can implicitly perform a non-linear mapping from input space into a high dimensional feature space, and the maximal margin algorithms will run virtually in the feature space without knowing the map explicitly. The role of the kernel function in this situation is to calculate the dot product between two vectors in some (inner product) space, and linear learning algorithms works on this space indirectly via kernel function. More formally, kernel is defined as follows

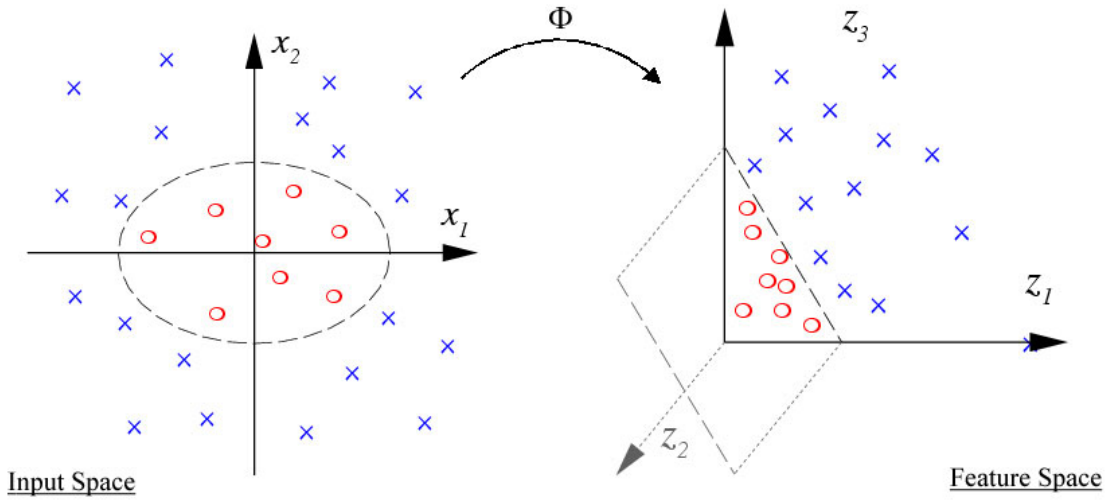


Figure 2.5: An illustration of kernel-based algorithms. By mapping the original input space to other high dimensional feature space, the linearly inseparable data may become linearly separable in the feature space.

Definition 3 *A kernel is a function K, such that for all $\boldsymbol{u}, \boldsymbol{v} \in X$*

$$K(\boldsymbol{u}, \boldsymbol{v}) = \Phi(\boldsymbol{u}) \cdot \Phi(\boldsymbol{v}) \tag{2.47}$$

where Φ is a map from input space X to an (inner product) feature space $\mathcal{F}$.

$$\Phi : \mathbb{R}^d \rightarrow \mathcal{F} \tag{2.48}$$

Once we have kernel function calculating dot product between two examples in feature space, we can find the optimal hyperplane in feature space by solving the following optimization

$$maximize_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j K(\boldsymbol{x_i}, \boldsymbol{x_j}) \tag{2.49}$$

$$subject\ to \quad \sum_{i=1}^{l} y_i \alpha_i = 0 \tag{2.50}$$

$$0 \leq \alpha_i \leq C, i = 1, ..., l \tag{2.51}$$

And the discriminant functions take the form

$$f(\boldsymbol{x}) = \sum_{\alpha_i \neq 0} y_i \alpha_i K(\boldsymbol{x_i}, \boldsymbol{x}) + b \tag{2.52}$$

In the next sections we will discuss several interested problems, such as what criteria make a function a kernel, and how can an inseparable training data becomes separable in feature space.

2.3.2 Kernels

How can we know that a two-variable function is a kernel or not? The Mercer's condition tells us whether a given function is actually a dot product in some space, thus working via this kernel enables maximal margin algorithms (as well as other algorithms) to work in feature space. This is essence of kernel-based algorithms.

Theorem 4 *(Mercer) To guarantee that a continuous symmetric function $K(u, v)$ in $L_2(C)$ has an expansion*

$$K(u, v) = \sum_{i=1}^{\infty} a_k z_k(u) z_k(v) \tag{2.53}$$

with positive coefficients $a_k > 0$ (i.e., $K(u, v)$ describes an inner product in some feature space), it is necessary and sufficient that the condition

$$\int_C \int_C K(u, v) g(u) g(v) du dv \geq 0 \tag{2.54}$$

is valid for all $g \in L_2(C)$ (C being a compact subset of $\mathbb{R}^d$)

In the following we examine two mostly used common type of kernels: polynomial and Gaussian Radial Basis function.

Polynomial kernels

Let's consider the quadratic homogeneous kernels acting on data in $\mathbb{R}^d$

$$K(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u} \cdot \boldsymbol{v})^2 \tag{2.55}$$

For $d = 2$, we can explicitly construct the map Φ from $\mathbb{R}^2$ to $\mathbb{R}^3$ as follows

$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \tag{2.56}$$

$$(u_1, u_2) \mapsto (u_1^2, \sqrt{2} u_1 u_2, u_2^2) \tag{2.57}$$

and $K(\boldsymbol{u}, \boldsymbol{v})$ is actually the dot product between two vectors $\Phi(u)$ and $\Phi(v)$ in $\mathbb{R}^3$ (note that with the same kernel function, we might have different ways to construct the map). For $d > 2$, we have the following relation

$$
\begin{aligned}
(\boldsymbol{u}\cdot\boldsymbol{v})^2 &= \left(\sum_{i=1}^{d} u_i v_i\right)^2 && (2.58)\\
&= \sum_{i=1}^{d}\sum_{j=1}^{d} u_i u_j v_i v_j && (2.59)\\
&= \sum_{(i,j)=(1,1)}^{(d,d)} (u_i u_j)(v_i v_j) && (2.60)\\
& && (2.61)
\end{aligned}
$$

which is equivalent to a dot product between two feature vectors

$$\Phi(\boldsymbol{u}) = (u_i u_j)_{(i,j)=(1,1)}^{(d,d)} \tag{2.62}$$

$$\Phi(\boldsymbol{v}) = (v_i v_j)_{(i,j)=(1,1)}^{(d,d)} \tag{2.63}$$

The number of features/dimensions of feature space in this case is $\begin{pmatrix} d+1 \\ 2 \end{pmatrix}$; all feature are monomials of degree 2. Generally, for both general homogeneous and inhomogeneous kernels

$$K(\boldsymbol{u},\boldsymbol{v}) = (\boldsymbol{u}\cdot\boldsymbol{v})^p \tag{2.64}$$

$$K(\boldsymbol{u},\boldsymbol{v}) = (\boldsymbol{u}\cdot\boldsymbol{v}+c)^p \tag{2.65}$$

The number of distinct features are $\begin{pmatrix} d+p-1 \\ p \end{pmatrix}$ and $\begin{pmatrix} d+p \\ p \end{pmatrix}$. The proof is given in detailed in [50].

Gaussian RBF kernels

The Gaussian kernels have the following form

$$K(\boldsymbol{u},\boldsymbol{v}) = e^{-\|\boldsymbol{u}-\boldsymbol{v}\|^2/2\delta^2} \tag{2.66}$$

where δ is the width of the function. In this case, dimensionality of feature space is infinite (we will discuss in more detailed in the next section), so it would not be easy to work with Φ explicitly. However, the maximal margin algorithm works in feature space by simply replacing $\boldsymbol{x_i}\cdot\boldsymbol{x_j}$ by $K(\boldsymbol{x_i},\boldsymbol{x_j})$ everywhere in the training algorithm, and the algorithm will produce a support vector machine which lives in an infinite dimensional feature space with roughly the same amount of time it would take to train on the original input space.

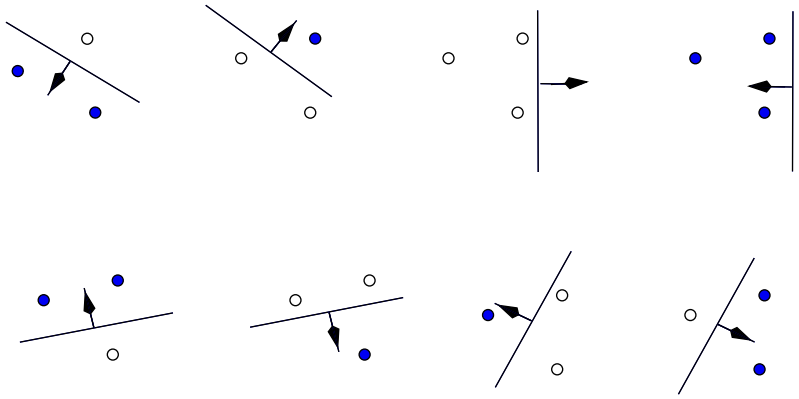


Figure 2.6: Three points in $\mathbb{R}^2$ shattered by oriented lines

2.3.3 VC Dimension and Generalization Ability of Support Vector Machine

Until now we have discussed the optimal hyperplane that maximizes its distance to the training data, the use of kernel to work in high or even infinite dimensional feature space. The question for this section is why the hyperplane does work better in a higher dimensional feature space; for example, why a data that is not linearly separable in the un-mapped input space becomes linearly separable in feature space (of course we cannot always say that working feature space ensures a higher generalization performance, and currently there exists no theory which guarantees that a given family of SVMs will have high accuracy on a given problem).

The VC (Vapnik-Chervonenkis) dimension is a measure of the capacity of a class of function $f(x, \alpha)$, e.g. a class of linear discriminant functions in our context. Given a set of l points, there are 2^l possible ways to assign them with label -1 or $+1$. For each labelling, if a member of the set $f(x, \alpha)$ can be found which correctly separates the points then we say that that set of points can be shattered by that set of function. The VC dimension for the set of function $f(x, \alpha)$ is defined as follows

Definition 4 *[3] The VC dimension of a set of indicator function $f(x, \alpha)$ is the maximum number h of vectors $x_1, ..., x_h$ that can be separated into two classes in all 2^l ways using functions of the set (e.g. the maximum number of vector that can be shattered by the set of functions). If for any n there exists a set of n vectors which can be shattered by the set $f(x, \alpha)$, then the VC dimension is equal to infinite.*

VC dimension plays a crucial role in a method to select the best suitable machine for a given task, the structural minimization principle (SRM) . Suppose that our task is to

learn the mapping $x_i \mapsto y_i$ by searching for a function $f(x, \alpha)$ where α is an adjustable parameter. For each particular choice of α, the expected error of the corresponding machine is

$$R(\alpha) = \int |y - f(x, \alpha)| dP(x, y) \tag{2.67}$$

The quantity $R(\alpha)$ is called the expected risk, or the actual risk, what we want to minimize. Besides, there is another risk called empirical risk $R_{emp}(\alpha)$ that measures the mean of error rate on the training data

$$R_{emp}(\alpha) = \frac{1}{l} \sum_{i=1}^{l} \frac{1}{2} |y_i - f(x_i, \alpha)| \tag{2.68}$$

The structure risk minimization principle suggests us to select the machine with minimum upper bound on generalization error, or the machine that keeps balance between data fitness and complexity (with $1 - \delta$ confidence).

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\frac{h(ln(2l/h) + 1) - ln(\delta/4)}{l}} \tag{2.69}$$

Now, let's coming back to our question: why our linear machine has higher capacity to work in feature space. The following theorem says that m points can be shattered if the remaining $m - 1$ points are linearly independent.

Theorem 5 *[50] Consider some set of m points in $\mathbb{R}^d$. Choose any one of the points as origin. Then the m points can be shattered by oriented hyperplanes if and only if the position vectors of the remaining points are linearly independent.*

The apparent corollary that could be draw from this theorem is that the VC dimension of the set of oriented hyperplane in $\mathbb{R}^d$ is $d + 1$ because we can always choose d linearly independent points, but not $d+1$. In the previous section we know that the dimensionality of feature space is very large, say $\begin{pmatrix} d+p-1 \\ p \end{pmatrix}$ for homogeneous polynomial kernels of degree p, or even infinite for radial basis function kernels. Thus by working in feature space via kernel, the (optimal) hyperplane has very high capacity.

Theorem 6 *[50] Consider the class of Mercer kernels for which $K(x_1, x_2) \to 0$ as $\|x_1 - x_2\| \to \infty$, and for which $K(x, x)$ is $O(1)$, and assume that the data can be chosen arbitrarily from $\mathbb{R}^d$. Then the family of classifiers consisting of support vector machines using these kernels, and for which the error penalty C is allowed to take all values, has infinite VC dimension.*

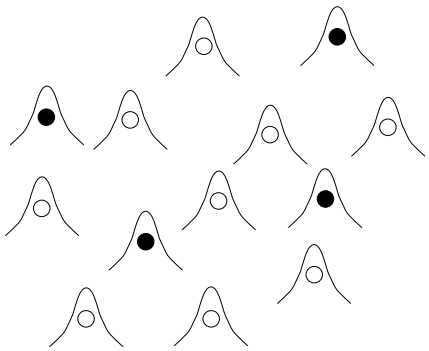


Figure 2.7: Gaussian RBF SVMs of sufficiently small width can classify an arbitrary large number of training points correctly, and thus have infinite VC dimension [50]

The detailed proof of this theorem (not very complicated) could be found in [50]. The main point is that we can choose training data such that all off-diagonal elements of the kernel matrix $K_{ij} = K(x_i, x_j)$ can be made arbitrary small, and because all diagonal elements K_{ii} are of $O(1)$, then the kernel matrix $\boldsymbol{K}$ is of full rank, or the set of vectors, whose dot products in the feature space form $\boldsymbol{K}$, are linearly independent. By theorem 5, the points can be shattered by hyperplanes in $\mathcal{F}$, and also by support vector machines with sufficiently large error penalty. Since this is true for any finite number of points, the VC dimension of these classifiers is infinite. An intuitive explanation is that for Gaussian RBF kernels, by choosing small enough RBF widths we can separate any l number of distinct training data. Illustration is provided in Figure 2.7.

2.4 Support Vector Regression

Suppose we are given a set of training data $S = \{(\boldsymbol{x}_i, y_i), i = 1, ..., l, \boldsymbol{x}_i \in X, y_i \in \mathbb{R}\}$, where X denotes space of the input pattern, e.g. $X = \mathbb{R}^d$ (the difference here is the target feature is in $\mathbb{R}$, not $\{-1, +1\}$ as in previous binary classification). The task of regression is to find a function $f : X \to \mathbb{R}$ that predicts the target value as accurate as possible. Because the target value is a real number, prediction of f can be tolerated an amount of θ from the true value, say, if the difference between predicted value and the true value is smaller than θ, then that prediction will not be considered as mistake. However, if we assess the training performance using the same θ, we are effectively using the real-valued regressors as classifiers and the worst case lower bounds on generalization performance apply (in two-class classification case, a training example with functional margin less than 1 is consider as an error though it might still be correctly classified when its functional margin is greater than 0). To avoid this we must allow a margin, called

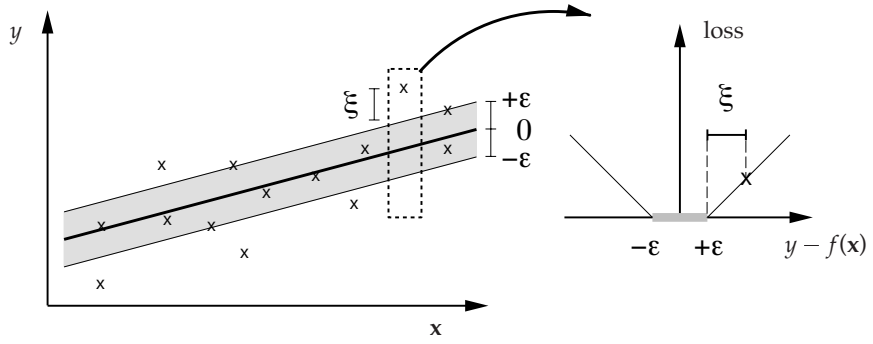


Figure 2.8: In ϵ-SV regression, training examples inside the tube of radius ϵ are not considered as mistakes. The trade-off between model complexity (or the flatness of the hyperplane) and points lying outside the tube is controlled by weighted ϵ-insensitive losses.

γ, in the regression accuracy that corresponds to the margin of a classifier, and we will use different loss functions during training and testing phases. In other words, a training example counts as a mistake if its accuracy is less than $\epsilon = \theta - \gamma$ (thus training tolerance is actually smaller than testing tolerance, or training condition is tighter than that in testing phase). In ϵ-SV regression, we define an ϵ-insensitive loss as follows

Definition 5 *The linear ϵ-insensitive loss of an example $(\boldsymbol{x}_i, y_i) \in (X, \mathbb{R})$ with respect to function is defined by*

$$L^{\epsilon}((\boldsymbol{x}_i, y_i), f) = |y_i - f(\boldsymbol{x}_i)|_{\epsilon} = \max(0, |y_i - f(\boldsymbol{x}_i)| - \epsilon) \tag{2.70}$$

where f is a real-valued function on domain X.

Similarly, the quadratic ϵ-insensitive loss is given by

$$L_2^{\epsilon}((\boldsymbol{x}_i, y_i), f) = |y_i - f(\boldsymbol{x}_i)|_{\epsilon}^2 \tag{2.71}$$

The margin slack variable of an example $(\boldsymbol{x}_i, y_i) \in (X, \mathbb{R})$ with respect to f, target accuracy θ, and loss margin δ is

$$\xi((\boldsymbol{x}_i, y_i), f, \theta, \delta) = \xi_i = \max(0, |y_i - f(\boldsymbol{x}_i)| - (\theta - \delta)) \tag{2.72}$$

Let's consider the following 1-norm bound on generalization performance.

Theorem 7 *Consider performing regression with linear functions $\mathcal{L}$ on an inner product space X, and fix $\gamma \leq \theta \in \mathbb{R}^+$. There is a constant c, such that for any probability distribution $\mathcal{D}$ on $X \times \mathbb{R}$ with support in a ball of radius R around the origin, with*

probability $1-\delta$ *over* l *random examples* S*, the probability that a hypothesis* $f(\boldsymbol{x}) = \boldsymbol{w}\cdot\boldsymbol{x}+b$ *has output more than* θ *away from its true value is bounded by*

$$err_{\mathcal{D}}(f) \leq \frac{c}{l}\left(\frac{\|\boldsymbol{w}\|_2^2 R^2 + \|\boldsymbol{\xi}\|_1^2 \log(1/\gamma)}{\gamma^2}\log^2 l + \log\frac{1}{\delta}\right) \tag{2.73}$$

where $\boldsymbol{\xi} = (\xi_1, ..., \xi_l)$ *is the margin slack vector with respect to* f*,* θ*, and* γ

The above theorem suggests that we can optimize the generalization of our regressor by minimizing the sum of the ϵ-insensitive losses

$$\frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{l} L^{\epsilon}((\boldsymbol{x}_i, y_i), f) \tag{2.74}$$

for some value of parameter C that measures the trade-off between complexity and losses. The equivalent primal optimization problem is as follows

$$\begin{aligned}
&\textit{minimize} && \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}(\xi_i^+ + \xi_i^-), && (2.75)\\
&\textit{subject to} && (\boldsymbol{w}\cdot\boldsymbol{x}_i + b) - y_i \leq \epsilon + \xi_i^+, && (2.76)\\
& && y_i - (\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \leq \epsilon + \xi_i^-, && (2.77)\\
& && \xi_i^+, \xi_i^- \geq 0, i = 1, ..., l && (2.78)
\end{aligned}$$

where two slack variables ξ^+ and ξ^- are introduced, one for exceeding the target value by more than ϵ, and the other for being more than ϵ below the target. The corresponding dual problem can be derived using standard techniques

$$\begin{aligned}
&\textit{maximize} && \sum_{i=1}^{l}(\alpha_i^- - \alpha_i^+)y_i - \epsilon\sum_{i=1}^{l}(\alpha_i^- + \alpha_i^+) && (2.79)\\
& && -\frac{1}{2}\sum_{i,j=1} l(\alpha_i^- - \alpha_i^+)(\alpha_j^- - \alpha_j^+)\boldsymbol{x_i}\cdot\boldsymbol{x_j}, &&\\
&\textit{subject to} && 0 \leq \alpha_i^+, \alpha_i^- \leq C, i = 1, ..., l && (2.80)\\
& && \sum_{i=1}^{l}(\alpha_i^- - \alpha_i^+) = 0, i = 1, ..., l &&
\end{aligned}$$

The corresponding Karush-Kuhn-Tucker complimentary conditions are

$$\alpha_i^+((\boldsymbol{w}\cdot\boldsymbol{x}_i+b)-y_i-\epsilon-\xi_i^+) = 0, \quad (2.81)$$

$$\alpha_i^-(y_i-(\boldsymbol{w}\cdot\boldsymbol{x}_i+b)-\epsilon-\xi_i^-) = 0, \quad (2.82)$$

$$\xi_i^+\xi_i^- = 0, \quad (2.83)$$

$$\alpha_i^+\alpha_i^- = 0, \quad (2.84)$$

$$(\alpha_i^+-C)\xi_i^+ = 0, \quad (2.85)$$

$$(\alpha_i^--C)\xi_i^- = 0, i=1,...,l \quad (2.86)$$

Substituting α_i for $\alpha_i^- - \alpha_i^+$ and $K(\boldsymbol{x_i}, \boldsymbol{x_j})$ for $\boldsymbol{x_i}\cdot\boldsymbol{x_j}$, we obtain the following proposition

Proposition 1 *Suppose that we wish to perform regression on a training samples* $S = \{\boldsymbol{x_i}, y_i)\}_{i=1,...,l}$ *using the feature space implicitly defined by the kernel* $K(\boldsymbol{u}, \boldsymbol{v})$*, and suppose the parameter* $\boldsymbol{\alpha}^*$ *solve the following quadratic optimization problem*

$$maximize \quad \sum_{i=1}^{l} y_i\alpha_i - \epsilon\sum_{i=1}^{l}|\alpha_i| - \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j K(\boldsymbol{x_i}, \boldsymbol{x_j}), \quad (2.87)$$

$$subject\ to \quad \sum_{i=1}^{l}\alpha_i = 0, \quad (2.88)$$

$$-C \le \alpha_i \le C, i=1,...,l \quad (2.89)$$

Let $f(\boldsymbol{x}) = \sum_{i=1}^{l}\alpha_i^* K(\boldsymbol{x_i}, \boldsymbol{x}) + b^*$*, where* b^* *is chosen so that* $f(\boldsymbol{x_i}) - y_i = -\epsilon$ *for any* i *with* $0 < \alpha_i^* < C$*. Then the function* $f(\boldsymbol{x})$ *is equivalent to the hyperplane in the feature space implicitly defined by the kernel* $K(\boldsymbol{u}, \boldsymbol{v})$ *that solves the optimization problem (2.79)*

2.5 Implementation Techniques

In previous sections we have shown that training a support vector machine is equivalent to solving a convex quadratic programming problem subject to a linear constraint. The problem of minimizing/maximizing a differentiable function of many variables has been widely studied, and most of the standard techniques can be directly applied to support vector training. However, these techniques are only suitable for small problems, or only suitable in some particular cases, e.g. most elements of the Gram matrix are zero. Unfortunately, the large training size is the main obstacle in the case of support vector training because just storing the kernel matrix requires a memory space that grows quadratically with the training size, hence easily exceeds the capacity of conventional computer when

the training size is large (e.g. with 30,000 training examples, the required memory for storing the whole kernel matrix is $30,000^2 \times 8/2 \approx 3GB$).

Among many particular algorithms designed for support vector training, we will briefly describe two methods that have been implemented in most of the commonly used SVM software, as well as in our implementation: the decomposition method and the sequential minimal optimization (SMO) algorithm.

Chunking and Decomposition

An important observation in training large scale SVM problem is the sparsity of the optimal solution. Depending on the problem, many of the α_i will be zero, or corresponding to inactive constraints in the primal problem. If we knew beforehand which α_i were zero, then we can remove the corresponding rows and columns from the kernel matrix without changing the value of the objective function. In other words, we can simplify the problem by discarding all of the inactive constraints. The *chunking* method starts with an arbitrary subset, or "chunk" of data, and train an SVM using a generic optimizer on that portion of data. The algorithm then retains the support vectors (those with corresponding $\alpha_i > 0$) from the chunk while temporally discarding the other points and then it uses the hypothesis found to test the points in the remaining part of the data. The points that most violate optimization condition, e.g. the KKT conditions, are added to the support vectors of the previous problem to form a new chunk. This procedure is iterated, initializing $\boldsymbol{\alpha}$ for each new sub-problem with the values output from previous stage, and optimizing sub-problem with a selected optimizer. The process will stop when the stopping condition is satisfied. The chunk of data being optimized at a particular stage is often referred to as the working set. The size of the working set varies, but is finally equals to the number of non-zero coefficients, or number of support vectors. This method assumes that the kernel matrix for the set of all support vectors fits in memory and can be fed to the optimization (we can alternatively recompute the kernel matrix every time when needed, but this becomes prohibitively expensive due to its frequently used). In practice, it can happen that the number of support vectors exceeds the capacity of computer. The decomposition methods overcome this difficulty by fixing the size of the subproblem. So every time a new point is added to the working set, another point has to removed. This allows to train arbitrary large datasets. However, the convergence of of this approach is very slow in practice. Practical implementations select several examples to add and remove from the subproblem plus efficient caching techniques to improve the efficiency. The general frame work for working set method is given in Table 2.1.

Table 2.1: Decomposition algorithm for SVM training.

Input:

a set S of l training examples $\{(x_i, y_i)\}_{i=1...l}$

size q of working set

Output:

a set of l coefficient $\{\alpha_i\}_{i=1...l}$

// Initialization

1. Set all α_i to zero
2. Select a working set B of size q

// Optimization

3. Repeat
4. Solve the local optimization on B
5. Update the working set B
6. Until the global optimization conditions are satisfied

Sequential Minimal Optimization Algorithm

The sequential minimal optimization (SMO) algorithm is the most extreme case of decomposition methods: it solves a quadratic optimization problem of size two in each iteration. The power of this algorithm is it gives analytical solution, thus quadratic optimizer is required. Based on the fact that the optimal solution has to satisfy the condition $\sum_{i=1}^{l} y_i\alpha_i = 0$, the SMO chooses two elements to jointly optimize in each iteration. Whenever one multiplier is changed, the other needs to be changed in order to keep the condition true. Because only two selected multipliers are involved in the optimization, the optimal update could be found analytically as follows.

Without loss of generality, assuming that the old values of two chosen elements are $(\alpha_1^{old}, \alpha_2^{old})$, and the new possible values of these two elements are $(\alpha_1^{new}, \alpha_2^{new})$. In order not to violate the condition $\sum_{i=1}^{l} y_i\alpha_i = 0$, the new values must lie on the line

$$y_1\alpha_1^{new} + y_2\alpha_2^{new} = y_1\alpha_1^{old} + y_1\alpha_1^{old} = constant \tag{2.90}$$

Fixing all other multipliers $\alpha_{i,i\neq1,i\neq2}$, the objective function can be rewritten as (detailed conversion can be found in [6])

$$L(\boldsymbol{\alpha}) = L(\alpha_2^{new}) = \frac{1}{2}\eta(\alpha_2^{new})^2 + (y_2(E_1^{old} - E_2^{old}) - \eta\alpha_2^{old})\alpha_2^{new} + constant \tag{2.91}$$

where $\eta = 2K_{12} - K_{11} - K_{22}, K_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $E_i^{old} = \sum_{k=1}^{l} y_k \alpha_k^{old} K(\boldsymbol{x}_k, \boldsymbol{x}_i) + b - y_i$. Note that E_i^{old} are prediction error on vector x_i with respect to the current solution, and the above objective function includes term $E_1^{old} - E_2^{old}$, so there is no need to calculate b for each iteration.

The objective function now becomes a one variable function of α_2^{new}. Its first and second derivatives are

$$\frac{dL}{d\alpha_2^{new}} = \eta\alpha_2^{new} + (y_2(E_1^{old} - E_2^{old}) - \eta\alpha_2^{old}) \tag{2.92}$$

$$\frac{d^2L}{d(\alpha_2^{new})^2} = \eta \tag{2.93}$$

Let $\frac{dL}{d\alpha_2^{new}} = 0$, we have

$$\alpha_2^{new} = \alpha_2^{old} + \frac{y_2(E_2^{old} - E_1^{old})}{\eta} \tag{2.94}$$

Because α_2^{new} must also satisfy the box constraint $0 \leq \alpha_2^{new} \leq C$, the new value of α_2 must be clipped to ensure a feasible solution

$$Low \leq \alpha_2^{new} \leq High \tag{2.95}$$

where

$$Low = \max(0, \alpha_2^{old} - \alpha_1^{old}) \tag{2.96}$$

$$High = \min(C, C - \alpha_1^{old} + \alpha_2^{old}) \tag{2.97}$$

if $y_1 \neq y_2$, and

$$Low = \max(0, \alpha_1^{old} + \alpha_2^{old} - C) \tag{2.98}$$

$$High = \min(C, \alpha_1^{old} + \alpha_2^{old}) \tag{2.99}$$

if $y_1 = y_2$. The new value of α_1 is obtained from α_2^{new} as follows

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2(\alpha_2^{old} - \alpha_2^{new}) \tag{2.100}$$

The heuristics for picking two α_i for optimization are as follows:

- The outer loop selects the first α_i, the inner loop selects the second α_j that maximize $|E_j - E_i|$.
- The outer loop first alternates between one sweep through all examples and as many as sweeps as possible through the non-boundary examples (those with $0 < \alpha_i < C$), selecting the example that violates the KKT condition.

- Given the first α_i, the inner loop looks for an example that maximizes $|E_j - E_i|$.

The advantage of SMO lies in the fact that solving for two Lagrangian multipliers can be done analytically. In practice, e.g. [10], [12], [11], SMO has been used to do optimization on the working set in the general decomposition framework in Table 2.1.

2.6 Summary

Support vector learning provides a new approach to classical problems like classification and regression. The solution of a support vector machine is unique for each parameter setting; this is a radical difference when comparing with other comparable learning approach such as neural networks. However, an SVM is largely characterized by the choice of its kernel, thus one of the biggest practical difficulty and the most time consuming task of this learning approach is the selection of the kernel. Currently, the best choice of kernel for a given problem is still a research issue. We will discuss more on this model selection problem in Chapter 5. Another limitation is speed and size in both training and testing phases. SVM training solves as optimization problem with quadratic requirement in memory. Training for million-size applications is still an unsolved problem. In testing phase, it seem to be that SVM is a kind of "lazy" machine where each testing pattern has to be compared with all support vectors. Despite of all these limitations, SVM has been emerging as a powerful learning approach and gaining success in many practical applications.

Chapter 3

Simplifying Support Vector Solutions

3.1 Introduction

Support Vector Machines (SVMs) [3, 54] have been demonstrated to be very robust in many applications, such as optical character recognition [56, 60], text categorization [55], face detection in images [42], and image denoising [61, 62]. The high generalization ability of SVMs is ensured by special properties of the optimal hyperplane that maximizes distance from it to the training patterns in a high dimensional feature space [2, 1, 3]. However SVMs are considerably slower in the test phase than other learning methods like decision trees or neural networks [36, 50, 48, 56, 60].

The solution of a SVM is parameterized by a set of input vectors, called support vectors (SVs), and their corresponding weights. When a new test sample is introduced, SVMs compare it with these SVs via kernel calculations; this computation becomes very expensive if the number of SVs is large. To reduce this computational complexity, reduced set methods, e.g., [36, 53, 38], try to approximate the original solution by another comprised by a much smaller number of newly constructed vectors, called the reduced vectors set. The former methods described in [36, 37, 53] start from approximating the solution comprised by all original SVs by only one new vector, and then incrementally construct the reduced set by finding vectors that minimize the differences between the original vector expansion and the reduced set expansion in feature space. This approach leads to the construction of each new vector required to solve an unconstrained optimization problem in a space of $d+1$ variables, where d is dimension of input space. Hence the computation is very expensive because the search must be repeated many times with different initial points to escape from local minimums [36, 61, 37]. Reduced vectors can also be selected from original SVs via kernel principal component analysis or L_1 shrinkage penalization as in [37], or by removing linearly dependent SVs in the feature space as method described in

[38]. In comparing with the previous reduced set construction methods, selection methods are less computationally expensive but also less effective in reducing the number of SVs and preserving generalization performance. The method described in [38] is based on the linearly dependency of SVs, so its applicability is very limited due to the fact that feature space's dimensionality is often very large or even infinite.

In this chapter we describe a new method to simplify support vector solutions in which the construction of new vectors only requires to find the unique maximum point of a one-variable function on (0,1). Instead of constructing reduced vectors set incrementally, two nearest SVs belonging to the same class will be iteratively considered and replaced by a newly constructed vector. This approach leads to a conceptually simpler and computationally less expensive method, the local extremum problem does not exist, and it also makes the vectors in the simplified solution look more meaningful (e.g. character-like in OCR applications). Experimental results on real life datasets show the effectiveness of our proposed method in reducing the number of support vectors and preserving generalization performance. On the US Postal Service (USPS) handwritten digit recognition database, a 91.3% (for polynomial kernel) and 90.0% (for Gaussian kernel) reduction rate were achieved, with a corresponding 0.2% and 0.3% loss in predictive accuracy. On the MNIST database, the numbers were 88.6% and 0.1%. The reduction rates on other four datasets in the StatLog collection were from 70.9% to 94.5% with almost no change in performance.

This chapter is organized as follows. We will briefly describe the SVM simplification problem and review the earlier reduced set methods in Section 3.2. In Section 3.3 we describe our proposed method, which constructs a new vector to replace two other support vectors, and the iterative process to simplify support vector solutions. Experiments on real world databases are described in Section 3.4. Section 3.5 discusses the results.

3.2 Simplifying Support Vector Machines

In this section we first briefly introduce the simplification of SVMs and then review former methods for reducing number of necessary SVs included in support vector solutions.

3.2.1 Reducing Complexity of SVMs in Testing Phase

SVMs work in feature space indirectly via a kernel function $K(x, y) = \Phi(x) \cdot \Phi(y)$ where $\Phi : \mathbb{R}^d \rightarrow F$ is a map from a d-dimensional input space to a possibly high-dimensional feature space [3]. For a two-class classification problem, the decision rule takes the form

$$y = sign\left(\sum_{i=1}^{N_S} \alpha_i K(x, x_i) + b\right) \tag{3.1}$$

where α_i are weights of support vectors x_i (for simplicity, we combine y_i into α_i, thus $\alpha_i < 0$ for negative SV i), x is the input vector needed to classify, $K(x, x_i) = \Phi(x) \cdot \Phi(x_i)$ is a kernel function calculating the dot product of two vectors $\Phi(x)$ and $\Phi(x_i)$ in the feature space, b is the bias, and N_S is the number of support vectors. The task of the SVMs training process is to determine all the parameters (x_i, α_i, b, N_S); the resulting x_i, $i = 1...N_S$ are a subset of the training set and are called *support vectors*.

The complexity of the computation (3.1) scales with the number of support vectors N_S. The expectation of N_S is bounded below by $(l-1)E(p)$, where $E(p)$ is the expectation of the probability of error on a test vector and l is the number of training samples [3]. Thus N_S can be expected to approximately scale with l. For practical applications like pattern recognition, this results in a machine that is considerably slower in the test phase than other systems [56, 60].

Reduced set methods try to approximate the normal vector Ψ of the separating hyperplane

$$\Psi = \sum_{i=1}^{N_S} \alpha_i \Phi(x_i) \tag{3.2}$$

expanded in images ($\Phi(x_i)$ is image of x_i under Φ) of input vectors $x_i \in \mathbb{R}^d$, $\alpha_i \in \mathbb{R}$, by a reduced set expansion

$$\Psi' = \sum_{i=1}^{N_Z} \beta_i \Phi(z_i) \tag{3.3}$$

with $N_Z < N_S$, $z_i \in \mathbb{R}^d$, $\beta_i \in \mathbb{R}$. To classify a new test point x, calculation (3.1) is replaced by

$$y = sign\left(\sum_{i=1}^{N_Z} \beta_i K(x, z_i) + b\right) \tag{3.4}$$

The goal of reduced set method is to choose the smallest $N_Z < N_S$, and construct the corresponding reduced set $\{(z_i, \beta_i)\}_{i=1...N_Z}$ such that any resulting loss in generation performance remains acceptable [36].

3.2.2 Reduced Set Construction

The method described in [36] starts by replacing the original expansion Ψ with the image of one input vector and its corresponding weight (z_1, β_1), and then iteratively finds

(z_{m+1}, β_{m+1}) so that their images approximate the complement vectors Ψ_m ($\Psi_0 = \Psi$)

$$\Psi_m = \sum_{i=1}^{N_S} \alpha_i \Phi(x_i) - \sum_{j=1}^{m} \beta_j \Phi(z_j) \tag{3.5}$$

Because in many situations it is impossible to find exactly z_m and β_m that make $\Psi_m = 0$, (e.g. the chosen kernel is a Gaussian RBF), z_m are vectors that minimize

$$\begin{aligned} \rho &= \|\Psi_{m-1} - \beta_m \Phi(z_m)\| && (3.6) \\ &= \left\| \left(\sum_{i=1}^{N_S} \alpha_i \Phi(x_i) - \sum_{j=1}^{m-1} \beta_j \Phi(z_j) \right) - \beta_m \Phi(z_m) \right\| && (3.7) \end{aligned}$$

When the first derivative of kernel K has been defined, the gradient of objective function $F = \rho^2/2$ can be computed. For example, assuming that $K(x, y)$ is a function of scalar $x \cdot y$:

$$\begin{aligned} \frac{\partial F}{\partial \beta_m} &= -\sum_{i=1}^{N_S} \alpha_i K(x_i \cdot z_m) + \sum_{j=1}^{m-1} \beta_j K(z_j \cdot z_m), && (3.8) \\ \frac{\partial F}{\partial z_{mk}} &= -\sum_{i=1}^{N_S} \alpha_i \beta_m K'(x_i \cdot z_m) x_{ik} + \sum_{j=1}^{m-1} \beta_j \beta_m K'(z_j \cdot z_m) z_{jk}, k = 1, ..., d && (3.9) \end{aligned}$$

In general, an unconstrained optimization technique is used to find the minimum of F. For a particular kind of kernel $K(x, y) = K(\|x - y\|^2)$ the fixed-point iteration method can be used to improve the speed of the finding. For example when the chosen kernel is a Gaussian $K(x, y) = \exp(-\gamma \|x - y\|^2)$, z_m can be found by iterating [49, 37, 53]

$$z_m^{(n+1)} = \frac{\sum_{i=1}^{N_x} \alpha_i \exp(-\gamma \left\| x_i - z_m^{(n)} \right\|^2) x_i}{\sum_{i=1}^{N_x} \alpha_i \exp(-\gamma \left\| x_i - z_m^{(n)} \right\|^2)} \tag{3.10}$$

where $N_x = N_s + m - 1$, and

$$\begin{aligned} (\alpha_1, ..., \alpha_{N_x}) &= (\alpha_1, ..., \alpha_{N_S}, -\beta_1, ..., -\beta_{m-1}) && (3.11) \\ (x_1, ..., x_{N_x}) &= (x_1, ..., x_{N_S}, -z_1, ..., -z_{m-1}) && (3.12) \end{aligned}$$

One drawback of the above methods is that they may suffer from numerical instability and get trapped in a local minimum of function F; to prevent this circumstance, the finding for each new vector must be repeated many times with different initial values [36, 61].

3.2.3 Reduced Set Selection

The idea of reduced set methods is that the null space of the Gram matrix $K_{ij} = (\Phi(x_i) \cdot \Phi(x_j))_{i,j=1...N_S}$ precisely tells us how many vector can be removed from an expansion while committing zero approximation error [37]. For example, when vectors $\Phi(x_i)$ are linearly dependent then any of the $\Phi(x_i)$ can be expressed in terms of the others. Hence, we may use the eigenvectors with eigenvalue 0 to eliminate certain SVs from any expansion in the $\Phi(x_i)$ [37], or this can be done using techniques from linear algebra like the row reduced echelon form [38].

However the dimensionality of feature space is usually very large or even infinite as in the case of Gaussian kernel, so there is no nonzero eigenvalue [63]. In 1999, Schoelkopf and coauthors proposed a methods in [37] to find coefficients β_j minimizing the error committed by replacing $\alpha_n\Phi(x_n), 0 \leq n \leq N_S$, with $\sum_{j\neq n}\beta_j\Phi(x_j)$

$$\rho(\boldsymbol{\beta}, n) = \left\| \alpha_n\Phi(x_n) - \sum_{j\neq n}\beta_j\Phi(x_j) \right\|^2 \tag{3.13}$$

By defining $\eta_j = 1$ for $j = n$, $\eta_j = -\beta_j/\alpha_n$ for $j \neq n$, (3.13) equals $|\alpha_n|^2 \left\| \sum_{j=1}^{N_S} \eta_j\Phi(x_j) \right\|^2$. Normalizing $\boldsymbol{\eta}$ to obtain $\boldsymbol{\gamma} := \boldsymbol{\eta}/\left\|\boldsymbol{\eta}\right\|$, hence $\gamma_n = 1/\left\|\boldsymbol{\eta}\right\|$, (3.13) is equivalent to the problem of minimizing

$$\rho(\boldsymbol{\gamma}, n) = \left|\frac{\alpha_n}{\gamma_n}\right|^2 (\boldsymbol{\gamma}\boldsymbol{K}\boldsymbol{\gamma}), \tag{3.14}$$

over $\|\boldsymbol{\gamma}\| = 1$, and we can recover the approximation coefficients β_j for $\alpha_n\Phi(x_n)$, i.e. the values to add to the $\alpha_{j,j\neq n}$ for leaving out $\alpha_n\Phi(x_n)$, as $\beta_j = -\frac{\alpha_n\gamma_j}{\gamma_n}$.

The fact is that $\boldsymbol{\gamma}\boldsymbol{K}\boldsymbol{\gamma}$) is minimized for the eigenvector with minimal eigenvalue. In that case, $\boldsymbol{\gamma}\boldsymbol{K}\boldsymbol{\gamma} = \lambda_{min}$. More generally, if γ^i is any normalized eigenvector of K, with eigenvalue λ_i, then

$$\rho(i, n) = \left|\frac{\alpha_n}{\gamma_n^i}\right| \lambda_i \tag{3.15}$$

Function 3.15 can be minimized by performing kernel PCA and scanning through the matrix $(\rho(i, n))_{in}$.

After choosing n, the original solution Ψ is approximated by

$$\Psi = \sum_{j\neq n}\alpha_j\Phi(x_j) + \alpha_n\Phi(x_n) \tag{3.16}$$

$$\approx \sum_{j\neq n}\left(\alpha_j - \frac{\alpha_n\gamma_j}{\gamma_n}\right)\Phi(x_j) \tag{3.17}$$

The selection process can be iterated until the expansion of Ψ is sparse enough. At each iteration all eigenvectors are computed using the Gram matrix computed from the SVs and then select n according to (3.15).

Another method for selecting a good subset of original SVs is to enforce the sparseness of the approximation which is inspired by L_1 shrinkage penalizers [64]. The given original expansion $\sum_i \alpha_i \Phi(x_i)$ will be approximated with $\sum_i \beta_i \Phi(x_i)$ by minimizing the following cost function

$$E(\boldsymbol{\beta}) = \left\| \sum_{i=1}^{N_S} \alpha_i \Phi(x_i) - \sum_{i=1}^{N_S} \beta_i \Phi(x_i) \right\|^2 + \lambda \sum_{i=1}^{l} c_i \left| \beta_i \right| \tag{3.18}$$

where λ is a constant determining the trade-off between sparseness and quality of approximation. The constants c_i are set to $\alpha/|\alpha_i|$ (α is the mean of all α_i) with an intention to shrink small terms [37].

To solve (3.18) it is necessary to remove the modulus by rewriting $\beta_i := \beta_i^+ + \beta_i^-$ with $\beta_i^{\pm} \geq 0$ and arriving at the following problem

$$\min_{\beta^+, \beta^-} \sum_{ij} (\beta_i^+ - \beta_i^-)(\beta_j^+ - \beta_j^-) K_{ij} + \sum_j \left(\beta_j^+ (\lambda c_j - 2 \sum_i K_{ij} \alpha_i) + \beta_j^- (\lambda c_j + 2 \sum_i K_{ij} \alpha_i) \right) \tag{3.19}$$

subject to

$$\beta_j^+, \beta_j^- > 0 \tag{3.20}$$

Problem (3.19) could be solved with standard quadratic optimization techniques, and their solution could be used directly as expansion coefficients.

Though still very complicated, the reduce set selection is less computationally expensive than the reduced set construction, but it performs practically worse [37]. In the next section we will introduce our proposed method that is conceptually simpler and computationally less expensive; experimental results indicated that new algorithm can reduce a big number of SVs while keeping well generalization performance.

3.3 A Bottom-up Method for Simplifying Support Vector Solutions

In this section we introduce a new method that iteratively replaces two support vectors belonging to the same class with a newly created vector. The simplification process could be considered as a bottom-up hierarchical clustering method and it will stop when an

estimated difference between the original solution and the simplified one exceeds a given threshold.

3.3.1 Simplification of Two Support Vectors

The solution of SVMs can be analyzed from a mechanical point of view: if each image of support vectors exerts a force $F_i = \alpha_i \hat{\Psi}$ on the decision hyperplane, then the SVMs solution satisfies the conditions of equilibrium: the sum of the forces and the torque all vanish ($\hat{\Psi}$ is the unit vector in the direction Ψ)[50]. In an equilibrium system, if we replace two member forces by an equivalent one, then the equilibrium state of the system will not change. In an SVM solution, if we replace two images $\Phi(x_i)$ and $\Phi(x_j)$ of two support vectors belonging to the same class x_i and x_j by a vector $M = m\Phi(x_i) + (1-m)\Phi(x_j)$, where $m = \alpha_i/(\alpha_i + \alpha_j)$ and weight vector M by $\alpha_m = (\alpha_i + \alpha_j)$, then for any point x in the input space, calculation (3.1) can be computed through $(N_S - 1)$ vectors:

$$y = sign\left(\sum_{k=1,k\neq i,k\neq j}^{N_S} \alpha_k K(x, x_k) + \alpha_m M \cdot \Phi(x) + b\right) \tag{3.21}$$

The difficulty is that M can not be used directly; we must use its pre-image (e.g. working via some input vector z that $\Phi(z) = M$), and in many situations, we cannot find the exact pre-image of M. For example, when a Gaussian RBF kernel is used, every point in the input space is mapped onto the surface of the unit hypersphere in feature space ($\Phi(x) \cdot \Phi(x) = 1$ for every input vector x). In this case, M lies on the segment connecting $\Phi(x_i)$ and $\Phi(x_j)$, or inside the hypersphere, and there is no pre-image of M. This problem was addressed in [53, 62] as the pre-image problem in kernel methods.

Rather than trying to find the exact pre-image, we will approximate M by an image $\Phi(z)$ of some input vector z. The optimal approximation can be made if we choose a vector z that gives a minimum value of $\|M - \Phi(z)\|^2$, or in other words, we have to solve the optimization problem:

$$\min_z \|M - \Phi(z)\|^2 \tag{3.22}$$

The following propositions will give us the way to find vector z efficiently. All that is required is to find the unique maximum point of a one-variable function on (0,1). The coefficient of z then can be calculated analytically.

Proposition 2 *For Gaussian RBF kernels $K(x, y) = \exp(-\gamma \|x - y\|^2)$, the 2-norm optimal approximation of $M = m\Phi(x_i) + (1-m)\Phi(x_j)$, $m = \alpha_i/(\alpha_i + \alpha_j)$, $\alpha_i\alpha_j > 0$, is the image of input vector z determined by*

$$z = kx_i + (1-k)x_j \tag{3.23}$$

where k is the maximum point of

$$f(k) = mC_{ij}^{(1-k)^2} + (1-m)C_{ij}^{k^2} \tag{3.24}$$

with $C_{ij} = K(x_i, x_j)$

Proof: For Gaussian RBF kernels, Φ maps each input vector onto the surface of the unit hypersphere in feature space, so we have $\|\Phi(z)\| = 1$ for every z, $\|M\|$ is a constant and can be calculated via $\Phi(x_i)$ and $\Phi(x_j)$. (3.22) is equivalent to

$$\max_z M \cdot \Phi(z) \tag{3.25}$$

For the extremum, we have $0 = \nabla_z(M \cdot \Phi(z))$. To get the gradient in terms of K, we substitute $M = m\Phi(x_i) + (1-m)\Phi(x_j)$ and $K(x,y) = \exp(-\gamma \|x-y\|^2)$ to get the sufficient condition

$$\begin{aligned} 0 &= \nabla_z(M \cdot \Phi(z)) \\ &= 2m\exp(-\gamma\|x_i - z\|^2)(x_i - z) + 2(1-m)\exp(-\gamma\|x_j - z\|^2)(x_j - z) \end{aligned} \tag{3.26}$$

leading to

$$z = \frac{\sum_{s=i,j} \alpha_s \exp(-\gamma\|x_s - z\|^2) x_s}{\sum_{s=i,j} \alpha_s \exp(-\gamma\|x_s - z\|^2)} \tag{3.27}$$

or

$$z = kx_i + (1-k)x_j \tag{3.28}$$

where

$$k = \frac{\alpha_i \exp(-\gamma\|x_i - z\|^2)}{\sum_{s=i,j} \alpha_s \exp(-\gamma\|x_s - z\|^2)} \tag{3.29}$$

Because $\alpha_i\alpha_j > 0$ (or x_i and x_j belong to the same positive or negative class) then $0 < k < 1$. (3.28) means that z always lies on the segment connecting x_i and x_j. To ease the finding of z we define $f(k) = M \cdot \Phi(z)$ and search for the maximum point of $f(k)$

$$\begin{aligned} f(k) &= M \cdot \Phi(kx_i + (1-k)x_j) \\ &= (m\Phi(x_i) + (1-m)\Phi(x_j)) \cdot \Phi(kx_i + (1-k)x_j) \\ &= m\Phi(x_i) \cdot \Phi(kx_i + (1-k)x_j) + (1-m)\Phi(x_j) \cdot \Phi(kx_i + (1-k)x_j) \\ &= m\exp(-\gamma\|x_i - x_j\|^2 (1-k)^2) + (1-m)\exp(-\gamma\|x_i - x_j\|^2 k^2) \\ &= mC_{ij}^{(1-k)^2} + (1-m)C_{ij}^{k^2} \end{aligned} \tag{3.30}$$

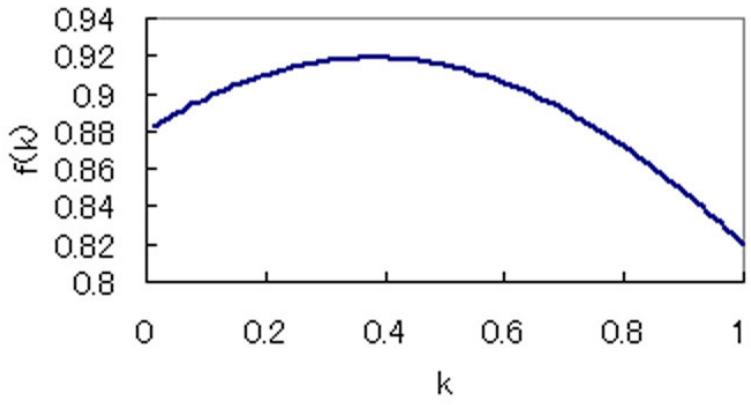


Figure 3.1: $f(k) = mC_{ij}^{(1-k)^2} + (1-m)C_{ij}^{k^2}$ with $m = 0.4, C_{ij} = 0.7$.

where $C_{ij} = \exp(-\gamma \|x_i - x_j\|^2) = K(x_i, x_j)$ ■

$f(k)$ is a one-variable function and has unique maximum point on $(0, 1)$ (as illustrated in Figure 3.1). The maximum point can be easily reached using common univariate parameter optimization methods. In our experiments, the inverse parabolic interpolation method [65] is used with three starting points $k = 0, k = m$ and $k = 1$, and the optimization process converges quickly after several iterations (if $m = 1/2$ then $k = m = 1/2$ is exactly the maximum point of $f(k)$).

Proposition 3 *For polynomial kernels $K(x, y) = (x \cdot y)^p$, the the 2-norm optimal approximation of $M = m\Phi(x_i) + (1-m)\Phi(x_j)$, $m = \alpha_i/(\alpha_i + \alpha_j)$, $\alpha_i\alpha_j > 0$, is the image of input vector z determined by*

$$z = \frac{\|M\|^{1/p}}{\|z^*\|} z^* \tag{3.31}$$

where $z^* = kx_i + (1-k)x_j$ and k is the maximum point of $h(k)$

$$h(k) = \|M\|\, u(k)v(k), \tag{3.32}$$

where

$$u(k) = \frac{1}{\left[x_i^2 k^2 + 2(x_i \cdot x_j)k(1-k) + x_j^2(1-k)^2\right]^{p/2}} \tag{3.33}$$

$$v(k) = m\left[x_i^2 k + (x_i \cdot x_j)(1-k)\right]^p + (1-m)\left[(x_i \cdot x_j)k + x_j^2(1-k)\right]^p \tag{3.34}$$

Proof: For polynomial kernels, Φ maps each input vector x lying on the surface of a hypersphere of radius r ($\|x\| = r$) onto the surface of a hypersphere of radius r^{2p} in the feature space ($(r^2+1)^p$ for inhomogeneous kernel $K(x, y) = (x \cdot y + 1)^p$). To approximate

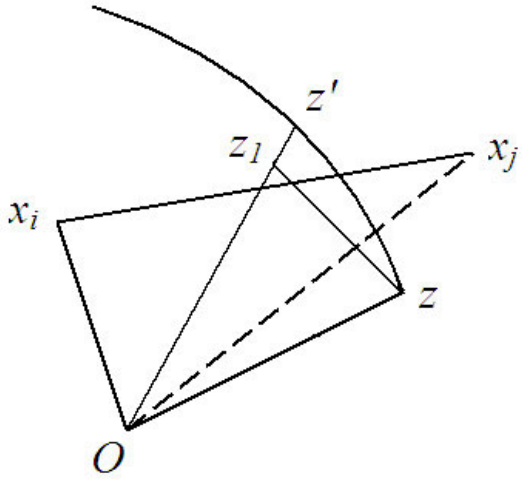


Figure 3.2: Projection of vector z on the plane (x_i, x_j) in the input space.

M by $\Phi(z)$ we can constrain $\Phi(z)$ to lay on the surface of the same hypersphere with M in feature space without any lost in generality. This is equivalent to constraining z to lie on the surface of the hypersphere of radius $\|M\|^{1/p}$ in the input space, and (3.22) becomes

$$\max_z M \cdot \Phi(z) \tag{3.35}$$

subject to

$$\|z\| = \|M\|^{1/p} \tag{3.36}$$

The following lemma shows that the (vector) solution of (3.35), x_i, and x_j are linearly dependent.

Lemma 1 *The input vector z that maximizes $M \cdot \Phi(z)$ in (3.35) is linearly dependent with x_i and x_j.*

Proof: Replacing $M = m\Phi(x_i) + (1-m)\Phi(x_j)$ into (3.35) we have

$$\begin{aligned} M \cdot \Phi(z) &= (m\Phi(x_i) + (1-m)\Phi(x_j)) \cdot \Phi(z) \\ &= m\Phi(x_i) \cdot \Phi(z) + (1-m)\Phi(x_j) \cdot \Phi(z) \\ &= m(x_i \cdot z)^p + (1-m)(x_j \cdot z)^p \end{aligned} \tag{3.37}$$

Suppose that z is an input vector satisfying constraint (3.36) and z_1 is the orthogonal projection of z on the plane determined by x_i and x_j (as described in Figure 3.2). Let's consider input vector z'

$$z' = \frac{\|z\|}{\|z_1\|} z_1 \tag{3.38}$$

We have z' satisfying constraint (3.36) and $x_i \cdot z' \geq x_i \cdot z$, $x_j \cdot z' \geq x_j \cdot z$, or $M \cdot \Phi(z') \geq M \cdot \Phi(z)$. This means that the optimal vector z_{opt} for maximizing $M \cdot \Phi(z)$ lies on the plane (x_i, x_j), or z_{opt} is linear dependent with x_i and x_j. ■

Because the solution of (3.35), called z_{opt}, lies on the plane (x_i, x_j) and $\|z_{opt}\| = \|M\|^{1/p}$, there exits a vector z^* and a scalar k such that

$$z^* = kx_i + (1-k)x_j \tag{3.39}$$

and

$$z_{opt} = \frac{\|M\|^{1/p}}{\|z^*\|} z^* \tag{3.40}$$

Call $g(z) = M \cdot \Phi(z)$, we have

$$\begin{aligned}
g(z_{opt}) &= M \cdot \Phi(z_{opt}) \\
&= (m\Phi(x_i) + (1-m)\Phi(x_j)) \cdot \Phi(z_{opt}) \\
&= m\Phi(x_i) \cdot \Phi(z_{opt}) + (1-m)\Phi(x_j) \cdot \Phi(z_{opt}) \\
&= m(x_i \cdot z_{opt})^p + (1-m)(x_j \cdot z_{opt})^p \\
&= m(\|x_i\| \, \|z_{opt}\| \cos(x_i, z_{opt}))^p + (1-m)(\|x_j\| \, \|z_{opt}\| \cos(x_j, z_{opt}))^p \\
&= \|z_{opt}\|^p \left(m \, \|x_i\|^p \cos^p(x_i, z^*) + (1-m) \, \|x_j\|^p \cos^p(x_j, z^*)\right) \\
&= \|z_{opt}\|^p \left[m \, \|x_i\|^p \left(\frac{x_i \cdot z^*}{\|x_i\| \, \|z^*\|}\right)^p + (1-m) \, \|x_j\|^p \left(\frac{x_j \cdot z^*}{\|x_j\| \, \|z^*\|}\right)^p\right] \\
&= \frac{\|z_{opt}\|^p}{\|z^*\|^p} \left[m(x_i \cdot z^*)^p + (1-m)(x_j \cdot z^*)^p\right]
\end{aligned} \tag{3.41}$$

Because z_{opt} satisfies (3.36) then $\|z_{opt}\|^p = \|M\|$. Replacing $z^* = kx_i + (1-k)x_j$ into (3.41) leads to

$$h(k) = \|M\| \, u(k)v(k) \tag{3.42}$$

where

$$u(k) = \frac{1}{\left[x_i^2 k^2 + 2(x_i \cdot x_j)k(1-k) + x_j^2(1-k)^2\right]^{p/2}} \tag{3.43}$$

$$v(k) = m\left[x_i^2 k + (x_i \cdot x_j)(1-k)\right]^p + (1-m)\left[(x_i \cdot x_j)k + x_j^2(1-k)\right]^p \tag{3.44}$$

■

$h(k)$ is also an one-variable function and has unique maximum points in $(0, 1)$ (corresponding to the unique vector z' in (3.38)). This means that the finding of each new vector in the reduced set is much easier and cheaper than that in former methods (in the space of $d+1$ variables with local minimums).

Proposition 4 *The optimal coefficient β for approximating $\alpha_m M = \alpha_i \Phi(x_i) + \alpha_j \Phi(x_j)$ by $\beta\Phi(z)$ is*

$$\beta = \frac{\alpha_m M \cdot \Phi(z)}{\|\Phi(z)\|^2} \tag{3.45}$$

Proof: Once we replace x_i and x_j by z, or approximate M by $\Phi(z)$ in the feature space, the difference between two solutions will be, for every input vector x

$$\begin{aligned} d(\beta) &= |\alpha_m M \cdot \Phi(x) - \beta\Phi(z) \cdot \Phi(x)| \\ &= |(\alpha_m M - \beta\Phi(z)) \cdot \Phi(x)| \end{aligned} \tag{3.46}$$

The difference will be minimized when $d(\beta)$ gets the minimum value. In (3.46) $d(\beta)$ can be minimized by minimizing $d_1(\beta) = \|\alpha_m M - \beta\Phi(z)\|$. Because $d_1(\beta)$ is a quadratic function of β, its minimum point is at

$$\beta = \frac{\alpha_m M \cdot \Phi(z)}{\|\Phi(z)\|^2} \tag{3.47}$$

■

Equation (3.45) is used to find the coefficient for one newly constructed vector. For the whole reduced vectors set, the following proposition is used to recompute all the coefficients to get a better approximation.

Proposition 5 ([37]) *The optimal coefficients $\boldsymbol{\beta} = (\beta_1, ..., \beta_{N_Z})$ for approximating $\Psi = \sum_{i=1}^{N_S} \alpha_i \Phi(x_i)$ by $\Psi' = \sum_{j=1}^{N_Z} \beta_j \Phi(z_j)$ (for linear independent $\Phi(z_1), ...,$ $\Phi(z_{N_Z})$) are given by*

$$\boldsymbol{\beta} = (\boldsymbol{K}^z)^{-1} \boldsymbol{K}^{zx} \boldsymbol{\alpha} \tag{3.48}$$

where $\boldsymbol{K}^z_{ij} = \Phi(z_i) \cdot \Phi(z_j)$ and $\boldsymbol{K}^{zx}_{ij} = \Phi(z_i) \cdot \Phi(x_j)$

Proof: We evaluate the derivatives of the distance in F

$$\frac{\partial F}{\partial \beta_m} = -\Phi(z_m) \left(\sum_{i=1}^{N_S} \alpha_i \Phi_{x_i} - \sum_{j=1}^{N_Z} \beta_j \Phi_(z_j) \right) \tag{3.49}$$

and set it to 0, we obtain

$$K^{zx}\alpha = K^z\beta \tag{3.50}$$

hence

$$\boldsymbol{\beta} = (\boldsymbol{K}^z)^{-1} \boldsymbol{K}^{zx} \boldsymbol{\alpha} \tag{3.51}$$

■

As mentioned in [37], (3.48) always gives the optimal coefficients to get a solution that is at least as good as the original one. In our experiments, (3.48) was used to recompute the final coefficients of all vectors in the reduced set after the iterative simplification process finished.

3.3.2 Simplification of Support Vector Solution

The simplification procedure iteratively replaces two support vectors (including newly created vectors) x_i and x_j by a new vector z using the method described in Section 3.3.1. This process can be viewed as a bottom-up hierarchical clustering procedure, and there are two problems we have to take into consideration. First, how to select a good pair of support vectors to simplify, and second, when the simplification process will stop.

Selection heuristic

In general, a pair of two support vectors that gives a minimum value of $d(\beta)$ in (3.46) will produce the minimum difference between two solutions (solutions at two consecutive steps). However, the cost of using this criterion is rather expensive because we have to try all possible pairs of support vectors and then evaluate (3.46) for each of them. Moreover, we are more concerned about the original solution and the final one, so the strictly good approximation of the solutions at every intermediate steps is not necessary. The alternative heuristic is based on the difference between two vectors $M = mx_i + (1 - m)x_j$ and $\Phi(z)$ in (3.22). For Gaussian RBF kernels, we can select x_i and x_j that give a maximum value of $C_{ij} = K(x_i, x_j)$ in (3.24) because the bigger the C_{ij}, the bigger the maximum value of $f(k)$, or smaller difference between M and $\Phi(z)$ ($f(k) = 1$ gives zero difference). This is equivalent to selecting two closest support vectors belonging to the same positive or negative class. Another interpretation for this selection heuristic is that we will try to approximate two Gaussian RBFs by one Gaussian RBF, and intuitively, the closer pair centers, the better approximation. This selection heuristic also be reasonably applied to polynomial kernels because the input vector z that maximizes $M \cdot \Phi(z)$ in (3.37) is linear dependent with x_i and x_j and the closer two vectors x_i and x_j (or smaller angle between two vectors x_i and x_j) will give a bigger maximum value of $M \cdot \Phi(z)$ (given fixed values of m, $\|x_i\|$, and $\|x_i\|$).

Stopping condition

The simplified solution is mostly different from the original one (except for linear kernels, or homogeneous quadratic polynomial kernels with a number of SVs greater than the dimension of input space [36]), so the simplification of support vector solutions will possibly cause a degradation in generalization performance. In the formed methods there is no specific way to manage this possibility [40]; instead, the size of the reduced set is first specified, and the resulting accuracy loss is determined experimentally [36].

To control this circumstance, we can monitor the difference between the two solutions

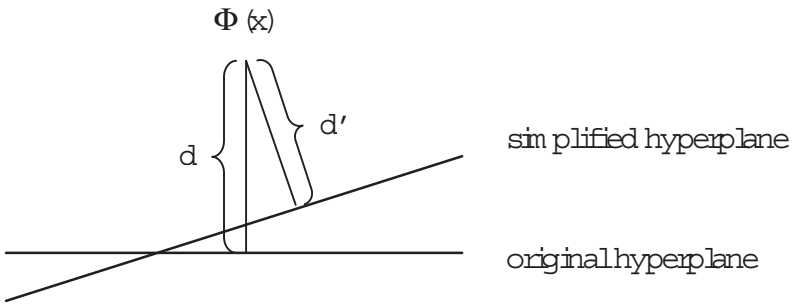


Figure 3.3: Illustration of the marginal difference of a (original) support vector x with respect to the original and simplified solutions

caused by the simplification process, and the simplification process will stop when any replacement of two SVs by a new one makes the difference exceed a given threshold. In the following we define a quantity called *Maximum Marginal Difference* (*MMD*) to estimate the difference between two support vector solutions.

Definition 6 *Suppose that the distance from a point $\Phi(x)$ to the original optimal hyperplane is d (d is 1 when x is a non-bounded support vector), and to the new hyperplane determined by the simplified solution is d'. The Marginal Difference (MD) on $\Phi(x)$ regarding to the two solutions is*

$$MD(\Phi(x)) \stackrel{def}{=} |d - d'| \tag{3.52}$$

and the difference between two solutions is defined as

$$MMD \stackrel{def}{=} \max_{i=1...N_S} MD(\Phi(x_i)) \tag{3.53}$$

where $x_1, ..., x_{N_S}$ are original support vectors.

The *MMD* uses the differences between two distances from the image of original support vectors to the two discriminant hyperplanes to estimate the difference between two support vector solutions. The reason for not using the difference between two normal vectors of the two hyperplanes $||\Psi - \Psi'||$ is that this quantity depends too much on $||\Psi||$ and $||\Psi'||$. For complicated problems ($||\Psi||$ is large), a small difference between two hyperplanes may cause a big difference $||\Psi - \Psi'||$, while for easy cases, a small $||\Psi - \Psi'||$ corresponds to a big difference between hyperplanes, so there is a big difference between the two solutions.

Table 3.1: The simplification algorithm

Input:

a set of N_S support vectors $x_1, ..., x_{N_S}$

a threshold θ of MMD

Output:

a set of N_Z reduced support vectors, $N_Z < N_S$

1. $PairList = \left\{(x_i, x_j) | i = 1...N_S, x_j = \arg\min_k(\|x_i - x_k\|^2), 1 \leq k \leq N_S, \alpha_i \alpha_k > 0\right\}$
2. Sort $PairList$ incrementally according to the distance between two vectors in pair
3. $PairID = 1$
4. Repeat
5. Call (x_i, x_j) the pair number $PairID$ in $PairList$
6. Try to replace x_i and x_j by z found by (3.23) or (3.31), weight z by (3.45)
7. If the replacement does not make MMD greater than θ
8. Then replace x_i and x_j by z, set $N_Z = N_Z - 1$, update $PairList$, set $PairID = 1$, and restart the loop
9. Otherwise set $PairID = PairID + 1$
10. Until all pairs in $PairList$ have been tried
11. Recompute coefficients of all support vectors using (3.48)
12. Optimize the whole reduced set using phase 2 described in Section 3.3.3
13. Return the reduced set

One note on the implementation of calculating $MD(\Phi(x_i))$ is that whenever two support vectors (v_1, α_1) and (v_2, α_2) are replaced by a vector (v, α), the marginal difference on $\Phi(x_i)$ will change an amount of $\alpha_1 K(v_1, x_i) + \alpha_2 K(v_2, x_i) - \alpha K(v, x_i)$(ref. (3.21)); therefore, during the simplification process the marginal differences on the original support vectors can be calculated accumulatively using only three vectors.

The Algorithm

The algorithm for simplifying support vector solution is described in Table 3.1. It iteratively selects two support vectors belonging to the same class and tries to replace them by a newly created vector. The process will stop when no replacement success, and finally all coefficients and reduced vectors are recomputed to get a better approximation.

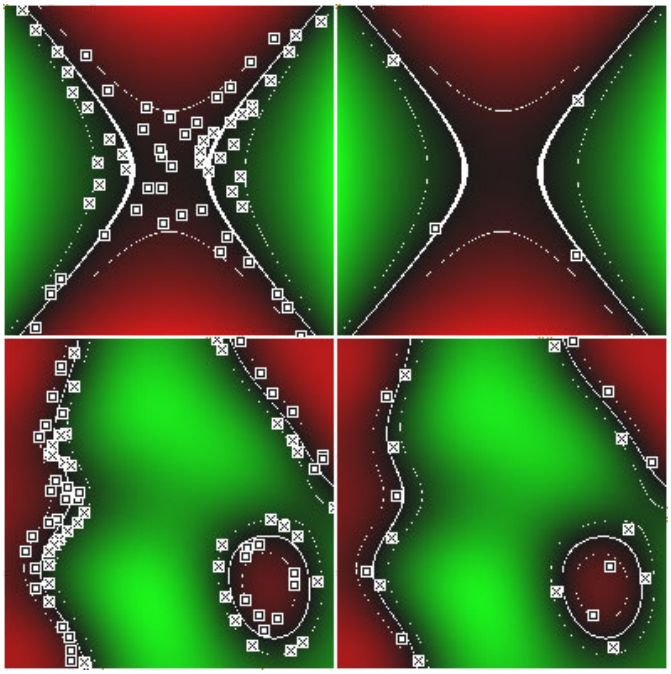


Figure 3.4: Illustration of simplified support vector solution using proposed method. The decision boundaries constructed by the simplified machines with 4 SVs (right-top) and 20 SVs (right-bottom) are almost identical with those constructed by the original machines with 61 SVs (left-top) and 75 SVs (left-bottom). The cracked lines represent vectors with approximately 1 marginal distance to the optimal hyperplane.

3.3.3 Pursuing a Better Approximation

A better approximate solution can be achieved by applying the unconstrained optimization process to minimize $F = \|\Psi - \Psi'\|$ with respect to all z_j and β_j together (phase 2 in [36]). Though the cost is high (working in a space of $(d+1)N_Z$ variables), this process can bring an effective reduction in the objective function F, or effective improvement of the simplified solution.

To illustrate how the proposed method works, in Figure 3.4 we show the results on two 2-dimensional datasets. The decision boundaries made by simplified machines, with much smaller number of vectors, are almost identical with those made by the original machines. Each reduced vector is constructed from and represents closed vectors in the same class.

3.4 Experiment

To assess its effectiveness on real world applications, we first applied proposed method to simplify ten binary classifiers trained to distinguish one digit from others in the US Postal

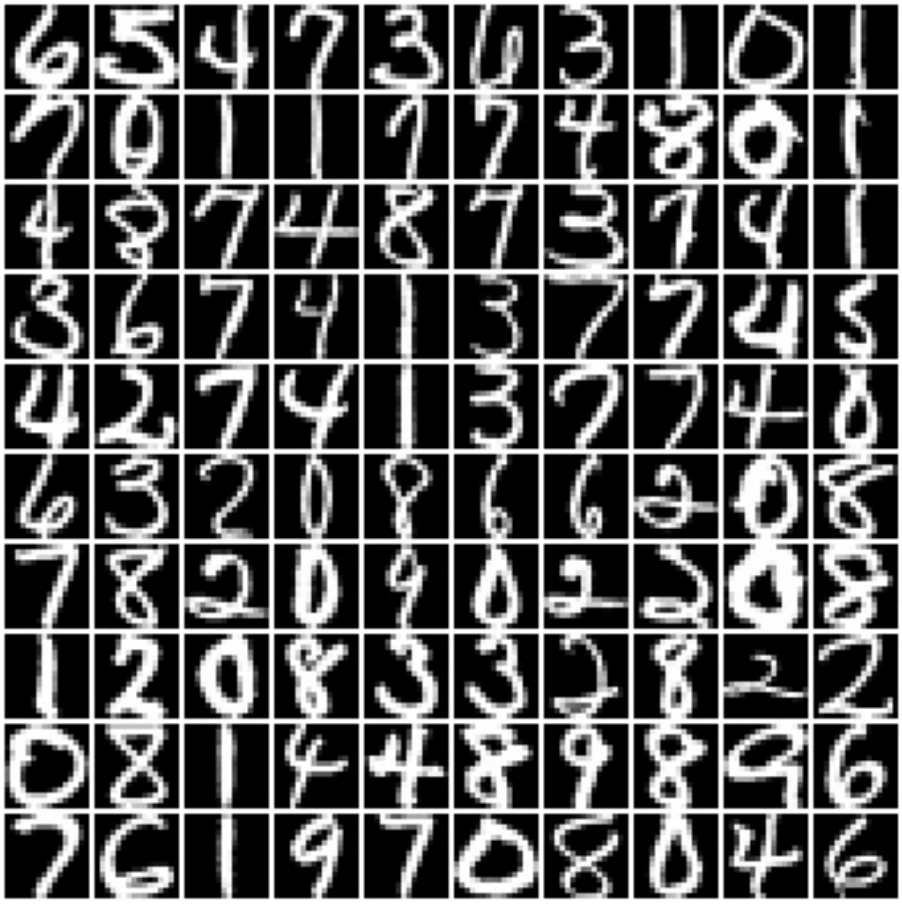


Figure 3.5: The first 100 digits in the USPS dataset

Service (USPS) handwritten digit recognition database. The dataset contains normalized grayscale images of handwritten digits taken from US zip codes; the size of each image is 16x16 pixels, and the data set is divided into a training set of 7,291 images and a test set of 2,007 images. For each binary classifier (using the one-versus-rest strategy) trained by a Gaussian RBF kernel or by a polynomial kernel, different values of MMD were used to give a different reduction rate in the number of SVs as well as different levels of the loss in generalization performance. The results are reported in Table 3.2. The first column displays different values of threshold MMD ($MMD = 0.0$ for original machines). The second column displays the total number of SVs in all ten binary classifiers. There are two kinds of errors. The first, named "Phase 1 Errors", were produced by the simplified classifiers using the simplification process described in Section 3.3.2 (phase 1), and the second, named "Phase 2 Errors", were produced by those using the optimization process described in 3.3.3 (phase 2) after phase 1 finished. For both kernels we could reduce the number of SVs by more than 90% with only a minor loss in generalization performance.

Note that the achieved reduction rate depends on the "*complexity*" of the solution, or

Table 3.2: Reduction in number of support vectors and the corresponding loss in generalization performance with different values of *MMD*. Original machines (the 3rd and 14th lines) were trained on the USPS training data using Gaussian and polynomial kernels. Errors were evaluated on the testing data.

RBF machines: $gamma = 0.0078$, $C = 10$			
MMD	# of SVs	Phase 1 Errors	Phase 2 Errors
0.0	5041	88(4.4%)	88(4.4%)
0.1	3476	85(4.2%)	88(4.4%)
0.2	2588	88(4.4%)	87(4.3%)
0.5	1285	91(4.5%)	90(4.5%)
0.7	864	97(4.8%)	94(4.7%)
1.0	502	108(5.4%)	95(4.7%)
1.2	343	124(6.2%)	97(4.8%)
1.5	246	144(7.2%)	101(5.0%)
Polynomial machines: $degree = 3$, $C = 10$			
MMD	# of SVs	Phase 1 Errors	Phase 2 Errors
0.0	4538	88(4.4%)	88(4.4%)
0.1	3024	88(4.4%)	88(4.4%)
0.2	2269	91(4.5%)	88(4.4%)
0.5	1114	93(4.6%)	89(4.4%)
0.7	795	104(5.2%)	89(4.4%)
1.0	522	110(5.5%)	91(4.5%)
1.2	397	116(5.8%)	93(4.6%)
1.5	270	147(7.3%)	95(4.7%)

Table 3.3: Experimental results on 45 binary classifiers learned from the USPS dataset using the first phase of the proposed method. Left-bottom: number of support vectors in original classifiers/number of vectors in simplified classifiers. Right-top: number of errors on the test data of original classifiers - simplified classifiers.

digit	1	2	3	4	5	6	7	8	9	0
1		4-4	3-3	6-6	4-4	6-6	4-4	4-4	4-4	4-4
2	39/7		10-10	10-10	**6-5**	7-7	3-3	**13-14**	5-5	9-9
3	45/2	140/55		2-2	17-17	4-4	7-7	9-9	**4-3**	8-8
4	63/3	161/65	72/9		5-5	**9-8**	9-9	4-4	**12-13**	3-3
5	54/2	148/42	**178/75**	120/37		2-2	4-4	9-9	3-3	8-8
6	38/7	127/30	**85/2**	100/27	123/42		2-2	2-2	0-0	2-2
7	36/3	88/18	69/15	88/16	82/12	62/2		3-3	12-12	5-5
8	50/8	127/45	133/34	102/33	131/51	89/9	82/24		3-3	**10-11**
9	60/2	83/9	79/15	131/21	90/26	72/2	166/50	97/17		3-3
0	35/2	119/17	96/12	81/13	143/49	111/30	55/10	99/11	73/2	

the difficulty of the problem. To judge this argument we conducted a second experiment on 45 binary classifiers trained to distinguish one digit from another in the USPS dataset (one-versus-one strategy). Chosen kernels were Gaussian RBFs $K(x, y) = \exp(-\gamma \|x - y\|^2)$ with the value of γ varied from 0.001 to 0.01 on a step of 0.002. The cost parameter C was fixed at 10. For each classifier, model selection consisted of varying γ and selecting the smallest value (or the simplest model) that gives the minimum train error (the train errors of these classifiers are almost zero, except for some mislabeling training examples).

The results reported in Table 3.3 show that the highest reduction rate achieved is 97.6% (83/85) on the classifier distinguishing character '3' from character '6', and the lowest rate is 57.9% (103/178) on the classifier distinguishing character '3' from character '5'. The difference in generalization performance on all these machines is almost zero (there were 6 differences on total 2007*9 tests).

To evaluate the performance on different applications, we conducted experiments on five other datasets: the MNIST database of handwritten digits [1], four datasets DNA,

[1]Available at http://yann.lecun.com/exdb/mnist/

Table 3.4: Experimental results on various applications.

Dataset	Dim.	# of Classes	Size		Original Machines		Simplified Machines	
			Train	Test	# of SVs	Error (%)	# of SVs	Error (%)
MNIST	784	10	60,000	10,000	22,294	1.5	2,538	1.6
Dna	180	3	2,000	1,186	1,686	6.0	93	6.4
Letter	16	26	15,000	5,000	10,284	5.0	2,993	5.2
Satimage	34	6	4,435	2,000	2,494	10.9	354	10.9
Shuttle	9	9	43,500	14,500	1,131	0.2	124	0.2

Letter Recognition, Satimage, and Shuttle in the StatLog collection [2]. These datasets are summarized in Table 3.4. Parameter settings for these datasets were polynomial kernel of degree five for the MNIST, Gaussian kernels with the width of $\frac{1}{0.6Variance}$ [60] for the StatLog datasets; the parameter MMD was fixed at 1.0. Experimental results in Table 3.4 show that with almost no change in generalization performance the achieved reduction rates could vary from 70.9% to 94.5% (corresponding to a speed-up rate from 3.4 to 18.2 times) depending on application.

3.5 Discussion

We have described a method to reduce the computational complexity of support vector machines by reducing the number of support vectors comprised in their solution. Our method has several advantages compared to the earlier reduced set methods. Firstly, the reduced vectors are constructed in a more *"natural"* way, leading to a more "meaningful" reduced set. Each vector in the reduced set could be considered as representative of several closed original SVs belonging to the same class. In Figure 3.7 we display the whole reduced vector set of 10 simplified classifiers. Each reduced vector is constructed from the same positive or negative close original SVs, so the original shape of these SVs is preserved. This is quite different from the former reduced set methods that construct each new vector from all original and newly constructed SVs. From Figure 3.7 we can also see that different machine requires a different number of reduced SVs. For example, the machine distinguishing character '1' from the other consists of only 6 SVs, while this number is 43

[2]Available at http://www.liacc.up.pt/ML/

SVs for machine '8'. This indicates that it is unreasonable to decide the same number of reduced SVs for all machines. The second advantage lies in the uniqueness of the result in finding the reduced set. With our proposed method, each reduced vector corresponds to the unique maximum point of a one-variable function on $(0, 1)$, and the result of the finding (for both two phases) is unique because we start from the same initial point and use the same search strategy. All the results described in this paper can be reproduced easily with a one-run test. Reproduction is difficult and very expensive, if not impossible, for the former methods because for each reduced vector they have to solve a multivariate parameter optimization problem, and the search has to restart many times with different initial points. For the second phase optimization, as noted in [37], the optimization also must be restarted to make sure that the global minimum of the cost function is actually found. The third advantage is its competitive SVs reduction rate while preserving well machine's performance. Experiments on the USPS dataset show that a reduction rate of 90.0% can be achieved with only a 0.3% loss in predictive accuracy (Gaussian kernel, $MMD = 1.0$), and 91.3% with a 0.2% lost (polynomial kernel, $MMD = 1.2$). The corresponding numbers reported in [37] are (for Gaussian kernel) 90% reduction rate with 0.3% loss (we report the reduction rate, not the number of SVs reduced because we did experiments on non-processed datasets and the total number of SVs were different).

The proposed method is applicable for common kernels like Gaussian RBFs and polynomial, and for both support vector classification and regression machines. For a further speed-up, other approximation methods, e.g., [39, 40], can be applied together with the reduced set methods to accelerate the test phase of the support vector machines.

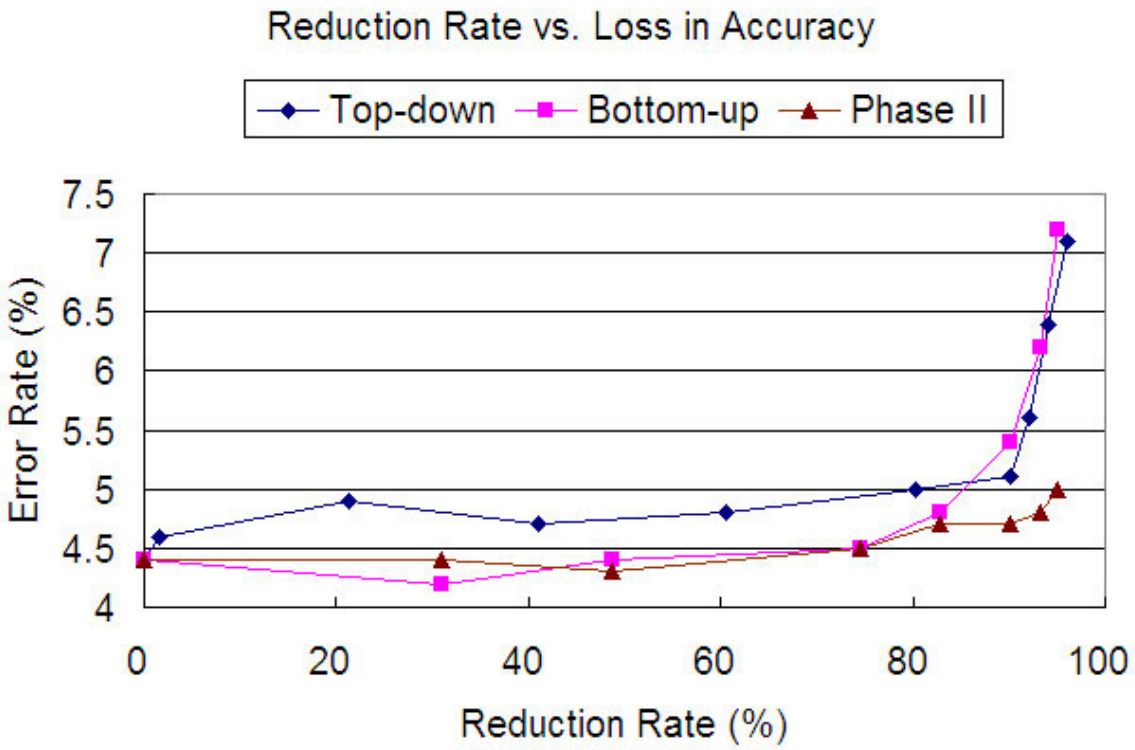


Figure 3.6: Performance comparison between the former top-down the the proposed bottom-up approach on the USPS dataset. With the same reduction rate the bottom-up preserved better predictive accuracy, while computational efficiency is guaranteed by theoretical result. Note: Top-down: the result of fix-point iteration method in [37] (Phase I); bottom-up: the result of proposed method (Phase I); Phase II: the result of proposed method running with both two phases optimization.

Figure 3.7: Display of all vectors in simplified solutions. The original 10 classifiers trained with polynomial kernel of degree three and the cost $C = 10$ consist of 4538 SVs and produce 88 errors (on 2007 testing data). The simplified 10 classifiers consist of 270 vectors and produce 95 errors. The number below each image indicates the new weight of a reduced vector.

Chapter 4

Speeding-up Support Vector Training in Model Selection

4.1 Introduction

In a tutorial paper on support vector machines (SVMs) for pattern recognition, Burges [50] pointed out three main limitations of the support vector learning approach. The biggest limitation lies in the choice of the kernel and its parameter setting. The second limitation is speed and size, in both the training and testing phases, and the third limitation is dealing with discrete data. Doing model selection for SVMs means we have to deal with the first two major limitations mentioned above.

In machine learning the model selection (MS) problem asks the following question [66], [67], [68]: given an observed data set, which model or learning algorithm running with which parameter setting will produce a model that performs best on unseen data? For SVMs, it is to select the most suitable kernel, its parameter value(s), and the appropriate error penalty on training error. To find the answer for this question, candidate models should be tried, and the model with the highest estimated performance will be selected. This is a very time consuming task because it requires multiple trials of training and testing models.

In this chapter we first introduce model selection problem, particularly for the support vector learning approach. We then describe our investigation on relation between two SVMs learned from the same training data using different kernels and parameter settings. By conducting intensive experiments on different datasets, we found/reconfirmed that every two SVMs share a big number of common support vectors. Based on this investigation, we propose a simple yet effective way to speed-up the training phase in MS process. The main idea is, in the sequence of trying models, the results of previous

trained machines are reused to train new machines. More concretely, the support vectors in previous trained machines are used to initialize the working set in training each new machine. This initialization of the working set helps to reduce the required number of optimization loops, so the optimization process can converge more quickly. Experimental results on three real-life datasets *sat-image*, *letter recognition*, and *shuttle* in the StatLog collection [69] show that the training time for each subsequent machine can be reduced as much as 85.5% depending on situations in the MS process. The method is applicable to common model search strategies like grid search [30], pattern search [31], or gradient-based methods [32], [70], and does not change the result of model selection.

The remainder of this chapter is organized as follows. In section 4.2, we describe the model selection problem for SVMs, and review several approaches to reduce its running time in section 4.3. Our proposed method for speeding up the training phase in model selection for SVMs is described in section 4.4. In section 4.5 we describe our experiments, and section 4.6 is a discussion.

4.2 Model Selection for Support Vector Machine

4.2.1 What is Model Selection

From a statistical point of view, models are sets of statistical hypothesis, e.g. [68], or predictive densities, e.g. [71], and model selection aims at the selection of approximately true models. For a more concrete definition of MS, let's consider the prediction problem in which a random observation $X \in \mathcal{X}$ is given and the task is to estimate $Y \in \mathcal{Y}$. A *predictive* model/rule is a measurable function $f : \mathcal{X} \to \mathcal{Y}$, with *loss* $L(f) = El(f(X), Y)$ where $l : \mathcal{Y} \times \mathcal{Y} \to [0, 1]$ is a bounded loss function. The data $D_n = (X_1, Y_1), ..., (X_n, Y_n)$ consist of a sequence of independent, identically distributed samples with the same distribution as (X, Y) and D_n is independent of (X, Y). The goal of MS is to choose a model f_n from some restricted class $\mathcal{F}$ such that the loss $L(f_n) = E\left[l(f_n(X), Y)|D_n\right]$ is as close as to the best possible loss, $L^* = \inf_f L(f)$, where the infimum is taken over all prediction rules $f : \mathcal{X} \to \mathcal{Y}$ [72].

Most models have their own parameters (except for non-parametric learning methods like nearest neighbor classification), and these parameters are not subject to change in training. For example: the number of neighbors k in k-nearest neighbor classification, number of hidden layers and units in each layer in a neural network, type of kernel, its parameter(s), and the error penalty in training a support vector machine. All these parameters must be set before we run a back propagation algorithm to minimize number of errors on training data by adjusting network's weights and thresholds in case of neural

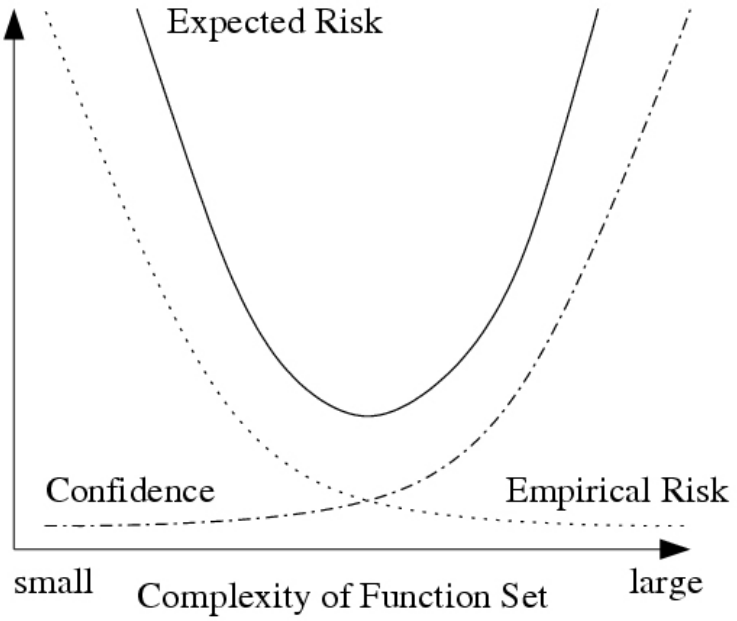


Figure 4.1: Relations among model complexity (horizontal axis), empirical risk (the dotted line), and expected risk (the solid line). The dash-dotted line is the upper-bound on the complexity term (confidence). [73]

network, or to minimize the objective function also on training data by adjusting the weight of support vectors in support vector learning. However, our ultimate goal is not to optimize the prediction on the given examples, but to optimize predictions on unseen data. By increasing the complexity of a model (e.g. in terms of VC dimension), we can achieve a perfect prediction on the given data, but this does not guarantee a good prediction on testing data. Figure 4.1 illustrates the relation between empirical risk, the loss achieved on training data, and the expected risk, or the actual loss what we desire to minimize. With higher complexity of function class $\mathcal{F}$ the empirical errors $L(f_n) = E\left[l(f_n(X), Y)|D_n\right]$ decrease but from a certain complexity of the function class the expected risk also increase. So the problem of model selection includes finding a good model evaluation/ ranking criterion and searching in model space.

Model Evaluation

There are two main approaches to the goodness estimation of a model [74]: the complexity-penalization methods that estimate a model based on its empirical loss and its complexity, and the hold-out testing methods that do the same thing by testing models on a pseudo data. The idea of complexity-penalization methods like minimum description length principle e.g. [75], structure risk minimization [3, 4], is that they prefer model that keeps

balance between training loss and model's complexity. For example. let's take again the bound (2.68). Our goal is to minimize $R[f]$, which can be achieve by obtaining small training error $R_{emp}[f]$ while keeping the function class as simple as possible (the second term in the right-hand side of inequality (2.68) that will be discussed in more detailed later).

The other most common approach is holdout-testing. The observed examples are partitioned into a pseudo training set $1, ..., k$ and a holdout test set $k+1, ..., t$. Models are learned on the pseudo training set and holdout test set is used to estimate the true errors. There are many variants method of this basic strategy, e.g. 10-fold cross validation, leave-one-out testing, bootstrapping, ect. Repeated testing in this manner does introduce some bias in the error estimates, but the results are still generally better than considering a single holdout partition [74].

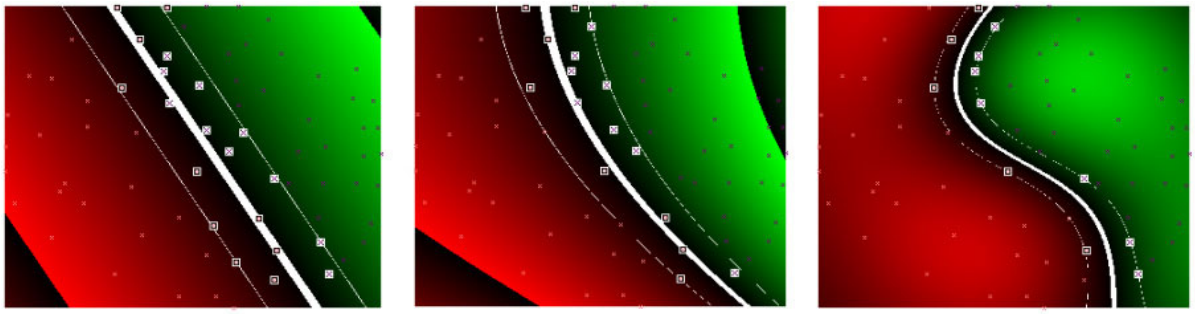
There are many other methods for evaluating a model like the method of maximum likelihood and classical hypothesis testing suggests model that has the greatest likelihood among competing models; Akaike's information criterion (AIC) and Bayesian information criterion (BIC) uses different estimations to evaluate a model based on its trade off between complexity and (training) data fitness [68]. Recently, new methods have been proposed that exploit the availability of unlabeled data in addition to the training data [74]. Also there has been a surge of interest in stability-based methods, e.g. [76]. These many different ways of evaluating models have their own advantages as well as weaknesses, and model evaluation is still a hard problem that does not have yet solutions that work across application domains and model families. There is no consensus right now about which approach works best.

Searching in Model Space

As stated above, most learning algorithm has its own parameter(s) and the changing of parameter setting affects prediction performance. Thus we do need a search strategy to explore parameter space. Exhaustive search is possible for discrete parameters, but often computationally impossible. Other search strategies include grid search, e.g. [30], pattern search [31], and all the common search techniques, when applicable (gradient descent, genetic algorithms, simulated annealing, greedy methods, relaxation techniques, etc.).

4.2.2 Model Selection for Support Vector Machines

Though possessing a very nice property that there is always an unique global optimum solution in training a SVM, selecting a suitable kernel function and its parameter(s), or



(a) (b) (c)

Figure 4.2: Different kernels produce different type of discriminant function.

adjusting the soft-margin parameter is outside of this nice optimization framework and requires an auxiliary technique. Figure 4.2 shows the difference in results of training a SVM with different type of kernels on the same dataset. One may have a linear classifier, or a polynomial function of some degree, or a Gaussian radial basis function, or a particular kind of two-layer sigmoidal neural network. The problem is that we do not know which machine is suitable for a given problem. Also, for each type of kernel function there is its own parameter. In case of Gaussian RBF kernel, parameter is the width of the function. By reducing the width, we can increase the complexity of corresponding machine [77]. In Figure 4.3, two different machines were trained on the same dataset using the same Gaussian kernel but with different values of width parameter and different values of the soft-margin parameter. The result was that SVM training algorithm produced a machine with 15 training errors and a machine with only one training errors. Also, we are not sure about which one will produce a better prediction on unseen data.

Beside general model evaluation methods described above, there have been a number of evaluation criteria specially designed for SVMs. The first one is the bound introduced in (2.68). In the second term on the right hand side

$$\sqrt{\frac{h(\log \frac{2l}{h} + 1) - \log(\delta/4)}{l}} \tag{4.1}$$

h is a non-negative integer called the Vapnik-Chervonenkis (VC) dimension that measures the capacity of a function class; l is the number of training examples. The main idea behind the bound (2.68) is that we want the model/function f with a minimum (true) risk on the left $R[f]$, but this is impossible. Instead, if we know h then we can easily compute the right hand side. Thus given several SVMs (or models), and choosing a fixed, sufficiently small δ, by then talking that machine which minimizes the right hand side, we are choosing that machine which give the lowest upper bound on the actual risk.

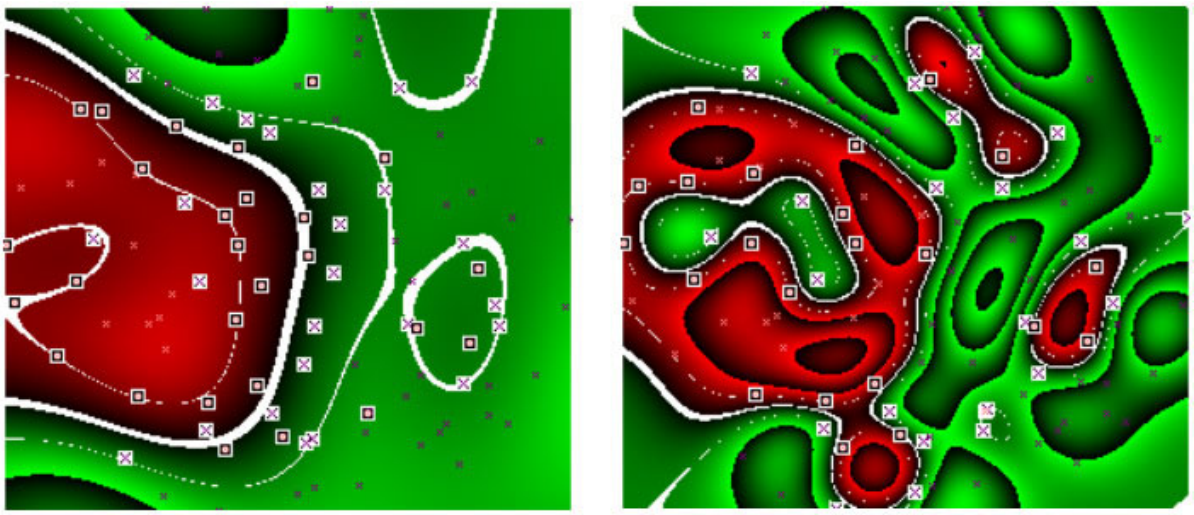


Figure 4.3: Trade off between model complexity and empirical risk.

This give a principled method for choosing a learning machine for a given task, and is the essence ideal of structural risk minimization. For computing h, readers are recommended to see [78]. In [4], another type of bound was obtained which demonstrated that for the separable case the expectation of probability of error for hyperplane passing though the origin depends on the expectation of ratio between radius R of the smallest hypersphere containing all SV in feature space and the margin $\|w\|$. More concretely the radius margin bound is

$$loo \leq 4R^2 \|w\|^2 \tag{4.2}$$

where *loo* is the number of leave-one-out cross validation errors. There are several more complicated method for approximating *loo* errors based on the concept of *span of support vectors* [79, 32].

Though there have been many criteria proposed as summarized above, k-fold cross validation is still one of the most widely used methods for performance evaluation and model selection, not only for SVMs but also for other learning approaches. In k-fold cross validation, the available dataset S is divided into k disjoint parts $S = S_1 \cup S_2 \cup ... \cup S_k$. Models are trained on the $k-1$ parts $S_1 \cup ... \cup S_{i-1} \cup S_{i+1} \cup ... \cup S_k$ and tested on the remaining part S_i. The performances are then averaged. Though the k-fold cross validation method is simple, consistent [80], almost unbiased [81], and works well in many applications [82]; the main drawback of k-fold cross validation is its high computational cost. For evaluating one model, 10-fold cross validation needs 10 times of training and

testing. In the next section we will discuss the problem of how to reduce this highly computational cost.

4.3 Speeding-up Model Selection SVM

4.3.1 Speeding-up by Improving Search Strategy

In order to speed-up the finding for the best model, a natural way is to apply a good search strategy working on model space. In support vector learning, model space consists of machines trained with different types of kernel, values of kernel's parameter, and a penalty constant C for each training error (e.g. for two-class classification machine). Thus the number of possible models is infinite. For example, if we consider Gaussian RBF kernel $K(x, y) = \exp(-\gamma \|x - y\|^2)$, the width γ of this kernel function takes value in R, so does the cost C. We cannot consider all possible pairs in the space R^2 in order to select the best model for our application. Grid search, e.g. [30] solves this problem by defining a grid of points in search space, and then all models trained with parameter values at these points are evaluated in order to find the best one. Usually the gird is not determined linearly but in a log scale to cover a large region. For example, if we transform the (γ, C) space into $(X = \log \gamma, Y = \log C)$ space, then the corresponding range $-2 \leq X \leq 10$ and $-2 \leq Y \leq 10$ will be $0.1353 \leq \gamma \leq 22026$ and $0.1353 \leq C \leq 22026$. In this type of search, the coarseness of the grid determines the quality of the solution found and the efficiency of the search. To avoid the explosion of number of models having to consider, the pattern search [31] could be a good solution. Beginning from a starting points in parameter space, pattern search investigates its neighbors and moves the focus point to one of them. The transition from one point to another is determined by a fix neighbor sampling pattern and the length of the search step. The length could be shrunk at each iteration until convergence is reached. To avoid local optimal as well as to increase robustness of the method, bagging or model averaging is suggested. Another way to reduce number of considered models is using gradient search when derivation of bounds could be calculated or approximated. In [32, 33], Chapelle and co-author proposed to use a gradient descent search to optimize the radius margin bound and span bound criteria. The limitation of this method is the need for a gradient computation which might be impossible or very difficult for general kernel.

4.3.2 Speeding-up by Improving Model Evaluation

Aside from improving search strategy, several methods have been proposed to quickly evaluate each considering model. In [70], authors proposed an efficient implementation for tuning SVM parameters using radius margin bound. Joachims [35] proposed the $\xi\alpha$-estimator to estimate generalization performance of a SVM without any computational intensive re-sampling. The $\xi\alpha$-estimator is much more efficient than cross validation because it can be computed immediately from the form of the hypothesis returned by the SVM. However, this estimator may not be differentiable, so it can not be used together with gradient search strategy.

Another approach to reduce computational cost of model selection is to apply a data filtering technique to reduce the size of the problem, e.g. [83]. However, the drawback of this approach is that the data filtering algorithm also has its own parameter(s), thus it is virtually required to solve another model selection problem.

4.4 Speeding-up SVM Training in Model Selection

All the above model selection methods require trying a series of models, or running the SVM training program many times with different values of parameters. In this section, we propose a simple yet effective way to speed-up the training phase for each considering model. The main idea is, in the process of trying a series of models, the support vectors in previously trained machines are used to initialize the working set in training a new machine. This initialization helps to reduce the number of required optimization loops, thus reducing the required running time.

4.4.1 The General Decomposition Algorithm for SVM Training

We would like to start introducing the decomposition framework for training a SVM. For convenience we rewrite the optimization problem as follows

$$minimize : W(\alpha) = -\sum_{i=1}^{l} \alpha_i + \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j K(x_i, x_j) \tag{4.3}$$

subject to:

$$\sum_{i=1}^{l} y_i \alpha_i = 0 \tag{4.4}$$

$$0 \leq \alpha_i \leq C, i = 1, ..., l \tag{4.5}$$

where l is the number of training examples, α is a vector of l variables where each component α_i corresponds to a training example (x_i, y_i). The solution of 4.3 is the vector α^* for which 4.3 is minimized and constraints 4.4 are satisfied.

Defining matrix Q as $Q_{ij} = y_i y_j K(x_i, x_j)$, the above optimization can be writtern as

$$minimize : W(\boldsymbol{\alpha}) = -\boldsymbol{\alpha}^T \mathbf{1} + \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} \tag{4.6}$$

subject to:

$$\boldsymbol{\alpha}^T \mathbf{y} = 0 \tag{4.7}$$

$$0 \leq \boldsymbol{\alpha} \leq C\mathbf{1} \tag{4.8}$$

This optimization works in a space of l variables $\alpha_i, i = 1...l$ with l^2 coefficients $Q_{ij}, i, j = 1...l$. Usually these coefficients need to be precomputed due to their very frequent use. For learning tasks with more than 10,000 training examples and more, it becomes impossible to keep Q in memory. An alternative would be to recomputed Q_{ij} whenever it is used, but this becomes very expensive.

The decomposition algorithm introduced in [5] suggests to split l training examples into two parts: active part B of free variable and inactive part N of fixed variables. Active variables are those which can be updated and inactive variables are those which temporally are fixed. Assuming that α, $\mathbf{y}$, and Q are properly arranged with respect to B and N, so that

$$\boldsymbol{\alpha} = \begin{pmatrix} \boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N \end{pmatrix} \tag{4.9}$$

$$\mathbf{y} = \begin{pmatrix} y_B \\ y_N \end{pmatrix} \tag{4.10}$$

$$Q = \begin{pmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{pmatrix} \tag{4.11}$$

Since Q is symmetric, we can write problem (4.6) as

$$minimize : W(\boldsymbol{\alpha}) = -\boldsymbol{\alpha}_B^T(\mathbf{1} - Q_{BN}\boldsymbol{\alpha}_N) + \frac{1}{2}\boldsymbol{\alpha}_B^T Q_{BB}\boldsymbol{\alpha}_B + \frac{1}{2}\boldsymbol{\alpha}_N^T Q_{NN}\boldsymbol{\alpha}_N - \boldsymbol{\alpha}_N^T\mathbf{1} \tag{4.12}$$

subject to:

$$\boldsymbol{\alpha}_B^T \mathbf{y}_B + \boldsymbol{\alpha}_N^T \mathbf{y}_N = 0 \tag{4.13}$$

$$0 \leq \boldsymbol{\alpha} \leq C\mathbf{1} \tag{4.14}$$

Because variables in N are fixed, term $\frac{1}{2}\boldsymbol{\alpha}_N^T Q_{NN}\boldsymbol{\alpha}_N$ and $\boldsymbol{\alpha}_N^T\mathbf{1}$ are constants. They can be omitted without changing the solution of (4.12). Problem (4.12) is also a quadratic programming problem with size smaller than (4.6). In the decomposition algorithm summarized in Table 2.1, a set of q variables are firstly selected for the working set B, remaining $l-q$ variables are fixed at their current values. In the main loop, an optimization technique works only on the working set B to find the local optimization solution. Usually, the sequential minimal optimization algorithm (SMO) [6, 7] is used to do this task. After that, working set B is updated by replacing vectors with zero coefficient with other vectors in N that does not satisfy the optimization conditions (the Karush-Kuhn-Tucker conditions). The process is repeated until there is no vector violating the KKT conditions, or the global optimization conditions are met. This common framework has been used in almost implementation of SVM learning like LibSVM [10], SVM^{light} [9], SVMTorch [11], HeroSVM [12]. To improve the efficiency some techniques like "shrinking" and "caching" are used.

It is clear that both initializing and updating working set B play an important role to the convergence of the decomposition algorithm. For example, if all SVs (those with the corresponding $\alpha_i > 0$) are selected in the first working set, then the main loop has to run just one time because the local optimized solution on B is also the global solution on the whole l training examples. In other words, a better initial working set with as many support vectors as possible will produce a local solution closer to the global solution, and will lead to a faster global convergence. However, in practice we don't know in advance which training vector will be the support vector, and in practice there is no way but to randomly initialize the working set, e.g. [12]. We can also increase the number of support vectors included in the initial working set by increasing the size of the working set, but the side effect is that this will also increase the local optimization time, and therefor will increase the training time [12]. Moreover, size of the working set is limited by available memory of computer, due to the requirement of kernel matrix.

4.4.2 Initializing Working Set

Our method starts with the fact that support vectors are training examples that lie close to the border between two classes (for two-class SVMs), and any two different machines may have many of them in common. This property had been reported in literature; for example, on the USPS hand-written digit dataset, two different machines trained by two different kernels, RBF and polynomial, share more than 80% of support vectors [4]. To reconfirm this argument we conducted intensive experiments on three datasets, *sat-image*, *letter recognition*, and *shuttle*, from the StatLog collection [69]. Details are reported in

Figure 4.4: Common support vectors in two different machines learned from three datasets *sat-image*, *letter recognition*, and *shuttle*: (a) linear machines learned with different error penalties $C = 1$ and $C = 2$, (b) polynomial machines of degree two and three learned with the same $C = 1$, (c) RBF machines learned with different error penalties $C = 1$ and $C = 2$.

Figure 4.4. The results of these experiments show that two different machines trained by different parameter settings share more than 86.0% number of support vectors. In model selection context, many machines must be tried, and we can benefit from using the information of previous trained models in training new ones.

One rather simple way is to select the SVs in trained machines as the initial working set for training a new machine. Especially, when two consecutive SVMs have the same kernel function, one solution can be reused directly to seed the search for another machine's solution, e.g. [84]. For example, supposed that training the first machine SVM_1 with kernel function K and error penalty $C = C_1$ resulted optimal solution $\boldsymbol{\alpha}^{SVM_1}$. To train a new machine SVM_2 with the same kernel function K and error penalty $C = C_2$, we can select all support vectors in SVM_1 into the first working set and initialize their corresponding Lagrangian coefficients

$$\alpha_i^{SVM_2} = C_2 \frac{\alpha_i^{SVM_1}}{C_1}, i = 1, ..., l \tag{4.15}$$

This initialization ensures that the new solution $\boldsymbol{\alpha}^{SVM_2}$ is a feasible solution of the optimization for SVM_2 training. For different kernels, we can just select support vectors into the working set and reset all corresponding Lagrangian coefficients α_i to zero. This method faces two difficulties in implementation. First, when the number of classes in the dataset is more than two, then the number of SVM required to build-up a classifier is m or $m * (m-1)/2$ binary classifiers depending on whether the selected strategy is one-versus-one or one-versus-rest, where m is the number of classes. In order to retain the information of previously trained machines, we need m or $m * (m-1)/2$ different sets of SVs. The second problem is that because the size of the working set is given in

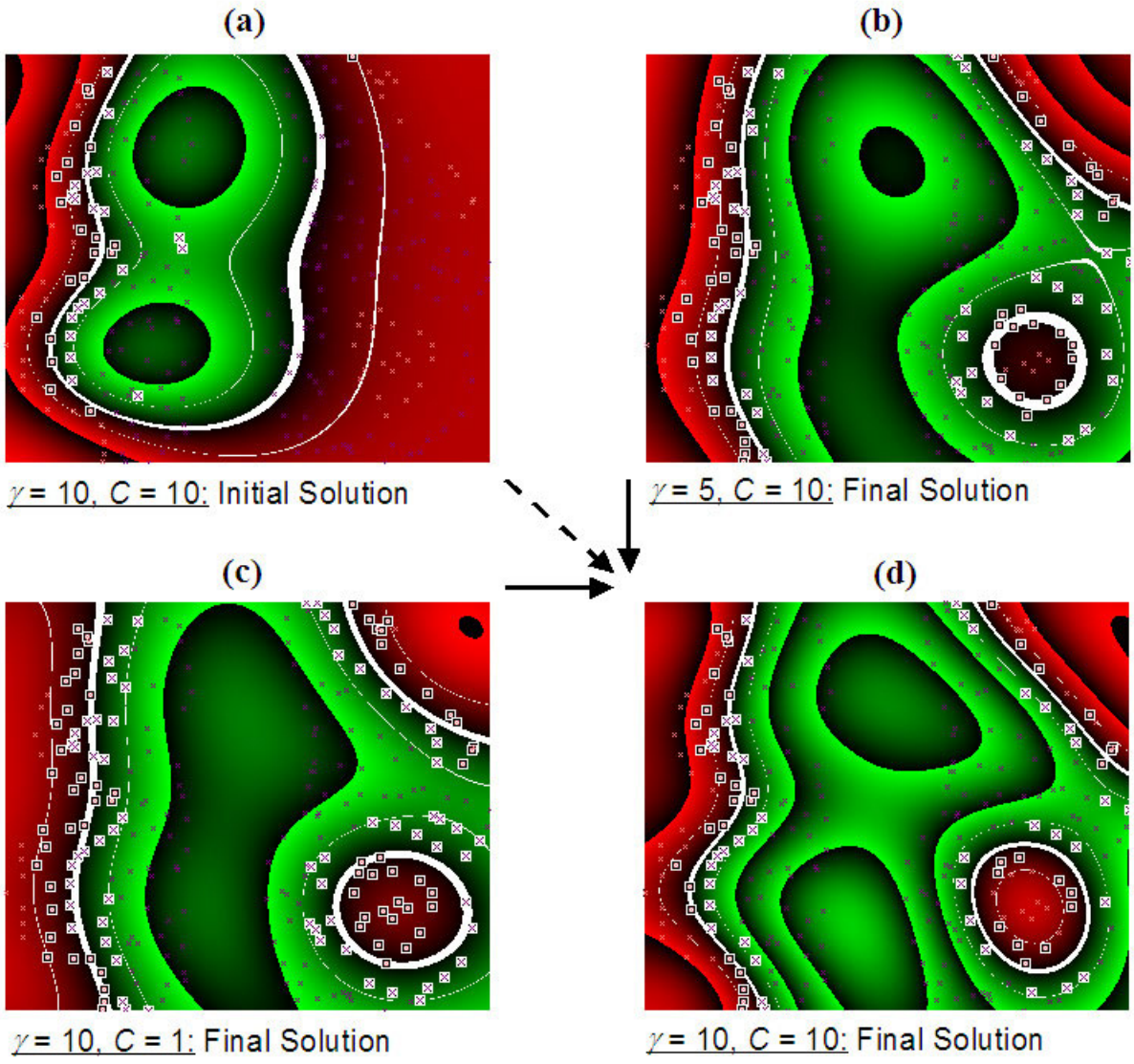


Figure 4.5: Illustration of initializing working set using result of previously trained SVM. The optimized solution for machine ($\gamma = 10, C = 10$) (d) can be reached normally from an random initial solution (a), or more efficiently from solution of a trained machine ($\gamma = 5, C = 10$) or ($\gamma = 10, C = 1$).

Table 4.1: Datasets used in experiments

Dataset	# Attribute	# Class	Size
Sat-image	36	6	4,435
Letter recognition	16	26	15,000
Shuttle	9	7	43,500

advance then the total number of SVs may exceed this limitation. In our experiments, a FIFO (First In First Out) queue structure with the same size as the working set was used to store the SVs of previous trained (binary) machines. With this structure, all the SVs of the latest trained machine (supposed to be closest to the next machine) are kept in the initial working set. Figure 4.5 illustrates the idea of using the result of a trained machine to train a new machine in model selection process. With a better initial step we can shorten the way to the optimized solution.

4.5 Experiments

We conducted experiments on three datasets in the StatLog collection: *sat-image*, *letter recognition*, and *shuttle*. These datasets are summarized in Table 4.1. They were chosen for their generality in dimension, size, number of classes, and the class distribution.

To see the effect of the method in real situations, we conducted experiments in different scenarios, including fixing the kernel and varying the cost parameter, fixing the cost and changing the kernel, and changing both kernel and cost parameter. In the first scenario we fixed the kernel to be linear and varied the cost parameter from 1 to 10. The second scenario was to fix the cost parameter at 1 and train the machines with polynomial kernels of degree from 2 to 9. The third scenario used Gaussian RBF kernels of the width γ changing from 0.01 to 0.1 with a step of 0.01 and varied the cost parameter from 1 to 10. The kernel cache sizes were 2000 (*sat-image*), 2000 (*letter recognition*), and 10,000 (*shuttle*). The optimization program was an implementation of the SMO algorithm and its improvement [6], [7]. Experiments were conducted on a PC Windows XP with 2.99 GH, 2GB RAM.

From the experimental results reported in Figure 4.6, we can see that in every situation the training time for each subsequent machine was reduced significantly, from 22.8% (*shuttle* dataset, RBF kernel, error cost 2) to 85.5% (*sat-image* dataset, linear kernel, cost 7).

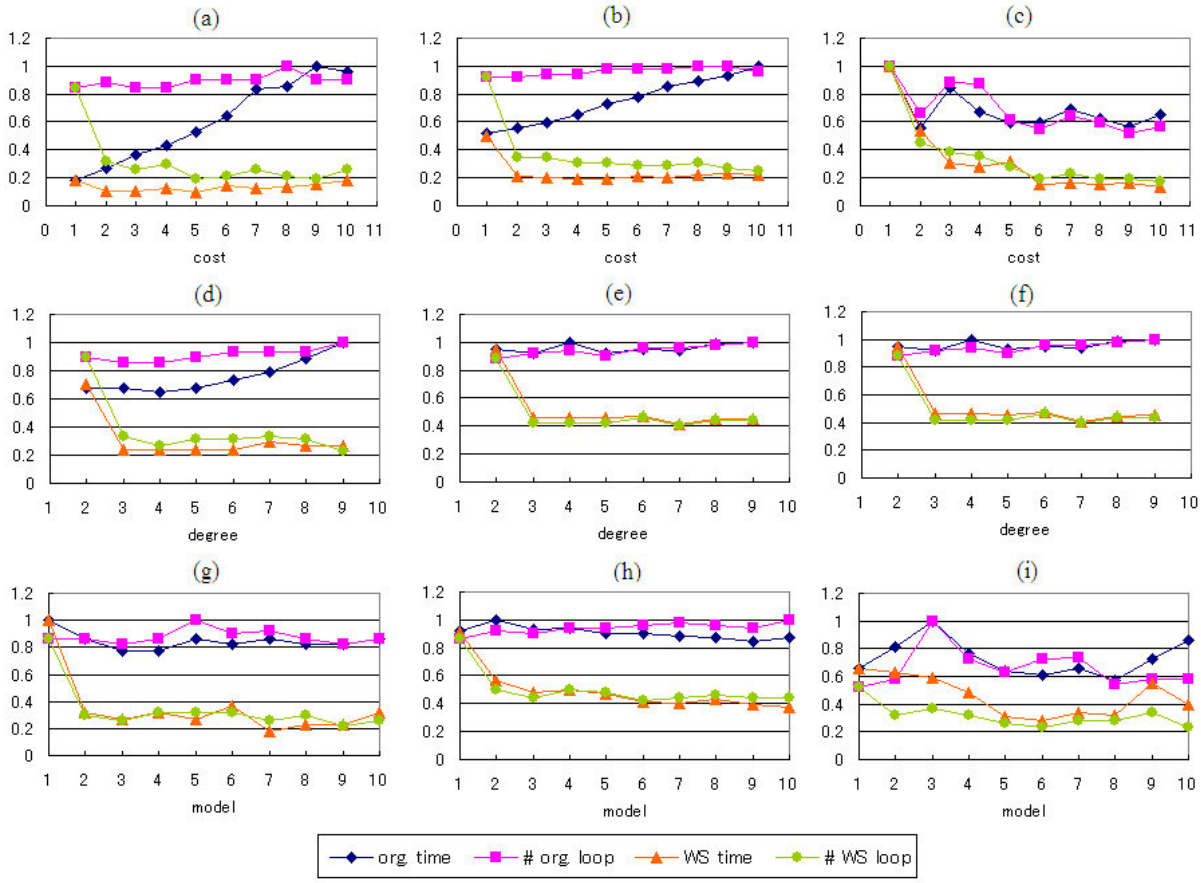


Figure 4.6: Reduction in number of required optimization loops and training time on three datasets *sat-image* (a-d-g), *letter recognition* (b-e-h), and *shuttle* (c-f-i), and in different situations: the same linear kernel with different cost (a-b-c), polynomial kernels of different degree with the same cost, and different RBF kernels with different costs. "org." denotes the original method with randomly working set selection; "WS" denotes the proposed method. All measures (average number of loops and training time) are normalized in to $[0, 1]$.

4.6 Discussion

We have described a method to speed up the training phase in model selection for support vector machines. The method utilizes the support vectors of previous trained machines to initialize the working set in training a new machine. This initialization scheme makes the training process converge more quickly. Experiment results on real life datasets show that the training time of subsequent machines can be reduced significantly.

In comparing with other speeding-up methods, the proposed one has two main advantages. First, it does not change the result of model selection. This is because the proposed method aims at initializing a better working set, leading to a faster convergence in training. In [83], a data filtering method is used to reduce the number of data in the dataset, or to reduce the size of the optimization problem. The data reduction makes the model selection process run faster, but the result is not the same as working on the entire available dataset due to the distortion of the training data. Moreover, the data-filtering algorithm has its own parameter k (in k-NN classification), so for each application it is necessary to do another model selection job in order to find the best value of k. The second advantage is the applicability of the proposed method in different situations and for different model search strategies like grid search [30], pattern search [31], and gradient-based methods [32], [70]. The alpha-seeding method in [84] is limited to the same kind of kernel and with a limited scheme of varying cost parameter. Experiment results on the *adult* dataset in the UCI corpus with linear kernel show the effectiveness of the alpha-seeding method (a reported of 5 times faster), but for machines with different kernels and different cost values, this method is not applicable.

The future work of this research is to enhance the way we utilize previous trained machines in initializing the working set, for example, using not only the support vectors (those with a distance to the separating hyperplane smaller than or equal one), but also the vectors that lie close to the separating plane (those with a distance to the separating hyperplane greater than one).

Chapter 5

Conclusions and Future Work

It is widely accepted that the support vector learning approach can produce machines with high generalization ability. Its solid theoretical background and success in many practical applications make support vector machine has received great attention in recent years. This dissertation introduces our two main contributions to the development of this learning approach: making a trained SVMs run faster and speeding-up SVM training in a model selection process.

For the first problem, we proposed a new method to reduce the complexity of a trained SVM by reducing the number of support vectors included in support vector solution. The reduction is done by iteratively replacing two support vectors by a newly created one, while trying to keep the whole solution unchanged. In comparing with former top-down approach, the proposed bottom-up approach leads to a conceptually simpler and computationally less expensive method. The construction of each new support vector is based on the finding of the unique maximum point of a one-variable function on $(0, 1)$, not to minimizing a multi-variable function with local minima. Experimental results on real life datasets show that the proposed method can reduce 57.9-97.6% of the number of support vectors, or making SVMs run 2.4 to 41.6 times faster with an almost unchanged generalization performance. Comparisons with previous methods also showed that the proposed one produced very competitive (even slightly better) results in terms of reduction rate and preserving predictive accuracy. The method is applicable for two-class support vector classifiers, support vector regressor, and one-class support vector machine for outlier detection task.

Several open problems are still remaining for a further research. Firstly, the construction of each new support vector is kernel-dependent. In this dissertation we have introduced the calculation of reduced support vectors for the two most commonly used kernels, the Gaussian RBF and polynomial kernels. The question is how does the pro-

posed method work for other types of kernel, like sigmoidal, inverse multi-quadric, spline kernels, or string kernels? Is the calculation simply to find the unique extremum of a one-variable function, or more complex? To answer these questions, it certainly requires an appropriate understanding above the kernel, and relation between support vectors in feature space which we cannot know explicitly. Another problem is that the proposed method suggests to represent and replace two close support vectors by another one, or more generally to represent a group of close support vectors by a representative. This representation is apparently reasonable and meaningful for the cases where input patterns are dense numerical vectors, e.g. in optical character recognition application. The question is how to combine patterns in other domains such as textual data, DNA and protein sequences in biology, graphs, or other structured data. And a more important question is that is this combination reasonable?

For the second problem, though the solution for each SVM is unique and global optimized, it does not mean that having a good machine for a given application is an easy task. Finding a suitable kernel and its parameter setting is still an open question in the field. Users must carry out an intensive model selection process with many trials of training and testing with different kernels and different values of parameters. We conducted intensive experiments and showed that two different machines trained by different parameter settings have many support vectors in common. Thus we can benefit from the result of trained machines to speed-up the optimization in training new machines. Our research demonstrated that, in a model selection process, by using solution of previously trained machine to initialize the search in training a new machine can reduce 22.8-85.5% training time. This method is applicable to any search strategy and does not change the result of model selection. One open question is that can the inheritance from previously trained machines be made more effectively? For example, SVMs are not only slow in training phase but also in testing phase, can we use trained machines to eliminate input patterns that are not support vectors in most machines under consideration? Thus we don't have to deal with these vectors in evaluating a model. And what can be effected by this elimination?. This idea is similar to using a data filtering technique to preprocess the data, but the advantage is that we don't have to solve another model selection task in chosing parameter for the data filtering algorithm. Another open problem for an efficient model selection method is how to conduct the process automatically. Because we cannot try all kernels and all possible values of paramters, so we firstly conduct model selection in some initial region in the whole space of hypothesis, e.g. when we use the common grid-search strategy. What happens if the best value does not belong to that region? Certainly we have to try again with another range of values. This problem demands more

effort of support vector learning researchers.

Bibliography

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

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

[3] V. N. Vapnik, *The Nature of Statistical Learning Theory.* N.Y.: Springer, 1995.

[4] V. N. Vapnik, *Statistical Learning Theory.* N.Y.: John Wiley & Sons, 1998.

[5] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in *Neural Networks for Signal Processing VII - Proceedings of the 1997 IEEE Workshop*, N. M. J. Principe, L. Gile and E. Wilson, Eds., New York, 1997, pp. 276–285.

[6] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods - Support Vector Learning*, B. Scholkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA: MIT Press, 1999, pp. 185–208.

[7] S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy, "Improvements to platt's smo algorithm for svm classifier design," *Neural Computation*, vol. 13, pp. 637–649, Mar. 2001.

[8] R.-E. Fan, P.-H. Chen, and C.-J. Lin, "Working set selection using the second order information for training svm," in *http://www.csie.ntu.edu.tw/ cjlin/libsvm*, 2005.

[9] T. Joachims, "Making large-scale support vector machine learning practical," in *Advances in Kernel Methods: Support Vector Machines*, A. S. B. Scholkopf, C. Burges, Ed. MIT Press, Cambridge, MA, 1998.

[10] C. Chih-Chung and L. Chi-Jen, "Libsvm : a library for support vector machines," in *http://www.csie.ntu.edu.tw/ cjlin/libsvm*, 2001.

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

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

[13] G. H. Peter, C. Eric, B. Léon, D. Igor, and V. Vladimir, "Parallel support vector machines: The Cascade SVM," in *Advances in Neural Information Processing Systems*, L. Saul, Y. Weiss, and L. Bottou, Eds., vol. 17. MIT Press, 2005.

[14] S. Romdhani, P. Torr, B. Scholkopf, and A. Blake, "Efficient face detection by a cascaded support-vector machine expansion," *Proceedings: Mathematical, Physical and Engineering Sciences*, pp. 3283 – 3297, 2004.

[15] R. Collobert, S. Bengio, and Y. Bengio, "A parallel mixture of svms for very large scale problems," *Neural Computation*, vol. 14, no. 5, pp. 1105–1114, 2002.

[16] X. Liu, L. O. Hall, and K. W. Bowyer, "Comments on a parallel mixture of svms for very large scale problems," *Neural Computation*, vol. 16, no. 7, pp. 1345–1351, 2004.

[17] K.-M. Lin and C.-J. Lin, "A study on reduced support vector machines," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1449–1459, 2003.

[18] Y.-J. Lee and O. L. Mangasarian, "Rsvm: Reduced support vector machines," in *Proceedings of the First SIAM International Conference on Data Mining.* Morgan Kaufmann, San Francisco, CA, 2001.

[19] B. Gokhan, B. Leon, and W. Jason, "Breaking svm complexity with cross-training," in *Advances in Neural Information Processing Systems*, L. Saul, Y. Weiss, and L. Bottou, Eds., vol. 17. MIT Press, 2005, pp. 81–88.

[20] J. Wang, X. Wu, and C. Zhang, "Support vector machines based on k-means clustering for real-time business intelligence systems," *International Journal of Business Intelligence and Data Mining*, vol. 1, no. 1, pp. 54–64, 2005.

[21] D. Boley and D. Cao, "Training support vector machine using adaptive clustering," in *2004 SIAM International Conference on Data Mining*, FL, USA, 2004.

[22] A. Bordes, S. Ertekin, J. Weston, and L. Bottou, "Fast kernel classifiers with online and active learning," *Journal of Machine Learning Research*, vol. 6, pp. 1579–1619, 2005.

[23] G. Schohn and D. Cohn, "Less is more: Active learning with support vector machines," in *Proc. 17th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 2000, pp. 839–846.

[24] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," in *The 17th International Conference on Machine Learning*, P. Langley, Ed. Stanford, US: Morgan Kaufmann, 2000, pp. 999–1006.

[25] C. K. I. Williams and M. Seeger, "Using the nystrom method to speed up kernel machines," *Advances in Neural Information Processing Systems*, vol. 13, pp. 682–688, 2001.

[26] A. J. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," in *Proc. 17th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 2000, pp. 911–918.

[27] S. Fine and K. Scheinberg, "Efficient svm training using low-rank kernel representations," *J. Mach. Learn. Res.*, vol. 2, pp. 243–264, 2002.

[28] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, "Core vector machines: Fast svm training on very large data sets," *J. Mach. Learn. Res.*, vol. 6, pp. 363–392, 2005.

[29] D. Anguita, S. Ridella, F. Rivieccio, and R. Zunino, "Quantum optimization for training support vector machines," *Neural Netw.*, vol. 16, no. 5-6, pp. 763–770, 2003.

[30] C. W. Hsu and C. J. Lin, "A comparison on methods for multi-class support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, pp. 415–425, 2002.

[31] M. Momma and K. Bennett, "A pattern search method for model selection of support vector regression," in *Proc. of SIAM Conference on Data Mining*, 2002.

[32] O. Chapelle and V. Vapnik, "Model selection for support vector machines," *Advances in Neural Information Processing Systems*, vol. 12, 2000.

[33] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing multiple parameters for support vector machines," *Machine Learning*, vol. 46, no. 1-3, pp. 131–159, 2002.

[34] H. Frohlich, O. Chapelle, and B. Scholkopf, "Feature selection for support vector machines using genetic algorithms," *International Journal on Artificial Intelligence Tools*, vol. 13, no. 4, pp. 791–800, 2004.

[35] T. Joachims, "Estimating the generalization performance of a SVM efficiently," in *Proceedings of ICML-00, 17th International Conference on Machine Learning*, P. Langley, Ed. Stanford, US: Morgan Kaufmann Publishers, San Francisco, US, 2000, pp. 431–438.

[36] C. J. C. Burges, "Simplified support vector decision rules," in *Proc. 13th International Conference on Machine Learning*, San Mateo, CA, 1996, pp. 71–77.

[37] B. Scholkopf, S. Mika, C. J. C. Burges, P. Knirsch, K. Muller, G. Ratsch, and A. J. Smola, "Input space versus feature space in kernel-based methods," *IEEE Trans. Neural Networks*, vol. 10, no. 5, pp. 1000–1017, 1999.

[38] T. Downs, K. E. Gates, and A. Masters, "Exact simplification of support vector solutions," *Journal of Machine Learning Research*, vol. 2, pp. 293–297, 2001.

[39] D. DeCoste, "Anytime interval-valued outputs for kernel machines: Fast support vector machine classification via distance geometry," in *Proceedings International Conference on Machine Learning (ICML-02)*, 2002, pp. 99–106.

[40] D. DeCoste and D. Mazzoni, "Fast query-optimized kernel machine classification via incremental approximate nearest support vectors," in *Proceedings International Conference on Machine Learning (ICML-03)*, 2003, pp. 115–122.

[41] R. Genov and G. Cauwenberghs, "Kerneltron: Support vector 'machine' in silicon," in *SVM '02: Proceedings of the First International Workshop on Pattern Recognition with Support Vector Machines*. London, UK: Springer-Verlag, 2002, pp. 120–134.

[42] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: an application to face detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1997.

[43] N. Ancona, G. Cicirelli, E. Stella, and A. Distante, "Object detection in images: Run-time complexity and parameter selection of support vector machines," in *16th International Conference on Pattern Recognition (ICPR'02)*, 2002.

[44] P. Michel and R. E. Kaliouby, "Real time facial expression recognition in video using support vector machines," in *ICMI '03: Proceedings of the 5th international*

conference on Multimodal interfaces. New York, NY, USA: ACM Press, 2003, pp. 258–264.

[45] S. Kang, H. Byun, and S.-W. Lee, "Real-time pedestrian detection using support vector machines," in *SVM '02: Proceedings of the First International Workshop on Pattern Recognition with Support Vector Machines*. London, UK: Springer-Verlag, 2002, pp. 268–277.

[46] M. Davy, F. Desobry, A. Gretton, and C. Doncarli, "An online support vector machine for abnormal events detections," *Signal Processing*, vol. 1, no. 1, pp. 1–42, 2005.

[47] A. Gretton and F. Desobry, "Online one-class nu-svm, an application to signal segmentation," in *IEEE ICASSP 2003*, 2003.

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

[49] B. Scholkopf, P. Knirsch, A. Smola, and C. Burges, "Fast approximation of support vector kernel expansions, and an interpretation of clustering as approximation in feature spaces," in *DAGM-Symposium, Informatik aktuell*, P. Levi, M. Schanz, R.-J. Ahlers, and F. May, Eds. Berlin: Springer, 1998, pp. 124–132.

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

[51] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386–408, 1958.

[52] K. Fukunaga, *Statistical Pattern Recognition*. New York: Academic Press, 1989.

[53] B. Scholkopf and A. Smola, *Learning with Kernels*. Cambridge, MA: MIT Press, 2002.

[54] C. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[55] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proceedings of the European Conference on Machine Learning*, C. Nedellec and C. Rouveirol, Eds. Berlin: Springer, 1998, pp. 137–142.

[56] Y. LeCun, L. Botou, L. Jackel, H. Drucker, C. Cortes, J. Denker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik, "Learning algorithms for classification: A comparison on handwritten digit recognition," *Neural Networks*, pp. 261–276, 1995.

[57] S. Dohkan, A. Koike, and T. Takagi, "Support vector machines for predicting protein-protein interactions," *Genome Informatics*, vol. 14, pp. 502–503, 2003.

[58] A. Kowalczyk and B. Raskutti, "One class svm for yeast regulation prediction," *SIGKDD Explor. Newsl.*, vol. 4, no. 2, pp. 99–100, 2002.

[59] R. Fletcher, *Practical Methods of Optimization*. New York: John Wiley, 1987.

[60] C. Liu, K. Nakashima, H. Sako, and H. Fujisawa, "Handwritten digit recognition: bench-marking of state-of-the-art techniques," *Pattern Recognition*, vol. 36, pp. 2271–2285, 2003.

[61] S. Mika, B. Scholkopf, A. Smola, K.-R. Muller, M. Scholz, and G. Ratsch, "Kernel pca and de-noising in feature spaces," in *Advances in Neural Information Processing Systems 11*, M. S. Kearns, S. A. Solla, and D. A. Cohn, Eds. Cambridge, MA: MIT Press, 1999, pp. 536–542.

[62] J. Kwok and I. Tsang, "The pre-image problem in kernel methods," in *In Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, Washington, D.C., USA, 2003, pp. 408–415.

[63] C. A. Micchelli, "Interpolation of scattered data: distance matrices and conditionally positive definite functions," *Constructive Approximation*, vol. 2, pp. 11–22, 1986.

[64] F. Girosi, "An equivalence between sparse approximation and support vector machines," *Neural Computation*, vol. 10, no. 6, pp. 1455–1480, 1998.

[65] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C++ : the art of scientific computing*. Cambridge University Press, 2002.

[66] T. Scheffer and T. Joachims, "Expected error analysis for model selection," in *Proceedings of ICML-99, 16th International Conference on Machine Learning*, I. Bratko and S. Dzeroski, Eds. Bled, SL: Morgan Kaufmann Publishers, San Francisco, US, 1999, pp. 361–370.

[67] M. J. Kearns, Y. Mansour, A. Y. Ng, and D. Ron, "An experimental and theoretical comparison of model selection methods," in *Computational Learing Theory*, 1995, pp. 21–30.

[68] M. Forster, "Key concepts in model selection: Performance and generalizability," *Journal of Mathematical Psychology*, vol. 44, no. 1, pp. 205–231, 2000.

[69] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification*. N.Y.: Ellis Horwood, 1994.

[70] S. Keerthi, "Efficient tuning of svm hyperparameters using radius/margin bound and iterative algorithms," *IEEE Transactions on Neural Networks*, vol. 13, pp. 1225–1229, Sept. 2002.

[71] W. Zucchini, "An introduction to model selection," *Journal of Mathematical Psychology*, vol. 44, pp. 41–46, 2000.

[72] P. L. Bartlett, S. Boucheron, and G. Lugosi, "Model selection and error estimationy," *Machine Learning*, vol. 48, no. 1-2, pp. 85–113, 2000.

[73] K. Muller, S.Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, "An introduction to kernel-based learning algorithms," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 181–201, 2001.

[74] D. Schuurmans, "A new metric-based approach to model selection," in *The Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 1997, pp. 552–558.

[75] M. H. Hansen and B. Yu, "Model selection and the principle of minimum description length," *Journal of the American Statistical Association*, vol. 96, no. 454, pp. 746–774, 2001.

[76] T. Lange, M. L. Braun, V. Roth, and J. M. Buhmann, "Stability-based model selection," *Advances in Neural Information Processing Systems*, vol. 15, pp. 746–774, 2003.

[77] S. Keerthi and C.-J. Lin, "Asymptotic behaviours of support vector machines with gaussian kernel," *Neural Computation*, vol. 15, pp. 1667–1689, 2003.

[78] V. Vapnik, E. Levin, and Y. L. Cun, "Measuring the vc-dimension of a learning machine," *Neural Comput.*, vol. 6, no. 5, pp. 851–876, 1994.

[79] V. Vapnik and O. Chapelle, "Bounds on error expectation for support vector machines," *Neural Computation*, vol. 12, no. 9, pp. 2013–2036, 2000.

[80] J. Shao and D. Tu, *The Jackknife and Bootstrap*. New York: Springer-Verlag, 1995.

[81] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*. London: Chapman & Hall, 1993.

[82] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1995, pp. 1137–1143.

[83] Y. Y. Ou, C. Y. Chen, S. C. Hwang, and Y. J. Oyang, "Expediting model selection for support vector machines based on data reduction," in *Proceedings of the 2003 IEEE International Conference on Systems, Man, and Cybernetics*, Washington D.C., Oct. 2003.

[84] D. DeCoste and K. Wagstaff, "Alpha seeding for support vector machines," in *International Conference on Knowledge Discovery & Data Mining (KDD-2000)*, 2000.

Publications

[1] **D.D. Nguyen**, T.B. Ho: "A Bottom-up Method for Simplifying Support Vector Solutions," *IEEE Transactions on Neural Networks*, (in press).

[2] **D.D. Nguyen**, T.B. Ho: "An Efficient Method for Simplifying Support Vector Machines," *The 22th International Conference on Machine Learning, ICML 2005*, Bonn, Germany, (August 2005).

[3] **D.D. Nguyen**, T.B. Ho: "Speeding-up Model Selection for Support Vector Machines," *The 18th International Conference of Florida Artificial Intelligence Research Society FLAIRS*, Florida, USA, (May 2005).

[4] **D.D. Nguyen**, T.B. Ho: "Efficient Model Selection for Support Vector Machines," The 5th International Symposium on Knowledge and Systems Sciences, Ishikawa, Japan (November 2004).

[5] T.B. Ho, **D.D. Nguyen**: "Chance Discovery and Learning Minority Classes," *Journal of New Generation Computing*, Ohmsha, Ltd. and Springer-Verlag, Vol. 21, No. 2, pp.147-160 (2003).

[6] T.B. Ho, T.D. Nguyen, S. Kawasaki, S.Q. Le, **D.D. Nguyen**, H. Yokoi, K. Takabayashi: "Mining Hepatitis Data with Temporal Abstraction," *ACM International Conference on Knowledge Discovery and Data Mining, ACM SIGKDD-03*, Washington DC, pp.369-377 (August 2003).

[7] T.B. Ho, T.D. Nguyen, **D.D. Nguyen**: "A User-Centered Visual Approach to Data Mining. The system D2MS," *Intelligent Information Processing*, M. Musen, B. Neumann, R. Studer (Eds.), Kluwer Academic Publishers, pp.213-224 (2002).

[8] T.B. Ho, T.D. Nguyen, **D.D. Nguyen**, S. Kawasaki: "Visualization of Data and Knowledge in the Knowledge Discovery Process," *Active Mining: New Directions of Data Mining*, H. Motoda (Ed.), IOS Press, pp.229-238 (2002).

[9] T.B. Ho, **D.D. Nguyen**, T.D. Nguyen, S. Kawasaki: "Extracting Knowledge from Hepatitis Data with Temporal Abstraction," *IEEE Conference on Data Mining, Workshop on Active Mining*, Maebashi, Japan, pp.91-96 (December 2002).

[10] S. Kawasaki, A. Saitou, **D.D. Nguyen**, T.B. Ho: "Mining from Medical Data: Case-Studies in Meningitis and Stomach Cancer Domains," *The 6th International Conference on Knowledge-based Intelligent Information & Engineering Systems*, Crema, Italy, pp.547-551 (September 2002).

[11] T.B. Ho, **D.D. Nguyen**, S. Kawasaki: "Learning Minority Classes in Unbalanced Datasets," *Third International Conference on Parallel and Distributed Computing*, Kanazawa, Japan, pp.196-203 (September 2002).

[12] T.B. Ho, **D.D. Nguyen**, S. Kawasaki, T.D. Nguyen: "Extracting Knowledge from Hepatitis Data with Temporal Abstraction," *ICML/PKDD 2002 Discovery Challenge, 6th European Conference on Principles of Data Mining and Knowledge Discovery PKDD 2002*, Helsinki, Finland (August 2002).

[13] T.B. Ho, T.D. Nguyen, **D.D. Nguyen**: "Visualization Support for a User-Centered KDD Process," *ACM International Conference on Knowledge Discovery and Data Mining, ACM SIGKDD-02*, Edmonton, Canada, pp. 519-524 (July 2002).

[14] T.B. Ho, A. Saito, S. Kawasaki, **D.D. Nguyen**, T.D. Nguyen: "Failure and Success Experience in Mining Stomach Cancer Data," *International Workshop Data Mining Lessons Learned, Inter. Conf. Machine Learning 2002*, Sydney, Australia, pp.40-47 (July 2002).

[15] S. Kawasaki, **D.D. Nguyen**, T.D. Nguyen, T.B. Ho: "Study of Hepatitis Data by Visual Data Mining System D2MS," *JSAI SIG-KBS-A201 Workshop Active Data Mining*, Pusan, Korea, pp.43-48 (May 2002).

[16] T.D. Nguyen, T.B. Ho, **D.D. Nguyen**: "Data and Knowledge Visualization in the Knowledge Discovery Process," *5th International Conference Recent Advances in Visual Information Systems*, Taiwan, (March 2002), Lecture Note in Computer Science 2314, Springer, pp. 311-321 (2002).

[17] T.B. Ho, T.D. Nguyen, **D.D. Nguyen**, S. Kawasaki: "Visualization Support for User-Centered Model Selection in Knowledge Discovery and Data Mining," *International Journal of Artificial Intelligence Tools*, World Scientific, Vol. 10, No. 4, pp.691-713 (2001).

[18] T.B. Ho, **D.D. Nguyen**, S. Kawasaki: "Mining Prediction Rules from Minority Classes," *14th International Conference on Applications of Prolog (INAP2001), International Workshop Rule-Based Data Mining RBDM 2001*, Tokyo, Japan, pp.254-264 (October 2001).

[19] T.B. Ho, S. Kawasaki, **D.D. Nguyen**: "Extracting Predictive Knowledge from Meningitis Data by Integration of Rule Induction and Association Mining," *International Workshop Challenge in KDD*, Shimaie, Japan, (May 2001), Lecture Notes in Artificial Intelligence 2253, Springer, pp.508-515 (2001).