APR Advances in Pattern Recognition

Support Vector Machines for Pattern Classification

Shigeo Abe

Springer

Advances in Pattern Recognition

Shigeo Abe

Support Vector Machines for Pattern Classification

With 110 Figures

Springer

Professor Dr Shigeo Abe
Kobe University, Kobe, Japan

Series editor
Professor Sameer Singh, PhD
Department of Computer Science, University of Exeter, Exeter, EX4 4PT, UK

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data
Abe, Shigeo, 1947–
Support vector machines for pattern classification / Shigeo Abe.
p. cm.
Includes bibliographical references and index.
ISBN 1-85233-929-9 (alk. paper)
1. Text processing (Computer science) 2. Pattern recognition systems. 3. Machine learning. I. Title.
QA76.9.T48A23 2005
005.52—dc22 2005040265

Advances in Pattern Recognition ISSN 1617-7916
ISBN-10: 1-85233-929-2 Printed on acid-free paper
ISBN-13: 978-1-85233-929-6

Printed in the United States of America (SB)

9 8 7 6 5 4 3 2 1

Springer Science+Business Media
springeronline.com

Preface

I was shocked to see a student's report on performance comparisons between support vector machines (SVMs) and fuzzy classifiers that we had developed with our best endeavors. Classification performance of our fuzzy classifiers was comparable, but in most cases inferior, to that of support vector machines. This tendency was especially evident when the numbers of class data were small. I shifted my research efforts from developing fuzzy classifiers with high generalization ability to developing support vector machine–based classifiers.

This book focuses on the application of support vector machines to pattern classification. Specifically, we discuss the properties of support vector machines that are useful for pattern classification applications, several multiclass models, and variants of support vector machines. To clarify their applicability to real-world problems, we compare performance of most models discussed in the book using real-world benchmark data. Readers interested in the theoretical aspect of support vector machines should refer to books such as [109, 215, 256, 257].

Three-layer neural networks are universal classifiers in that they can classify any labeled data correctly if there are no identical data in different classes [3, 279]. In training multilayer neural network classifiers, network weights are usually corrected so that the sum-of-squares error between the network outputs and the desired outputs is minimized. But because the decision boundaries between classes acquired by training are not directly determined, classification performance for the unknown data, i.e., the generalization ability, depends on the training method. And it degrades greatly when the number of training data is small and there is no class overlap.

On the other hand, in training support vector machines the decision boundaries are determined directly from the training data so that the separating margins of decision boundaries are maximized in the high-dimensional space called *feature space*. This learning strategy, based on statistical learning theory developed by Vapnik [256, 257], minimizes the classification errors of the training data and the unknown data.

Therefore, the generalization abilities of support vector machines and other classifiers differ significantly, especially when the number of training data is small. This means that if some mechanism to maximize the margins of decision boundaries is introduced to non-SVM-type classifiers, their performance degradation will be prevented when the class overlap is scarce or nonexistent.[1]

In the original support vector machine, an n-class classification problem is converted into n two-class problems, and in the ith two-class problem we determine the optimal decision function that separates class i from the remaining classes. In classification, if one of the n decision functions classifies an unknown datum into a definite class, it is classified into that class. In this formulation, if more than one decision function classify a datum into definite classes, or if no decision functions classify the datum into a definite class, the datum is unclassifiable.

Another problem of support vector machines is slow training. Because support vector machines are trained by solving a quadratic programming problem with the number of variables equal to the number of training data, training is slow for a large number of training data.

To resolve unclassifiable regions for multiclass support vector machines we propose fuzzy support vector machines and decision-tree-based support vector machines.

To accelerate training, in this book, we discuss two approaches: selection of important data for training support vector machines before training and training by decomposing the optimization problem into two subproblems.

To improve generalization ability of non-SVM-type classifiers, we introduce the ideas of support vector machines to the classifiers: neural network training incorporating maximizing margins and a kernel version of a fuzzy classifier with ellipsoidal regions [3, pp. 90–3, 119–39].

In Chapter 1, we discuss two types of decision functions: direct decision functions, in which the class boundary is given by the curve where the decision function vanishes; and the indirect decision function, in which the class boundary is given by the curve where two decision functions take on the same value.

In Chapter 2, we discuss the architecture of support vector machines for two-class classification problems. First we explain hard-margin support vector machines, which are used when the classification problem is linearly separable, namely, the training data of two classes are separated by a single hyperplane. Then, introducing slack variables for the training data, we extend hard-margin support vector machines so that they are applicable to inseparable problems. There are two types of support vector machines: L1 soft-margin support vector machines and L2 soft-margin support vector machines. Here, L1 and L2 denote the linear sum and the square sum of the slack variables that are added to the objective function for training. Then we investigate the charac-

[1]To improve generalization ability of a classifier, a regularization term, which controls the complexity of the classifier, is added to the objective function.

teristics of solutions extensively and survey several techniques for estimating the generalization ability of support vector machines.

In Chapter 3, we discuss some methods for multiclass problems: one-against-all support vector machines, in which each class is separated from the remaining classes; pairwise support vector machines, in which one class is separated from another class; the use of error-correcting output codes for resolving unclassifiable regions; and all-at-once support vector machines, in which decision functions for all the classes are determined at once. To resolve unclassifiable regions, in addition to error-correcting codes, we discuss fuzzy support vector machines with membership functions and decision-tree-based support vector machines. To compare several methods for multiclass problems, we show performance evaluation of these methods for the benchmark data sets.

Since support vector machines were proposed, many variants of support vector machines have been developed. In Chapter 4, we discuss some of them: least squares support vector machines whose training results in solving a set of linear equations, linear programming support vector machines, robust support vector machines, and so on.

In Chapter 5, we discuss some training methods for support vector machines. Because we need to solve a quadratic optimization problem with the number of variables equal to the number of training data, it is impractical to solve a problem with a huge number of training data. For example, for 10,000 training data, 800 MB memory is necessary to store the Hessian matrix in double precision. Therefore, several methods have been developed to speed training. One approach reduces the number of training data by preselecting the training data. The other is to speed training by decomposing the problem into two subproblems and repeatedly solving the one subproblem while fixing the other and exchanging the variables between the two subproblems.

Optimal selection of features is important in realizing high-performance classification systems. Because support vector machines are trained so that the margins are maximized, they are said to be robust for nonoptimal features. In Chapter 6, we discuss several methods for selecting optimal features and show, using some benchmark data sets, that feature selection is important even for support vector machines. Then we discuss feature extraction that transforms input features by linear and nonlinear transformation.

Some classifiers need clustering of training data before training. But support vector machines do not require clustering because mapping into a feature space results in clustering in the input space. In Chapter 7, we discuss how we can realize support vector machine–based clustering.

One of the features of support vector machines is that by mapping the input space into the feature space, nonlinear separation of class data is realized. Thus the conventional linear models become nonlinear if the linear models are formulated in the feature space. They are usually called *kernel-based methods.* In Chapter 8, we discuss typical kernel-based methods: kernel least squares, kernel principal component analysis, and the kernel Mahalanobis distance.

The concept of maximum margins can be used for conventional classifiers to enhance generalization ability. In Chapter 9, we discuss methods for maximizing margins of multilayer neural networks, and in Chapter 10 we discuss maximum-margin fuzzy classifiers with ellipsoidal regions and polyhedral regions.

Support vector machines can be applied to function approximation. In Chapter 11, we discuss how to extend support vector machines to function approximation and compare the performance of the support vector machine with that of other function approximators.

Acknowledgments

We are grateful to those who are involved in the research project, conducted at the Graduate School of Science and Technology, Kobe University, on neural, fuzzy, and support vector machine–based classifiers and function approximators, for their efforts in developing new methods and programs. Discussions with Dr. Seiichi Ozawa were always helpful. Special thanks are due to then and current graduate and undergraduate students: T. Inoue, K. Sakaguchi, T. Takigawa, F. Takahashi, Y. Hirokawa, T. Nishikawa, K. Kaieda, Y. Koshiba, D. Tsujinishi, Y. Miyamoto, S. Katagiri, T. Yamasaki, T. Kikuchi, and K. Morikawa; and Ph.D. student T. Ban.

I thank A. Ralescu for having used my draft version of the book as a graduate course text and having given me many useful comments. Thanks are also due to H. Nakayama, S. Miyamoto, J. A. K. Suykens, F. Anouar, G. C. Cawley, H. Motoda, A. Inoue, F. Schwenker, N. Kasabov, and B.-L. Lu for their valuable discussions and useful comments.

The Internet was a valuable source of information in writing the book. Most of the papers listed in the References were obtained from the Internet, from either authors' home pages or free downloadable sites such as:

ESANN: www.dice.ucl.ac.be/esann/proceedings/electronicproceedings.htm
JMLR: www.jmlr.org/papers/
NEC Research Institute CiteSeer: citeseer.nj.nec.com/cs
NIPS: books.nips.cc/

Kobe, October 2004 *Shigeo Abe*

Contents

Nomenclature

We use lowercase bold letters to denote vectors and uppercase italic letters to denote matrices. The following list shows the symbols used in the book:

α_i :	Lagrange multiplier for $\mathbf{x}_i$
ξ_i :	slack variable associated with $\mathbf{x}_i$
A^{-1} :	inverse of matrix A
A^T :	transpose of matrix A
B :	set of bounded support vector indices
b_i :	bias term of the ith hyperplane
C :	margin parameter
d :	degree of a polynomial kernel
$\mathbf{g}(\mathbf{x})$:	mapping function from $\mathbf{x}$ to the feature space
γ :	parameter for a radial basis function kernel
$H(\mathbf{x}, \mathbf{x}')$:	kernel
l :	dimension of the feature space
M :	number of training data
m :	number of input variables
n :	number of classes
S :	set of support vector indices
U :	set of unbounded support vector indices
$\|\mathbf{x}\|$:	Euclidean norm of vector $\mathbf{x}$
$\mathbf{w}_i$:	coefficient vector of the ith hyperplane
X_i :	set for class i training data
$\vert X_i\vert$:	number of data in the set X_i
$\mathbf{x}_i$:	ith m-dimensional training data
y_i :	class label 1 or -1 for input $\mathbf{x}_i$ for pattern classification and a scalar output for function approximation

1 Introduction

Pattern classification is to classify some object into one of the given categories called *classes*. For a specific pattern classification problem, a classifier, which is computer software, is developed so that objects are classified correctly with reasonably good accuracy. Inputs to the classifier are called *features*, because they are determined so that they represent each class well or so that data belonging to different classes are well separated in the input space.

In general there are two approaches to develop classifiers: a parametric approach [90], in which a priori knowledge of data distributions is assumed, and a nonparametric approach, in which no a priori knowledge is assumed.

Neural networks [1, 35, 108], fuzzy systems [3, 33, 183], and support vector machines [69] are typical nonparametric classifiers. Through training using input-output pairs, classifiers acquire decision functions that classify an input datum into one of the given classes.

In this chapter we first classify decision functions into direct and indirect decision functions. For a two-class problem, the class boundary given by a direct decision function corresponds to the curve where the function vanishes, while the class boundary given by two indirect decision functions corresponds to the curve where the two functions give the same values. Then we discuss how to define and determine the direct decision functions for multiclass problems.

1.1 Decision Functions

1.1.1 Decision Functions for Two-Class Problems

Consider classifying an m-dimensional vector $\mathbf{x} = (x_1, \ldots, x_m)^T$ into one of two classes. Suppose that we are given functions $g_1(\mathbf{x})$ and $g_2(\mathbf{x})$ for Classes 1 and 2, respectively, and we classify $\mathbf{x}$ into Class 1 if

$$g_1(\mathbf{x}) > 0, \quad g_2(\mathbf{x}) < 0, \tag{1.1}$$

and into Class 2 if

$$g_1(\mathbf{x}) < 0, \quad g_2(\mathbf{x}) > 0. \tag{1.2}$$

We call these functions *decision functions*. By the preceding decision functions, if for $\mathbf{x}$

$$g_1(\mathbf{x})\, g_2(\mathbf{x}) > 0 \tag{1.3}$$

is satisfied, $\mathbf{x}$ is not classifiable (see the hatched regions in Fig. 1.1; the arrows show the positive sides of the functions).

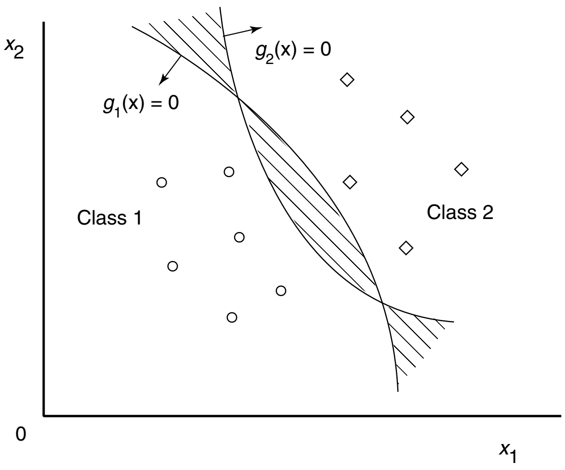


Fig. 1.1. Decision functions in a two-dimensional space

To resolve unclassifiable regions, we may change (1.1) and (1.2) as follows. We classify $\mathbf{x}$ into Class 1 if

$$g_1(\mathbf{x}) > g_2(\mathbf{x}) \tag{1.4}$$

and into Class 2 if

$$g_1(\mathbf{x}) < g_2(\mathbf{x}). \tag{1.5}$$

In this case, the class boundary is given by (see the dotted curve in Fig. 1.2)

$$g_1(\mathbf{x}) = g_2(\mathbf{x}). \tag{1.6}$$

This means that the class boundary is indirectly obtained by solving (1.6) for $\mathbf{x}$. We call this type of decision function an *indirect decision function*.

If we define the decision functions by

$$g_1(\mathbf{x}) = -g_2(\mathbf{x}), \tag{1.7}$$

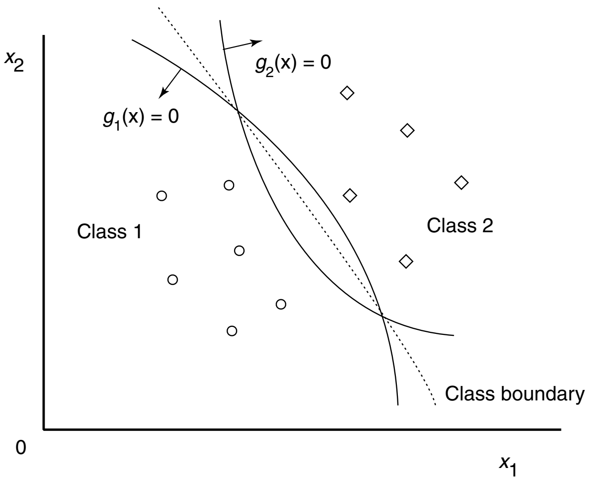


Fig. 1.2. Class boundary for Fig. 1.1

we classify $\mathbf{x}$ into Class 1 if

$$g_1(\mathbf{x}) > 0 \tag{1.8}$$

and into Class 2 if

$$g_2(\mathbf{x}) > 0. \tag{1.9}$$

Thus the class boundary is given by

$$g_1(\mathbf{x}) = -g_2(\mathbf{x}) = 0. \tag{1.10}$$

Namely, the class boundary corresponds to the curve where the decision function vanishes. We call this type of decision function a *direct decision function.*

If the decision function is linear, namely, $g_1(\mathbf{x})$ is given by

$$g_1(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b, \tag{1.11}$$

where $\mathbf{w}$ is an m-dimensional vector and b is a bias term, and if one class is on the positive side of the hyperplane, i.e., $g_1(\mathbf{x}) > 0$, and the other class is on the negative side, the given problem is said to be *linearly separable.*

1.1.2 Decision Functions for Multiclass Problems

Indirect Decision Functions

For an $n(> 2)$-class problem, suppose we have indirect decision functions $g_i(\mathbf{x})$ for classes i. To avoid unclassifiable regions, we classify $\mathbf{x}$ into class j given by

$$j = \arg \max_{i=1,\ldots,n} g_i(\mathbf{x}), \tag{1.12}$$

where arg returns the subscript with the maximum value of $g_i(\mathbf{x})$. If more than one decision function take the same maximum value for $\mathbf{x}$, namely, $\mathbf{x}$ is on the class boundary, it is not classifiable.

In the following we discuss several methods to obtain the direct decision functions for multiclass problems.

One-against-All Formulation

The first approach is to determine the decision functions by the one-against-all formulation [256]. We determine the ith decision function $g_i(\mathbf{x})$ $(i = 1, \ldots, n)$, so that when $\mathbf{x}$ belongs to class i,

$$g_i(\mathbf{x}) > 0, \tag{1.13}$$

and when $\mathbf{x}$ belongs to one of the remaining classes,

$$g_i(\mathbf{x}) < 0. \tag{1.14}$$

When $\mathbf{x}$ is given, we classify $\mathbf{x}$ into class i if $g_i(\mathbf{x}) > 0$ and $g_j(\mathbf{x}) < 0\,(j \neq i, j = 1, \ldots, n)$. But by these decision functions, unclassifiable regions exist when more than one decision function are positive or no decision functions are positive, as seen from Fig. 1.3. To resolve these unclassifiable regions we introduce membership functions in Chapter 3.

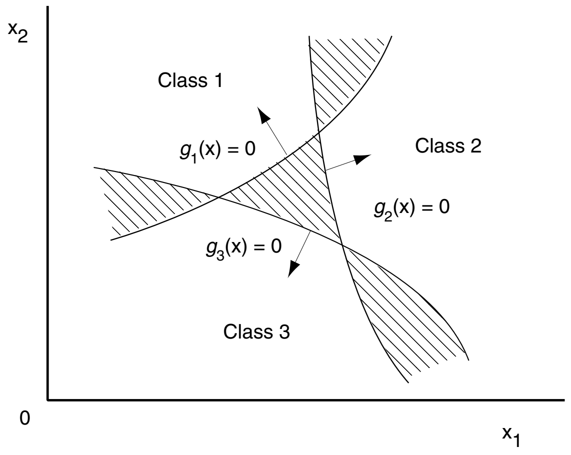


Fig. 1.3. Class boundaries by one-against-all formulation

Decision-Tree Formulation

The second approach is based on a decision tree. It is considered to be a variant of one-against-all formulation. We determine the ith decision function $g_i(\mathbf{x})\ (i = 1, \ldots, n-1)$, so that when $\mathbf{x}$ belongs to class i,

$$g_i(\mathbf{x}) > 0, \tag{1.15}$$

and when $\mathbf{x}$ belongs to one of the classes $\{i+1, \ldots, n\}$,

$$g_i(\mathbf{x}) < 0. \tag{1.16}$$

In classifying $\mathbf{x}$, starting from $g_1(\mathbf{x})$, we find the first positive $g_i(\mathbf{x})$ and classify $\mathbf{x}$ into class i. If there is no such i among $g_i(\mathbf{x})\ (i = 1, \ldots, n-1)$, we classify $\mathbf{x}$ into class n.

Figure 1.4 shows an example of decision functions for four classes. The decision functions change if we determine decision functions in descending order or in an arbitrary order of class labels. Therefore, in this architecture, we need to determine the decision functions so that classification performance in the upper level of the tree is more accurate than in the lower one. Otherwise, the classification performance may not be good.

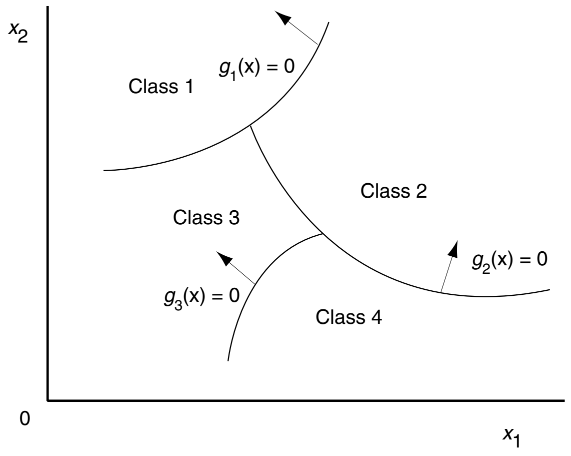


Fig. 1.4. Decision-tree-based decision functions

Pairwise Formulation

The third approach is to determine the decision functions by pairwise formulation [140]. For classes i and j we determine the decision function $g_{ij}(\mathbf{x})\ (i \neq j, i, j = 1, \ldots, n)$, so that

$$g_{ij}(\mathbf{x}) > 0 \tag{1.17}$$

when $\mathbf{x}$ belongs to class i and

$$g_{ij}(\mathbf{x}) < 0 \tag{1.18}$$

when $\mathbf{x}$ belongs to class j.

In this formulation, $g_{ij}(\mathbf{x}) = -g_{ji}(\mathbf{x})$, and we need to determine $n(n-1)/2$ decision functions. Classification is done by voting, namely, we calculate

$$g_i(\mathbf{x}) = \sum_{j \neq i, j=1}^{n} \operatorname{sign}(g_{ij}(\mathbf{x})), \tag{1.19}$$

where

$$\operatorname{sign}(x) = \begin{cases} 1 & x \geq 0, \\ -1 & x < 0, \end{cases} \tag{1.20}$$

and we classify $\mathbf{x}$ into the class with the maximum $g_i(\mathbf{x})$.[1] By this formulation also, unclassifiable regions exist if $g_i(\mathbf{x})$ take the maximum value for more than one class (see the hatched region in Fig. 1.5). These can be resolved by decision-tree formulation or by introducing membership functions as discussed in Chapter 3.

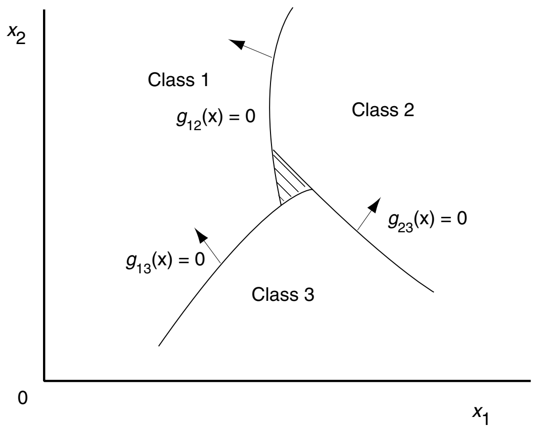


Fig. 1.5. Class boundaries by pairwise formulation

[1]We may define the sign function by

$$\operatorname{sign}(x) = \begin{cases} 1 & x > 0, \\ 0 & x = 0, \\ -1 & x < 0. \end{cases}$$

Error-Correcting Output Codes

The fourth approach is to use error-correcting codes for encoding outputs [73]. One-against-all formulation is a special case of error-correcting code with no error-correcting capability, and so is pairwise formulation, as discussed in Chapter 3, if "don't" care bits are introduced.

All-at-Once Formulation

The fifth approach is to determine decision functions at all once. Namely, we determine the decision functions $g_i(\mathbf{x})$ by

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \qquad \text{for} \quad j \neq i,\, j = 1, \ldots, n. \tag{1.21}$$

In this formulation we need to determine n decision functions at all once [19, 20, 29, 40, 100, 269, 270], [257, pp. 437–40], [77, pp. 174–6]. This results in solving a problem with a larger number of variables than the previous methods.

An example of class boundaries is shown in Fig. 1.6. Unlike one-against-all and pairwise formulations, there is no unclassifiable region.

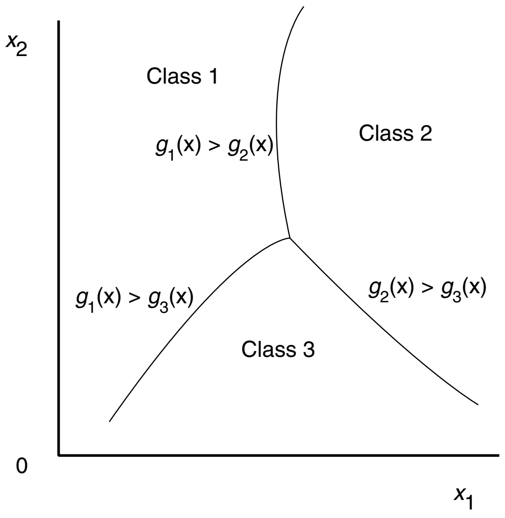


Fig. 1.6. Class boundaries by all-at-once formulation

1.2 Determination of Decision Functions

Determination of decision functions using input-output pairs is called *training*. In training a multilayer neural network for a two-class problem, we can determine a direct decision function if we set one output neuron instead of two. But because for an n-class problem we set n output neurons with the ith neuron corresponding to the class i decision function, the obtained functions are indirect. Similarly, decision functions for fuzzy classifiers are indirect because membership functions are defined for each class.

Conventional training methods determine the indirect decision functions so that each training input is correctly classified into the class designated by the associated training output. Figure 1.7 shows an example of the decision functions obtained when the training data of two classes do not overlap. Assuming that the circles and rectangles are training data for Classes 1 and 2, respectively, even if the decision function $g_2(\mathbf{x})$ moves to the right as shown in the dotted curve, the training data are still correctly classified. Thus there are infinite possibilities of the positions of the decision functions that correctly classify the training data. Although the generalization ability is directly affected by the positions, conventional training methods do not consider this.

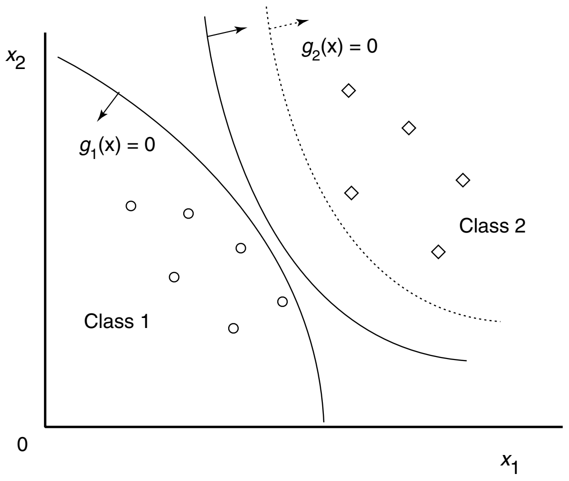


Fig. 1.7. Class boundary when classes do not overlap

In a support vector machine, the direct decision function that maximizes the generalization ability is determined for a two-class problem. Assuming that the training data of different classes do not overlap, the decision function is determined so that the distance from the training data is maximized. We call this the *optimal decision function*. Because it is difficult to determine a nonlinear decision function, the original input space is mapped into a high-dimensional space called *feature space*. And in the feature space, the optimal decision function, namely, the optimal hyperplane is determined.

Support vector machines outperform conventional classifiers, especially when the number of training data is small and the number of input variables is large. This is because the conventional classifiers do not have the mechanism to maximize the margins of class boundaries. Therefore, if we introduce some mechanism to maximize margins, the generalization ability is improved.

1.3 Data Sets Used in the Book

Table 1.1 shows the data sets used in this book to evaluate the performance of classifiers and function approximators. The first eight data sets are for pattern classification, the last three for function approximation.

The data sets for pattern classification are the iris data [32, 84], the numeral data for license plate recognition [244], the thyroid data [264],[2] the blood cell data [106], hiragana data [3, 141], and the MNIST data [143].[3]

Table 1.1. Benchmark data specification

Data	Inputs	Classes	Training data	Test data
Iris	4	3	75	75
Numeral	12	10	810	820
Thyroid	21	3	3772	3428
Blood cell	13	12	3097	3100
Hiragana-50	50	39	4610	4610
Hiragana-105	105	38	8375	8356
Hiragana-13	13	38	8375	8356
MNIST	784	10	60,000	10,000
Mackey-Glass	4	1	500	500
Water Purif. (Stationary)	10	1	241	237
Water Purif. (Nonstationary)	10	1	45	40

The Fisher iris data are widely used for evaluating classification performance of classifiers. They consist of 150 data with four features and three

[2]ftp://ftp.ics.uci.edu/pub/machine-learning-databases/
[3]http://yann.lecun.com/exdb/mnist/

classes; there are 50 data per class. We used the first 25 data of each class as the training data and the remaining 25 data of each class as the test data.

The numeral data were collected to identify Japanese license plates of running cars. They include numerals, hiragana, and kanji characters. The original image taken from a TV camera was preprocessed and each numeral was transformed into 12 features, such as the number of holes and the curvature of a numeral at some point.

The thyroid data include 15 digital features and more than 92 percent of the data belong to one class. Thus the recognition rate lower than 92 percent is useless.

The blood cell classification involves classifying optically screened white blood cells into 12 classes using 13 features. This is a very difficult problem; class boundaries for some classes are ambiguous because the classes are defined according to the growth stages of white blood cells.

Hiragana-50 and hiragana-105 data were gathered from Japanese license plates. The original grayscale images of hiragana characters were transformed into (5×10)-pixel and (7×15)-pixel images, respectively, with the grayscale range being from 0 to 255. Then by performing grayscale shift, position shift, and random noise addition to the images, the training and test data were generated. Then for the hiragana-105 data to reduce the number of input variables, i.e., $7 \times 15 = 105$, the hiragana-13 data were generated by calculating the 13 central moments for the (7×15)-pixel images [50, 141].

The MNIST data are handwritten numerals consisting of (28×28)-pixel inputs with 256 grayscale levels; they are often used to evaluate support vector machines.

In addition to these data sets, in Chapter 10, we use two-class data sets[4] used in [173, 199].

For function approximation we use the Mackey-Glass time series data [70] and water purification plant data [22] listed in Table 1.1. In the table, the number of classes corresponds to the number of outputs and is 1.

The Mackey-Glass differential equation generates time series data with a chaotic behavior and is given by

$$\frac{dx(t)}{dt} = \frac{0.2\,x(t-\tau)}{1 + x^{10}(t-\tau)} - 0.1\,x(t), \tag{1.22}$$

where t and τ denote time and time delay, respectively.

By integrating (1.22), we can obtain the time series data $x(0), x(1), x(2), \ldots, x(t), \ldots$. Using x prior to time t, we predict x after time t. Setting $\tau = 17$ and using four inputs $x(t-18), x(t-12), x(t-6), x(t)$, we estimate $x(t+6)$.

The first 500 data from the time series data, $x(118), \ldots, x(1117)$, are used to train function approximators, and the remaining 500 data are used to test performance. This data set is often used as the benchmark data for function approximation and the normalized root-mean-square error (NRMSE), i.e., the

[4]http://ida.first.fraunhofer.de/projects/bench/benchmarks.htm

root-mean-square error divided by the standard deviation of the time series data is used to measure the performance.

In a water purification plant, to eliminate small particles floating in the water taken from a river, coagulant is added and the water is stirred while these small particles begin sticking to each other. As more particles stick together they form flocs, which fall to the bottom of a holding tank. Potable water is obtained by removing the precipitated flocs and adding chlorine. Careful implementation of the coagulant injection is very important to obtain high-quality water. Usually an operator determines the amount of coagulant needed according to an analysis of the water qualities, observation of floc formation, and prior experience.

To automate this operation, as inputs for water quality, (1) turbidity, (2) temperature, (3) alkalinity, (4) pH, and (5) flow rate were used, and to replace the operator's observation of floc properties by image processing, (1) floc diameter, (2) number of flocs, (3) floc volume, (4) floc density, and (5) illumination intensity were used [22].

The 563 input-output data, which were gathered over a one-year period, were divided into 478 stationary data and 95 nonstationary data according to whether turbidity values were smaller or larger than a specified value. Then each type of data was further divided into two groups to form a training data set and a test data set; division was done in such a way that both sets had similar distributions in the output space.

2

Two-Class Support Vector Machines

In training a classifier, usually we try to maximize classification performance for the training data. But if the classifier is too fit for the training data, the classification ability for unknown data, i.e., the generalization ability is degraded. This phenomenon is called *overfitting*. Namely, there is a trade-off between the generalization ability and fitting to the training data. Various methods have been proposed to prevent overfitting [35, pp. 11–5], [1, pp. 86–91], [80].[1]

For a two-class problem, a support vector machine is trained so that the direct decision function maximizes the generalization ability [256, pp. 127–51], [60, pp. 92–129]. Namely, the m-dimensional input space $\mathbf{x}$ is mapped into the l-dimensional $(l \geq m)$ feature space $\mathbf{z}$. Then in $\mathbf{z}$, the quadratic programming problem is solved to separate two classes by the optimal separating hyperplane.

In this chapter we discuss support vector machines for two-class problems. First, we discuss hard-margin support vector machines, in which training data are linearly separable in the input space. Then we extend hard-margin support vector machines to the case where training data are not linearly separable and map the input space into the high-dimensional feature space to enhance linear separability in the feature space. The characteristics of support vector machines are then studied theoretically and by computer simulations.

2.1 Hard-Margin Support Vector Machines

Let M m-dimensional training inputs $\mathbf{x}_i$ $(i = 1, \ldots, M)$ belong to Class 1 or 2 and the associated labels be $y_i = 1$ for Class 1 and -1 for Class 2. If these data are linearly separable, we can determine the decision function:

$$D(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b, \tag{2.1}$$

[1]One of the main ideas is, like support vector machines, to add a regularization term, which controls the generalization ability, to the objective function.

where $\mathbf{w}$ is an m-dimensional vector, b is a bias term, and for $i = 1, \ldots, M$

$$\mathbf{w}^T \mathbf{x}_i + b \begin{cases} > 0 & \text{for} \quad y_i = 1, \\ < 0 & \text{for} \quad y_i = -1. \end{cases} \tag{2.2}$$

Because the training data are linearly separable, no training data satisfy $\mathbf{w}^T \mathbf{x} + b = 0$. Thus, to control separability, instead of (2.2), we consider the following inequalities:

$$\mathbf{w}^T \mathbf{x}_i + b \begin{cases} \geq 1 & \text{for} \quad y_i = 1, \\ \leq -1 & \text{for} \quad y_i = -1. \end{cases} \tag{2.3}$$

Here, 1 and -1 on the right-hand sides of the inequalities can be a constant $a\,(> 0)$ and $-a$, respectively. But by dividing both sides of the inequalities by a, (2.3) is obtained. Equation (2.3) is equivalent to

$$y_i\,(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad \text{for} \quad i = 1, \ldots, M. \tag{2.4}$$

The hyperplane

$$D(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = c \quad \text{for} \quad -1 < c < 1 \tag{2.5}$$

forms a separating hyperplane that separates $\mathbf{x}_i\,(i = 1, \ldots, M)$. When $c = 0$, the separating hyperplane is in the middle of the two hyperplanes with $c = 1$ and -1. The distance between the separating hyperplane and the training datum nearest to the hyperplane is called the *margin*. Assuming that the hyperplanes $D(\mathbf{x}) = 1$ and -1 include at least one training datum, the hyperplane $D(\mathbf{x}) = 0$ has the maximum margin for $-1 < c < 1$. The region $\{\mathbf{x} \mid -1 \leq D(\mathbf{x}) \leq 1\}$ is the generalization region for the decision function.

Figure 2.1 shows two decision functions that satisfy (2.4). Thus there are an infinite number of decision functions that satisfy (2.4), which are separating hyperplanes. The generalization ability depends on the location of the separating hyperplane, and the hyperplane with the maximum margin is called the *optimal separating hyperplane* (see Fig. 2.1). Assume that no outliers are included in the training data and that unknown test data will obey the same probability law as that of the training data. Then it is intuitively clear that the generalization ability is maximized if the optimal separating hyperplane is selected as the separating hyperplane.

Now consider determining the optimal separating hyperplane. The Euclidean distance from a training datum $\mathbf{x}$ to the separating hyperplane is given by $|D(\mathbf{x})|/\|\mathbf{w}\|$. This can be shown as follows. Because the vector $\mathbf{w}$ is orthogonal to the separating hyperplane, the line that goes through $\mathbf{x}$ and that is orthogonal to the hyperplane is given by $a\mathbf{w}/\|\mathbf{w}\| + \mathbf{x}$, where $|a|$ is the Euclidean distance from $\mathbf{x}$ to the hyperplane. It crosses the hyperplane at the point where

$$D(a\mathbf{w}/\|\mathbf{w}\| + \mathbf{x}) = 0 \tag{2.6}$$

is satisfied. Solving (2.6) for a, we obtain $a = -D(\mathbf{x})/\|\mathbf{w}\|$.

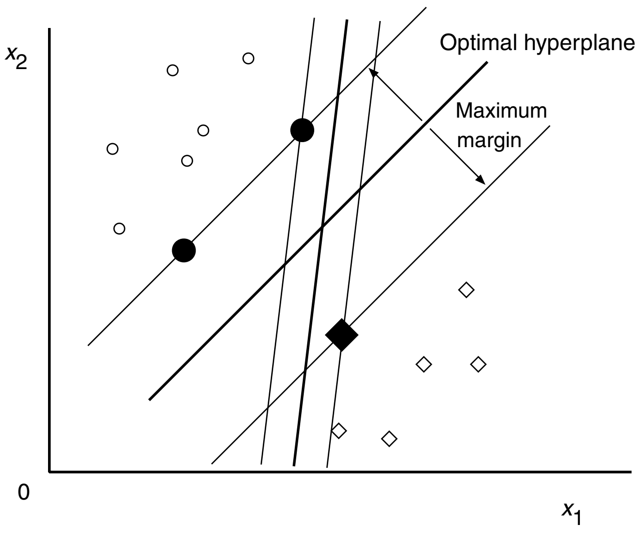


Fig. 2.1. Optimal separating hyperplane in a two-dimensional space

Then all the training data must satisfy

$$\frac{y_k D(\mathbf{x}_k)}{\|\mathbf{w}\|} \geq \delta \qquad \text{for} \quad k = 1, \ldots, M, \tag{2.7}$$

where δ is the margin.

Now if $(\mathbf{w}, b)$ is a solution, $(a\mathbf{w}, ab)$ is also a solution, where a is a scalar. Thus we impose the following constraint:

$$\delta \, \|\mathbf{w}\| = 1. \tag{2.8}$$

From (2.7) and (2.8), to find the optimal separating hyperplane, we need to find $\mathbf{w}$ with the minimum Euclidean norm that satisfies (2.4).

Therefore, the optimal separating hyperplane can be obtained by minimizing

$$Q(\mathbf{w}) = \frac{1}{2} \, \|\mathbf{w}\|^2 \tag{2.9}$$

with respect to $\mathbf{w}$ and b subject to the constraints:

$$y_i \left(\mathbf{w}^T \, \mathbf{x}_i + b\right) \geq 1 \qquad \text{for} \qquad i = 1, \ldots, M. \tag{2.10}$$

Here, the square of the Euclidean norm $\|\mathbf{w}\|$ in (2.9) is to make the optimization problem quadratic programming. The assumption of linear separability means that there exist $\mathbf{w}$ and b that satisfy (2.10). We call the solutions that satisfy (2.10) *feasible solutions.* Because the optimization problem has the quadratic objective function with the inequality constraints, even if the solutions are nonunique, the value of the objective function is unique (see Section 2.6.4). Thus nonuniqueness is not a problem for support vector machines. This is one of the advantages of support vector machines over neural networks, which have numerous local minima.

Because we can obtain the same optimal separating hyperplane even if we delete all the data that satisfy the strict inequalities in (2.10), the data that satisfy the equalities are called *support vectors*.[2] In Fig. 2.1, the data corresponding to the filled circles and the filled rectangle are support vectors.

The variables of the convex optimization problem given by (2.9) and (2.10) are $\mathbf{w}$ and b. Thus the number of variables is the number of input variables plus 1: $m+1$. When the number of input variables is small, we can solve (2.9) and (2.10) by the quadratic programming technique. But, as will be discussed later, because we map the input space into a high-dimensional feature space, in some cases, with infinite dimensions, we convert (2.9) and (2.10) into the equivalent dual problem whose number of variables is the number of training data.

To do this, we first convert the constrained problem given by (2.9) and (2.10) into the unconstrained problem

$$Q(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\, \mathbf{w}^T\, \mathbf{w} - \sum_{i=1}^{M} \alpha_i \, \{y_i\, (\mathbf{w}^T\, \mathbf{x}_i + b) - 1\}, \tag{2.11}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)^T$ and α_i are the nonnegative Lagrange multipliers. The optimal solution of (2.11) is given by the saddle point, where (2.11) is minimized with respect to $\mathbf{w}$ and b and maximized with respect to $\alpha_i \, (\geq 0)$, and it satisfies the following Karush-Kuhn-Tucker (KKT) conditions (see Theorem C.1):

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} = \mathbf{0}, \tag{2.12}$$

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial b} = 0, \tag{2.13}$$

$$\alpha_i \, \{y_i\, (\mathbf{w}^T\, \mathbf{x}_i + b) - 1\} = 0 \quad \text{for} \quad i = 1, \ldots, M, \tag{2.14}$$

$$\alpha_i \geq 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{2.15}$$

Especially, the relations between the inequality constraints and their associated Lagrange multipliers given by (2.14) are called *KKT complementarity conditions*. In the following, if there is no confusion, the KKT complementarity conditions are called the KKT conditions.

[2]This definition is imprecise. As shown in Definition 2.11, there are data that satisfy $y_i\,(\mathbf{w}^T\,\mathbf{x} + b) = 1$ but that can be deleted without changing the optimal separating hyperplane. Support vectors are defined using the solution of the dual problem, as discussed later.

From (2.14), $\alpha_i = 0$, or $\alpha_i \neq 0$ and $y_i\,(\mathbf{w}^T\,\mathbf{x}_i + b) = 1$ must be satisfied. The training data $\mathbf{x}_i$ with $\alpha_i \neq 0$ are called *support vectors*.[3]

Using (2.11), we reduce (2.12) and (2.13), respectively, to

$$\mathbf{w} = \sum_{i=1}^{M} \alpha_i\, y_i\, \mathbf{x}_i \tag{2.16}$$

and

$$\sum_{i=1}^{M} \alpha_i\, y_i = 0. \tag{2.17}$$

Substituting (2.16) and (2.17) into (2.11), we obtain the following dual problem. Maximize

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,\,j=1}^{M} \alpha_i\, \alpha_j\, y_i\, y_j\, \mathbf{x}_i^T\, \mathbf{x}_j \tag{2.18}$$

with respect to α_i subject to the constraints

$$\sum_{i=1}^{M} y_i\, \alpha_i = 0, \qquad \alpha_i \geq 0 \qquad \text{for} \quad i = 1, \ldots, M. \tag{2.19}$$

The formulated support vector machine is called the *hard-margin support vector machine*. Because

$$\frac{1}{2} \sum_{i,\,j=1}^{M} \alpha_i\, \alpha_j\, y_i\, y_j\, \mathbf{x}_i^T\, \mathbf{x}_j = \frac{1}{2} \left(\sum_{i=1}^{M} \alpha_i\, y_i\, \mathbf{x}_i \right)^T \left(\sum_{i=1}^{M} \alpha_i\, y_i\, \mathbf{x}_i \right) \geq 0, \tag{2.20}$$

maximizing (2.18) under the constraints (2.19) is a concave quadratic programming problem. If a solution exists, namely, if the classification problem is linearly separable, the global optimal solution α_i $(i = 1, \ldots, M)$ exists. For quadratic programming, the values of the primal and dual objective functions coincide at the optimal solutions if they exist. This is called the *zero duality gap*.

Data that are associated with positive α_i are support vectors for Classes 1 and 2. Then from (2.16) the decision function is given by

$$D(\mathbf{x}) = \sum_{i \in S} \alpha_i\, y_i\, \mathbf{x}_i^T\, \mathbf{x} + b, \tag{2.21}$$

where S is the set of support vector indices, and from the KKT conditions given by (2.14), b is given by

[3]In the definition of support vectors, we exclude the data in which both $\alpha_i = 0$ and $y_i\,(\mathbf{w}^T\,\mathbf{x}_i + b) = 1$ hold.

$$b = y_i - \mathbf{w}^T \mathbf{x}_i, \tag{2.22}$$

where $\mathbf{x}_i$ is a support vector. From the standpoint of precision of calculations, it is better to take the average among the support vectors as follows:

$$b = \frac{1}{|S|} \sum_{i \in S} (y_i - \mathbf{w}^T \mathbf{x}_i). \tag{2.23}$$

Then unknown datum $\mathbf{x}$ is classified into:

$$\begin{cases} \text{Class 1} & \text{if } D(\mathbf{x}) > 0, \\ \text{Class 2} & \text{if } D(\mathbf{x}) < 0. \end{cases} \tag{2.24}$$

If $D(\mathbf{x}) = 0$, $\mathbf{x}$ is on the boundary and thus is unclassifiable. When training data are separable, the region $\{\mathbf{x} \mid 1 > D(\mathbf{x}) > -1\}$ is a generalization region.

Example 2.1. Consider a linearly separable case shown in Fig. 2.2. The inequality constraints given by (2.10) are

$$-w + b \geq 1, \tag{2.25}$$
$$-b \geq 1, \tag{2.26}$$
$$-(w + b) \geq 1. \tag{2.27}$$

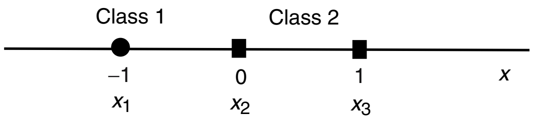


Fig. 2.2. Linearly separable one-dimensional case

The region of (w, b) that satisfies (2.25) to (2.27) are given by the shaded region shown in Fig. 2.3. Thus the solution that minimizes $\|w\|^2$ is given by

$$b = -1, \qquad w = -2. \tag{2.28}$$

Namely, the decision function is given by

$$D(x) = -2\,x - 1. \tag{2.29}$$

The class boundary is given by $x = -1/2$. Because the solution is determined by (2.25) and (2.26), $x = 0$ and -1 are support vectors.

The dual problem is given as follows. Maximize

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 + \alpha_3 - \frac{1}{2}\,(\alpha_1 + \alpha_3)^2 \tag{2.30}$$

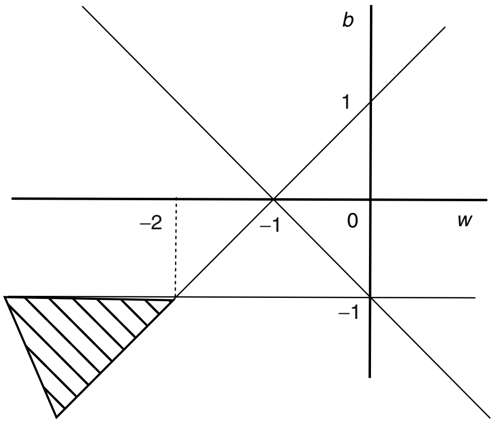


Fig. 2.3. Region that satisfies constraints

subject to

$$\alpha_1 - \alpha_2 - \alpha_3 = 0, \tag{2.31}$$

$$\alpha_i \geq 0 \qquad \text{for} \quad i = 1, 2, 3. \tag{2.32}$$

From (2.31), $\alpha_2 = \alpha_1 - \alpha_3$. Substituting it into (2.30), we obtain

$$Q(\boldsymbol{\alpha}) = 2\,\alpha_1 - \frac{1}{2}\,(\alpha_1 + \alpha_3)^2, \tag{2.33}$$

$$\alpha_i \geq 0 \qquad \text{for} \quad i = 1, 2, 3. \tag{2.34}$$

Because $\alpha_1 \geq 0$ and $\alpha_3 \geq 0$, (2.33) is maximized when $\alpha_3 = 0$. Thus, (2.33) reduces to

$$\begin{aligned} Q(\boldsymbol{\alpha}) &= 2\,\alpha_1 - \frac{1}{2}\,\alpha_1^2 \\ &= -\frac{1}{2}\,(\alpha_1 - 2)^2 + 2, \end{aligned} \tag{2.35}$$

$$\alpha_1 \geq 0. \tag{2.36}$$

Because (2.35) is maximized for $\alpha_1 = 2$, the optimal solution for (2.30) is

$$\alpha_1 = 2, \quad \alpha_2 = 2, \quad \alpha_3 = 0. \tag{2.37}$$

Therefore, $x = -1$ and 0 are support vectors and $w = -2$ and $b = -1$, which are the same as the solution obtained by solving the primary problem. In addition, because $Q(\mathbf{w}) = Q(\boldsymbol{\alpha}) = 2$, the duality gap is zero.

Consider changing the label of x_3 into that of the opposite class, i.e., $y_3 = 1$. Then the problem becomes inseparable and (2.27) becomes $w + b \geq 1$. Thus, from Fig 2.3 there is no feasible solution.

2.2 L1 Soft-Margin Support Vector Machines

In hard-margin support vector machines, we assumed that the training data are linearly separable. When the data are linearly inseparable, there is no feasible solution, and the hard-margin support vector machine is unsolvable. Here we extend the support vector machine so that it is applicable to an inseparable case.

To allow inseparability, we introduce the nonnegative slack variables ξ_i (≥ 0) into (2.4):

$$y_i\,(\mathbf{w}^T\,\mathbf{x}_i + b) \geq 1 - \xi_i \qquad \text{for} \quad i = 1, \ldots, M. \tag{2.38}$$

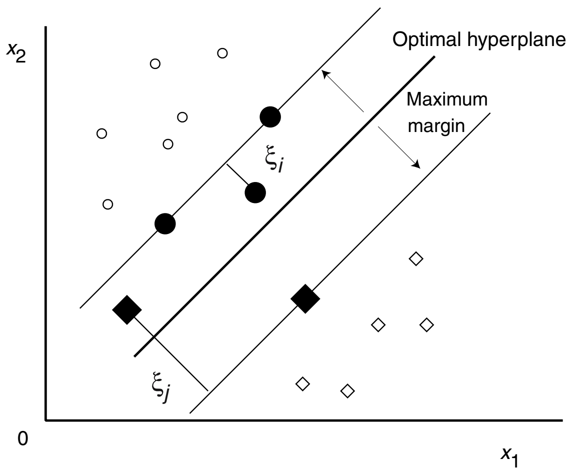


Fig. 2.4. Inseparable case in a two-dimensional space

By the slack variables ξ_i, feasible solutions always exist. For the training data $\mathbf{x}_i$, if $0 < \xi_i < 1$ (ξ_i in Fig. 2.4), the data do not have the maximum margin but are still correctly classified. But if $\xi_i \geq 1$ (ξ_j in Fig. 2.4) the data are misclassified by the optimal hyperplane. To obtain the optimal hyperplane in which the number of training data that do not have the maximum margin is minimum, we need to minimize

$$Q(\mathbf{w}) = \sum_{i=1}^{M} \theta(\xi_i),$$

where

$$\theta(\xi_i) = \begin{cases} 1 & \text{for} \quad \xi_i > 0, \\ 0 & \text{for} \quad \xi_i = 0. \end{cases}$$

But this is a combinatorial optimization problem and difficult to solve. Instead, we consider minimizing

$$Q(\mathbf{w}, b, \boldsymbol{\xi}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{M} \xi_i^p \tag{2.39}$$

subject to the constraints

$$y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right) \geq 1 - \xi_i \qquad \text{for} \qquad i = 1, \ldots, M, \tag{2.40}$$

where $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_M)^T$ and C is the margin parameter that determines the trade-off between the maximization of the margin and minimization of the classification error. We select the value of p as either 1 or 2. We call the obtained hyperplane the *soft-margin hyperplane*. When $p = 1$, we call the support vector machine the *L1 soft-margin support vector machine* or the L1 support vector machine for short (L1 SVM) and when $p = 2$, the L2 soft-margin support vector machine or L2 support vector machine (L2 SVM). In this section, we discuss L1 soft-margin support vector machines.

Similar to the linearly separable case, introducing the nonnegative Lagrange multipliers α_i and β_i, we obtain

$$\begin{aligned} Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = {} & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{M} \xi_i \\ & - \sum_{i=1}^{M} \alpha_i \left(y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right) - 1 + \xi_i\right) - \sum_{i=1}^{M} \beta_i \, \xi_i, \end{aligned} \tag{2.41}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)^T$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_M)^T$.

For the optimal solution, the following KKT conditions are satisfied (see Theorem C.1):

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \mathbf{w}} = \mathbf{0}, \tag{2.42}$$

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial b} = 0, \tag{2.43}$$

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \boldsymbol{\xi}} = \mathbf{0}. \tag{2.44}$$

$$\alpha_i \left(y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right) - 1 + \xi_i\right) = 0 \quad \text{for} \quad i = 1, \ldots, M, \tag{2.45}$$

$$\beta_i \, \xi_i = 0 \quad \text{for} \quad i = 1, \ldots, M, \tag{2.46}$$

$$\alpha_i \geq 0, \quad \beta_i \geq 0, \quad \xi_i \geq 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{2.47}$$

Using (2.41), we reduce (2.42) to (2.44), respectively, to

$$\mathbf{w} = \sum_{i=1}^{M} \alpha_i \, y_i \, \mathbf{x}_i, \tag{2.48}$$

$$\sum_{i=1}^{M} \alpha_i \, y_i = 0, \tag{2.49}$$

$$\alpha_i + \beta_i = C \quad \text{for} \quad i = 1, \ldots, M. \tag{2.50}$$

Thus substituting (2.48) to (2.50) into (2.41), we obtain the following dual problem. Maximize

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \, \alpha_j \, y_i \, y_j \, \mathbf{x}_i^T \, \mathbf{x}_j \tag{2.51}$$

subject to the constraints

$$\sum_{i=1}^{M} y_i \, \alpha_i = 0, \qquad C \geq \alpha_i \geq 0 \qquad \text{for} \quad i = 1, \ldots, M. \tag{2.52}$$

The only difference between L1 soft-margin support vector machines and hard-margin support vector machines is that α_i cannot exceed C.

Especially, (2.45) and (2.46) are called *KKT (complementarity) conditions.* From these and (2.50), there are three cases for α_i:

1. $\alpha_i = 0$. Then $\xi_i = 0$. Thus $\mathbf{x}_i$ is correctly classified.
2. $0 < \alpha_i < C$. Then $y_i \, (\mathbf{w}^T \, \mathbf{x}_i + b) - 1 + \xi_i = 0$ and $\xi_i = 0$. Therefore, $y_i \, (\mathbf{w}^T \, \mathbf{x}_i + b) = 1$ and $\mathbf{x}_i$ is a support vector. Especially, we call the support vector with $C > \alpha_i > 0$ an *unbounded support vector.*
3. $\alpha_i = C$. Then $y_i \, (\mathbf{w}^T \, \mathbf{x}_i + b) - 1 + \xi_i = 0$ and $\xi_i \geq 0$. Thus $\mathbf{x}_i$ is a support vector. We call the support vector with $\alpha_i = C$ a *bounded support vector.* If $0 \leq \xi_i < 1$, $\mathbf{x}_i$ is correctly classified, and if $\xi_i \geq 1$, $\mathbf{x}_i$ is misclassified.

The decision function is the same as that of the hard-margin support vector machine and is given by

$$D(\mathbf{x}) = \sum_{i \in S} \alpha_i \, y_i \, \mathbf{x}_i^T \, \mathbf{x} + b, \tag{2.53}$$

where S is the set of support vector indices. Because α_i are nonzero for the support vectors, the summation in (2.53) is added only for the support vectors. For the unbounded α_i,

$$b = y_i - \mathbf{w}^T \mathbf{x}_i \tag{2.54}$$

is satisfied. To ensure the precision of calculations, we take the average of b that is calculated for unbounded support vectors,

$$b = \frac{1}{|U|} \sum_{i \in U} (y_i - \mathbf{w}^T \mathbf{x}_i), \tag{2.55}$$

where U is the set of unbounded support vector indices.

Then unknown datum $\mathbf{x}$ is classified into

$$\begin{cases} \text{Class 1} & \text{if } D(\mathbf{x}) > 0, \\ \text{Class 2} & \text{if } D(\mathbf{x}) < 0. \end{cases} \tag{2.56}$$

If $D(\mathbf{x}) = 0$, $\mathbf{x}$ is on the boundary and thus is unclassifiable. When there are no bounded support vectors, the region $\{\mathbf{x} \mid 1 > D(\mathbf{x}) > -1\}$ is a generalization region, which is the same as the hard-margin support vector machine.

2.3 Mapping to a High-Dimensional Space

2.3.1 Kernel Tricks

In a support vector machine the optimal hyperplane is determined to maximize the generalization ability. But if the training data are not linearly separable, the obtained classifier may not have high generalization ability although the hyperplanes are determined optimally. Thus to enhance linear separability, the original input space is mapped into a high-dimensional dot-product space called the *feature space.*

Now using the nonlinear vector function $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_l(\mathbf{x}))^T$ that maps the m-dimensional input vector $\mathbf{x}$ into the l-dimensional feature space, the linear decision function in the feature space is given by

$$D(\mathbf{x}) = \mathbf{w}^T \mathbf{g}(\mathbf{x}) + b, \tag{2.57}$$

where $\mathbf{w}$ is an l-dimensional vector and b is a bias term.

According to the Hilbert-Schmidt theory (see Appendix D), if a symmetric function $H(\mathbf{x}, \mathbf{x}')$ satisfies

$$\sum_{i,j=1}^{M} h_i \, h_j H(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \tag{2.58}$$

for all M, $\mathbf{x}_i$, and h_i, where M takes on a natural number and h_i take on real numbers, there exists a mapping function, $\mathbf{g}(\mathbf{x})$, that maps $\mathbf{x}$ into the dot-product feature space and $\mathbf{g}(\mathbf{x})$ satisfies

$$H(\mathbf{x}, \mathbf{x}') = \mathbf{g}^T(\mathbf{x}) \, \mathbf{g}(\mathbf{x}'). \tag{2.59}$$

If (2.59) is satisfied,

$$\sum_{i,j=1}^{M} h_i \, h_j H(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{i=1}^{M} h_i \, \mathbf{g}^T(\mathbf{x}_i) \right) \left(\sum_{i=1}^{M} h_i \, \mathbf{g}(\mathbf{x}_i) \right) \geq 0. \tag{2.60}$$

The condition (2.58) or (2.60) is called *Mercer's condition*, and the function that satisfies (2.58) or (2.60) is called the *positive semidefinite kernel* or the

Mercer kernel. In the following, if there is no confusion, we simply call it the *kernel.*

The advantage of using kernels is that we need not treat the high-dimensional feature space explicitly. This technique is called *kernel trick.* Namely, we use $H(\mathbf{x}, \mathbf{x}')$ in training and classification instead of $\mathbf{g}(\mathbf{x})$ as shown later.

Using the kernel, the dual problem in the feature space is given as follows. Maximize

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) \tag{2.61}$$

subject to the constraints

$$\sum_{i=1}^{M} y_i \, \alpha_i = 0, \qquad 0 \leq \alpha_i \leq C \qquad \text{for} \quad i = 1, \ldots, M. \tag{2.62}$$

Because $H(\mathbf{x}, \mathbf{x}')$ is a positive semidefinite kernel, the optimization problem is a concave quadratic programming problem. And because $\boldsymbol{\alpha} = \mathbf{0}$ is a feasible solution, the problem has the global optimum solution.

The KKT complementarity conditions are given by

$$\alpha_i \left(y_i \left(\sum_{j=1}^{M} y_j \, \alpha_j \, H(\mathbf{x}_i, \mathbf{x}_j) + b \right) - 1 + \xi_i \right) = 0$$
$$\text{for} \quad i = 1, \ldots, M, \tag{2.63}$$
$$(C - \alpha_i) \, \xi_i = 0 \quad \text{for} \quad i = 1, \ldots, M, \tag{2.64}$$
$$\alpha_i \geq 0, \quad \xi_i \geq 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{2.65}$$

The decision function is given by

$$D(\mathbf{x}) = \sum_{i \in S} \alpha_i \, y_i \, H(\mathbf{x}_i, \mathbf{x}) + b, \tag{2.66}$$

where b is given by

$$b = y_j - \sum_{i \in S} \alpha_i y_i H(\mathbf{x}_i, \mathbf{x}_j). \tag{2.67}$$

Here, $\mathbf{x}_j$ is an unbounded support vector. To ensure stability of calculations, we take the average:

$$b = \frac{1}{U} \sum_{j \in U} \left(y_j - \sum_{i \in S} \alpha_i y_i H(\mathbf{x}_i, \mathbf{x}_j) \right), \tag{2.68}$$

where U is the set of unbounded support vector indices.

Then unknown data are classified using the decision function as follows:

$$\mathbf{x} \in \begin{cases} \text{Class 1} & \text{if} \quad D(\mathbf{x}) > 0, \\ \text{Class 2} & \text{if} \quad D(\mathbf{x}) < 0. \end{cases} \tag{2.69}$$

If $D(\mathbf{x}) = 0$, $\mathbf{x}$ is unclassifiable.

2.3.2 Kernels

In the following we discuss some of the kernels that are used in support vector machines. For the properties of kernels, see Appendix D.1.

Linear Kernels

If a classification problem is linearly separable in the input space, we need not map the input space into a high-dimensional space. In such a situation we use linear kernels:

$$H(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \, \mathbf{x}'. \tag{2.70}$$

Polynomial Kernels

The polynomial kernel with degree d, where d is a natural number, is given by

$$H(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \, \mathbf{x}' + 1)^d. \tag{2.71}$$

Here, 1 is added so that cross terms with degrees equal to or less than d are all included.

When $d = 1$, the kernel is the linear kernel plus 1. Thus, by adjusting b in the decision function, it is equivalent to the linear kernel. For $d = 2$ and $m = 2$, the polynomial kernel given by (2.71) becomes

$$\begin{aligned} H(\mathbf{x}, \mathbf{x}') &= 1 + 2x_1 \, x_1' + 2x_2 \, x_2' + 2x_1 \, x_1' \, x_2 \, x_2' + x_1^2 \, x_1'^2 + x_2^2 \, x_2'^2 \\ &= \mathbf{g}^T(\mathbf{x}) \, \mathbf{g}(\mathbf{x}), \end{aligned} \tag{2.72}$$

where $\mathbf{g}(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2)^T$. Thus for $d = 2$ and $m = 2$, polynomial kernels satisfy Mercer's condition. In general, we can prove that polynomial kernels satisfy Mercer's condition (see Appendix D.1).

Instead of (2.71), the following polynomial kernel can be used:

$$H(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \, \mathbf{x}')^d. \tag{2.73}$$

However, in this book we use (2.71), which is more general.[4]

Radial Basis Function Kernels

The radial basis function (RBF) kernel is given by

$$H(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \, \|\mathbf{x} - \mathbf{x}'\|^2), \tag{2.74}$$

where γ is a positive parameter for controlling the radius. Rewriting (2.74),

[4]In Section 2.3.4, we will show that the mapping functions associated with (2.73) are many to one for even d, which is unfavorable.

$$H(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \, \|\mathbf{x}\|^2) \exp(-\gamma \, \|\mathbf{x}'\|^2) \exp(2\gamma \, \mathbf{x}^T \mathbf{x}'). \tag{2.75}$$

Because

$$\exp(2\gamma \, \mathbf{x}^T \mathbf{x}') = 1 + 2\gamma \, \mathbf{x}^T \mathbf{x}' + 2\gamma^2 \, (\mathbf{x}^T \mathbf{x}')^2 + \frac{(2\gamma)^3}{3!} \, (\mathbf{x}^T \mathbf{x}')^3 + \cdots, \tag{2.76}$$

$\exp(2\gamma \, \mathbf{x}^T \mathbf{x}')$ is an infinite summation of polynomials. Thus it is a kernel. In addition, $\exp(-\gamma \, \|\mathbf{x}\|^2)$ and $\exp(-\gamma \, \|\mathbf{x}'\|^2)$ are proved to be kernels and the product of kernels is also a kernel (see Appendix D.1). Thus (2.75) is a kernel.

From (2.66), the resulting decision function is given by

$$D(\mathbf{x}) = \sum_{i \in S} \alpha_i \, y_i \, \exp(-\gamma \, \|\mathbf{x}_i - \mathbf{x}\|^2) + b. \tag{2.77}$$

Here, the support vectors are the centers of the radial basis functions.

Because RBF kernels use the Euclidean distance, they are not robust to outliers. To overcome this, Chen [57] proposed the *M*-estimator-based robust kernels, which are robust versions of RBF kernels, introducing the idea of robust statistics.

Three-Layer Neural Network Kernels

The three-layer neural network kernel is given by

$$H(\mathbf{x}, \mathbf{x}') = \frac{1}{1 + \exp(\nu \, \mathbf{x}^T \, \mathbf{x}' - a)}, \tag{2.78}$$

where ν and a are constants. This kernel does not always satisfy Mercer's condition; we need to determine the values of ν and a so that (2.58) is satisfied [256, p. 141], [60, p. 369].[5]

From (2.66), the resulting decision function is given by

$$D(\mathbf{x}) = \sum_{i \in S} \frac{\alpha_i \, y_i}{1 + \exp(\nu \, \mathbf{x}_i^T \mathbf{x} - a)} + b. \tag{2.79}$$

The support vectors correspond to the weights between the input and hidden neurons in the three-layer neural network.

Because Mercer's condition is not always satisfied for three-layer neural network kernels, several approaches are made to overcome this problem [178, 239] (see Chapter 9).

[5]In [215], neural network kernels are shown to be indefinite.

Other Kernels

There are other kernels, such as spline functions [102, 144, 257] and the Mahalanobis kernels [109] that are used for support vector machines. In addition, many kernels have been developed for specific applications such as image processing, text classification, and speech recognition, where inputs have variable lengths. Because support vector machines are based on fixed-length inputs, we need to extract fixed-length features or extend kernels that handle variable-length inputs. In the following, we discuss some of their kernels.

Because correlation among image pixels is localized, global kernels such as polynomial kernels are inadequate for image classification, and local kernels are studied [39, 214]. Barla et al. [25] discussed two image kernels: histogram intersection kernels, which measure the similarity of two color images, and Hausdorff kernels, which measure the similarity of two grayscale images. Hotta [113] used the summation of RBF kernels for robust occluded face recognition.

In text classification, documents are classified into one of several topics. The common approach uses, as features, a histogram of words called a *bag of words* [123]. Lodhi et al. [155] used string kernels that give the similarity of a common substring in two documents. Let the substring length be 2 and the words be *cat* and *bat*. Then we obtain five substrings: c-a, c-t, a-t, b-a, and b-t, which constitute variables in the feature space. The mapping function is defined so that the substring that is nearer in a document (in this case a word) has a higher score, as shown in Table 2.1, where $0 < \lambda < 1$. For example, c-a in *cat*, which is a contiguous substring, has a higher score than c-t in *cat*. The resulting kernel is calculated as follows:

$$\begin{aligned} H(cat, bat) &= \lambda^4 \\ H(cat, cat) &= H(bat, bat) = 2\lambda^4 + \lambda^6. \end{aligned}$$

The whole document is mapped into one feature space, ignoring punctuation and retaining spaces. For a long substring length, evaluation of kernels is sped up by dynamic programming.

Table 2.1. Mapping function for *cat* and *bat*

	c-a	c-t	a-t	b-a	b-t
$\mathbf{g}(cat)$	λ^2	λ^3	λ^2	0	0
$\mathbf{g}(bat)$	0	0	λ^2	λ^2	λ^3

Leslie et al. [145] developed mismatch kernels, which belong to a class of string kernels, for protein classification. The mapping function of the (k, m)-mismatch kernel maps the space of all finite strings with an alphabet of size l

to an l^k-dimensional space, i.e., all the combinations of k-length strings. For the input string of length k, the value of a variable in the feature space is 1 if the input string differs from the variable string at at most k mismatches and 0 otherwise. For an arbitrary input, the value of a variable is a mismatch count of all k-substrings included in the input. Then the kernel value is calculated by the dot-product of the variables in the feature space.

In speech recognition, sequences of data with different lengths need to be matched [64, 223, 227]. Shimodaira et al. [223] proposed dynamic time-alignment kernels. Let two sequences of vectors be $X = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$ and $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$, where m and n are, in general, different. The dynamic time-alignment kernel is defined by

$$\begin{aligned} H_s(X, V) &= \max_{\phi,\theta} \frac{1}{M_{\phi\theta}} \sum_{k=1}^{L} m(k)\, \mathbf{g}^T(\mathbf{x}_{\phi(k)})\, \mathbf{g}(\mathbf{x}_{\theta(k)}) \\ &= \max_{\phi,\theta} \frac{1}{M_{\phi\theta}} \sum_{k=1}^{L} m(k)\, H(\mathbf{x}_{\phi(k)}, \mathbf{x}_{\theta(k)}), \end{aligned} \tag{2.80}$$

where L is the normalized length, $m(k)$ are weights, $M_{\phi\theta}$ is a normalizing factor, $\phi(\cdot)$ and $\theta(\cdot)$ are time-alignment functions that align two sequences by dynamic programming so that the two become similar.

2.3.3 Normalizing Kernels

If the number of input variables is very large, the value of a kernel becomes so small or large that training of support vector machines becomes difficult. To overcome this, it is advisable to normalize kernels.

For a polynomial kernel with degree d, the maximum value is $(m+1)^d$ for the input range of $[0, 1]$, where m is the number of input variables. Thus we use the following normalized polynomial:

$$H(\mathbf{x}, \mathbf{x}') = \frac{(\mathbf{x}^T\mathbf{x}' + 1)^d}{(m+1)^d} \tag{2.81}$$

for large m. In the computer experiments in this book, we used (2.81) for m larger than or equal to 100.

For an RBF kernel, the maximum value of $\|\mathbf{x} - \mathbf{x}'\|^2$ is m for the input range of $[0, 1]$. Thus we use the following normalized RBF kernel:

$$H(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\gamma}{m}\, \|\mathbf{x} - \mathbf{x}'\|^2\right). \tag{2.82}$$

The use of (2.82) is favorable for choosing a proper value of γ for problems with different numbers of input variables.

Example 2.2. In Fig. 2.2 let x_3 $(= 1)$ belong to class 1 so that the problem is inseparable (see Fig. 2.5). Using the polynomial kernel with degree 2, the dual problem is given as follows. Maximize

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 + \alpha_3 - (2\,\alpha_1^2 + \frac{1}{2}\,\alpha_2^2 + 2\,\alpha_3^2 - \alpha_2\,(\alpha_1 + \alpha_3)) \qquad (2.83)$$

subject to

$$\alpha_1 - \alpha_2 + \alpha_3 = 0, \qquad (2.84)$$
$$C \geq \alpha_i \geq 0 \qquad \text{for} \quad i = 1, 2, 3. \qquad (2.85)$$

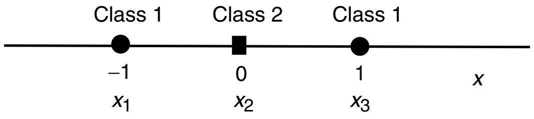


Fig. 2.5. Inseparable one-dimensional case

From (2.84), $\alpha_2 = \alpha_1 + \alpha_3$. Then substituting it into (2.83), we obtain

$$Q(\boldsymbol{\alpha}) = 2\,\alpha_1 + 2\,\alpha_3 - (2\,\alpha_1^2 - \frac{1}{2}\,(\alpha_1 + \alpha_3)^2 + 2\,\alpha_3^2), \qquad (2.86)$$
$$C \geq \alpha_i \geq 0 \qquad \text{for} \quad i = 1, 2, 3. \qquad (2.87)$$

From

$$\frac{\partial Q(\boldsymbol{\alpha})}{\partial \alpha_1} = 2 - 3\,\alpha_1 + \alpha_3 = 0, \qquad (2.88)$$
$$\frac{\partial Q(\boldsymbol{\alpha})}{\partial \alpha_3} = 2 + \alpha_1 - 3\,\alpha_3 = 0, \qquad (2.89)$$

$\alpha_1 = \alpha_3 = 1$. Thus, for $C \geq 2$, the optimal solution is

$$\alpha_1 = 1, \quad \alpha_2 = 2, \quad \alpha_3 = 1. \qquad (2.90)$$

Therefore, for $C \geq 2$, $x = -1$, 0, and 1 are support vectors. From (2.67), $b = -1$. Then the decision function is given by (see Fig. 2.6)

$$\begin{aligned} D(x) &= (x-1)^2 + (x+1)^2 - 3 \\ &= 2x^2 - 1. \end{aligned} \qquad (2.91)$$

The decision boundaries are given by $x = \pm\sqrt{2}/2$. Therefore, in the input space the margin for Class 2 is larger than that for Class 1, although they are the same in the feature space.

2.3.4 Properties of Mapping Functions Associated with Kernels

If the mapping function associated with a nonlinear kernel is many-to-one mapping, namely, more than one point in the input space are mapped to a

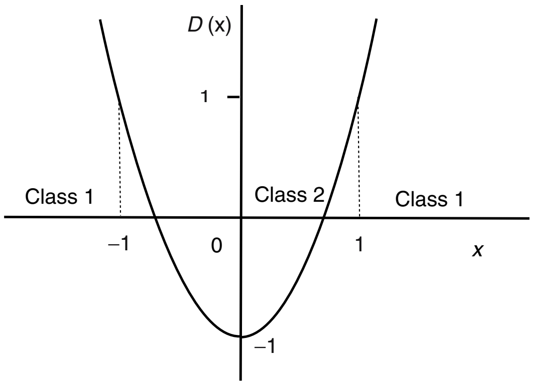


Fig. 2.6. Decision function for the inseparable one-dimensional case

single point in the feature space, a classification problem may become inseparable in the feature space even if the problem is separable in the input space. In this section, we discuss that this does not happen for polynomial kernels with constant terms and RBF kernels. But for polynomial kernels without constant terms, many-to-one mapping may occur. Then we clarify that the mapped region is not convex. Namely, the preimage of a linear sum of the mapped input data does not exist.

One-to-One Mapping

The mapping function $\mathbf{g}(\mathbf{x})$ associated with the polynomial kernel given by (2.71) is

$$\mathbf{g}(\mathbf{x}) = (1, \sqrt{d}\, x_1, \ldots, \sqrt{d}\, x_m, \ldots, x_1^d, \ldots, x_m^d)^T. \tag{2.92}$$

Thus, if $\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{x}')$, $\mathbf{x} = \mathbf{x}'$. Therefore, mapping is one to one.

But for $H(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^d$ with even d, the mapping is many to one. For example, for $d = 2$,

$$\mathbf{g}(\mathbf{x}) = (x_1^2, \ldots, x_m^2, \sqrt{2}\, x_1\, x_2, \ldots, \sqrt{2}\, x_{m-1}\, x_m)^T. \tag{2.93}$$

Thus, if $\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{x}')$,

$$\begin{aligned} x_1^2 &= x_1'^2, \\ &\cdots \\ x_m^2 &= x_m'^2, \\ x_1\, x_2 &= x_1'\, x_2', \\ &\cdots \\ x_{m-1}\, x_m &= x_{m-1}'\, x_m'. \end{aligned}$$

This set of simultaneous equations is satisfied if $\mathbf{x} = \mathbf{x}'$ or $\mathbf{x} = -\mathbf{x}'$. For the input range of $[0, 1]$, the mapping is one to one, because we can exclude $\mathbf{x} = -\mathbf{x}'$. But for the range of $[-1, 1]$, the mapping is two to one. A similar discussion holds for even d. Therefore, we should avoid using the scale $[-1, 1]$ for even d.

For the RBF kernel, if $\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{x}')$,

$$\begin{aligned} H(\mathbf{x}, \mathbf{x}') &= \mathbf{g}^T(\mathbf{x})\, \mathbf{g}(\mathbf{x}') = \exp\left(-\gamma\, \|\mathbf{x} - \mathbf{x}'\|\right) \\ &= \mathbf{g}^T(\mathbf{x})\, \mathbf{g}(\mathbf{x}) \\ &= H(\mathbf{x}, \mathbf{x}) \\ &= 1. \end{aligned}$$

Therefore, $\mathbf{x} = \mathbf{x}'$. Thus, the mapping is one to one.

Nonconvexity of Mapped Regions

Let G be the mapped region of the m-dimensional input space R^m by $\mathbf{g}(\cdot)$:

$$G = \{\mathbf{g}(\mathbf{x}) \,|\, \mathbf{x} \in R^m\}. \tag{2.94}$$

Then for polynomial kernels and RBF kernels, G is not convex. Namely, for $\mathbf{z}$ and $\mathbf{z}' \in G$, there exists α $(1 > \alpha > 0)$ such that $\alpha\, \mathbf{z} + (1 - \alpha)\, \mathbf{z}'$ is not included in G. This can be shown as follows.

For a polynomial kernel, it is sufficient to show nonconvexity for the axis x_i^d. Because $\alpha\, x_i^d + (1 - \alpha)\, x_i'^d$ $(1 > \alpha > 0)$ is a line segment that connects x_i^d and $x_i'^d$, there is no x that satisfies $x^d = \alpha\, x_i^d + (1 - \alpha)\, x_i'^d$ (see Fig. 2.7). Thus, G is not convex.

For an RBF kernel, because $H(\mathbf{x}, \mathbf{x}) = \mathbf{g}^T(\mathbf{x})\, \mathbf{g}(\mathbf{x}) = 1$, $\mathbf{g}(\mathbf{x})$ is on the surface of the hypersphere with the radius of 1 and the center being at the origin. Thus $\alpha\, \mathbf{z} + (1 - \alpha)\, \mathbf{z}'$ for $1 > \alpha > 0$ is not in G.

Now, for $\mathbf{z} \notin G$, there is no $\mathbf{x}$ that satisfies $\mathbf{g}(\mathbf{x}) = \mathbf{z}$. Namely, the preimage of $\mathbf{z}$ exists if and only if $\mathbf{z} \in G$. And, in general for $\mathbf{z}_1, \ldots, \mathbf{z}_k \in G$, the preimage of a linear sum of $\mathbf{z}_1, \ldots, \mathbf{z}_k$ does not exist.

2.3.5 Implicit Bias Terms

If a kernel includes a constant term instead of (2.57), we can use the decision function without an explicit bias term:

$$D(\mathbf{x}) = \mathbf{w}^T \mathbf{g}(\mathbf{x}). \tag{2.95}$$

In this case, the dual optimization problem becomes as follows. Maximize

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i\, \alpha_j\, y_i\, y_j\, H(\mathbf{x}_i, \mathbf{x}_j) \tag{2.96}$$

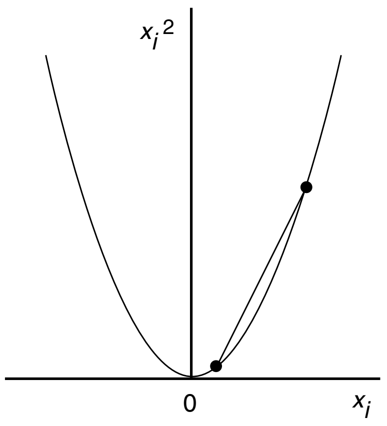


Fig. 2.7. Nonconvexity of the mapped region. For $d = 2$, the line segment in the x_i^2 axis is outside of the mapped region.

subject to

$$C \geq \alpha_i \geq 0 \qquad \text{for} \quad i = 1, \ldots, M. \tag{2.97}$$

The decision function is given by

$$D(\mathbf{x}) = \sum_{i \in S} \alpha_i \, y_i \, H(\mathbf{x}_i, \mathbf{x}). \tag{2.98}$$

Elimination of the bias term results in elimination of the equality constraint in the dual problem. Thus the problem is more easily solved. Polynomial kernels and RBF kernels include bias terms. And even if a kernel does not include a constant term, adding 1, we obtain the kernel with a constant term. For $H(\mathbf{x}, \mathbf{x}') = \mathbf{g}^T(\mathbf{x}) \, \mathbf{g}(\mathbf{x})$, the kernel

$$\begin{aligned} H'(\mathbf{x}, \mathbf{x}') &= H(\mathbf{x}, \mathbf{x}') + 1 \\ &= (\mathbf{g}^T(\mathbf{x}), 1) \, (\mathbf{g}^T(\mathbf{x}), 1)^T \end{aligned} \tag{2.99}$$

has a constant term. For example, for the linear kernel $H(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}$, $H'(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}' + 1$ has a constant term.

Unfortunately, the solutions of both formulations are, in general, different even for the linear kernels [101, p. 22] as Example 2.3 shows. According to the computer experiment by Huang and Kecman [115], the solution of the support vector machine without bias terms has a lager number of support vectors. Therefore, to suppress the increase, they developed a model that includes Mangasarian and Musicant's model as a special case.

Example 2.3. Suppose that there are only two data: $\mathbf{x}_1$ belonging to Class 1 and $\mathbf{x}_2$ belonging to Class 2. Let $H'(\mathbf{x}, \mathbf{x}') = H(\mathbf{x}, \mathbf{x}') + 1$ and assume that $H(\mathbf{x}_i, \mathbf{x}_j) \geq 0$ for $i, j = 1, 2$.[6]

We study the relation of the solutions with and without bias terms. Assume that $\mathbf{x}_1 \neq \mathbf{x}_2$. Then the problem is linearly separable. So we consider the hard-margin support vector machine. The dual problem with the explicit bias term is as follows. Maximize

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 - \frac{1}{2}\,(\alpha_1^2\, H_{11} + \alpha_2^2\, H_{22} - 2\,\alpha_1\,\alpha_2\, H_{12}) \tag{2.100}$$

subject to

$$\alpha_1 - \alpha_2 = 0, \quad \alpha_1 \geq 0, \quad \alpha_2 \geq 0. \tag{2.101}$$

Here, for example, $H_{12} = H(\mathbf{x}_1, \mathbf{x}_2)$.

Substituting the first equation in (2.101) into (2.100), we obtain the solution:

$$\alpha_1 = \alpha_2 = \frac{2}{H}, \tag{2.102}$$

$$b = \frac{1}{H}\,(H_{22} - H_{11}), \tag{2.103}$$

where $H = H_{11} + H_{22} - 2H_{12} > 0$.

Thus the decision function is given by

$$D(\mathbf{x}) = \frac{2}{H}\,(H(\mathbf{x}_1, \mathbf{x}) - H(\mathbf{x}_2, \mathbf{x})) + \frac{1}{H}\,(H_{22} - H_{11}). \tag{2.104}$$

From (2.104), $D(\mathbf{x}_1) = 1$ and $D(\mathbf{x}_2) = -1$, which satisfy the KKT conditions. For $H(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$, (2.104) becomes

$$D(\mathbf{x}) = \frac{1}{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}\,(2\,(\mathbf{x}_1 - \mathbf{x}_2)^T \mathbf{x} + \mathbf{x}_2^T\,\mathbf{x}_2 - \mathbf{x}_1^T\,\mathbf{x}_1). \tag{2.105}$$

Thus the decision function is orthogonal to $(\mathbf{x}_1 - \mathbf{x}_2)$ and the decision boundary passes through the middle point of $\mathbf{x}_1$ and $\mathbf{x}_2$.

Now consider the special case where the input variable has one dimension. For $H(x, x') = (x\, x')^d$, the decision function is

$$D(\mathbf{x}) = \frac{2x^d - x_1^d - x_2^d}{x_1^d - x_2^d}. \tag{2.106}$$

Thus the decision function vanishes when

$$x = \sqrt[d]{\frac{x_1^d + x_2^d}{2}}. \tag{2.107}$$

[6]This assumption is satisfied when the input variables are nonnegative and $H(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x})^d$.

This means that for $d = 1$, the decision boundary passes through the middle point of x_1 and x_2, but for d larger than 1, it passes through the middle point of $g(x_1) = x_1^d$ and $g(x_2) = x_2^d$, which is different from $(x_1 + x_2)/2$.

The dual problem with the implicit bias term is as follows. Maximize

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 - \frac{1}{2}\,(\alpha_1^2\,H'_{11} + \alpha_2^2\,H'_{22} - 2\,\alpha_1\,\alpha_2\,H'_{12}) \tag{2.108}$$

subject to

$$\alpha_1 \geq 0,\, \alpha_2 \geq 0. \tag{2.109}$$

Partially differentiating (2.108) with respect to α_1 and α_2 and equating them to 0, we obtain the solution:

$$\alpha_1 = \frac{H'_{12} + H'_{22}}{H'_{11}H'_{22} - (H'_{12})^2}, \tag{2.110}$$

$$\alpha_2 = \frac{H'_{11} + H'_{12}}{H'_{11}H'_{22} - (H'_{12})^2}. \tag{2.111}$$

According to the assumption, α_1 and α_2 are nonnegative.

Then the decision function is given by

$$\begin{aligned} D(\mathbf{x}) &= \frac{1}{H'_{11}H'_{22} - (H'_{12})^2} \\ &\quad \times((H'_{12} + H'_{22})\,H'(\mathbf{x}_1, \mathbf{x}) - (H'_{11} + H'_{12})\,H'(\mathbf{x}_2, \mathbf{x})) \\ &= \frac{1}{(H_{11} + 1)\,(H_{22} + 1) - (H_{12} + 1)^2} \\ &\quad \times((H_{12} + H_{22} + 2)\,H(\mathbf{x}_1, \mathbf{x}) - (H_{11} + H_{12} + 2)\,H(\mathbf{x}_2, \mathbf{x}) \\ &\quad + H_{22} - H_{11}). \end{aligned} \tag{2.112}$$

Thus, $D(\mathbf{x}_1) = 1$ and $D(\mathbf{x}_2) = -1$, which satisfy the KKT conditions. For $H'(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}' + 1$, (2.112) becomes

$$\begin{aligned} D(\mathbf{x}) &= \frac{1}{(\mathbf{x}_1^T\mathbf{x}_1 + 1)\,(\mathbf{x}_2^T\mathbf{x}_2 + 1) - (\mathbf{x}_1^T\mathbf{x}_2 + 1)^2} \\ &\quad \times((((\mathbf{x}_1 + \mathbf{x}_2)^T\mathbf{x}_2 + 2)\,\mathbf{x}_1^T - (\mathbf{x}_1^T(\mathbf{x}_1 + \mathbf{x}_2) + 2)\,\mathbf{x}_2^T)\,\mathbf{x} \\ &\quad + \mathbf{x}_2^T\,\mathbf{x}_2 - \mathbf{x}_1^T\,\mathbf{x}_1). \end{aligned} \tag{2.113}$$

Because $D(\mathbf{x}_1) = 1$ and $D(\mathbf{x}_2) = -1$ and (2.113) is linear, the decision boundary passes through the middle point of $\mathbf{x}_1$ and $\mathbf{x}_2$. But because in general (2.113) is not orthogonal to $(\mathbf{x}_1 - \mathbf{x}_2)$, (2.106) and (2.113) are different.

Now consider the special case where the two decision functions are equal. Let the dimension of the input variable be 1 and $H'(x, x') = (x\,x')^d + 1$. Then the decision function (2.112) becomes

$$D(\mathbf{x}) = \frac{2x^d - x_2^d - x_1^d}{x_1^d - x_2^d}, \tag{2.114}$$

which is equivalent to (2.106).

2.4 L2 Soft-Margin Support Vector Machines

Instead of the linear sum of the slack variables ξ_i in the objective function, the L2 soft-margin support vector machine (L2 SVM) uses the square sum of the slack variables. Namely, training is done by minimizing

$$Q(\mathbf{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{2}\sum_{i=1}^{M}\xi_i^2 \tag{2.115}$$

with respect to $\mathbf{w}$, b, and $\boldsymbol{\xi}$ subject to the inequality constraints:

$$y_i\left(\mathbf{w}^T\,\mathbf{g}(\mathbf{x}_i) + b\right) \geq 1 - \xi_i \quad \text{for} \quad i = 1, \ldots, M. \tag{2.116}$$

Here, $\mathbf{w}$ is the l-dimensional vector, b is the bias term, $\mathbf{g}(\mathbf{x})$ is the mapping function that maps the m-dimensional vector $\mathbf{x}$ into the l-dimensional feature space, ξ_i is the slack variable for $\mathbf{x}_i$, and C is the margin parameter.

Introducing the Lagrange multipliers α_i (≥ 0), we obtain

$$\begin{aligned} Q(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}) = {} & \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{2}\sum_{i=1}^{M}\xi_i^2 \\ & - \sum_{i=1}^{M}\alpha_i\left(y_i\left(\mathbf{w}^T\mathbf{g}(\mathbf{x}_i) + b\right) - 1 + \xi_i\right). \end{aligned} \tag{2.117}$$

Here, we do not need to introduce the Lagrange multipliers associated with $\boldsymbol{\xi}$. As is shown immediately, $C\xi_i = \alpha_i$ is satisfied for the optimal solution. Hence ξ_i is nonnegative, so long as α_i is nonnegative.

For the optimal solution the following KKT conditions are satisfied:

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{M} y_i\,\alpha_i\,\mathbf{g}(\mathbf{x}_i) = \mathbf{0}, \tag{2.118}$$

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi})}{\partial \xi_i} = C\,\xi_i - \alpha_i = 0, \tag{2.119}$$

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi})}{\partial b} = \sum_{i=1}^{M} y_i\,\alpha_i = 0, \tag{2.120}$$

$$\alpha_i\left(y_i\left(\mathbf{w}^T\,\mathbf{g}(\mathbf{x}_i) + b\right) - 1 + \xi_i\right) = 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{2.121}$$

Equation (2.121) gives the KKT complementarity conditions; from (2.118), (2.119), and (2.121), the optimal solution must satisfy either $\alpha_i = 0$ or

$$y_i\left(\sum_{j=1}^{M}\alpha_j\,y_j\left(H(\mathbf{x}_j, \mathbf{x}_i) + \frac{\delta_{ij}}{C}\right) + b\right) - 1 = 0, \tag{2.122}$$

where $H(\mathbf{x}, \mathbf{x}') = \mathbf{g}^T(\mathbf{x})\,\mathbf{g}(\mathbf{x})$ and δ_{ij} is Kronecker's delta function, in which $\delta_{ij} = 1$ for $i = j$ and 0 otherwise. Thus the value of the bias term b is calculated for $\alpha_i > 0$:

$$b = y_i - \sum_{j=1}^{M} \alpha_j\, y_j \left(H(\mathbf{x}_j, \mathbf{x}_i) + \frac{\delta_{ij}}{C} \right), \tag{2.123}$$

which is different from that of the L1 support vector machine. But the decision function is the same:

$$D(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i\, y_i\, H(\mathbf{x}_i, \mathbf{x}) + b. \tag{2.124}$$

Substituting (2.118) to (2.120) into (2.117), we obtain the dual objective function:

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} y_i\, y_j\, \alpha_i\, \alpha_j \left(H(\mathbf{x}_i, \mathbf{x}_j) + \frac{\delta_{ij}}{C} \right). \tag{2.125}$$

Thus the following dual problem is obtained. Maximize (2.125) subject to

$$\sum_{i=1}^{M} y_i\, \alpha_i = 0, \qquad \alpha_i \geq 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{2.126}$$

This is similar to a hard-margin support vector machine. The difference is the addition of δ_{ij}/C in (2.125). Therefore, for the L1 support vector machine, if we replace $H(\mathbf{x}_j, \mathbf{x}_i)$ with $H(\mathbf{x}_j, \mathbf{x}_i) + \delta_{ij}/C$ and remove the upper bound given by C for α_i, we obtain the L2 support vector machine. But we must notice that when we calculate the decision function in (2.124) we must not add δ_{ij}/C.

Because $1/C$ is added to the diagonal elements of the matrix $H = \{H(\mathbf{x}_i, \mathbf{x}_j)\}$ called the *kernel matrix*, the resulting matrix becomes positive definite. Thus the associated optimization problem is more computationally stable than that of the L1 support vector machine.

L2 soft-margin support vector machines look similar to hard-margin support vector machines. Actually, letting

$$\tilde{\mathbf{w}} = \begin{pmatrix} \mathbf{w} \\ \sqrt{C}\boldsymbol{\xi} \end{pmatrix}, \quad \tilde{b} = b, \quad \tilde{\mathbf{g}}(\mathbf{x}_i) = \begin{pmatrix} \mathbf{g}(\mathbf{x}_i) \\ \dfrac{y_i}{\sqrt{C}} \mathbf{e}_i \end{pmatrix}, \tag{2.127}$$

where $\mathbf{e}_i$ is the M-dimensional vector with the ith element being 1 and the remaining elements 0, training of the L2 support vector machine given by (2.115) and (2.116) is converted into the following problem. Minimize

$$\frac{1}{2}\tilde{\mathbf{w}}^T\tilde{\mathbf{w}} \tag{2.128}$$

subject to

$$y_i\,(\tilde{\mathbf{w}}^T\tilde{\mathbf{g}}(\mathbf{x}_i)+\tilde{b}) \geq 1 \quad \text{for} \quad i=1,\ldots,M. \tag{2.129}$$

Therefore, the L2 support vector machine is equivalent to the hard-margin support vector machine with the augmented feature space. Because the L2 support vector machine always has a solution because of the slack variables, the equivalent hard-margin support vector machine also has a solution. But this only means that the solution is nonoverlapping in the augmented feature space. Therefore, there may be cases where the solution is overlapped in the original feature space, and thus the recognition rate of the training data for the L2 support vector machine is not 100 percent.

2.5 Advantages and Disadvantages

Based on the support vector machines explained so far, we discuss their advantages over multilayer neural networks, which are representative nonparametric classifiers. Then we discuss disadvantages of support vector machines and some ways to solve these problems.

2.5.1 Advantages

The advantages of support vector machines over multilayer neural network classifiers are as follows.

1. **Maximization of generalization ability.** In training a multilayer neural network classifier, the sum-of-squares error between outputs and desired training outputs is minimized. Thus, the class boundaries change as the initial weights change. So does the generalization ability. Thus, especially when training data are scarce and linearly separable, the generalization ability deteriorates considerably. But because a support vector machine is trained to maximize the margin, the generalization ability does not deteriorate very much, even under such a condition [3].
2. **No local minima.** A multilayer neural network classifier is known to have numerous local minima, and there have been extensive discussions on how to avoid a local minimum in training. But because a support vector machine is formulated as a quadratic programming problem, there is a global optimum solution.
3. **Robustness to outliers.** Multilayer neural network classifiers are vulnerable to outliers because they use the sum-of-squares errors. Thus to prevent the effect of outliers, outliers need to be eliminated before training, or some mechanism for suppressing outliers needs to be incorporated

in training. In support vector machines the margin parameter C controls the misclassification error. If a large value is set to C, misclassification is suppressed, and if a small value is set, training data that are away from the gathered data are allowed to be misclassified. Thus by properly setting a value to C, we can suppress outliers.

2.5.2 Disadvantages

The disadvantages of support vector machines explained so far are as follows.

1. **Extension to multiclass problems.** Unlike multilayer neural network classifiers, support vector machines use direct decision functions. Thus an extension to multiclass problems is not straightforward, and there are several formulations. One of the purposes of this book is to clarify relations between these formulations (see Chapter 3).
2. **Long training time.** Because training of a support vector machine is done by solving the associated dual problem, the number of variables is equal to the number of training data. Thus for a large number of training data, solving the dual problem becomes difficult from both the memory size and the training time. (See Chapter 5 for training speedup.)
3. **Selection of parameters.** In training a support vector machine, we need to select an appropriate kernel and its parameters, and then we need to set the value to the margin parameter C. To select the optimal parameters to a given problem is called *model selection*. This is the same situation as that of neural network classifiers. Namely, we need to set the number of hidden units, initial values of weights, and so on. In support vector machines, model selection is done by estimating the generalization ability through repeatedly training support vector machines. But because this is time-consuming, several indices for estimating the generalization ability have been proposed (see Section 2.8).

2.6 Characteristics of Solutions

Here we analyze characteristics of solutions for the L1 and L2 soft-margin support vector machines. For the L1 soft-margin support vector machine we find $\alpha_i \; (i = 1, \ldots, M)$ that maximize

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) \tag{2.130}$$

subject to the constraints

$$\sum_{i=1}^{M} y_i \, \alpha_i = 0, \qquad C \geq \alpha_i \geq 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{2.131}$$

For the L2 soft-margin support vector machine we find $\alpha_i \ (i = 1, \ldots, M)$ that maximize

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \, \alpha_j \, y_i \, y_j \, \left(H(\mathbf{x}_i, \mathbf{x}_j) + \frac{\delta_{ij}}{C} \right) \tag{2.132}$$

subject to the constraints

$$\sum_{i=1}^{M} y_i \, \alpha_i = 0, \qquad \alpha_i \geq 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{2.133}$$

2.6.1 Hessian Matrix

Rewriting (2.130) for the L1 support vector machine using the mapping function $\mathbf{g}(\mathbf{x})$, we have

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \left(\sum_{i=1}^{M} \alpha_i \, y_i \, \mathbf{g}(\mathbf{x}_i) \right)^T \sum_{i=1}^{M} \alpha_i \, y_i \, \mathbf{g}(\mathbf{x}_i). \tag{2.134}$$

Solving (2.131) for α_s $(s \in \{1, \ldots, M\})$,

$$\alpha_s = -y_s \sum_{\substack{i \neq s, \\ i=1}}^{M} y_i \, \alpha_i. \tag{2.135}$$

Substituting (2.135) into (2.134), we obtain

$$\begin{aligned} Q(\boldsymbol{\alpha}') = & \sum_{\substack{i \neq s, \\ i=1}}^{M} (1 - y_s \, y_i) \, \alpha_i \\ & -\frac{1}{2} \left(\sum_{\substack{i \neq s, \\ i=1}}^{M} \alpha_i \, y_i \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)) \right)^T \sum_{\substack{i \neq s, \\ i=1}}^{M} \alpha_i \, y_i \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)), \end{aligned} \tag{2.136}$$

where $\boldsymbol{\alpha}'$ is obtained by deleting α_s from $\boldsymbol{\alpha}$. Thus the Hessian matrix of $-Q(\boldsymbol{\alpha}')$, H_{L1},[7] which is an $(M-1) \times (M-1)$ matrix, is given by

$$\begin{aligned} H_{L1} &= -\frac{\partial^2 Q(\boldsymbol{\alpha}')}{\partial \boldsymbol{\alpha}'^2} \\ &= \left(\cdots y_i \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)) \cdots \right)^T \left(\cdots y_j \, (\mathbf{g}(\mathbf{x}_j) - \mathbf{g}(\mathbf{x}_s)) \cdots \right). \end{aligned} \tag{2.137}$$

[7]Conditionally positive semidefiniteness, which is positive semidefiniteness under an equality constraint discussed in Appendix D.1, is equivalent to positive semidefiniteness of H_{L1}.

Because H_{L1} is expressed by the product of the transpose of a matrix and the matrix, H is positive semidefinite. Let $N_{\mathbf{g}}$ be the maximum number of independent vectors among $\{\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s) | \, i \in \{i, \ldots, M, i \neq s\}\}$. Then the rank of H_{L1} is $N_{\mathbf{g}}$ [3, pp. 311–12]. Because $N_{\mathbf{g}}$ does not exceed the dimension of the feature space l,

$$N_{\mathbf{g}} \leq l \tag{2.138}$$

is satisfied. Therefore, if $M > (l+1)$, H_{L1} is positive semidefinite. For the linear kernel, $l = m$, where m is the number of input variables, for the polynomial kernel with degree d, $l = {}_{m+d}C_d$ [109, pp. 38–41], and for the RBF kernel, $l = \infty$.

The Hessian matrix H_{L2} in which one variable is eliminated, for the L2 support vector machine, is expressed by

$$H_{L2} = H_{L1} + \left\{ \frac{y_i \, y_j + \delta_{ij}}{C} \right\}. \tag{2.139}$$

The matrix H_{L1} is positive semidefinite, and the matrix $\{\delta_{ij}/C\}$ is positive definite. Because

$$\{y_i \, y_j\} = (y_1 \cdots y_M)^T \, (y_1 \cdots y_M) \, , \tag{2.140}$$

the matrix $\{y_i y_j / C\}$ is positive semidefinite. Here, the sum of positive definite and positive semidefinite matrices is positive definite. Therefore, unlike H_{L1}, H_{L2} is positive definite irrespective of the dimension of the feature space.

2.6.2 Dependence of Solutions on *C*

According to the analysis of Pontil and Verri [193], we discuss the dependence of the solution of the L1 support vector machine on the margin parameter C.

Let the sets of support vector indices be

$$U = \{i \,|\, 0 < \alpha_i < C\}, \tag{2.141}$$

$$B = \{i \,|\, \alpha_i = C\}, \tag{2.142}$$

$$S = U \cup B. \tag{2.143}$$

From the KKT complementarity conditions given by (2.63) to (2.65), for $i \in U$,

$$y_i \left(\sum_{j \in S} \alpha_j \, y_j \, H(\mathbf{x}_j, \mathbf{x}_i) + b \right) = 1. \tag{2.144}$$

Thus, from (2.144) for a fixed $s \in U$, b is given by

$$b = y_s - \sum_{j \in S} \alpha_j \, y_j \, H(\mathbf{x}_j, \mathbf{x}_s). \tag{2.145}$$

Solving the equality constraint (2.131) for $\alpha_s \, (s \in U)$, we obtain

$$\alpha_s = -\sum_{\substack{j \neq s,\\ j \in S}} y_j\, y_s\, \alpha_j. \tag{2.146}$$

Substituting (2.145) and (2.146) into (2.144), we obtain

$$\sum_{\substack{j \neq s,\\ j \in S}} \alpha_j H_{ji} = 1 - y_i\, y_s, \tag{2.147}$$

where

$$H_{ji} = y_j\, y_i\, \left(H(\mathbf{x}_j, \mathbf{x}_i) - H(\mathbf{x}_s, \mathbf{x}_i) - H(\mathbf{x}_j, \mathbf{x}_s) + H(\mathbf{x}_s, \mathbf{x}_s) \right). \tag{2.148}$$

Separating the unbounded and bounded α_i in (2.147), we obtain

$$\sum_{\substack{j \neq s,\\ j \in U}} \alpha_j\, H_{ji} + C \sum_{j \in B} H_{ji} = 1 - y_i\, y_s. \tag{2.149}$$

In a matrix form, (2.149) becomes

$$H_{U'}\, \boldsymbol{\alpha}_{U'} + C\, H_{U'B} \mathbf{1}_{U'} = \mathbf{1}_{U'} - \mathbf{y}_{U'}, \tag{2.150}$$

where $U' = U - \{s\}$, $H_{U'} = \{H_{ij} \,|\, i, j \in U'\}$, $\boldsymbol{\alpha}_{U'} = (\cdots\ \alpha_i\ \cdots)^T\ (i \in U')$, $H_{U'B} = \{H_{ij} \,|\, i \in U', j \in B\}$, $\mathbf{y}_{U'} = (\cdots\ y_s\, y_i\ \cdots)^T\ (i \in U')$, and $\mathbf{1}_U$ is a $|U'|$-dimensional vector with all elements equal to 1.

From (2.59), H_{ij} is expressed by

$$H_{ij} = y_i\, y_j\, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s))^T (\mathbf{g}(\mathbf{x}_j) - \mathbf{g}(\mathbf{x}_s)). \tag{2.151}$$

Thus $H_{U'}$ is a symmetric positive semidefinite matrix. As will be stated in Theorem 2.14, $H_{U'}$ is not always positive definite. But here we assume that it *is* positive definite. Then, from (2.150) and (2.146), we obtain the following theorem.

Theorem 2.4. *If the Hessian matrix $H_{U'}$ is positive definite, the unbounded α_i are given by*

$$\boldsymbol{\alpha}_{U'} = H_{U'}^{-1}\, (\mathbf{1}_{U'} - \mathbf{y}_{U'}) - C\, H_{U'}^{-1}\, H_{U'B}\, \mathbf{1}_B, \tag{2.152}$$

$$\alpha_s = -\mathbf{y}_{U'}^T\, \boldsymbol{\alpha}_{U'} - C\, \mathbf{y}_B^T\, \mathbf{1}_B, \tag{2.153}$$

where $\mathbf{y}_B = (\cdots\ y_s\, y_i\ \cdots)^T\ (i \in B)$ and $\mathbf{1}_B$ is a $|B|$-dimensional vector with all elements equal to 1.

Therefore, if $B = \phi$, namely, $0 < \alpha_i < C$ for all support vectors, $\boldsymbol{\alpha}_{U'} = H_{U'}^{-1} (\mathbf{1}_{U'} - \mathbf{y}_{U'})$.

Theorem 2.5. *The coefficient vector of the separating hyperplane, $\mathbf{w}$, in the feature space is given by*

$$\mathbf{w} = \mathbf{w}_1 + C\, \mathbf{w}_2, \tag{2.154}$$

where $\mathbf{w}_1$ and $\mathbf{w}_2$ are given by (2.158) and (2.159), and

$$\mathbf{w}_1^T \mathbf{w}_2 = 0. \tag{2.155}$$

Proof. Because

$$
\begin{aligned}
\mathbf{w} &= \sum_{i \in S} \alpha_i \, y_i \, \mathbf{g}(\mathbf{x}_i) \\
&= \sum_{i \in U'} \alpha_i \, y_i \, \mathbf{g}(\mathbf{x}_i) + \alpha_s \, y_s \, \mathbf{g}(\mathbf{x}_s) + C \sum_{i \in B} y_i \, \mathbf{g}(\mathbf{x}_i) \\
&= \sum_{i \in U'} \alpha_i \, y_i \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)) + C \sum_{i \in B} y_i \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)) \quad (\text{from (2.146)}) \\
&= \sum_{i \in U'} r_i \, y_i \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)) + C \sum_{i \in U'} t_i \, y_i \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)) \\
&\quad + C \sum_{i \in B} y_i \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)),
\end{aligned}
$$

where from (2.152)

$$\mathbf{r} = (\cdots r_i \cdots)^T = H_{U'}^{-1} (\mathbf{1}_{U'} - \mathbf{y}_{U'}), \tag{2.156}$$

$$\mathbf{t} = (\cdots t_i \cdots)^T = -H_{U'}^{-1} H_{U'B} \, \mathbf{1}_B, \tag{2.157}$$

we obtain

$$\mathbf{w}_1 = \sum_{i \in U'} r_i \, y_i \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)), \tag{2.158}$$

$$\mathbf{w}_2 = \sum_{i \in U'} t_i \, y_i \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)) + \sum_{i \in B} y_i \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)). \tag{2.159}$$

Then

$$
\begin{aligned}
\mathbf{w}_1^T \mathbf{w}_2 &= \sum_{i,\, j \in U'} r_i \, H_{ij} \, t_j + \sum_{i \in U',\, j \in B} r_i \, H_{ij} \\
&= \mathbf{r} \, H_{U'} \, \mathbf{t} + \mathbf{r} \, H_{U'B} \, \mathbf{1}_B \\
&= 0 \quad (\text{from (2.157)}). \blacksquare
\end{aligned}
\tag{2.160}
$$

Now consider changing the margin parameter C. Let $[C_k, C_{k+1}]$ be the interval of C, in which the set of support vectors does not change. Here, we consider that if unbounded support vectors change to bounded support vectors, the set is changed. We add suffix k to the sets of support vectors for the interval $[C_k, C_{k+1}]$, namely, S_k, U_k, and B_k. Then, from (2.152) and (2.154), for C in $[C_k, C_{k+1})$,

$$\alpha_i = a_i + C \, b_i \quad \text{for} \quad i \in U_k, \tag{2.161}$$

$$\alpha_i = C \quad \text{for} \quad i \in B_k, \tag{2.162}$$

where a_i and b_i are constant. Notice that so long as $U_k = U_{k'}$ and $B_k = B_{k'}$, a_i and b_i are the same for the intervals $[C_k, C_{k+1})$ and $[C_{k'}, C_{k'+1})$.

Because the number of training data is finite, the combinations of sets of support vectors are finite. But if some sets of support vectors appear infinitely as we increase C to infinity, the intervals $[C_k, C_{k+1})$ may be infinite. The following theorem shows that this does not happen.

Theorem 2.6. *For C in $[0, \infty)$, there are finite points $C_i \, (i = 1, \ldots, \max)$ where the set of support vectors changes. And for any C in $[C_i, C_{i+1})$ or $[C_{\max}, \infty)$ the set of support vectors does not change.*

Proof. First we show that as C approaches infinity, the weight vector $\mathbf{w}_2$ in (2.154) approaches $\mathbf{0}$. From (2.154), the quadratic term in the dual objective function of an L1 support vector machine is expressed by

$$\begin{aligned}\frac{1}{2}\mathbf{w}^T\mathbf{w} &= \frac{1}{2}(\mathbf{w}_1 + C\mathbf{w}_2)^T(\mathbf{w}_1 + C\mathbf{w}_2) \\ &= \frac{1}{2}\mathbf{w}_1^T\mathbf{w}_1 + C\mathbf{w}_1^T\mathbf{w}_2 + \frac{1}{2}C^2\mathbf{w}_2^T\mathbf{w}_2, \end{aligned} \tag{2.163}$$

where $\mathbf{w}$ is the weight vector for the optimal solution with C.

Therefore, if $\mathbf{w}_2$ is not a zero vector, the value of (2.163) goes to infinity as C approaches infinity quadratically. Thus, the dual objective function becomes negative for a large value of C. But because $\alpha_i = 0 \, (i = 1, \ldots, M)$ is a feasible solution with the objective function being 0, $\mathbf{w}$ cannot be optimal. Therefore, there exists C_0 where for C larger than C_0, $\mathbf{w} = \mathbf{w}_1$.

Assume that for C larger than C_0, two sets of support vectors S_k and S_{k+1} alternately appear infinitely. For $C > C_0$, $\mathbf{w} = \mathbf{w}_1$. Thus, from (2.161) and (2.162), the dual objective function $Q(C)$ for C in $[C_k, C_{k+1})$ is given by

$$Q(C) = \sum_{i \in U_k} a_i + C\left(\sum_{i \in U_k} b_i + |B_k|\right) - \frac{1}{2}\mathbf{w}_1^T\mathbf{w}_1, \tag{2.164}$$

where $\mathbf{w}_1$is a constant vector in $[C_k, C_{k+1})$.

For the infinite intervals in $[C_0, \infty)$, where the sets of support vectors are the same as S_k, the objective function is given by (2.164). According to the assumption, the sets of support vectors S_k and S_{k+1} alternate, namely,

$$S_k = S_{k+2} = S_{k+4} = \cdots, \tag{2.165}$$

$$S_{k+1} = S_{k+3} = S_{k+5} = \cdots. \tag{2.166}$$

Thus, the objective functions (2.164) for S_k and S_{k+1} cross at $C = C_{k+1}$, but because they are monotonic functions, they do not cross at C larger than C_{k+1} (see Fig. 2.8). Because the optimization problem is continuous, namely, the objective function is continuous for the change of C, discontinuity of the objective function does not occur. Thus, the assumption that the two sets of support vectors alternate infinitely does not happen. Therefore the number of intervals $[C_k, C_{k+1}]$ is finite and the interval ends with $[C_{\max}, \infty]$, where "max" is the maximum number of intervals. Thus, the theorem holds. ■

For the L2 support vector machine, the following theorem, similar to Theorem 2.6, holds.

Theorem 2.7. *For an L2 support vector machine, there are finite points $C'_i \, (i = 1, \ldots, \max')$ for $C \in [0, \infty)$ where the set of support vectors changes.*

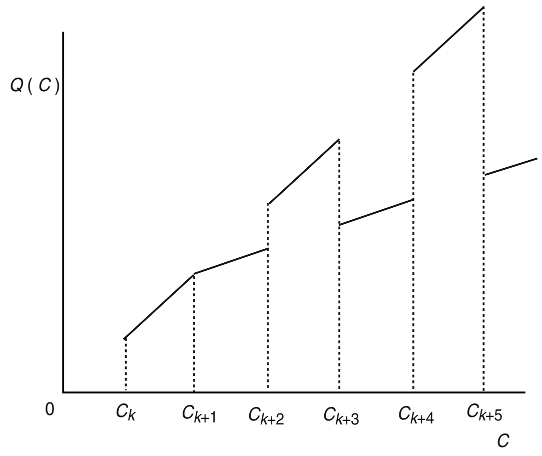


Fig. 2.8. Counterexample of infinite intervals of C

And for any C in (C'_i, C'_{i+1}) or $(C'_{\max'}, \infty)$, the set of support vectors is the same.

Proof. Similar to (2.152) and (2.153), the support vectors for an L2 support vector machine are given by

$$\boldsymbol{\alpha}_{S'} = H_{S'}^{-1}(\mathbf{1}_{S'} - \mathbf{y}_{S'}), \tag{2.167}$$

$$\alpha_s = -\mathbf{y}_{S'}^T \boldsymbol{\alpha}_{S'}, \tag{2.168}$$

where $S' = S - \{s\}$, $H_{S'} = \{H'_{ij} \mid i, j \in S'\}$, $H'_{ij} = H_{ij} + \delta_{ij}/C + y_i\, y_j/C$, H_{ij} is given by (2.148), $\boldsymbol{\alpha}_{S'} = (\cdots \alpha_i \cdots)^T$ $(i \in S')$, $\mathbf{y}_{S'} = (\cdots y_s\, y_i \cdots)^T$ $(i \in S')$, and $\mathbf{1}_{S'}$ is an $|S'|$-dimensional vector with all elements equal to 1.

Similar to (2.136), the objective function for the L2 support vector machine, in a matrix form, is given by

$$Q(C) = (\mathbf{1}_{S'} - \mathbf{y}_{S'})^T \boldsymbol{\alpha}_{S'}^T - \frac{1}{2}\, \boldsymbol{\alpha}_{S'}^T\, H_{S'}\, \boldsymbol{\alpha}_{S'}. \tag{2.169}$$

Substituting (2.167) into (2.169) gives

$$Q(C) = \frac{1}{2}\, (\mathbf{1}_{S'} - \mathbf{y}_{S'})^T H_{S'}^{-1} (\mathbf{1}_{S'} - \mathbf{y}_{S'}). \tag{2.170}$$

Because the matrix $H_{S'}$ is positive definite, $H_{S'}^{-1}$ is also positive definite. By increasing the value of C, the eigenvalues of $H_{S'}$ decrease, thus those of $H_{S'}^{-1}$ increase. Therefore, $Q(C)$ monotonically increases and saturates as C approaches infinity.

Suppose that there are infinite intervals of $[C'_k, C'_{k+1}]$ and that the sets of support vectors S'_k and S'_{k+1} alternate infinitely. Then the objective functions

(2.170) for S'_k and S'_{k+1} cross at $C = C'_{k+1}$ and they do not cross at C larger than C'_{k+1}. But because the objective function is continuous for the change of C, discontinuity of the objective function does not occur. Thus, the assumption that the two sets of support vectors alternate infinitely does not happen. Therefore the number of intervals $[C'_k, C'_{k+1}]$ is finite and the interval ends with $[C'_{\max'}, \infty]$, where "max′" is the maximum number of intervals. ■

2.6.3 Equivalence of L1 and L2 Support Vector Machines

In this section, we clarify the condition in which L1 and L2 support vector machines are equivalent.

Setting $C = \infty$ in L1 and L2 support vector machines, we obtain the hard-margin support vector machines as follows. Namely, we find α_i $(i = 1, \dots, M)$ that maximize

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) \tag{2.171}$$

subject to the constraints

$$\sum_{i=1}^{M} y_i \, \alpha_i = 0, \qquad \alpha_i \geq 0. \tag{2.172}$$

In other words, as C approaches infinity, the solution of the L1 support vector machine approaches that of the associated L2 support vector machine. In the following, we discuss this in more detail.

For the L2 support vector machine, the weight vector is not expressed by (2.154), but as C approaches infinity, from our previous discussions, the weight vector converges to $\mathbf{w}_1$. Namely, the following theorem holds.

Theorem 2.8. *For C in $[\max(C_{\max}, C'_{\max'}), \infty]$, the sets of support vectors $S_{\max}$ and $S'_{\max'}$ are the same, and for L1 and L2 support vector machines, the weight vectors in the feature space converges to vector $\mathbf{w}_1$ as C approaches infinity.*

In the following, we discuss the equivalence for the separable and inseparable classification problems.

If the problem is separable, the L1 support vector machine has a finite optimum solution:

$$0 \leq \alpha_i < \infty \qquad \text{for} \quad i = 1, \dots, M. \tag{2.173}$$

Let

$$C_{\max} = \max_{i=1,\dots,M} \alpha_i. \tag{2.174}$$

Then, for the L1 support vector machine, the solution is the same for any $C \in [C_{\max}, \infty)$. Namely, for any $C \in [C_{\max}, \infty)$, the support vectors are unbounded and do not change. Thus the resulting optimal hyperplane does not change.

Now assume that the given problem is inseparable in the feature space. Then the solution for (2.171) and (2.172) diverges without bounds. Let $C_{\max}$ and $C'_{\max'}$ be as defined before. Then for any $C \in [\max(C_{\max}, C'_{\max}), \infty)$, the sets of support vectors for the L1 and L2 support vector machines are the same. Then although α_i goes to infinity as C approaches infinity, from Theorem 2.8, the weight vector converges to a constant vector.

Example 2.9. Consider an inseparable case with a linear kernel, shown in Fig. 2.9. The dual problem of the L1 support vector machine is to maximize

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2}\,(-0.2\,\alpha_2 + 0.4\,\alpha_3 - \alpha_4)^2 \qquad (2.175)$$

subject to

$$\alpha_1 + \alpha_3 = \alpha_2 + \alpha_4, \quad C \geq \alpha_i \geq 0, \;\; i = 1, \ldots, 4. \qquad (2.176)$$

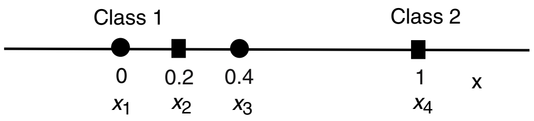


Fig. 2.9. Inseparable one-dimensional case

From Fig. 2.9, we can assume that x_2 and x_3 are bounded support vectors. Namely, $\alpha_2 = \alpha_3 = C$. Thus from (2.176), $\alpha_1 = \alpha_4$. Therefore, (2.175) reduces to

$$Q(\boldsymbol{\alpha}) = 2\,\alpha_1 + 2\,C - \frac{1}{2}\,(\alpha_1 - 0.2\,C)^2. \qquad (2.177)$$

Equation (2.177) reaches the maximum value of $2.4\,C + 2$ when $\alpha_1 = 0.2\,C + 2$. Thus, the optimal solution is given by

$$\alpha_1 = \alpha_4 = 0.2\,C + 2, \quad \alpha_2 = \alpha_3 = C. \qquad (2.178)$$

Therefore, the solution diverges indefinitely as C approaches infinity. Substituting (2.178) into the quadratic term of (2.175), we obtain 4. Namely, the quadratic term is constant for $C \geq 2.5$. If this does not hold, an unbounded solution is not obtained.

Using (2.178), the decision function is given by

$$D(x) = -2\,x + 1. \qquad (2.179)$$

Thus the class boundary is $x = 0.5$ for $C \geq 2.5$.

Example 2.10. To evaluate the equivalence of L1 and L2 support vector machines, we used the iris data listed in Table 1.1. We selected the test data for Classes 2 and 3, which are not linearly separable for the linear kernels. For $d = 3$, the data are linearly separable, and for the L1 support vector machine, $C_{\max} = 4290.39$. Thus, for $C \geq C_{\max}$, the L1 support vector machine gives the same solution. Table 2.2 lists the support vectors and their α_i.

Table 2.2. Support vectors for the iris data with $d = 3$

i	Class	α_i
7	2	4290.39
21	2	4.28
24	2	2011.89
26	3	763.04
30	3	16.74
32	3	4123.90
37	3	240.29
38	3	1162.58

Table 2.3 shows the number of support vectors and α_7 of the L2 support vector machine for the change of C. Except for $C = 10^3$ with the number of support vectors of 11, the support vectors are the same as those of the L1 support vector machine. For $C = 10^{10}$, the value of α_7 coincides with that for the L1 support vector machine for the first six digits.

Next, we examined the nonseparable case, i.e., $d = 1$. Table 2.4 shows the numbers of support vectors and coefficients of the optimal hyperplanes for the L1 and L2 support vector machines against C. For the L1 support vector machine, the numerals in parentheses in the "SVs" column show the bounded support vectors. From the table, there are five unbounded L1 support vectors, which corresponds to the maximum number of support vectors, i.e., $m+1 = 5$. For the L1 support vector machine, the coefficient w_1 reached almost constant at $C = 10^4$.

In theory, for $C = \infty$ the optimal hyperplane of the L2 support vector machine converges to that of the L1 support vector machine. But from Table 2.4, the difference between the two is still large for $C = 10^{10}$. Although the optimal hyperplane of the L2 support vector machine changes as C is increased, the change is small and convergence to that of the L1 support vector machine is very slow compared to the case where the problem is separable in the feature space.

Table 2.3. L2 support vectors for the change of C ($d = 3$)

C	SVs	α_7
10^3	11	1015.69
10^4	8	3215.73
10^5	8	4151.95
10^6	8	4276.14
10^7	8	4288.96
10^8	8	4290.24
10^9	8	4290.38
10^{10}	8	4290.39

Table 2.4. Coefficients of the optimal hyperplane for L1 and L2 support vector machines ($d = 1$)

C	L1 SVM		L2 SVM	
	SVs	w_1	SVs	w_1
10^2	11 (6)	1.07	15	1.58
10^3	8 (3)	4.18	10	2.75
10^4	7 (2)	11.49	10	3.45
10^5	7 (2)	11.50	10	3.53
10^6	7 (2)	11.50	10	3.54
10^{10}	7 (2)	11.49	10	3.54

2.6.4 Nonunique Solutions

Because support vector machines are formulated as quadratic optimization problems, there is a global maximum (minimum) of the objective function. This is one of the advantages over multilayer neural networks with numerous local minima. Although the objective function has the global maximum (minimum), there may be cases where solutions are not unique [46]. This is not a serious problem because the values of the objective function are the same. In the following, we discuss nonuniqueness of the solution.

If a convex function gives a minimum or maximum at a point, not in an interval, the function is called *strictly convex*. In general, if the objective function of a quadratic programming problem constrained in a convex set is strictly convex or the associated Hessian matrix is positive (negative) definite, the solution is unique. And if the objective function is convex, there may be cases where the solution is nonunique (see Fig. 2.10). Convexity of objective functions for different support vector machine architectures is summarized in Table 2.5. The symbols in parentheses show the variables.

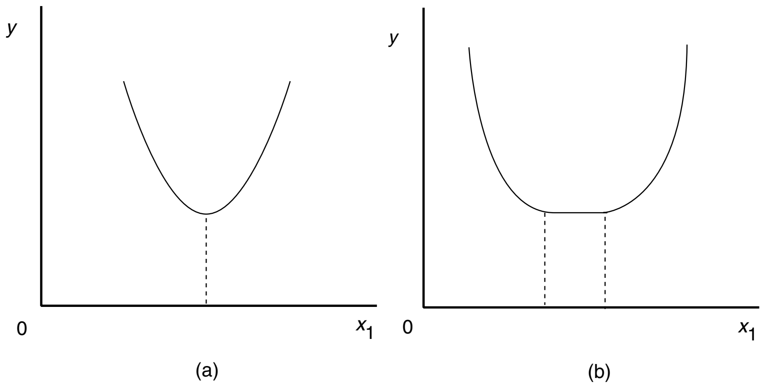


Fig. 2.10. Convex functions: (**a**) Strictly convex function. (**b**) Convex function. From [4, p. 92, ©IEEE 2002]

Table 2.5. Convexity of objective functions. From [4, p. 92, ©IEEE 2002]

	Hard margin	L1 soft margin	L2 soft margin
Primal	Strictly convex $(\mathbf{w}, b)$	Convex $(\mathbf{w}, b, \boldsymbol{\xi})$	Strictly convex $(\mathbf{w}, b, \boldsymbol{\xi})$
Dual	Convex $(\boldsymbol{\alpha})$	Convex $(\boldsymbol{\alpha})$	Strictly convex $(\boldsymbol{\alpha})$

We must notice that because b is not included in the dual problem, even if the solution of the dual problem is unique, the solution of the primal problem may not be unique.

Assume that the hard-margin support vector machine has a solution, i.e., the given problem is separable in the feature space. Then because the objective function of the primal problem is $\|\mathbf{w}\|^2/2$, which is strictly convex, the primal problem has a unique solution for $\mathbf{w}$ and b. But because the Hessian matrix of the dual problem is positive semidefinite, the solution for α_i may be nonunique. As discussed previously, the hard-margin support vector machine for a separable problem is equivalent to the L1 soft-margin support vector machine with an unbounded solution.

The objective function of the primal problem for the L2 soft-margin support vector machine is strictly convex. Thus, $\mathbf{w}$ and b are uniquely determined if we solve the primal problem. In addition, because the Hessian matrix of the dual objective function is positive definite, α_i are uniquely determined. And because of the uniqueness of the primal problem, b is determined uniquely using the KKT conditions.

Because the L1 soft-margin support vector machine includes the linear sum of ξ_i, the primal objective function is convex. Likewise, the Hessian matrix of the dual objective function is positive semidefinite. Thus the primal and dual solutions may be nonunique [4].

Before we discuss nonuniqueness of the solution, we clarify some properties of support vectors.

The KKT complementarity conditions for the L1 support vector machine are given by (2.63) to (2.65). Thus, in some situation, $\alpha_i = 0$ and $y_i\,(\mathbf{w}^T\mathbf{g}(\mathbf{x}_i) + b) = 1$ are satisfied simultaneously. In this case, $\mathbf{x}_i$ is not a support vector.

Definition 2.11. For the L1 support vector machine, we call the data that satisfy $y_i\,(\mathbf{w}^T\mathbf{g}(\mathbf{x}_i)+b) = 1$ and that are not support vectors *boundary vectors.*

Example 2.12. Consider the two-dimensional case shown in Fig. 2.11, in which $\mathbf{x}_1$ belongs to Class 1, $\mathbf{x}_2$ and $\mathbf{x}_3$ belong to Class 2, and $\mathbf{x}_1 - \mathbf{x}_2$ and $\mathbf{x}_3 - \mathbf{x}_2$ are orthogonal. The dual problem with linear kernels is given as follows. Maximize

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 + \alpha_3 - \frac{1}{2}\,(\alpha_1\,\mathbf{x}_1 - \alpha_2\,\mathbf{x}_2 - \alpha_3\,\mathbf{x}_3)^T(\alpha_1\,\mathbf{x}_1 - \alpha_2\,\mathbf{x}_2 - \alpha_3\,\mathbf{x}_3) \quad (2.180)$$

subject to

$$\alpha_1 - \alpha_2 - \alpha_3 = 0, \quad C \geq \alpha_i \geq 0, \quad i = 1, 2, 3. \quad (2.181)$$

Substituting $\alpha_3 = \alpha_1 - \alpha_2$ and $\alpha_2 = a\,\alpha_1$ $(a \geq 0)$ into (2.180), we obtain

$$Q(\boldsymbol{\alpha}) = 2\,\alpha_1 - \frac{1}{2}\,\alpha_1^2 \times (\mathbf{x}_1 - \mathbf{x}_3 - a\,(\mathbf{x}_2 - \mathbf{x}_3))^T(\mathbf{x}_1 - \mathbf{x}_3 - a\,(\mathbf{x}_2 - \mathbf{x}_3)). \quad (2.182)$$

Defining

$$d^2(a) = (\mathbf{x}_1 - \mathbf{x}_3 - a(\mathbf{x}_2 - \mathbf{x}_3))^T(\mathbf{x}_1 - \mathbf{x}_3 - a(\mathbf{x}_2 - \mathbf{x}_3)), \quad (2.183)$$

(2.182) becomes

$$Q(\boldsymbol{\alpha}) = 2\,\alpha_1 - \frac{1}{2}\alpha_1^2 d^2(a). \quad (2.184)$$

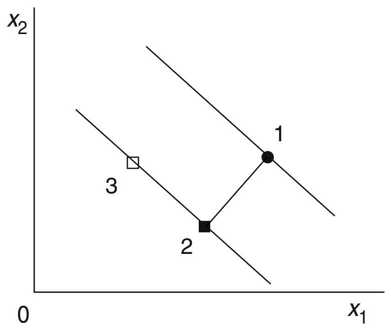


Fig. 2.11. An example of a nonsupport vector. From [4, p. 93, ©IEEE 2002]

When

$$C \geq \frac{2}{d^2(a)}, \tag{2.185}$$

$Q(\boldsymbol{\alpha})$ is maximized at $\alpha_1 = 2/d^2(a)$ and takes the maximum

$$Q\left(\frac{2}{d^2(a)}\right) = \frac{2}{d^2(a)}. \tag{2.186}$$

Because $\mathbf{x}_1 - \mathbf{x}_2$ and $\mathbf{x}_3 - \mathbf{x}_2$ are orthogonal, $d(a)$ is minimized at $a = 1$. Thus $Q(2/d^2(a))$ is maximized at $a = 1$. Namely, $\alpha_1 = \alpha_2 = 2/d^2(a)$ and $\alpha_3 = 0$. Because $y_3\,(\mathbf{w}^T\mathbf{x}_3 + b) = 1$, $\mathbf{x}_3$ is a boundary vector.

Definition 2.13. For the L1 support vector machine, a set of support vectors is *irreducible* if deletion of all the boundary vectors and any support vector results in the change of the optimal hyperplane. It is *reducible* if the optimal hyperplane does not change for deletion of all the boundary vectors and some support vectors.

Deletion of nonsupport vectors from the training data set does not change the solution. In [74], after training, an irreducible set is obtained by deleting linearly dependent support vectors in the feature space.

In the following theorem, the Hessian matrix associated with a set of support vectors means that the Hessian matrix is calculated for the support vectors, not for the entire training data.

Theorem 2.14. *For the L1 support vector machine, let all the support vectors be unbounded. Then the Hessian matrix associated with an irreducible set of support vectors is positive definite and the Hessian matrix associated with a reducible set of support vectors is positive semidefinite.*

Proof. Let the set of support vectors be irreducible. Then, because deletion of any support vector results in the change of the optimal hyperplane, any

$\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)$ cannot be expressed by the remaining $\mathbf{g}(\mathbf{x}_j) - \mathbf{g}(\mathbf{x}_s)$. Thus the associated Hessian matrix is positive definite. If the set of support vectors is reducible, deletion of some support vector, e.g., $\mathbf{x}_i$ does not result in the change of the optimal hyperplane. This means that $\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)$ is expressed by the linear sum of the remaining $\mathbf{g}(\mathbf{x}_j) - \mathbf{g}(\mathbf{x}_s)$. Thus the associated Hessian matrix is positive semidefinite. ■

Theorem 2.15. *For the L1 support vector machine, let the dimension of the feature space be finite. Then the number of unbounded support vectors in the irreducible set cannot exceed the dimension of the feature space plus 1. For the infinite feature space, the maximum number of unbounded support vectors is the number of training data. For the L2 support vector machine, the maximum number of support vectors is the number of training data.*

Proof. It is clear from Theorem 2.14 that the theorem holds for the L1 support vector machine. For the L2 support vector machine, because the Hessian matrix associated with the training data set is positive definite, the theorem holds. ■

Theorem 2.16. *For the L1 support vector machine, if there is only one irreducible set of support vectors and the support vectors are all unbounded, the solution is unique.*

Proof. Delete the nonsupport vectors from the training data. Then because the set of support vectors is irreducible, the associated Hessian matrix is positive definite. Thus, the solution is unique for the irreducible set. Because there is only one irreducible set, the solution is unique for the given problem. ■

Example 2.17. Consider the two-dimensional case shown in Fig. 2.12, in which $\mathbf{x}_1$ and $\mathbf{x}_2$ belong to Class 1 and $\mathbf{x}_3$ and $\mathbf{x}_4$ belong to Class 2. Because $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$ form a rectangle, $\{\mathbf{x}_1, \mathbf{x}_3\}$ and $\{\mathbf{x}_2, \mathbf{x}_4\}$ are irreducible sets of support vectors for the linear kernel.

The training of the L1 support vector machine is to maximize

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2}\left((\alpha_1 + \alpha_4)^2 + (\alpha_2 + \alpha_3)^2\right) \qquad (2.187)$$

subject to

$$\alpha_1 + \alpha_2 = \alpha_3 + \alpha_4, \quad C \geq \alpha_i \geq 0, \quad i = 1, \ldots, 4. \qquad (2.188)$$

For $C \geq 1$, $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (1, 0, 1, 0)$ and $(0, 1, 0, 1)$ are two solutions. Thus,

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (\beta, 1 - \beta, \beta, 1 - \beta), \qquad (2.189)$$

where $0 \leq \beta \leq 1$, is also a solution. Then $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (0.5, 0.5, 0.5, 0.5)$ is a solution.

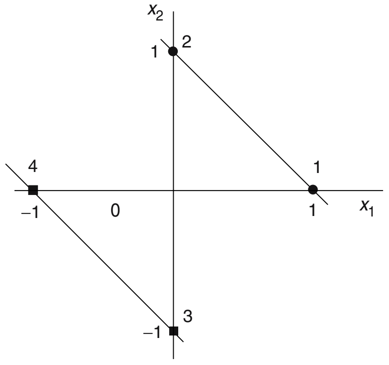


Fig. 2.12. Nonunique solutions. From [4, p. 95, ©IEEE 2002]

For the L2 support vector machine, the objective function becomes

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2}\left((\alpha_1 + \alpha_4)^2 + (\alpha_2 + \alpha_3)^2 + \frac{\alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2}{C}\right). \qquad (2.190)$$

Then for $\alpha_i = 1/(2 + 1/C)$ $(i = 1, \ldots, 4)$, (2.190) becomes

$$Q(\boldsymbol{\alpha}) = \frac{1}{1 + \dfrac{1}{2C}}. \qquad (2.191)$$

For $\alpha_1 = \alpha_3 = 1/(1 + 1/C)$ and $\alpha_2 = \alpha_4 = 0$, (2.190) becomes

$$Q(\boldsymbol{\alpha}) = \frac{1}{1 + \dfrac{1}{C}}. \qquad (2.192)$$

Thus, for $C > 0$, $Q(\boldsymbol{\alpha})$ given by (2.192) is smaller than that given by (2.191). Therefore, $\alpha_1 = \alpha_3 = 1/(1+1/C)$ and $\alpha_2 = \alpha_4 = 0$ or $\alpha_2 = \alpha_4 = 1/(1+1/C)$ and $\alpha_1 = \alpha_3 = 0$ are not optimal, but $\alpha_i = 1/(2 + 1/C)$ $(i = 1, \ldots, 4)$ are.

In general, the number of support vectors for an L2 support vector machine is equal to or greater than that for an L1 support vector machine. And for sufficiently large C, the sets of support vectors are the same as shown in Theorem 2.8. In the original formulation of support vector machines, there is no mechanism for reducing the number of support vectors. Drezet and Harrison [75] reformulated support vector machines so that the number of support vectors is reduced.

Example 2.17 shows nonuniqueness of the dual problem, but the primal problem is unique because there are unbounded support vectors. Nonunique solutions occur when there are no unbounded support vectors. Burges and

Crisp [46] derived conditions in which the dual problem is unique but the primal solution is nonunique. In the following, we discuss nonuniqueness of primal problems. First consider an example discussed in [46].

Example 2.18. Consider a one-dimensional example shown in Fig. 2.13.

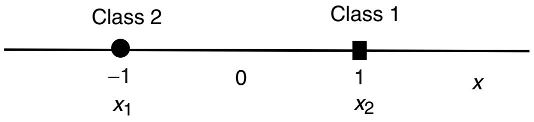


Fig. 2.13. Nonunique solutions

The dual problem for linear kernels is given as follows. Maximize

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 - \frac{1}{2}\,(\alpha_1^2 + \alpha_2^2 + 2\,\alpha_1\,\alpha_2) \tag{2.193}$$

subject to

$$\alpha_1 - \alpha_2 = 0, \tag{2.194}$$

$$C \geq \alpha_i \geq 0 \qquad \text{for} \quad i = 1, 2. \tag{2.195}$$

From (2.194), $\alpha_2 = \alpha_1$. Then substituting it into (2.193), we obtain

$$\begin{aligned} Q(\boldsymbol{\alpha}) &= 2\,\alpha_1 - 2\,\alpha_1^2 \\ &= -2\left(\alpha_1 - \frac{1}{2}\right)^2 + \frac{1}{2}, \end{aligned} \tag{2.196}$$

$$C \geq \alpha_i \geq 0 \qquad \text{for} \quad i = 1, 2. \tag{2.197}$$

For $C \geq 1/2$, (2.196) is maximized when $\alpha_1 = 1/2$. Thus from (2.48), $w = 1$, and from (2.54), $b = 0$. For $C < 1/2$, the optimal solution is given by

$$\alpha_1 = \alpha_2 = C. \tag{2.198}$$

Therefore, $w = 2C$. But because both α_1 and α_2 are bounded, from (2.45)

$$\xi_1 = 1 + b - 2C \geq 0, \tag{2.199}$$

$$\xi_2 = 1 - b - 2C \geq 0. \tag{2.200}$$

Thus, because $\xi_1 + \xi_2 = 2 - 4C$, the primal objective function does not change if ξ_1 and ξ_2 change so long as $\xi_1 + \xi_2 = 2 - 4C$. Therefore

$$-1 + 2C \leq b \leq 1 - 2C. \tag{2.201}$$

Because $\alpha_1 = \alpha_2 = C$ and b is not included in the dual problem, the dual problem is unique, but the primal problem is not. Namely, the primal objective function $w^2/2 + C(\xi_1 + \xi_2)$ is maximized for $w = 2C$ and (ξ_1, ξ_2) that satisfy (2.199) and (2.200).

For the L2 soft-margin support vector machine, the dual problem is given as follows. Maximize

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 - \frac{1}{2}\left(\left(1 + \frac{1}{C}\right)(\alpha_1^2 + \alpha_2^2) + 2\,\alpha_1\,\alpha_2\right) \tag{2.202}$$

subject to

$$\alpha_1 - \alpha_2 = 0, \tag{2.203}$$
$$\alpha_i \geq 0 \quad \text{for} \quad i = 1, 2. \tag{2.204}$$

Substituting $\alpha_2 = \alpha_1$ into (2.202), we obtain

$$Q(\boldsymbol{\alpha}) = 2\,\alpha_1 - \left(2 + \frac{1}{C}\right)\alpha_1^2 \tag{2.205}$$
$$\text{subject to} \quad \alpha_1 \geq 0. \tag{2.206}$$

Thus, (2.205) is maximized when

$$\alpha_1 = \alpha_2 = \frac{C}{2\,C + 1}, \tag{2.207}$$

and from (2.122), $b = 0$. Therefore, the problem is uniquely solved.

Burges and Crisp [46] derived general conditions in which primal problems have nonunique solutions. Here, we discuss a simplified version.

Theorem 2.19. *If the support vectors are all bounded, b is not uniquely determined and the numbers of support vectors belonging to Classes 1 and 2 are the same.*

Proof. The KKT conditions are given by

$$(C - \alpha_i)\,\xi_i = 0, \tag{2.208}$$
$$\alpha_i\,\left(y_i\,(\mathbf{w}^T\mathbf{g}(\mathbf{x}_i) + b) - 1 + \xi_i\right) = 0. \tag{2.209}$$

If $\alpha_i \neq 0$ and $\alpha_i \neq C$, b is uniquely determined by (2.209). Thus, if $\alpha_i \neq 0$, then $\alpha_i = C$. Because $\sum_{i\in S} y_i\,\alpha_i = 0$, the numbers of support vectors for Classes 1 and 2 need to be the same for $\alpha_i = C\,(i \in S)$.

For $\alpha_i = C$,

$$\xi_i = 1 - y_i\,(\mathbf{w}^T\mathbf{g}(\mathbf{x}_i) + b) \geq 0. \tag{2.210}$$

Because the primal objective function concerning $\mathbf{w}$ is strictly positive definite, $\mathbf{w}$ is uniquely determined. In addition, because the numbers of support vectors for Classes 1 and 2 are the same,

$$\sum_{i\in S}\xi_i = |S| - \sum_{i\in S} y_i\,\mathbf{w}^T\mathbf{g}(\mathbf{x}_i), \tag{2.211}$$

which is constant irrespective of the values of ξ_i. This means that the primal objective function is constant for different values of ξ_i. Therefore, b is not unique, and from (2.210),

$$\min_{\substack{i \in S, \\ y_i = 1}} \left(1 - \mathbf{w}^T \mathbf{g}(\mathbf{x}_i)\right) \geq b \geq \max_{\substack{i \in S, \\ y_i = -1}} \left(-1 - \mathbf{w}^T \mathbf{g}(\mathbf{x}_i)\right). \blacksquare \tag{2.212}$$

2.6.5 Reducing the Number of Support Vectors

One of the problems of support vector machines is that when the numbers of input variables and support vectors are large, classification is slow. To overcome this problem, Burges [43] proposed simplifying decision rules by approximating the weight vector by the major eigenvalues. Some extended support vector methods [164], and others used support vector–like methods [58, 59].

Here we first show the geometrical interpretation of hard-margin support vector machines [132], and then we consider the possibility of reducing the number of support vectors [252].

The solution of a hard-margin support vector machine can be geometrically interpreted [132]. Let U_1 and U_2 be the convex hulls for Class 1 and Class 2 training data, respectively. Because the solution exists, U_1 and U_2 do not overlap. Let $\mathbf{u}_1^*$ and $\mathbf{u}_2^*$ give the minimum distance between U_1 and U_2:

$$\min_{\mathbf{u}_1 \in U_1, \mathbf{u}_2 \in U_2} \|\mathbf{u}_1 - \mathbf{u}_2\|. \tag{2.213}$$

Then the optimal hyperplane passes through the middle point of $\mathbf{u}_1^*$ and $\mathbf{u}_2^*$ and perpendicular to $\mathbf{u}_1^* - \mathbf{u}_2^*$. Thus, $\mathbf{w}$ and b are given by

$$\mathbf{w} = \frac{2(\mathbf{u}_1^* - \mathbf{u}_2^*)}{\|\mathbf{u}_1^* - \mathbf{u}_2^*\|^2}, \tag{2.214}$$

$$b = -\frac{\|\mathbf{u}_1^*\|^2 - \|\mathbf{u}_2^*\|^2}{\|\mathbf{u}_1^* - \mathbf{u}_2^*\|^2}. \tag{2.215}$$

It is noted that $\mathbf{u}_1^*$ and $\mathbf{u}_2^*$ are not necessarily training data. In Fig. 2.14 (a), one is a training datum but the other is not; in Fig. 2.14 (b), both are the training data. Thus in a special case, the optimum separating hyperplane is expressed by two support vectors. This interpretation holds for L1 and L2 support vector machines so long as the problem is linearly separable in the feature space.

From this discussion, we can show the following theorem.

Theorem 2.20. *An optimal hyperplane, which is expressed by a set of support vectors and a bias term, can be expressed by one unbounded support vector, the associated vector in the feature space, and the bias term.*

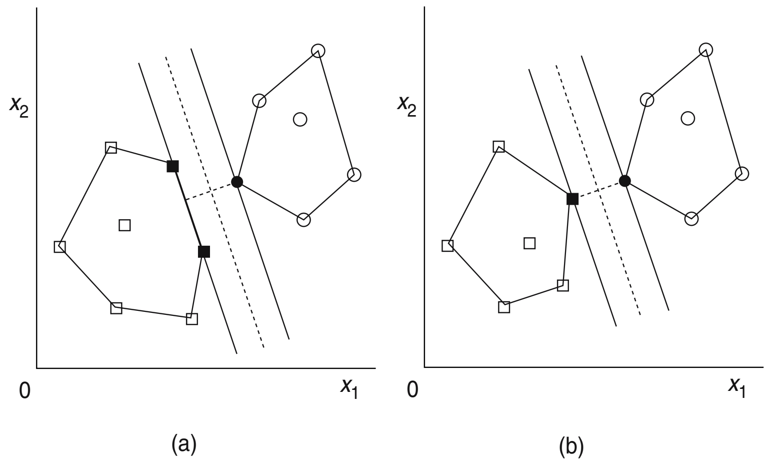


Fig. 2.14. Geometrical interpretation of linearly separable solution: (a) More than two support vectors. (b) Two support vectors

Proof. Let the index set of support vectors for a given problem be S. Then the weight vector is given by

$$\mathbf{w} = \sum_{i \in S} y_i \, \alpha_i \, \mathbf{g}(\mathbf{x}_i). \tag{2.216}$$

Select one unbounded support vector and let this be $\mathbf{x}^+$, which belongs to Class 1. Define $\mathbf{z}^-$ in the feature space by

$$\mathbf{z}^- = \mathbf{g}(\mathbf{x}^+) - 2\,\delta^2\,\mathbf{w}, \tag{2.217}$$

where δ is the margin of the separating hyperplane. Because $\mathbf{w}$ is orthogonal to the hyperplane and 2δ is the distance between the two hyperplanes on which the unbounded support vectors reside, $\mathbf{z}^-$ satisfies $D(\mathbf{z}^-) = -1$. For the two vectors $\mathbf{x}^+$ and $\mathbf{z}^-$, the optimal separating hyperplane goes through $(\mathbf{g}(\mathbf{x}^+) + \mathbf{z}^-)/2$ and is orthogonal to $(\mathbf{g}(\mathbf{x}^+) - \mathbf{z}^-)/2$. Thus it is the same as the original separating hyperplane. ■

If $\mathbf{x}^-$ that satisfies $\mathbf{z}^- = \mathbf{g}(\mathbf{x}^-)$, i.e., the preimage of $\mathbf{z}^-$, exists, the support vectors can be reduced to two. Or if $\mathbf{v}$ that satisfies $\mathbf{g}(\mathbf{v}) = \mathbf{w}$ exists, we can evaluate the decision function by $D(\mathbf{x}) = H(\mathbf{x}, \mathbf{v}) + b$, which will result in a considerable speedup.

In [215, pp. 544–5], a simple calculation method of the preimage is proposed if it exists and if $H(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}^T \mathbf{x}')$, where $f(\cdot)$ is some scalar function. Let $\{\mathbf{e}_1, \ldots, \mathbf{e}_m\}$ be the basis of the input space. Then

$$H(\mathbf{v}, \mathbf{e}_j) = f(v_j)$$

$$= \mathbf{w}^T \mathbf{g}(\mathbf{e}_j)$$
$$= \sum_{i \in S} \alpha_i \, y_i \, f(x_{ij}), \qquad (2.218)$$

where x_{ij} is the jth element of $\mathbf{x}_i$. For the polynomial kernel, $f(v_j) = (v_j + 1)^d$. Thus, if d is odd, the inverse exists and

$$v_j = f^{-1}\left(\sum_{i \in S} \alpha_i \, y_i \, f(x_{ij})\right). \qquad (2.219)$$

This seems to be correct. Indeed, with linear kernels, $\mathbf{g}^{-1}(\mathbf{z}^-) = \mathbf{z}^-$, or $\mathbf{g}^{-1}(\mathbf{w}) = \mathbf{w}$. But if an m-dimensional vector $\mathbf{x}$ is mapped into an l-dimensional space $(l > m)$, the inverse of $\mathbf{w}$ does not exist. Consider the case where $H(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}' + 1$ and $\mathbf{g}(\mathbf{v}) = a_1 \mathbf{g}(\mathbf{x}_1) + a_2 \mathbf{g}(\mathbf{x}_2)$. Thus, $\mathbf{g}(\mathbf{x}) = (1, \mathbf{x}^T)^T$ and $l = m + 1$. Then the following $(m + 1)$ equations must be satisfied for m variables:

$$1 = a_1 + a_2, \qquad (2.220)$$
$$v_1 = a_1 \, x_{11} + a_2 \, x_{21}, \qquad (2.221)$$
$$\dots$$
$$v_m = a_1 \, x_{1m} + a_2 \, x_{2m}. \qquad (2.222)$$

This set of simultaneous equations is solved only when (2.220) is satisfied.

For the polynomial kernel with degree 2 with a one-dimensional input x, $H(x, x') = (1 + x \, x')^2$ and $\mathbf{g}(x)$ is given by

$$\mathbf{g}(x) = (1, \sqrt{2}\, x, x^2)^T. \qquad (2.223)$$

Thus, the following equations must be satisfied:

$$1 = a_1 + a_2, \qquad (2.224)$$
$$v = a_1 \, x_1 + a_2 \, x_2, \qquad (2.225)$$
$$v^2 = a_1 \, x_1^2 + a_2 \, x_2^2, \qquad (2.226)$$

which is, in general, unsolvable.

In general, a set of l equations must be satisfied for m variables. Thus, if $l \neq m$, the inverse does not exist. This is caused by the fact that the region of $\mathbf{g}(\mathbf{x})$ is nonconvex, as discussed in Section 2.3.4.

This is discouraging because Theorem 2.20 is useful only when $d = 1$, which is trivial.

To evaluate the speedup by reducing the number of support vectors to two, we used the data sets listed in Table 1.1. Table 2.6 shows the results for $d = 1$. The columns "SVs," "Original," "Reduced," and "Speedup" show the number of support vectors, the classification time of the training and test data using the support vectors, the classification time using the reduced support vectors, and the speedup by reduction, respectively. From the table, it is seen, except for the thyroid data, that the speedup is roughly the number of support vectors divided by two.

Table 2.6. Classification speedup by reducing the number of support vectors to two ($d = 1$)

Data	SVs	Original (s)	Reduced (s)	Speedup
Blood cell	14	8	1	8.0
Thyroid	80	33	10	3.3
Hiragana-50	11	263	49	5.4
Hiragana-13	7	113	31	3.6
Hiragana-105	13	1029	158	6.5

2.6.6 Degenerate Solutions

Rifkin, Pontil, and Verri [201] discussed degenerate solutions in which $\mathbf{w} = \mathbf{0}$ for L1 support vector machines. Fernández [82] derived similar results for L1 support vector machines, although he did not refer to degeneracy. Degeneracy occurs also for L2 support vector machines. In the following, we discuss degenerate solutions following the proof in [82].

Theorem 2.21. *Let $C = K\,C_0$, where K and C_0 are positive parameters and $\boldsymbol{\alpha}^*$ be the solution of the L1 support vector machine with $K = 1$. Define*

$$\mathbf{w}(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i\, y_i\, \mathbf{g}(\mathbf{x}_i). \tag{2.227}$$

Then the necessary and sufficient condition for

$$\mathbf{w}(\boldsymbol{\alpha}^*) = \mathbf{0} \tag{2.228}$$

is that $K\,\boldsymbol{\alpha}^*$ is also a solution for any $K\,(> 1)$.

Proof. We prove the theorem for L2 support vector machines. The proof for L1 support vector machines is obtained by deleting $\boldsymbol{\alpha}^T\boldsymbol{\alpha}/(2C)$ in the following proof.

Necessary condition. Let $\boldsymbol{\alpha}'$ be the optimal solution for $C = KC_0\,(K > 1)$ and $\mathbf{w}(\boldsymbol{\alpha}') \neq 0$. Define $\boldsymbol{\alpha}' = K\boldsymbol{\alpha}''$. Then, because $\boldsymbol{\alpha}''$ satisfies the equality constraint, it is a nonoptimal solution for $C = C_0$. Then for $C = C_0$,

$$\begin{aligned} Q(\boldsymbol{\alpha}^*) &= \sum_{i=1}^{M} \alpha_i^* - \frac{\boldsymbol{\alpha}^{*T}\boldsymbol{\alpha}^*}{2\,C_0} \\ &\geq Q(\boldsymbol{\alpha}'') = \sum_{i=1}^{M} \alpha_i'' - \frac{1}{2}\mathbf{w}(\boldsymbol{\alpha}'')^T\mathbf{w}(\boldsymbol{\alpha}'') - \frac{\boldsymbol{\alpha}''^T\boldsymbol{\alpha}''}{2\,C_0}. \end{aligned} \tag{2.229}$$

For $C = KC_0\,(K > 1)$,

$$\begin{aligned} Q(K\boldsymbol{\alpha}^*) &= K\sum_{i=1}^{M} \alpha_i^* - \frac{K\boldsymbol{\alpha}^{*T}\boldsymbol{\alpha}^*}{2\,C_0} \leq Q(K\boldsymbol{\alpha}'') \\ &= K\sum_{i=1}^{M} \alpha_i'' - \frac{K^2}{2}\mathbf{w}(\boldsymbol{\alpha}'')^T\mathbf{w}(\boldsymbol{\alpha}'') - \frac{K\boldsymbol{\alpha}''^T\boldsymbol{\alpha}''}{2\,C_0}. \qquad (2.230) \end{aligned}$$

Multiplying all the terms in (2.229) by K and comparing it with (2.230), we see the contradiction. Thus, $K\boldsymbol{\alpha}^*$ is the optimal solution for $K > 1$.
Sufficient condition. Suppose $K\boldsymbol{\alpha}^*$ is the optimal solution for any $C = K\,C_0\,(\geq 1)$. Thus for any $C = K\,C_0\,(\geq 1)$,

$$\begin{aligned} Q(K\boldsymbol{\alpha}^*) &= \sum_{i=1}^{M} K\,\alpha_i^* - \frac{1}{2}\,K^2\,\mathbf{w}^T(\boldsymbol{\alpha}^*)\,\mathbf{w}(\boldsymbol{\alpha}^*) - \frac{K\,\boldsymbol{\alpha}^{*T}\boldsymbol{\alpha}^*}{2\,C_0} \\ &\geq Q(\boldsymbol{\alpha}^*) = \sum_{i=1}^{M} \alpha_i^* - \frac{1}{2}\,\mathbf{w}^T(\boldsymbol{\alpha}^*)\,\mathbf{w}(\boldsymbol{\alpha}^*) - \frac{\boldsymbol{\alpha}^{*T}\boldsymbol{\alpha}^*}{2\,C_0}. \qquad (2.231) \end{aligned}$$

Rewriting (2.231), we have

$$\sum_{i=1}^{M} \alpha_i^* \geq \frac{K+1}{2}\,\mathbf{w}^T(\boldsymbol{\alpha}^*)\,\mathbf{w}(\boldsymbol{\alpha}^*) + \frac{\boldsymbol{\alpha}^{*T}\boldsymbol{\alpha}^*}{2\,C_0}. \qquad (2.232)$$

Because (2.232) is satisfied for any large K, $\mathbf{w}(\boldsymbol{\alpha}^*) = \mathbf{0}$ must be satisfied. Otherwise $K\boldsymbol{\alpha}^*$ cannot be the optimal solution. ■

Example 2.22. Now reconsider the case shown in Fig. 2.5. Here, we use the linear kernel. The inequality constraints given by (2.40) are

$$-w + b \geq 1 - \xi_1, \qquad (2.233)$$

$$-b \geq 1 - \xi_2, \qquad (2.234)$$

$$w + b \geq 1 - \xi_3. \qquad (2.235)$$

The dual problem for the L1 support vector machine is given as follows. Maximize

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 + \alpha_3 - \frac{1}{2}\,(-\alpha_1 + \alpha_3)^2 \qquad (2.236)$$

subject to

$$\alpha_1 - \alpha_2 + \alpha_3 = 0, \qquad (2.237)$$

$$C \geq \alpha_i \geq 0 \qquad \text{for} \quad i = 1, 2, 3. \qquad (2.238)$$

From (2.237), $\alpha_2 = \alpha_1 + \alpha_3$. Then substituting it into (2.236), we obtain

$$Q(\boldsymbol{\alpha}) = 2\,\alpha_1 + 2\,\alpha_3 - \frac{1}{2}\,(-\alpha_1 + \alpha_3)^2, \tag{2.239}$$

$$C \geq \alpha_i \geq 0 \qquad \text{for} \quad i = 1, 2, 3. \tag{2.240}$$

Because $Q(\boldsymbol{\alpha})$ is symmetric for α_1 and α_3, it is maximized when $\alpha_1 = \alpha_3$. Thus the optimal solution is given by

$$\alpha_1 = \frac{C}{2}, \quad \alpha_2 = C, \quad \alpha_3 = \frac{C}{2}. \tag{2.241}$$

Therefore, $x = -1$, 0, and 1 are support vectors; and $w = 0$ and $b = 1$. Because the two unbounded support vectors, α_1 and α_3, belong to the same class, this is an abnormal solution; all the data are classified into Class 1, irrespective of the input.

The dual objective function for the L2 support vector machine is given by

$$Q(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 + \alpha_3 - \frac{1}{2}\,(-\alpha_1 + \alpha_3)^2 - \frac{\alpha_1^2 + \alpha_2^2 + \alpha_3^2}{2\,C}. \tag{2.242}$$

The objective function is maximized when

$$\alpha_1 = \alpha_3 = \frac{2\,C}{3}, \quad \alpha_2 = \frac{4\,C}{3}. \tag{2.243}$$

Thus, $w = 0$ and $b = 1$. Therefore, any datum is classified into Class 1.

2.6.7 Duplicate Copies of Data

Nonunique solutions occur when duplicate copies of the datum with the same label are included in the training data. If $\mathbf{x}_i = \mathbf{x}_j$, a solution $\boldsymbol{\alpha}^*$ satisfies $\alpha_i^* = \alpha_j^*$. This can be shown as follows. Suppose $\alpha_i^* \neq \alpha_j^*$. Then because of the symmetry of the variables, $(\alpha_i, \alpha_j) = (\alpha_j^*, \alpha_i^*)$ is also a solution. Because for quadratic programming problems the nonunique solutions cannot be isolated [46], the nonunique solution satisfies

$$\alpha_i = \beta\,\alpha_i^* + (1-\beta)\,\alpha_j^*, \tag{2.244}$$

$$\alpha_j = (1-\beta)\,\alpha_i^* + \beta\,\alpha_j^*, \tag{2.245}$$

where $0 \leq \beta \leq 1$. Then setting $\beta = 1/2$,

$$\alpha_i = \alpha_j = \frac{\alpha_i^* + \alpha_j^*}{2}. \tag{2.246}$$

Let $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ be a set of training data with each datum different from the others. Assume that for each $\mathbf{x}_i$ we add $\zeta_i - 1$ copies of $\mathbf{x}_i$ to the set. Then using the preceding result, the same ζ_i data share the same Lagrange multiplier. Thus the training of the L1 support vector machine for the set is to maximize [52]

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \zeta_i \, \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \zeta_i \, \alpha_i \, \zeta_j \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) \tag{2.247}$$

subject to the constraints

$$\sum_{i=1}^{M} y_i \, \zeta_i \, \alpha_i = 0, \qquad 0 \leq \alpha_i \leq C \quad \text{for} \quad i = 1, \ldots, M. \tag{2.248}$$

Changing the variables by $\alpha'_i = \zeta_i \, \alpha_i$, (2.247) and (2.248) become, respectively,

$$Q(\boldsymbol{\alpha}') = \sum_{i=1}^{M} \alpha'_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha'_i \, \alpha'_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j), \tag{2.249}$$

$$\sum_{i=1}^{M} y_i \, \alpha'_i = 0, \qquad 0 \leq \alpha'_i \leq \zeta_i \, C \quad \text{for} \quad i = 1, \ldots, M. \tag{2.250}$$

Thus, if the solution does not have bounded support vectors, the solution is the same as that without copies.[8]

The training of the L2 support vector machine for the set of training data is to maximize

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \zeta_i \, \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \zeta_i \, \alpha_i \, \zeta_j \, \alpha_j \, y_i \, y_j \left(H(\mathbf{x}_i, \mathbf{x}_j) + \frac{\delta_{ij}}{\zeta_i \, C} \right) \tag{2.251}$$

subject to the constraints

$$\sum_{i=1}^{M} y_i \, \zeta_i \, \alpha_i = 0, \qquad \alpha_i \geq 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{2.252}$$

Changing the variables by $\alpha'_i = \zeta_i \, \alpha_i$, (2.251) and (2.252) become, respectively,

$$Q(\boldsymbol{\alpha}') = \sum_{i=1}^{M} \alpha'_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha'_i \, \alpha'_j \, y_i \, y_j \left(H(\mathbf{x}_i, \mathbf{x}_j) + \frac{\delta_{ij}}{\zeta_i \, C} \right), \tag{2.253}$$

$$\sum_{i=1}^{M} y_i \, \alpha'_i = 0, \qquad \alpha'_i \geq 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{2.254}$$

Thus, an addition of copied data affects the solution of the L2 support vector machine.

To estimate the generalization ability of classifiers using a small set of data, resampling with replacement is often used. To speed up evaluation, it is advisable to use (2.249) and (2.250), or (2.253) and (2.254) instead of copying data.

[8]The formulation given by (2.249) and (2.250) is the same as that of the fuzzy support vector machine discussed in [151].

2.6.8 Imbalanced Data

In medical diagnosis, usually, misclassification of abnormal data into the normal class is less favorable than misclassification of normal data into the abnormal class. To control misclassification, Veropoulos, Campbell, and Cristianini [260] proposed preparing different margin parameters C^+ and C^- for the normal ($+1$) and abnormal (-1) classes, respectively, and setting $C^- > C^+$. Lee, Gunn, Harris, and Reed [144] applied this method to diagnosis problems with imbalanced data. By setting the ratio of C^+ and C^- about the ratio of the number of data for Classes -1 and 1, better classification performance was obtained than by the conventional support vector machine. Xu and Chan [275] determined the ratios of a one-against-all support vector machine by genetic algorithms.

For imbalanced data, resampling, which allows multiple selection of the same datum, is usually used. In that situation, we had better use the formulation of support vector machines discussed in Section 2.6.7.

In some classification problems, a priori class probabilities are given, and they are different from the ratios of class data to the total data. To compensate for this disparity, Cawley and Talbot [52] proposed the formulation discussed in Section 2.6.7. Namely, for the class j datum $\mathbf{x}_i$ we set

$$\zeta_i = \frac{p_j^o}{p_j^t}, \tag{2.255}$$

where p_j^o is the a priori probability for class j and p_j^t is given by the number of class j data divided by the number of total training data.

In [94], the bias term is adjusted for the unbalanced data from the link between the least squares support vector machines and kernel discriminant analysis.

2.6.9 Classification for the Blood Cell Data

In this section we show the effect of parameters on classification using the blood cell data for Classes 2 and 3 in Table 1.1. Except for the Class 2 training data set, which includes 399 data, training and test data sets include 400 data each. The blood cell data are very difficult to classify, and Classes 2 and 3 are the most difficult.

We investigated the effect of kernel parameters and the margin parameter C on the recognition rates and the number of support vectors using L1 and L2 support vector machines.

Figure 2.15 shows the recognition rates for the change of the polynomial degree fixing $C = 5000$. For $d = 1$ (i.e., linear kernels), the recognition rates of the training data are about 90 percent for L1 and L2 support vector machines and for d larger than 2, the recognition rates are 100 percent. Thus the training data are not linearly separable. For the change of d, the recognition rates of

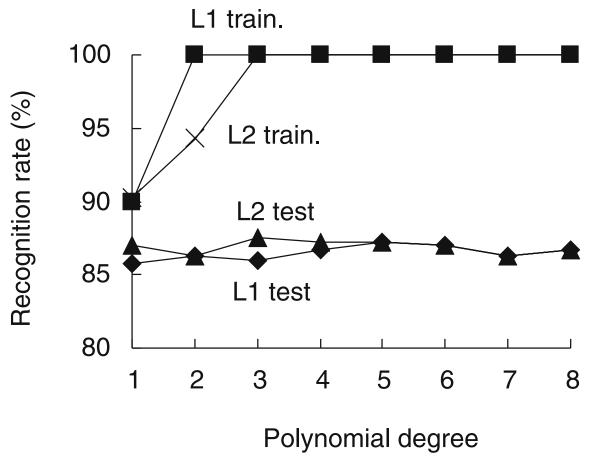


Fig. 2.15. Recognition rates of the blood cell data with polynomial kernels

the test data do not change very much for L1 and L2 support vector machines. This means that overfitting does not occur in this case.

Figure 2.16 shows the number of support vectors for the change of the polynomial degree under the same conditions as those in Fig. 2.15. As seen from Fig. 2.16, because the blood cell data are not linearly separable, especially for the L2 support vector machine, a large number of support vectors are necessary for $d = 1$. For the polynomial degree lower than 4, the number of support vectors for the L2 support vector machine is larger than that for the L1 support vector machine, but for degrees higher than 3, they are almost the

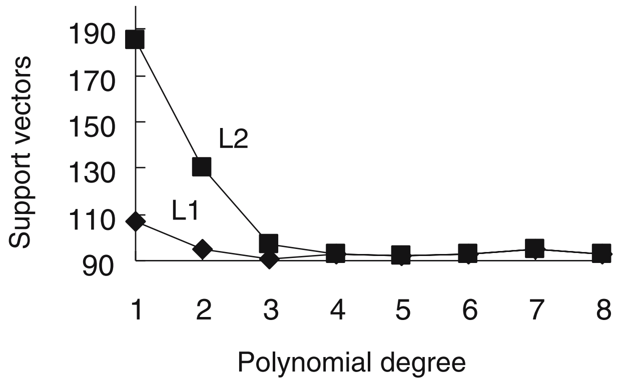


Fig. 2.16. Number of support vectors for the blood cell data with polynomial kernels

same. From Figs. 2.15 and 2.16, for d larger than 3, behaviors of the L1 and L2 support vector machines are almost the same.

Figure 2.17 shows the recognition rates for the RBF kernel with different values of γ fixing $C = 5000$. The recognition rates of the L1 and L2 support vector machines are almost the same, and for γ lager than 1, the recognition rates of the training data are 100 percent. For the change of γ, the recognition rates of the test data do not change very much for L1 and L2 support vector machines.

Figure 2.18 shows the number of support vectors for the RBF kernel with different values of γ under the same conditions as those in Fig. 2.17. For γ smaller than 1, the number of support vectors for the L2 support vector machine is larger than for the L1 support vector machine, but for γ larger than or equal to 1, they are almost the same. For $\gamma = 100$, the numbers of support vectors are 396. As γ becomes smaller, the radius of RBF becomes smaller. Thus, a larger number of support vectors are necessary to classify the training data correctly. Similar to the case with the polynomial kernels, the behaviors of the L1 and L2 support vector machines are almost the same for γ larger than 1.

Figure 2.19 shows the recognition rates when C is changed with the polynomial degree of 4. As the value of C is increased, the weight for the sum (square sum) of the slack variables is increased. Thus the recognition rate of the training data is improved, and the recognition rate reaches 100 percent for $C = 1000$. On the other hand, the recognition rate of the test data gradually decreases. Thus there is a tendency of overfitting. Similar to the previous results, the recognition rates of L1 and L2 support vector machines are almost the same.

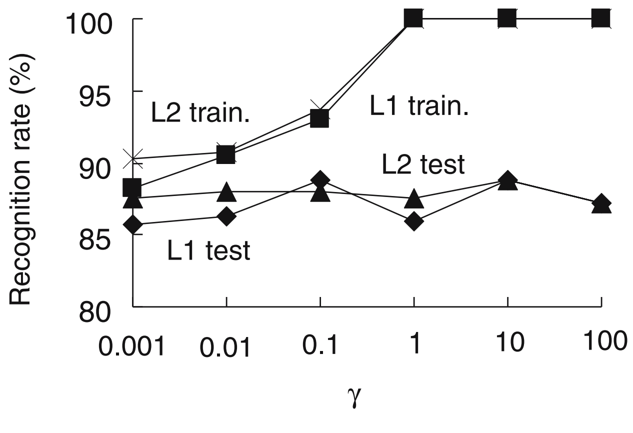


Fig. 2.17. Recognition rates of the blood cell data with RBF kernels

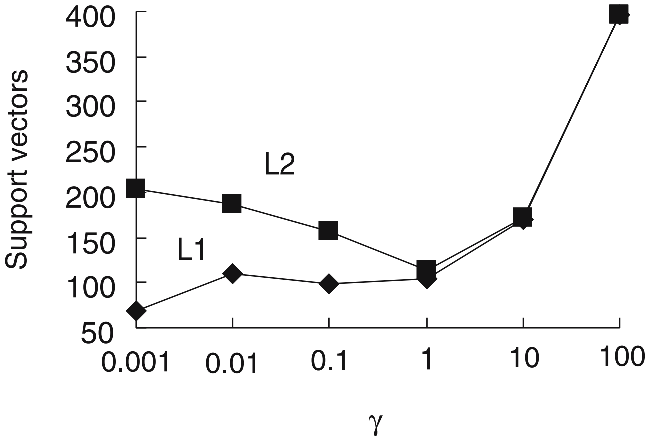


Fig. 2.18. Number of support vectors for the blood cell data with RBF kernels

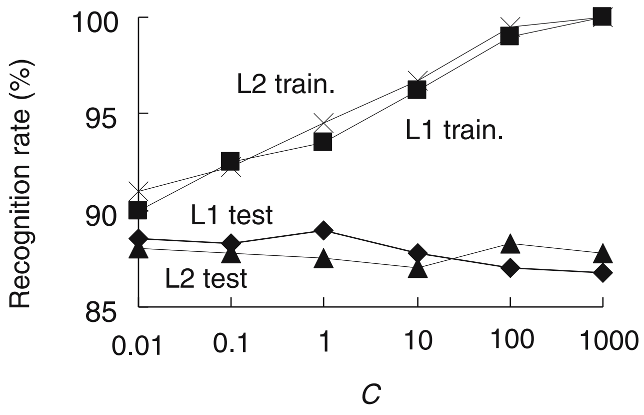


Fig. 2.19. Recognition rates of the blood cell data with the polynomial kernel ($d = 4$) for the change of C

Figure 2.20 shows the numbers of support vectors for the change of C. As the value of C increases, the weight of the (square) sum of slack variables is increased. Thus the number of support vectors decreases. At $C = 1000$, the number of support vectors for the L1 and L2 support vector machines is 93 each.

To investigate the robustness of support vector machines, as outliers, we added 10 data belonging to classes other than 2 and 3 to Class 2 training

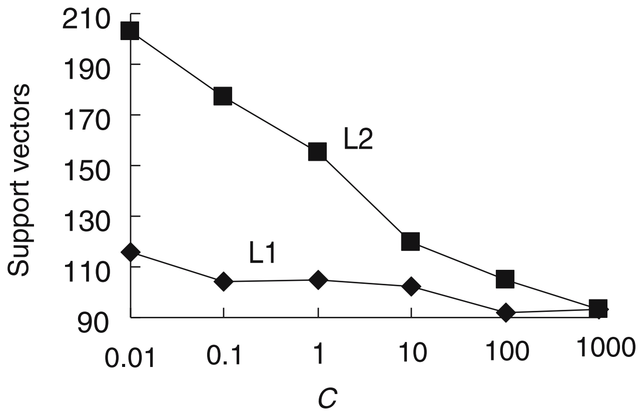


Fig. 2.20. Number of support vectors for the blood cell data with polynomial kernels ($d = 4$) for the change of C

data. Figure 2.21 shows the recognition rate of the test data against the margin parameter C when outliers were added and not added. We used the L2 support vector machine. For $C = 0.01$, the recognition rate with outliers is much worse than without outliers. But with C larger than or equal to 0.01, the difference is not so significant. Thus if an appropriate value is set to C, the support vector machine is robust against outliers.

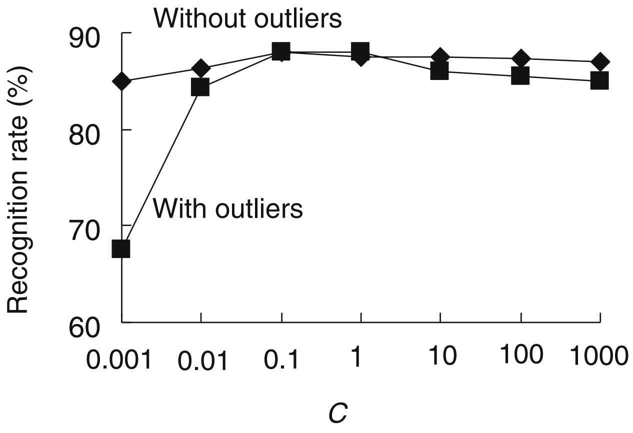


Fig. 2.21. Recognition rates against the margin parameter C for the inclusion of outliers

From the computer simulations for the blood cell data, the following tendencies are seen.

1. The recognition rate of the training data increased as the polynomial degree, γ in RBF kernels, or the value of C is increased. But the recognition rate of the test data does not change very much.
2. In theory, for $C = \infty$, L1 and L2 support vector machines are equivalent. But by computer simulations, this condition may be loosened; for appropriately large d or γ and C, the recognition rates of the training and test data and the number of support vectors are almost the same for the L1 and L2 support vector machines.
3. For an appropriately chosen value of C, the effect of outliers on the recognition rate is not significant.

2.7 Class Boundaries for Different Kernels

In this section we discuss how class boundaries change as kernels are changed for a two-class problem. For the linear kernel, the class boundary is a hyperplane. For the polynomial kernel with degree 2, the class boundary is a hyperplane, a hyperellipsoid (see Fig. 2.22), or a hyperparabola (see Fig. 2.23). This class boundary is equivalent to that of the fuzzy classifier with ellipsoidal regions.

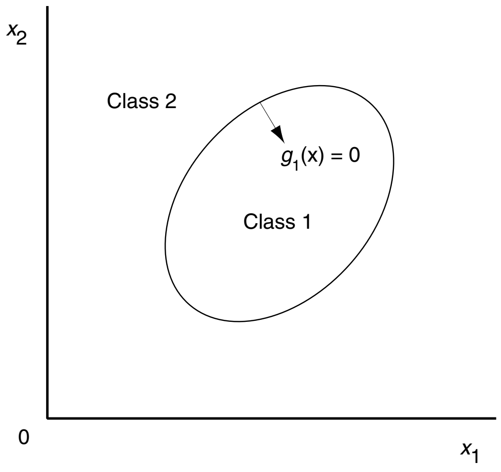


Fig. 2.22. An ellipsoidal boundary by a polynomial kernel with the degree of two

To clarify how class regions are clustered for a given kernel, we restrict our discussion to one input variable. For a polynomial kernel with degree 3,

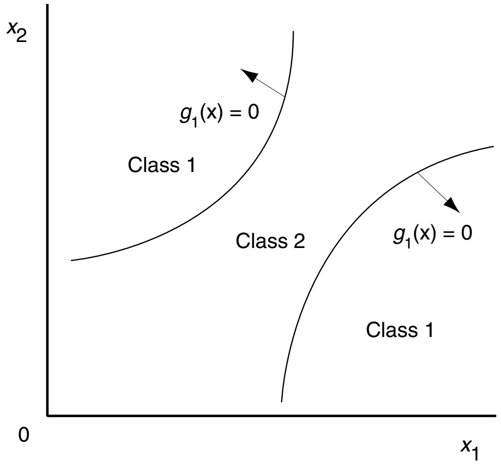


Fig. 2.23. Parabolic boundaries by a polynomial kernel with the degree of two

$$\begin{aligned} H(x, x') &= (x\,x' + 1)^3 \\ &= 1 + 3\,x'\,x + 3\,x'^2\,x^2 + x'^3\,x^3 \\ &= (1, \sqrt{3}\,x, \sqrt{3}\,x^2, x^3)\,(1, \sqrt{3}\,x', \sqrt{3}\,x'^2, x'^3)^T. \end{aligned} \tag{2.256}$$

Thus the function that maps x into the feature space $\mathbf{z}$ is given by $\mathbf{g}(x) = (1, \sqrt{3}\,x, \sqrt{3}\,x^2, x^3)^T$. Therefore, the hyperplane in the feature space is given by

$$w_0 + w_1\,x + w_2\,x^2 + w_3\,x^3 = 0. \tag{2.257}$$

Because the maximum number of solutions of (2.257) for x is 3, the input regions of both Classes 1 and 2 are divided into two subregions in maximum (see Fig. 2.24). In general, for a polynomial kernel with degree n, $n + 1$ subregions are generated in maximum. If n is even, the input regions of both Classes 1 and 2 are divided into $(n + 2)/2$ and $n/2$ subregions in maximum. And if n is odd, those of Classes 1 and 2 are divided into $(n+1)/2$ subregions in maximum.

For an RBF kernel, the induced feature space has infinite dimensions. Therefore, the hyperplane in the feature space can divide the input space into any number of subregions. This is also true for a three-layer neural network kernel. Thus we need not worry about clustering the training data before training.

In a feature space induced by an RBF kernel, data are on the surface of the unit hypersphere centered at the origin because $\mathbf{g}^T(\mathbf{x})\,\mathbf{g}(\mathbf{x}) = \exp(-\gamma\|\mathbf{x}-\mathbf{x}\|^2) = 1$ [280]. Thus, the optimal separating hyperplane $D(\mathbf{x}) = 0$ and hyperplanes $D(\mathbf{x}) = \pm 1$ must intersect with the hypersphere. Figure 2.25 shows an illustration of class regions for a hard-margin support vector ma-

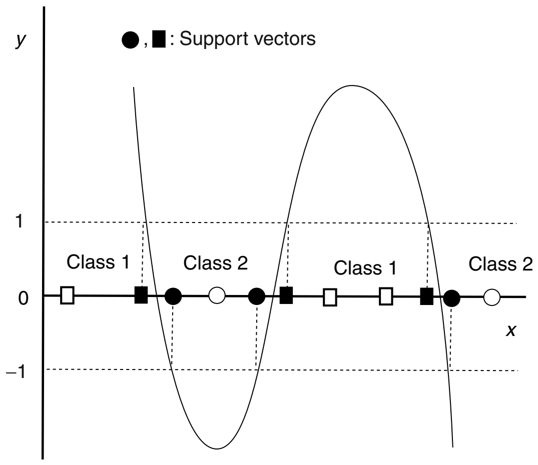


Fig. 2.24. Class regions by a polynomial kernel with the degree of three. Regions for Classes 1 and 2 are divided into two

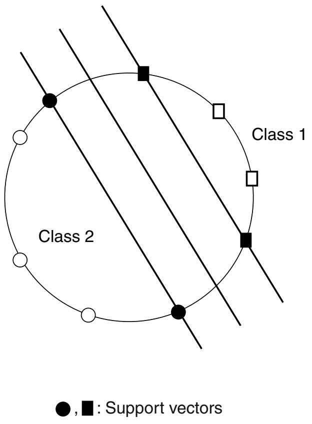


Fig. 2.25. Class regions by an RBF kernel in the feature space

chine. In this way training data that are not linearly separable in the input space are separated in the feature space.

2.8 Developing Classifiers

In this section, we discuss how to develop classifiers with high generalization ability. Here we do not consider optimizing the input features. Thus our aim is to develop the classifier that realizes the best generalization ability for the given input-output training pairs. Here we call the classifier with the best generalization ability the *optimal classifier.*

2.8.1 Model Selection

In training a support vector machine we need to select a kernel and set a value to the margin parameter C. Thus to develop the optimal classifier, we need to determine the optimal kernel parameter and the optimal value of C. Determining the optimal classifier is called *model selection.*

The model selection is usually done by estimating the generalization abilities for the grid points in the kernel-parameter-and-C plane, and selecting the classifier that realizes the highest generalization ability.

The most reliable but time-consuming method of estimating the generalization ability is cross-validation based on repetitive training of support vector machines. Thus to shorten model selection time, several measures for estimating the generalization ability have been proposed.

2.8.2 Estimating Generalization Errors

Cross-validation is a widely used technique to estimate the generalization error of a classifier. Instead of using cross-validation, many measures to estimate the generalization error of support vector machines, which are based on statistical learning theory, are proposed. In the following, we briefly summarize these measures.

Cross-Validation

Cross-validation is used to measure the generalization error of classifiers for a limited number of gathered data. In cross-validation, the M given data are divided into two data sets, S_i^{tr} $(i = 1, \ldots, {}_M C_l)$, which includes l training data, and S_i^{ts}, which includes $M - l$ test data. Then for the training data set S_i^{tr} the classifier is trained and tested for the test data set S_i^{ts}. This is repeated for all the combinations $({}_M C_l)$ of the partitioned training and test data sets, and the total recognition rate for all the test data sets is calculated as the estimation of the classification performance. But because this is a time-consuming task, we usually use k-fold cross-validation.

In k-fold cross validation, training data are randomly divided into approximately equal sized k subsets, and a classifier is trained using $k - 1$ subsets and tested using the remaining subset. Training is repeated k times, and the total recognition rate for all the k subsets that are not included in the training data is calculated. A leave-one-out method (LOO) is a special case of cross-validation $(l = M - 1)$ and k-fold cross validation. The leave-one-out error is known to be an unbiased estimate of the test error [258, p. 265].

For classifiers other than support vector machines, LOO is a time-consuming task when the number of training data is large. But for support vector machines, once we have trained a support vector machine, we need to apply LOO only to support vectors. This is because even if we delete the training data other than support vectors, these data are correctly classified. Cauwenberghs and Poggio's decremental training [51] further speeds up LOO by deleting one support vector from the trained support vector machine.

Saadi, Cawley, and Talbot [208] discussed acceleration of the leave-one-out procedure of least squares support vector machines (see Section 4.1). For leaving the jth datum out, we strike out the jth row and jth column of the coefficient matrix in the set of linear equations that determines $\boldsymbol{\alpha}$ and b. Thus in solving the set of equations by the Gauss-Jordan elimination, we can avoid calculation of the first to $(j-2)$nd elimination by caching these results obtained for the $(j-1)$st datum. Ying and Keong [278] trained a least squares support vector machine using all the training data and then used the matrix inversion lemma to speed up the leave-one-out procedure.

Error Bound by VC Dimension

The VC (Vapnik-Chervonenkis) dimension is the theoretical basis of support vector machines and is defined as the maximum number of samples that can be separated into any combination of two sets by the set of functions. Because the set of m-dimensional hyperplanes can separate at most $m+1$ samples, the VC dimension of the set is $m+1$ (see Fig. 2.26).

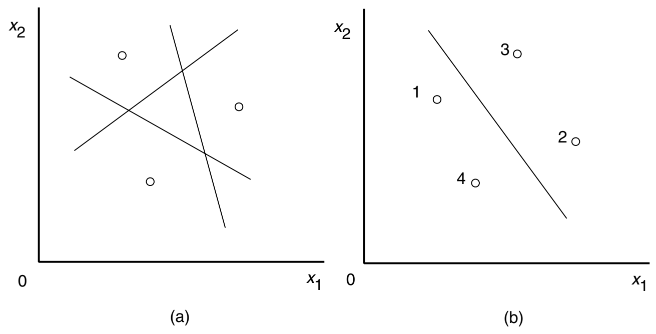


Fig. 2.26. VC dimension of a set of lines: (**a**) Any three data can be separated into any combination of two sets of data by a single line. (**b**) Sets of data {1, 4} and {2, 4} can be separated into the two sets by a line but sets of data {1, 2} and {3, 4} cannot. Thus the VC dimension of the set of lines is three

According to Vapnik's theory [212, pp. 1–15], the generalization error of a support vector machine is bounded with the probability of at least $1-\eta$ by

$$R(\mathbf{w}, b) \leq R_{\text{emp}}(\mathbf{w}, b) + \phi, \qquad (2.258)$$

where $R_{\mathrm{emp}}(\mathbf{w}, b)$ is the empirical risk (classification error) for the M training data and ϕ is the confidence interval (classification error) for the unknown data:

$$\phi = \sqrt{\frac{h\left[\ln\left(\frac{2M}{h}\right)+1\right] - \ln\left(\frac{\eta}{4}\right)}{M}}. \tag{2.259}$$

Here h is the VC dimension of a set of hyperplanes, and for the hard-margin optimal separating hyperplane, the VC dimension of the hyperplanes with margin $\|\mathbf{w}\|^{-1}$, h, is bounded by [44, 256, 258]

$$h \leq \min(D^2\|\mathbf{w}\|^2, l) + 1. \tag{2.260}$$

Here D is the diameter of the smallest hypersphere that includes all the training data. We can determine R by the method discussed in Section 7.1.

When the training data are linearly separable in the feature space, the empirical risk $R_{\mathrm{emp}}(\mathbf{w}, b)$ is zero. Thus if we minimize ϕ, we can maximize the generalization ability. From (2.259), ϕ is the monotonic function of the VC dimension h. Thus ϕ is minimized by minimizing h. From (2.260), this is realized by maximizing the margin $\|\mathbf{w}\|^{-1}$. When the training data are not linearly separable in the feature space, $R_{\mathrm{emp}}(\mathbf{w}, b)$ is not zero. In this case the generalization ability is determined by the trade-off between the empirical risk and the confidence interval via the margin parameter C.

LOO Error Rate Estimators

Several measures for estimating the LOO error rate have been proposed. By taking the average of the LOO error rate for possible combinations of training data, we obtain the error bound for the test data. The error rate estimators discussed in the following need to train the support vector machine using the training data once.

1. Vapnik [256]
   The LOO error rate is bounded by

   $$\varepsilon_{\mathrm{loo}} \leq \frac{|S|}{M}, \tag{2.261}$$

   where $\varepsilon_{\mathrm{loo}}$ is the LOO error rate for the given training data, $|S|$ is the number of support vectors, and M is the number of training data.

   Assume that we delete a datum, which is not a support vector, from the training data set. Then the support vector machine trained using the reduced training data set correctly classifies the deleted datum. But if a support vector is deleted, the classification result cannot be estimated. Thus assuming that all the support vectors are misclassified, (2.261) is obtained.

2. Joachims $\xi\,\alpha$ estimators [122, 123]
The LOO error rate is bounded by
$$\varepsilon_{\mathrm{loo}} \leq \frac{|\{i \mid 2\,\alpha_i\,R_{\Delta}^2 + \xi_i \geq 1\}|}{M}, \tag{2.262}$$
where R_{Δ}^2 is the upper bound that satisfies
$$c \leq H(\mathbf{x}, \mathbf{x}') \leq c + R_{\Delta}^2 \tag{2.263}$$
for all $\mathbf{x}$ and $\mathbf{x}'$. Because the inequality on the right-hand side of (2.262) does not hold for zero α_i, (2.262) is an improved version of (2.261).
3. Vapnik and Chapelle [56, 258]
The LOO error rate is bounded by
$$\varepsilon_{\mathrm{loo}} \leq \frac{S_m \max(D, 1/\sqrt{C}) \sum_{i \in U} \alpha_i + |B|}{M}, \tag{2.264}$$
where D is the diameter of the smallest hypersphere that includes the training data, U is the set of unbounded support vector indices, B is the set of bounded support vector indices, and S_m is the span of support vectors and is defined as follows:
$$\begin{aligned} S_m &= \max_p S_p, \\ S_p^2 &= \min_{\mathbf{x} \in \Lambda_p} (\mathbf{x}_p - \mathbf{x})^2, \\ \Lambda_p &= \left\{ \sum_{i \in S, i \neq p} \lambda_i \mathbf{x}_i \mid \sum_{i \in S, i \neq p} \lambda_i = 1, \alpha_i + y_i\, y_p\, \alpha_p\, \lambda_i \geq 0 \text{ for } \forall i \neq p \right\}. \end{aligned}$$
Here S_p is the distance between support vector $\mathbf{x}_p$ and the set Λ_p, and S is the set of support vector indices. The bounded support vectors are assumed to be misclassified when deleted by the LOO procedure.

To optimize parameters of support vector machines, the error bound need not be accurate so long as it shows the correct tendency for the parameter change. Duan, Keerthi, and Poo [76] evaluated the five-fold cross-validation, the LOO bounds by (2.262), ϕ in (2.258) with h evaluated by (2.260), and (2.264) for three benchmark data sets with 400 to 1300 training data. The RBF kernels were used, and C and γ were tuned using the preceding four measures for the training data. Then the best recognition rates of the test data sets were compared with the recognition rates obtained by model selection. The five-fold cross-validation gave the best results, and the bound by (2.262) gave the second best. Because S_m in (2.264) was replaced by D_U, which is the diameter of the smallest hypersphere that includes the unbounded support vectors, the results were not good.

Anguita, Boni, and Ridella [18] evaluated the bound given by (2.260) for the six data sets with 80 to 958 data by the first-order accurate bootstrap [78]. For a fixed γ for the RBF kernels, 1000 training data sets were generated by drawing from the original training data set with replacement. Then for each data set, the support vector machine was trained and the average and standard deviation of the right-hand side of (2.260) were evaluated. Although the bound given by (2.260) was loose, it had the same tendency with the recognition rates evaluated for the data not included in the generated training data.

2.8.3 Sophistication of Model Selection

Because model selection is time-consuming, several methods have been developed to ease model selection [68, 180, 181, 213, 220]. Cristianini and Campbell [68] showed that the bound on the generalization error is smooth in the kernel parameter. Namely, when the margin is optimal, small variations in the kernel parameter result in small variations in the margin. Then the model selection for the RBF kernel is done as follows. Train the support vector machine with a small value of σ, evaluate the error bound, increment the value of σ, and repeat the procedure until the optimal parameter is obtained.

Friedrichs and Igel [87] used evolution strategies to further tune the parameters obtained by grid search. They showed, by computer experiments, that using the generalized RBF kernel (Mahalanobis kernel):

$$H(\mathbf{x}, \mathbf{x}') = \exp\left(-(\mathbf{x} - \mathbf{x}')^T A\, (\mathbf{x} - \mathbf{x}')\right), \tag{2.265}$$

where A is a positive definite matrix, the generalization ability was improved with a smaller number of support vectors.

Lebrun, Charrier, and Cardot [142] used the vector quantization technique to replace training data with a smaller number of prototypes and then to speed up model selection.

2.9 Invariance for Linear Transformation

Because fuzzy classifiers with ellipsoidal regions [3, pp. 208–9] are based on the Mahalanobis distance, they are invariant for the linear transformation of input variables; specifically translation, scaling, and rotation invariant. But most classifiers, such as multilayer neural networks, are not. Thus, to avoid the influence of variables with large input ranges on the generalization ability, we usually scale the ranges of input variables into $[0, 1]$ or $[-1, 1]$.

In [215, pp. 333–58] and [45], invariance of the kernel method, in which a small variance of the input does not affect the classification results, is discussed.

In this section, we discuss invariance of support vector machines for linear transformation of input variables [6]. Here, we consider translation, scaling, and rotation. Then, we clarify the relationships between the input ranges $[0, 1]$ and $[-1, 1]$.

The Euclidean distance is used to calculate the margins, and it is rotation- and translation-invariant but not scale-invariant. Therefore, support vector machines with linear kernels are rotation- and translation-invariant but not scale-invariant. In general, the Euclidean distance is not scale-invariant, but if all the input variables are scaled with the same factor, the Euclidean distance changes with that factor. Therefore here we consider the following transformation:

$$\mathbf{z} = s\,A\,\mathbf{x} + \mathbf{c}, \tag{2.266}$$

where $s\,(> 0)$ is a scaling factor, A is an orthogonal matrix and satisfies $A^T A = I$, and $\mathbf{c}$ is a constant vector.

Now the RBF kernel $H(\mathbf{z}, \mathbf{z}')$ is given by

$$\begin{aligned} H(\mathbf{z}, \mathbf{z}') &= \exp(-\gamma' \, \|s\,A\,\mathbf{x} + \mathbf{c} - s\,A\,\mathbf{x}' - \mathbf{c}\|^2) \\ &= \exp(-\gamma' \, \|s\,A\,(\mathbf{x} - \mathbf{x}')\|^2) \\ &= \exp(-\gamma' \, s^2 \, \|\mathbf{x} - \mathbf{x}'\|^2). \end{aligned} \tag{2.267}$$

Therefore, RBF kernels are translation- and rotation-invariant. For $s \neq 1$, if

$$\gamma' s^2 = \gamma, \tag{2.268}$$

$H(\mathbf{z}, \mathbf{z}') = H(\mathbf{x}, \mathbf{x}')$. Thus, if (2.268) is satisfied, the optimal solutions for a training data set and the data set transformed by (2.266) are the same.

The neural network kernel $H(\mathbf{z}, \mathbf{z}')$ is given by

$$H(\mathbf{z}, \mathbf{z}') = \frac{1}{1 + \exp(\nu' \, (s\,A\,\mathbf{x} + \mathbf{c})^T (s\,A\,\mathbf{x}' + \mathbf{c}) - a)}. \tag{2.269}$$

If $\mathbf{c} \neq 0$, (2.269) is not invariant. Setting $\mathbf{c} = 0$, (2.269) becomes

$$H(\mathbf{z}, \mathbf{z}') = \frac{1}{1 + \exp(\nu' \, s^2 \, \mathbf{x}^T \mathbf{x}' - a)}. \tag{2.270}$$

Therefore, neural network kernels are rotation-invariant. If

$$\nu' s^2 = \nu \tag{2.271}$$

is satisfied, $H(\mathbf{z}, \mathbf{z}') = H(\mathbf{x}, \mathbf{x}')$. Thus, if (2.271) is satisfied, the optimal solutions for a training data set and the data set transformed by (2.266) with $\mathbf{c} = \mathbf{0}$ are the same.

For the linear kernel, $H(\mathbf{z}, \mathbf{z}')$ is given by

$$\begin{aligned} H(\mathbf{z}, \mathbf{z}') &= (s\,A\,\mathbf{x} + \mathbf{c})^T (s\,A\,\mathbf{x}' + \mathbf{c}) \\ &= s^2 \, \mathbf{x}^T \, \mathbf{x}' + s\,\mathbf{c}^T A\,\mathbf{x}' + s\,\mathbf{x}^T A^T \mathbf{c} + \mathbf{c}^T \mathbf{c}. \end{aligned} \tag{2.272}$$

Training of the L1 support vector machine with a data set transformed by (2.266) is as follows. Find $\alpha'_i \; (i = 1, \ldots, M)$ that maximize

$$Q(\boldsymbol{\alpha}') = \sum_{i=1}^{M} \alpha'_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha'_i \, \alpha'_j \, y_i \, y_j \times (s^2 \, \mathbf{x}_i^T \mathbf{x}_j + s \, \mathbf{c}^T A \, \mathbf{x}_j + s \, \mathbf{x}_i^T A^T \mathbf{c} + \mathbf{c}^T \mathbf{c}) \qquad (2.273)$$

subject to the constraints

$$\sum_{i=1}^{M} y_i \, \alpha'_i = 0, \qquad 0 \le \alpha'_i \le C' \quad \text{for} \quad i = 1, \dots, M. \qquad (2.274)$$

Using (2.274), (2.273) becomes

$$\begin{aligned} Q(\boldsymbol{\alpha}') &= \sum_{i=1}^{M} \alpha'_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha'_i \, \alpha'_j \, y_i \, y_j \, s^2 \, \mathbf{x}_i^T \mathbf{x}_j \\ &= s^{-2} \left(\sum_{i=1}^{M} s^2 \, \alpha'_i - \frac{1}{2} \sum_{i,j=1}^{M} s^2 \, \alpha'_i \, s^2 \, \alpha'_j \, y_i \, y_j \, \mathbf{x}_i^T \mathbf{x}_j \right). \end{aligned} \qquad (2.275)$$

Thus, setting $\alpha_i = s^2 \, \alpha'_i$, the inequality constraint in (2.274) becomes

$$0 \le \alpha_i \le s^2 \, C' \quad \text{for} \quad i = 1, \dots, M. \qquad (2.276)$$

Therefore, the optimal solutions of the L1 support vector machine with the linear kernel for a training data set and the data set transformed by (2.266) are the same when

$$C = s^2 C'. \qquad (2.277)$$

This also holds for L2 support vector machines.

For the polynomial kernel, $H(\mathbf{z}, \mathbf{z}')$ is given by

$$\begin{aligned} H(\mathbf{z}, \mathbf{z}') &= \left((s \, A \, \mathbf{x} + \mathbf{c})^T (s \, A \, \mathbf{x}' + \mathbf{c}) + 1 \right)^d \\ &= \left(s^2 \, \mathbf{x}^T \, \mathbf{x}' + s \, \mathbf{c}^T A \, \mathbf{x}' + s \, \mathbf{x}^T A^T \mathbf{c} + \mathbf{c}^T \mathbf{c} + 1 \right)^d. \end{aligned} \qquad (2.278)$$

Therefore, polynomial kernels are rotation-invariant but neither scale- nor translation-invariant. This is also true for L2 support vector machines using polynomial kernels. Assuming that (2.278) is approximated by the term with the highest degree of s (at least $s > 1$ is necessary):

$$H(\mathbf{z}, \mathbf{z}') = s^{2d} \, (\mathbf{x}^T \mathbf{x}')^d, \qquad (2.279)$$

similar to the discussions for the linear kernel, the support vector machines with a data set and the data set transformed by (2.266) perform similarly when

$$C = s^{2d} \, C'. \qquad (2.280)$$

In training support vector machines, we normalize the range of input variables into $[0, 1]$ or $[-1, 1]$, without knowing their difference. Using our previous discussions, however, we can clarify relations of the solutions. Because the transformation from $[0, 1]$ to $[-1, 1]$ is given by

$$\mathbf{z} = 2\,\mathbf{x} - \mathbf{1}, \tag{2.281}$$

it is a combination of translation and scaling. Thus according to the previous discussions, we can obtain the parameter values that give the same or roughly the same results for the two input ranges. Table 2.7 summarizes this result.

Table 2.7. Parameters that give the same or roughly the same solutions

Kernel	$[0, 1]$	$[-1, 1]$
Linear	$4\,C$	C
Polynomial	$\approx 4^d\,C$	C
RBF	$4\,\gamma$	γ
NN	$\approx 4\,\nu$	ν

To see the validity of Table 2.7, especially for the polynomial kernels, we conducted the simulation using the blood cell data and the thyroid data sets listed in Table 1.1. For both data sets, we selected data for Classes 2 and 3. The numbers of training and test data are listed in Table 2.8. We trained the L1 support vector machine for the blood cell data and the L2 support vector machine for the thyroid data. For the input range of $[-1, 1]$, we set $C = 5000$ and for $[0, 1]$, we set it appropriately according to Table 2.7. For the polynomial kernels, we changed C for $[0, 1]$ from $4 \times 5000 = 20{,}000$ to $4^d \times 5000$.

Table 2.8. Training and test data for the blood cell and thyroid data

Data	Training data		Test data	
	Class 2	Class 3	Class 2	Class 3
Blood cell	399	400	400	400
Thyroid	191	3488	177	3178

Table 2.9. Solutions of the L1 SVM for the blood cell data

Kernel	Range	PARM	Test rate (%)	Train. rate (%)	SVs	$Q(\boldsymbol{\alpha})$
Linear	$[0, 1]$	C20,000	87.00	90.23	103 (89)	1,875,192
	$[-1, 1]$	C5000	87.00	90.23	103 (89)	1,875,192/4
	$[0, 1]$	C5000	88.50	92.23	101 (52)	331,639
$d2$	$[0, 1]$	C20,000	86.25	94.24	103 (51)	1,191,424
	$[0, 1]$	C80,000	86.75	95.99	96 (34)	4,060,006
	$[-1, 1]$	C5000	86.75	95.49	99 (35)	4,137,900/16
	$[0, 1]$	C5000	88.25	96.24	99 (31)	237,554
	$[0, 1]$	C20,000	86.00	97.49	98 (19)	672,345
$d3$	$[0, 1]$	C80,000	85.75	99.00	97 (4)	1,424,663
	$[0, 1]$	C320,000	86.50	100	93	1,839,633
	$[-1, 1]$	C5000	86.00	100	90 (1)	2,847,139/64
RBF	$[0, 1]$	γ4	89.00	92.48	99 (58)	358,168
	$[-1, 1]$	γ1	89.00	92.48	99 (58)	358,168

Table 2.9 lists the recognition rates of the blood cell test and training data, the number of support vectors, and the value of $Q(\boldsymbol{\alpha})$ for the L1 support vector machine. The numerals in parentheses show the numbers of bounded support vectors. For the linear kernel, as the theory tells us, the solution with the ranges of $[0, 1]$ and $C = 20{,}000$ and that with $[-1, 1]$ and $C = 5000$ are the same. For the RBF kernels also, the solution with $[0, 1]$ and $\gamma = 4$ and that with $[-1, 1]$ and $\gamma = 1$ are the same.

For the polynomial kernel with $d = 2$, the solution with $[0, 1]$ and $C = 4^d \times 5000 = 80{,}000$ and that with $[-1, 1]$ and $C = 5000$ are similar. The value of $Q(\boldsymbol{\alpha})$ with $C = 80{,}000$ is near the value of $Q(\boldsymbol{\alpha}) \times 16$ with $C = 5000$. Similar results hold for $d = 3$, although the difference between the values of $Q(\boldsymbol{\alpha})$ are widened compared to that for $d = 2$.

Table 2.10 lists the recognition rates of the thyroid test and training data, the number of support vectors, and the value of $Q(\boldsymbol{\alpha})$ for the L2 support vector machine.

For the linear and RBF kernels, the solutions with the range of $[0, 1]$ and the associated solutions are the same.

For the polynomial kernels, the solution with $[0, 1]$ and $C = 5000$ and that with $[-1, 1]$ and $4^d \times 5000$ are similar.

Table 2.10. Solutions of the L2 SVM for the thyroid data

Kernel	Range	PARM	Test rate (%)	Train. rate (%)	SVs	$Q(\boldsymbol{\alpha})$
Linear	$[0,1]$	C20,000	97.50	98.34	474	2,096,156
	$[-1,1]$	C5000	97.50	98.34	474	2,096,156/4
$d2$	$[0,1]$	C5000	98.12	99.18	275	298,379
	$[0,1]$	C20,000	98.18	99.29	216	974,821
	$[0,1]$	C80,000	98.21	99.37	191	3,360,907
	$[-1,1]$	C5000	97.85	99.37	201	3,357,314/16
$d3$	$[0,1]$	C5000	97.97	99.40	217	206,335
	$[0,1]$	C20,000	98.15	99.57	168	642,435
	$[0,1]$	C80,000	98.06	99.76	131	1,993,154
	$[0,1]$	C320,000	97.94	99.86	106	5,626,809
	$[-1,1]$	C5000	97.65	99.92	125	4,691,633/64
γ	$[0,1]$	γ4	97.91	97.35	237	3,816,254
	$[-1,1]$	γ1	97.91	97.35	237	3,816,254

Theoretical analysis and the computer experiments showed that the input ranges of $[0,1]$ and $[-1,1]$ are interchangeable for polynomial kernels with constant terms and RBF kernels. Namely, we can use either range. But for polynomial kernels without constant terms, it is unfavorable to use $[-1,1]$ as discussed in Section 2.3.4.

3

Multiclass Support Vector Machines

As discussed in Chapter 2, support vector machines are formulated for two-class problems. But because support vector machines employ direct decision functions, an extension to multiclass problems is not straightforward. There are roughly four types of support vector machines that handle multiclass problems:

1. one-against-all support vector machines,
2. pairwise support vector machines,
3. error-correcting output code (ECOC) support vector machines, and
4. all-at-once support vector machines.

According to Vapnik's formulation [256], in one-against-all support vector machines, an n-class problem is converted into n two-class problems and for the ith two-class problem, class i is separated from the remaining classes. But by this formulation unclassifiable regions exist if we use the discrete decision functions.

To solve this problem, in pairwise support vector machines, Kreßel [140] converts the n-class problem into $n(n-1)/2$ two-class problems, which cover all pairs of classes. But by this method unclassifiable regions also exist.

We can resolve unclassifiable regions by introducing membership functions [10, 116], decision trees [134, 192, 194, 240, 242], or error-correcting output codes [73] or by determining the decision functions all at once [65, 257, 269, 270].

Especially for one-against-all support vector machines, if we use continuous decision functions instead of discrete decision functions, unclassifiable regions are resolved.

In the preceding methods the code words are discrete and fixed before training. But there are some approaches to optimize codes using continuous code words [66, 200].

In the following, we discuss the four types of support vector machines and their variants that resolve unclassifiable regions, and we clarify their relationships, advantages, and disadvantages through theoretical analysis and computer experiments. Specifically, we prove that one-against-all support vector machines with continuous decision functions are equivalent to one-against-all fuzzy support vector machines [5]. We show that generalization ability of decision-tree-based support vector machines [134, 192, 194] depends on their structure and discuss how to optimize their structure [240, 242]. And we clarify the relationship between fuzzy support vector machines and ECOC support vector machines.

3.1 One-against-All Support Vector Machines

In this section, first we discuss a one-against-all support vector machine with discrete decision functions and its problem that unclassifiable regions exist. Then we discuss three methods to solve the problem: one-against-all support vector machines with continuous decision functions, fuzzy support vector machines, and decision-tree-based support vector machines. We show that one-against-all support vector machines with continuous decision functions and fuzzy support vector machines are equivalent.

3.1.1 Conventional Support Vector Machines

Consider an n-class problem. For a one-against-all support vector machine, we determine n direct decision functions that separate one class from the remaining classes. Let the ith decision function, with the maximum margin that separates class i from the remaining classes, be

$$D_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{g}(\mathbf{x}) + b_i, \tag{3.1}$$

where $\mathbf{w}_i$ is the l-dimensional vector, $\mathbf{g}(\mathbf{x})$ is the mapping function that maps $\mathbf{x}$ into the l-dimensional feature space, and b_i is the bias term.

The hyperplane $D_i(\mathbf{x}) = 0$ forms the optimal separating hyperplane, and if the classification problem is separable, the training data belonging to class i satisfy $D_i(\mathbf{x}) \geq 1$ and those belonging to the remaining classes satisfy $D_i(\mathbf{x}) \leq -1$. Especially, support vectors satisfy $y_i\, D_i(\mathbf{x}) = 1$. If the problem is inseparable, unbounded support vectors satisfy $y_i\, D_i(\mathbf{x}) = 1$ and bounded support vectors satisfy $y_i\, D_i(\mathbf{x}) \leq 1$. The remaining training data satisfy $y_i\, D_i(\mathbf{x}) \geq 1$.

In classification, if for the input vector $\mathbf{x}$

$$D_i(\mathbf{x}) > 0 \tag{3.2}$$

is satisfied for one i, $\mathbf{x}$ is classified into class i. Because only the sign of the decision function is used, the decision is discrete.

If (3.2) is satisfied for plural is or if there is no i that satisfies (3.2), $\mathbf{x}$ is unclassifiable. Consider the three-class problem with two-dimensional input as shown in Fig. 3.1, where the arrows show the positive sides of the hyperplanes. For Datum 1, $\mathbf{x}_1$, the three decision functions are

$$D_1(\mathbf{x}_1) > 0, \quad D_2(\mathbf{x}_1) > 0, \quad D_3(\mathbf{x}_1) < 0.$$

Because $\mathbf{x}_1$ belongs to both Classes 1 and 2, $\mathbf{x}_1$ is unclassifiable. Likewise, for Datum 2, $\mathbf{x}_2$, the three decision functions are

$$D_1(\mathbf{x}_2) < 0, \quad D_2(\mathbf{x}_2) < 0, \quad D_3(\mathbf{x}_2) < 0.$$

Thus, $\mathbf{x}_2$ is unclassifiable.

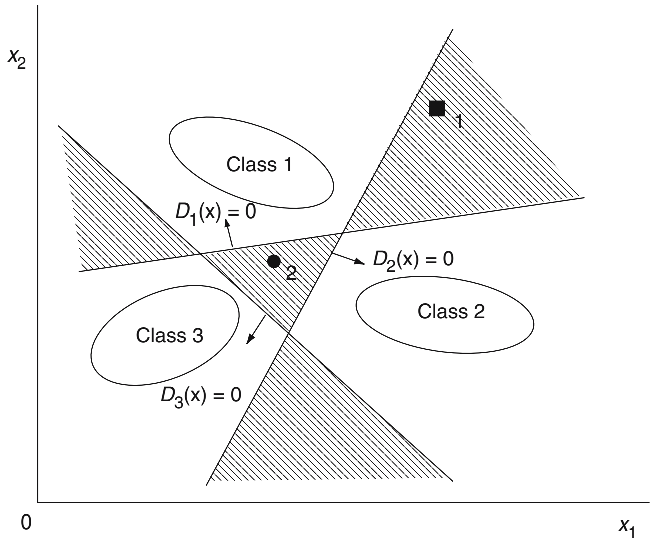


Fig. 3.1. Unclassifiable regions by the one-against-all formulation

To avoid this, instead of discrete decision functions, continuous decision functions are proposed for classification. Namely, datum $\mathbf{x}$ is classified into the class

$$\arg \max_{i=1,\ldots,n} D_i(\mathbf{x}). \tag{3.3}$$

Then Datum 1 in Fig. 3.1 is classified into Class 1 because $D_1(\mathbf{x}_1)$ is the maximum among the three. Likewise, Datum 2 is classified into Class 1.

In Section 3.1.3, the meaning of (3.3) is explained from the membership functions defined in the directions orthogonal to the optimal hyperplanes.

3.1.2 Fuzzy Support Vector Machines

In this section, we introduce membership functions into one-against-all support vector machines to resolve unclassifiable regions, while realizing the same classification results for the data that are classified by conventional one-against-all support vector machines. We introduce two operators: minimum and average operators to define membership functions for classes.

One-Dimensional Membership Functions

For class i we define one-dimensional membership functions $m_{ij}(\mathbf{x})$ in the directions orthogonal to the optimal separating hyperplanes $D_j(\mathbf{x}) = 0$ as follows:

1. For $i = j$

$$m_{ii}(\mathbf{x}) = \begin{cases} 1 & \text{for} \quad D_i(\mathbf{x}) \geq 1, \\ D_i(\mathbf{x}) & \text{otherwise.} \end{cases} \tag{3.4}$$

2. For $i \neq j$

$$m_{ij}(\mathbf{x}) = \begin{cases} 1 & \text{for} \quad D_j(\mathbf{x}) \leq -1, \\ -D_j(\mathbf{x}) & \text{otherwise.} \end{cases} \tag{3.5}$$

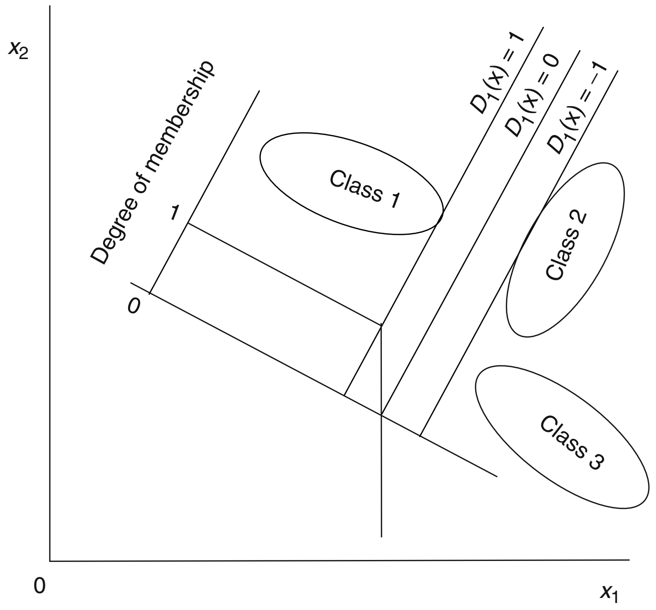


Fig. 3.2. Definition of one-dimensional membership function

Figure 3.2 shows the membership function $m_{11}(\mathbf{x})$ for the two-dimensional input space. Because for a separable classification problem only the class i

training data exist when $D_i(\mathbf{x}) \geq 1$, we assume that the degree of class i membership is 1 for $D_i(\mathbf{x}) \geq 1$ and $D_i(\mathbf{x})$ otherwise. We assume that the same is true even if inseparable. Here, we allow the negative degree of membership so that any data not on the boundary can be classified.

For $i \neq j$, class i is on the negative side of $D_j(\mathbf{x}) = 0$. In this case, support vectors may not include class i data (as is the case with Class 3 in Fig. 3.2), but when $D_j(\mathbf{x}) \leq -1$, we assume that the degree of class i degree of membership is 1, otherwise $-D_j(\mathbf{x})$.

Membership Functions for Classes

We define the class i membership function of $\mathbf{x}$ by the minimum operation for $m_{ij}(\mathbf{x})$ $(j = 1, \ldots, n)$:

$$m_i(\mathbf{x}) = \min_{j=1,\ldots,n} m_{ij}(\mathbf{x}), \tag{3.6}$$

or the average operation:

$$m_i(\mathbf{x}) = \frac{1}{n} \sum_{j=1,\ldots,n} m_{ij}(\mathbf{x}). \tag{3.7}$$

The datum $\mathbf{x}$ is classified into the class

$$\arg \max_{i=1,\ldots,n} m_i(\mathbf{x}). \tag{3.8}$$

Now consider the difference of membership functions given by (3.6) and (3.7). By the definition of $m_{ij}(\mathbf{x})$ given by (3.4) and (3.5), for $\mathbf{x} \in R_i$ where

$$R_i = \{\mathbf{x} \,|\, D_i(\mathbf{x}) > 1, D_j(\mathbf{x}) < -1, \ j \neq i, \ j = 1, \ldots, n\}, \tag{3.9}$$

$m_i(\mathbf{x}) = 1$ for both (3.6) and (3.7). Because $m_i(\mathbf{x}) = 1$ is satisfied for only one i, $\mathbf{x} \in R_i$ is classified into class i. Thus, both membership functions give the same classification result for the data in R_i. Therefore, the difference of the membership functions occurs for $m_i(\mathbf{x}) < 1$. It is shown in Section 3.1.3 that the class boundaries by the support vector machine with the minimum or average operator are the same as those by the one-against-all support vector machine with continuous decision functions.

Figure 3.3 shows the membership functions $m_1(\mathbf{x})$ for the minimum and average operators for two decision functions. For the minimum operator, a contour line, which has the same degree of membership, lies in parallel to the surface of R_1. The membership function with the average operator has a similar shape to that with the minimum operator for the region, where the degree of one of the two one-dimensional functions is 1.

According to the formulation, the unclassifiable regions shown in Fig. 3.1 are resolved as shown in Fig. 3.4. This gives the similar class boundaries proposed by Bennett [29].

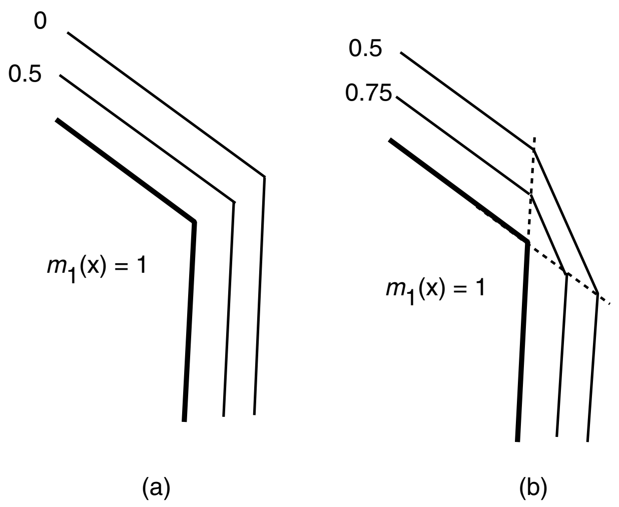


Fig. 3.3. Membership functions: (a) Minimum operator. (b) Average operator

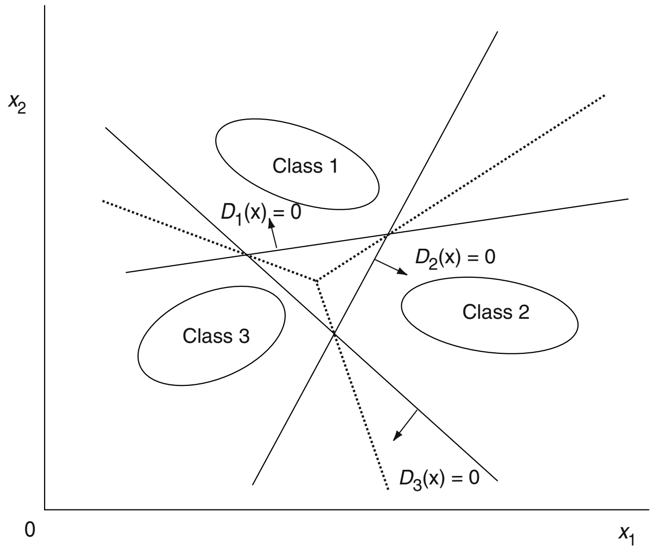


Fig. 3.4. Extended generalization regions

Because the decision boundary between classes i and j is given by $m_i(\mathbf{x}) = m_j(\mathbf{x})$, the decision boundary changes as the output of the decision functions is normalized. Mayoraz and Alpaydin [165] discussed three ways to normalize the outputs. By this normalization, however, the classification results change only for the data in the unclassifiable regions caused by discrete decision functions.

3.1.3 Equivalence of Fuzzy Support Vector Machines and Support Vector Machines with Continuous Decision Functions

Here, we show that one-against-all support vector machines with continuous decision functions and one-against-all fuzzy support vector machines with minimum or average operators are equivalent in that they give the same classification result for the same input [136].

Let $m_i^{\mathrm{m}}(\mathbf{x})$ and $m_i^{\mathrm{a}}(\mathbf{x})$ be the membership functions for class i using the minimum and average operators, respectively.

Then (3.6) and (3.7) are rewritten as follows:

$$m_i^{\mathrm{m}}(\mathbf{x}) = \min\left(\min(1, D_i(\mathbf{x})), \min_{\substack{k\neq i,\\ k=1,\ldots,n}} \min(1, -D_k(\mathbf{x}))\right), \tag{3.10}$$

$$m_i^{\mathrm{a}}(\mathbf{x}) = \frac{1}{n}\left(\min(1, D_i(\mathbf{x})) + \sum_{k=1,k\neq i}^{n} \min(1, -D_k(\mathbf{x}))\right). \tag{3.11}$$

Thus $m_i^{\mathrm{a}}(\mathbf{x}) - m_j^{\mathrm{a}}(\mathbf{x})$ is given by

$$\begin{aligned} m_i^{\mathrm{a}}(\mathbf{x}) - m_j^{\mathrm{a}}(\mathbf{x}) = \frac{1}{n}(&\min(1, D_i(\mathbf{x})) + \min(1, -D_j(\mathbf{x})) \\ &- \min(1, D_j(\mathbf{x})) - \min(1, -D_i(\mathbf{x}))). \end{aligned} \tag{3.12}$$

Now we prove the equivalence classifying the cases into three:

1. $D_i(\mathbf{x}) > 0$, $D_j(\mathbf{x}) \leq 0$ $(j = 1, \ldots, n, j \neq i)$

   By the support vector machine with continuous decision functions, input $\mathbf{x}$ is classified into class i.

   From (3.10) and the conditions on the signs of D_k $(k = 1, \ldots, n)$,

$$m_i^{\mathrm{m}}(\mathbf{x}) > 0, \quad m_j^{\mathrm{m}}(\mathbf{x}) \leq 0. \tag{3.13}$$

   Thus by the fuzzy support vector machine with minimum operators, input $\mathbf{x}$ is classified into class i.

   From (3.12),

$$\begin{aligned} m_i^{\mathrm{a}}(\mathbf{x}) - m_j^{\mathrm{a}}(\mathbf{x}) = \frac{1}{n}\,(&\min(1, D_i(\mathbf{x})) + \min(1, -D_j(\mathbf{x})) \\ &-D_j(\mathbf{x}) + D_i(\mathbf{x})) > 0. \end{aligned} \tag{3.14}$$

Thus by the fuzzy support vector machine with average operators, input $\mathbf{x}$ is classified into class i.

2. $0 > D_i(\mathbf{x}) > D_j(\mathbf{x})$ $(j = 1, \ldots, n, j \neq i)$

By the support vector machine with continuous decision functions, $\mathbf{x}$ is classified into class i.

From (3.10) and the conditions on the signs of $D_k(\mathbf{x})$ $(k = 1, \ldots, n)$,

$$m_i^{\mathrm{m}}(\mathbf{x}) > m_j^{\mathrm{m}}(\mathbf{x}). \tag{3.15}$$

Thus input $\mathbf{x}$ is classified into class i by the fuzzy support vector machine with minimum operators.

From (3.12),

$$\begin{aligned} m_i^{\mathrm{a}}(\mathbf{x}) - m_j^{\mathrm{a}}(\mathbf{x}) = \frac{1}{n}\,&(D_i(\mathbf{x}) - D_j(\mathbf{x}) \\ &- \min(1, -D_i(\mathbf{x})) + \min(1, -D_j(\mathbf{x}))) > 0. \end{aligned} \tag{3.16}$$

Thus input $\mathbf{x}$ is classified into class i by the fuzzy support vector machine with average operators.

3. $D_i(\mathbf{x}) > D_j(\mathbf{x}) > 0 > D_k(\mathbf{x})$, where $j \in N_1$, $k \in N_2$, $N_1 \cap N_2 = \phi$, $(N_1 \cup N_2) \cap \{i\} = \phi$, $N_1 \cup N_2 \cup \{i\} = \{1, \ldots, n\}$

Input $\mathbf{x}$ is classified into class i by the support vector machine with continuous decision functions.

From (3.10),

$$m_i^{\mathrm{m}}(\mathbf{x}) = \min_{j \in N_1} -D_j(\mathbf{x}), \tag{3.17}$$

$$m_j^{\mathrm{m}}(\mathbf{x}) = -D_i(\mathbf{x}) \quad \text{for} \quad j \in N_1, \tag{3.18}$$

$$m_k^{\mathrm{m}}(\mathbf{x}) = \min(-D_i(\mathbf{x}), D_k(\mathbf{x})) \quad \text{for} \quad k \in N_2. \tag{3.19}$$

Thus,

$$m_i^{\mathrm{m}}(\mathbf{x}) > m_j^{\mathrm{m}}(\mathbf{x}) \quad \text{for} \quad j \in N_1 \cup N_2. \tag{3.20}$$

Therefore, $\mathbf{x}$ is classified into class i by the fuzzy support vector machine with minimum operators.

From

$$\begin{aligned} m_i^{\mathrm{a}}(\mathbf{x}) - m_j^{\mathrm{a}}(\mathbf{x}) = \frac{1}{n}\,&(\min(1, D_i(\mathbf{x})) - D_j(\mathbf{x}) \\ - \min(1, D_j(\mathbf{x})) + D_i(\mathbf{x})) > 0 &\quad \text{for} \quad j \in N_1 \end{aligned} \tag{3.21}$$

and from (3.14),

$$m_i^{\mathrm{a}}(\mathbf{x}) > m_j^{\mathrm{a}}(\mathbf{x}) \quad \text{for} \quad j \in N_1 \cup N_2. \tag{3.22}$$

Thus, input $\mathbf{x}$ is classified into class i by the fuzzy support vector machine with average operators.

Therefore, one-against-all support vector machines with continuous decision functions and the fuzzy support vector machines with minimum or average operators are equivalent.

3.1.4 Decision-Tree-Based Support Vector Machines

To resolve unclassifiable regions in one-against-all support vector machines, in this section we discuss decision-tree-based support vector machines [240]. Namely, we train $n-1$ support vector machines; the ith ($i = 1, \ldots, n-1$) support vector machine is trained so that it separates class i data from data belonging to one of classes $i+1, i+2, \ldots, n$. After training, classification is performed from the first to the $(n-1)$st support vector machines. If the ith support vector machine classifies a datum into class i, classification terminates. Otherwise, classification is performed until the datum is classified into the definite class.

Figure 3.5 shows an example of class boundaries for four classes, when linear kernels are used. As seen from the figure, the classes with smaller class numbers have larger class regions. Thus the processing order affects the generalization ability. In some applications, the structure of a decision tree is determined by the relationships of inclusion among classes [125], but in most cases we need to determine the structure. In a usual decision tree, each node separates one set of classes from another set of classes. And to divide the set of classes into two, in [218, 219] the k-means clustering algorithm is used. With $k = 2$, the data in the set are clustered into two clusters. And if the data in one class are clustered into the two clusters, the class data are considered to reside in the cluster with the larger number of data.

In the following, we discuss determining the structure of decision trees using distance measures.

Architecture of Decision Trees

Because the more data are misclassified at the upper node of the decision tree, the worse the classification performance becomes, the classes that are easily separated need to be separated at the upper node of the decision tree. To determine the decision tree, we use the fact that the neighborhood relations of data in the input space are kept in the feature space. We use four types of decision trees as follows:

1. **Type 1 decision tree.** At each node one class is separated from the remaining classes using the Euclidean distance as a separability measure.

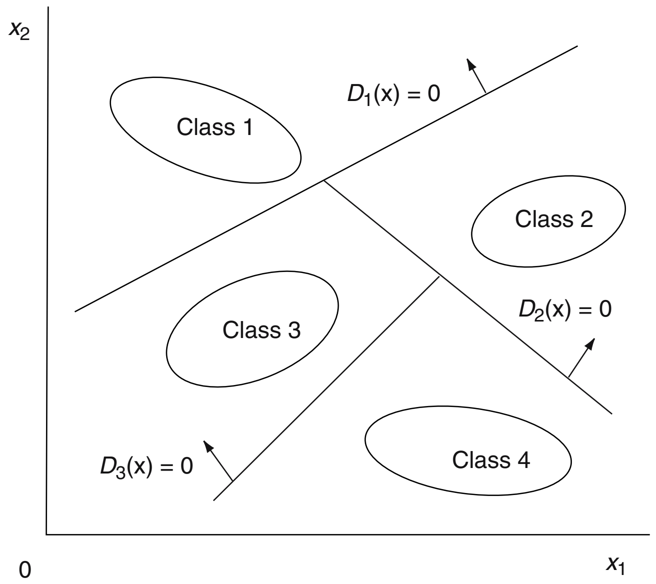


Fig. 3.5. Resolution of unclassifiable regions by decision-tree formulation

2. **Type 2 decision tree.** At each node some classes are separated from the remaining classes using the Euclidean distance as a separability measure.
3. **Type 3 decision tree.** At each node one class is separated from the remaining classes using classification errors by the Mahalanobis distance as a separability measure.
4. **Type 4 decision tree.** At each node some classes are separated from the remaining classes using classification errors by the Mahalanobis distance as a separability measure.

In the following, we discuss these algorithms in detail for an n-class problem.

Type 1 Decision Tree

In this method, we calculate the Euclidean distances between the class centers and recursively separate the farthest class from the remaining classes.

1. Calculate the class centers $\mathbf{c}_i \ (i = 1, \dots, n)$ by

$$\mathbf{c}_i = \frac{1}{|X_i|} \sum_{\mathbf{x} \in X_i} \mathbf{x} \tag{3.23}$$

and the distance between class i and class j, $d_{ij} \ (i, j = 1, \dots, n)$, by

$$d_{ij} \ (= d_{ji}) = \|\mathbf{c}_i - \mathbf{c}_j\|. \tag{3.24}$$

Here X_i is a set of training data included in class i and $|X_i|$ is the number of elements in X_i.

2. Find the smallest value of d_{ij} for class i,

$$l_i = \min_{j \neq i} d_{ij}, \tag{3.25}$$

and regard the class that has the largest l_i as the farthest class and calculate the optimal hyperplane that separates this class. Namely, separate class k from the others. Here $k = \arg\max_i l_i$. If plural ks exist for these classes, compare the next smallest distance l'_i and $k = \arg\max_i l'_i$.
3. If the remaining classes exist, repeat Step 2. Otherwise, terminate the algorithm.

Type 2 Decision Tree

Using the distances between class centers, repeat merging the two nearest classes until two clusters are obtained, and separate the clusters by the optimal hyperplane.

1. Using (3.23), calculate the class centers and the distances between class i and class j, $d_{ij}(i, j = 1, \ldots, n)$ by (3.24). Initially, we assume that all the classes belong to different clusters.
2. For the classes that belong to different clusters, calculate the smallest value of distances by (3.25) and let the associated two classes belong to the same cluster.
3. Repeat Step 2 $(n-2)$ times so that all the clusters merge into two clusters.
4. Calculate the optimal hyperplane that separates the clusters generated in Step 3.
5. If the separated cluster in Step 4 has $n'(> 2)$ classes, regard the classes as belonging to different clusters and repeat Step 2 $(n' - 2)$ times and go to Step 4. If $n' = 2$, calculate the optimal hyperplane that separates the two classes. If no cluster has more than one class, terminate the algorithm.

Type 3 Decision Tree

First, we classify the training data using the Mahalanobis distance and determine the optimal hyperplane that separates the class with the smallest misclassifications from the remaining classes.

1. For each class, calculate the covariance matrix $Q_i \; (i = 1, \ldots, n)$ by

$$Q_i = \frac{1}{|X_i|} \sum_{\mathbf{x} \in X_i} (\mathbf{x} - \mathbf{c}_i)(\mathbf{x} - \mathbf{c}_i)^T, \tag{3.26}$$

where $\mathbf{c}_i$ is the center vector of class i given by (3.23). Calculate the Euclidean distance between class centers using (3.24). For all the data, calculate the Mahalanobis distance $d_i(\mathbf{x})$:

$$d_i{}^2(\mathbf{x}) = (\mathbf{x} - \mathbf{c}_i)^T Q_i{}^{-1} (\mathbf{x} - \mathbf{c}_i) \qquad \text{for} \quad i = 1, \ldots, n \tag{3.27}$$

and classify them to the nearest classes.

2. Let e_{ij} be the number of class i data misclassified into class j.
3. Calculate the number of misclassified data for class i by

$$\sum_{\substack{j \neq i, \\ j=1,\ldots,n}} (e_{ij} + e_{ji})$$

and separate the class that has the smallest value from the others. If plural classes have the same value, separate the class with the farthest Euclidean distance among these classes.

4. If the remaining classes exist, repeat Step 3. Otherwise, terminate the algorithm.

Type 4 Decision Tree

In this method, we first classify the data using the Mahalanobis distance and then repeat merging the two most misclassified classes.

1. Initially, we assume that all classes belong to different clusters. Do Steps 1 and 2 in the Type 3 algorithm.
2. For the two classes in different clusters, find the largest value of e_{ij} and regard classes i and j as belonging to the same cluster. If plural classes have the same value, similar to a Type 2 decision tree, merge the two nearest classes.
3. Repeat Step 2 until the number of clusters becomes two, and calculate a hyperplane that separates these two clusters.
4. If the number of classes in a cluster separated in Step 3, n', is larger than 2, let these classes belong to different clusters and repeat Steps 2 and 3. If $n' = 2$, calculate the optimal hyperplane that separates these two classes. If no cluster includes more than one class, terminate the algorithm.

Performance Evaluation

Because there is not much difference of generalization abilities among Type 1 to Type 4 decision trees, in the following we show only the results for Type 1 decision trees.

We compared the recognition rates of the test data for Type 1 decision trees and those of the conventional one-against-all and fuzzy support vector machines using the data sets listed in Table 1.1. We used the polynomial kernel with degrees 2 to 4. The ranges of the input variables were normalized into $[0, 1]$. We trained the support vector machine by the primal-dual interior-point method combined with the decomposition technique.

We set C = 20,000 for the thyroid data set; C = 10,000 for the MNIST data set; and C = 2000 for the remaining data sets. We used an Athron MP 2000 personal computer.

Table 3.1 shows the recognition rates of the test data for the conventional one-against-all support vector machine, fuzzy support vector machine (FSVM), and Type 1 decision tree. The best recognition rate in a row is shown in bold. Column "Train." on the left lists the training time for SVM and FSVM, and the column on the right lists the training time for the Type 1 decision tree.

Table 3.1. Performance of decision-tree SVMs

Data	Kernel	SVM (%)	FSVM (%)	Train. (s)	Type 1 (%)	Train. (s)
Iris	*d*2	92.00	**94.67**	–	93.33	–
	*d*3	93.33	**94.67**	–	93.33	–
Numeral	*d*2	99.02	99.39	0.5	**99.76**	0.2
	*d*3	98.90	99.51	0.5	**99.76**	0.3
Thyroid	*d*2	95.13	97.20	245	**97.78**	5
	*d*3	95.51	97.40	290	**97.84**	5
	*d*4	95.57	97.58	21	**97.72**	5
Blood cell	*d*2	88.77	**93.03**	24	92.45	9
	*d*3	88.84	**93.10**	23	92.19	8
	*d*4	86.61	**92.68**	22	91.48	7
Hiragana-50	*d*2	95.73	**99.07**	126	97.74	50
	*d*3	96.20	**99.35**	123	98.00	52
	*d*4	96.33	**99.37**	136	98.07	53
Hiragana-105	*d*2	99.99	**100**	530	99.99	247
	*d*3	**100**	**100**	560	**100**	237
Hiragana-13	*d*2	96.25	**99.50**	238	98.17	64
	*d*3	96.09	**99.35**	229	98.28	63
	*d*4	96.12	**99.34**	231	98.44	64
MNIST	*d*2	96.06	**98.17**	2760	97.71	870
	*d*3	96.56	**98.38**	4166	97.74	967

From the table, the recognition rate of the Type 1 decision tree is, in most cases, better than that of the SVM, but except for the numeral and thyroid data sets, it is lower than that of the FSVM, although the difference is small.

The training times of the SVM and the FSVM are the same. From the table, the training time of Type 1 decision tree is usually two to four times shorter than that of the SVM. This is because in training a Type 1 decision tree, the number of training data decreases as training proceeds from the top node to the leaf nodes, but for the SVM all the training data are used to determine n decision functions.

3.2 Pairwise Support Vector Machines

In this section, we discuss pairwise support vector machines and their variants. Pairwise support vector machines reduce the unclassifiable regions that occur for one-against-all support vector machines. But unclassifiable regions still exist. To resolve unclassifiable regions, we discuss fuzzy support vector machines and decision-tree-based support vector machines.

3.2.1 Conventional Support Vector Machines

In pairwise support vector machines, we determine the decision functions for all the combinations of class pairs. In determining a decision function for a class pair, we use the training data for the corresponding two classes. Thus, in each training session, the number of training data is reduced considerably compared to one-against-all support vector machines, which use all the training data. But the number of decision functions is $n(n-1)/2$, compared to n for one-against-all support vector machines, where n is the number of classes.

Let the decision function for class i against class j, with the maximum margin, be

$$D_{ij}(\mathbf{x}) = \mathbf{w}_{ij}^T\,\mathbf{g}(\mathbf{x}) + b_{ij}, \tag{3.28}$$

where $\mathbf{w}_{ij}$ is the l-dimensional vector, $\mathbf{g}(\mathbf{x})$ is a mapping function that maps $\mathbf{x}$ into the l-dimensional feature space, b_{ij} is the bias term, and $D_{ij}(\mathbf{x}) = -D_{ji}(\mathbf{x})$.

The regions

$$R_i = \{\mathbf{x} \,|\, D_{ij}(\mathbf{x}) > 0, j = 1, \ldots, n, j \neq i\} \tag{3.29}$$

do not overlap, and if $\mathbf{x}$ is in R_i, we classify $\mathbf{x}$ into class i. If $\mathbf{x}$ is not in $R_i\ (i = 1, \ldots, n)$, we classify $\mathbf{x}$ by voting. Namely, for the input vector $\mathbf{x}$ we calculate

$$D_i(\mathbf{x}) = \sum_{j \neq i, j=1}^{n} \operatorname{sign}(D_{ij}(\mathbf{x})), \tag{3.30}$$

where

$$\operatorname{sign}(x) = \begin{cases} 1 & \text{for } x \geq 0, \\ -1 & \text{for } x < 0, \end{cases} \tag{3.31}$$

and we classify $\mathbf{x}$ into the class

$$\arg \max_{i=1,\ldots,n} D_i(\mathbf{x}). \tag{3.32}$$

If $\mathbf{x} \in R_i$, $D_i(\mathbf{x}) = n - 1$ and $D_k(\mathbf{x}) < n - 1$ for $k \neq i$. Thus $\mathbf{x}$ is classified into i. But if any of $D_i(\mathbf{x})$ is not $n - 1$, (3.32) may be satisfied for plural is. In this case, $\mathbf{x}$ is unclassifiable. In the shaded region in Fig. 3.6, $D_i(\mathbf{x}) = 0\,(i = 1, 2,$ and $3)$. Thus the shaded region is unclassifiable, although the unclassifiable region is much smaller than that for the one-against-all support vector machine shown in Fig. 3.1.

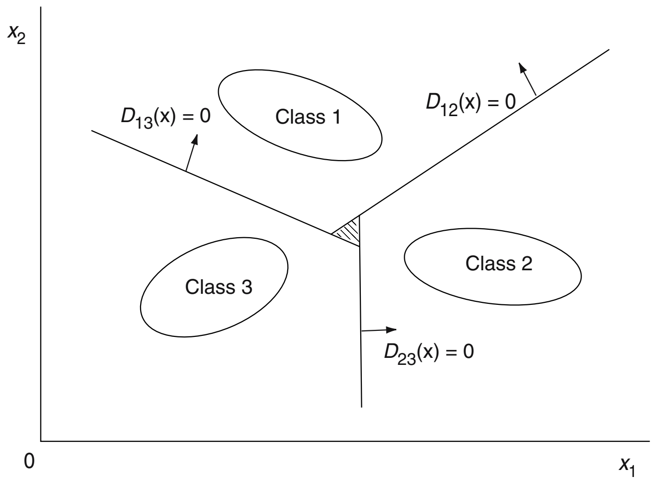


Fig. 3.6. Unclassifiable regions by the pairwise formulation. From [10, p. 115]

3.2.2 Fuzzy Support Vector Machines

Architecture

Similar to the one-against-all formulation, we introduce the membership function to resolve unclassifiable regions while realizing the same classification results with that of the conventional pairwise classification. To do this, for the optimal separating hyperplane $D_{ij}(\mathbf{x}) = 0\,(i \neq j)$ we define one-dimensional membership functions $m_{ij}(\mathbf{x})$ in the directions orthogonal to $D_{ij}(\mathbf{x}) = 0$ as follows:

$$m_{ij}(\mathbf{x}) = \begin{cases} 1 & \text{for} \quad D_{ij}(\mathbf{x}) \geq 1, \\ D_{ij}(\mathbf{x}) & \text{otherwise.} \end{cases} \tag{3.33}$$

We define the class i membership function of $\mathbf{x}$ by the minimum operation for $m_{ij}(\mathbf{x})$ ($j \neq i, j = 1, \ldots, n$):

$$m_i(\mathbf{x}) = \min_{\substack{j \neq i, \\ j=1,\ldots,n}} m_{ij}(\mathbf{x}), \tag{3.34}$$

or the average operation:

$$m_i(\mathbf{x}) = \frac{1}{n-1} \sum_{\substack{j \neq i, \\ j=1}}^{n} m_{ij}(\mathbf{x}). \tag{3.35}$$

Now an unknown datum $\mathbf{x}$ is classified into the class

$$\arg \max_{i=1,\ldots,n} m_i(\mathbf{x}). \tag{3.36}$$

Equation (3.34) is equivalent to

$$m_i(\mathbf{x}) = \min \left(1, \min_{\substack{j \neq i, \\ j=1,\ldots,n}} D_{ij}(\mathbf{x}) \right). \tag{3.37}$$

Because $m_i(\mathbf{x}) = 1$ holds for only one class, classification with the minimum operator is equivalent to classifying $\mathbf{x}$ into the class

$$\arg \max_{i=1,\ldots,n} \min_{\substack{j \neq i, \\ j=1,\ldots,n}} D_{ij}(\mathbf{x}). \tag{3.38}$$

Thus, the unclassifiable region shown in Fig. 3.6 is resolved as shown in Fig. 3.7 for the fuzzy support vector machine with the minimum operator.

3.2.3 Performance Comparison of Fuzzy Support Vector Machines

In this section we compare L1 and L2 support vector machines for one-against-all and pairwise classification for the data sets listed in Table 1.1. We scaled the input ranges into $[0, 1]$. Except for the MNIST data set, we determined the kernels and parameters by five-fold cross-validation. We used linear kernels; polynomial kernels with degrees 2, 3, and 4; and RBF kernels with $\gamma = 0.1, 1,$ and 10. The values of C were selected from 1, 50, 100, 1000, 3000, 5000, 7000, 10,000, and 100,000. We selected the kernel and parameters with the highest recognition rate for the validation set. If the same recognition rate was obtained, we broke the tie by selecting the simplest structure as follows:

1. Select the kernel and parameters with the highest recognition rate for the training data.
2. Select polynomial kernels from polynomial and RBF kernels.

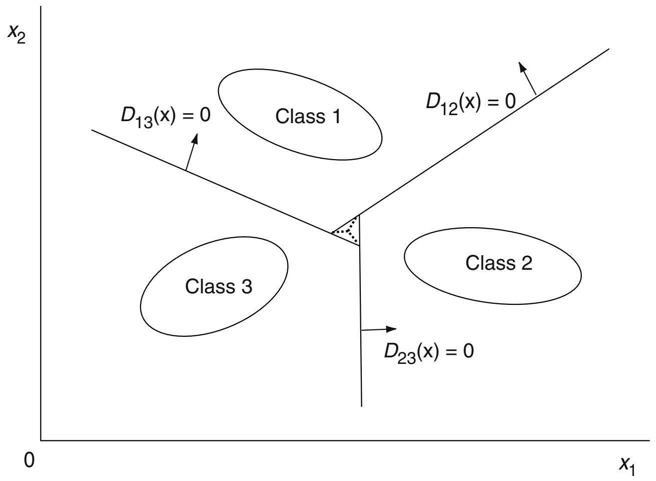


Fig. 3.7. Extended generalization regions. From [10, p. 116]

3. Select the polynomial kernel with the smallest degree from polynomial kernels.
4. Select the RBF kernel with the smallest value of γ from RBF kernels.
5. Select the value with the largest value of C from different values of C.

For the MNIST data set, we set $C = 10{,}000$ and selected the kernel that realized the maximum recognition rate for the test data.

Tables 3.2 and 3.3 list the results for one-against-all L1 and L2 SVMs and pairwise L1 and L2 SVMs. The "Parm" row lists the kernel and the value of C selected. If the values of C selected by cross-validation are different for the minimum and average operators, we only show the value for the minimum operator. The "Dis.," "Min," and "Avg." rows list the recognition rates of the test data, those of the training data in parentheses if lower than 100 percent, with discrete functions, minimum, and average operators, respectively. The "SVs" row lists the number of average support vectors per decision function and the numeral in parentheses shows the number of bounded support vectors for L1 SVMs. The maximum recognition rate of the test data for a data set is shown in bold.

For some cases, such as pairwise SVMs for the iris data, the recognition rates of the discrete SVMs and fuzzy SVMs are the same. This means that no test data were in the unclassifiable regions. Except for these cases, the recognition rates of the fuzzy SVMs are better. As the theory tells us, the recognition rates by minimum operators and average operators are the same for the one-against-all SVMs. In addition, for pairwise SVMs there is not much difference.

Table 3.2. Performance comparison of support vector machines

Data	Item	One-against-all		Pairwise	
		L1 SVM	L2 SVM	L1 SVM	L2 SVM
Iris	Parm	$\gamma 0.1$, $C5000$	$d1$, $C2000$	$\gamma 1$, $C100$	$\gamma 0.1$, $C100$
	Dis.	92.00 (97.33)	69.33 (74.67)	**97.33** (98.67)	**97.33**
	Min	94.67	94.67	**97.33** (98.67)	**97.33**
	Avg.	94.67	94.67	**97.33** (98.67)	**97.33**
	SVs	10 (5)	25	10 (7)	21
Numeral	Parm	$\gamma 1$, $C50$	$d3$, $C1$	$d2$, $C1$	$\gamma 0.1$, $C1000$
	Dis.	99.02 (99.51)	99.15	99.63	**99.76**
	Min	99.27 (99.88)	99.63	99.63	**99.76**
	Avg.	99.27 (99.88)	99.63	99.63	**99.76**
	SVs	15 (3)	47	6	13
Thyroid	Parm	$d4$, $C10^5$	$d4$, $C10^5$	$d1$, $C10^5$	$d3$, $C10^4$
	Dis.	95.97 (99.84)	96.09 (99.89)	97.29 (98.59)	97.67 (99.95)
	Min	**97.93** (99.97)	97.81	97.61 (98.75)	**97.93** (99.95)
	Avg.	**97.93** (99.97)	97.81	97.64 (98.73)	**97.93** (99.95)
	SVs	87 (7)	96	68 (52)	55
Blood cell	Parm	$d2$, $C3000$	$\gamma 10$, $C100$	$d1$, $C50$	$d2$, $C10$
	Dis.	86.87 (94.93)	90.39 (94.80)	91.58 (95.41)	92.87 (96.51)
	Min	93.16 (97.68)	**93.58** (97.19)	92.03 (95.58)	92.97 (96.61)
	Avg.	93.16 (97.68)	**93.58** (97.19)	92.06 (96.03)	92.94 (96.58)
	SVs	92 (29)	188	19 (11)	34
H-50	Parm	$\gamma 10$, $C5000$	$\gamma 10$, $C1000$	$\gamma 10$, $C10^4$	$\gamma 10$, $C10^4$
	Dis.	97.72	97.74	99.00	99.09
	Min	99.26	**99.28**	99.11	99.11
	Avg.	99.26	**99.28**	99.11	99.11
	SVs	71	97	21	21
H-13	Parm	$\gamma 10$, $C3000$	$\gamma 10$, $C500$	$\gamma 10$, $C7000$	$\gamma 10$, $C2000$
	Dis.	98.10	98.68 (99.83)	99.63	99.70
	Min	99.63	99.69 (99.96)	99.74	**99.76**
	Avg.	99.63	99.69 (99.96)	99.70	99.72
	SVs	39 (1)	71	10	11

Table 3.3. Performance comparison of support vector machines

Data	Item	One-against-all		Pairwise	
		L1 SVM	L2 SVM	L1 SVM	L2 SVM
H-105	Parm	$\gamma 10$, $C 10^4$	$\gamma 10$, $C 10^4$	$\gamma 10$, $C 10^4$	$\gamma 10$, $C 10^4$
	Dis.	**100**	**100**	99.93	**100**
	Min	**100**	**100**	99.95	**100**
	Avg.	**100**	**100**	99.94	**100**
	SVs	91	91	13	26
MNIST	Parm	$\gamma 10$, $C 10^4$	$\gamma 10$, $C 10^4$	$\gamma 10$, $C 10^4$	$\gamma 10$, $C 10^4$
	Min	**98.55**	**98.55**	98.32	98.32
	SVs	2287	2295	602	603

Comparing L1 and L2 SVMs, there is not much difference in the recognition rates of the test data, but usually L2 SVMs require more support vectors.

Comparing one-against-all and pairwise SVMs, we can see the following observations:

1. The recognition improvement of pairwise classification by the introduction of membership functions is small. Thus, the unclassifiable regions by pairwise SVMs are smaller than those by one-against-all SVMs.
2. In most cases, the recognition rates of the test data by the pairwise fuzzy SVMs are almost the same as those by the one-against-all SVMs.
3. The number of support vectors of pairwise SVMs is smaller than that of one-against-all SVMs. This is because in pairwise SVMs one class needs to be separated from another class but in one-against-all SVMs, one class needs to be separated from the remaining classes; thus more support vectors are necessary. But the total number of support vectors for the pairwise SVMs may be larger for $n > 3$ because $n\,(n-1)/2$ decision functions are necessary.

3.2.4 Cluster-Based Support Vector Machines

In a one-against-all support vector machine, all the training data are used for training, but for a pairwise support vector machine, training data for two classes are used at a time. Thus for a large problem, a pairwise support vector machine handles much a smaller number of training data than a one-against-all support vector machine. But if the number of data for one class is very large, training becomes prohibitive, even for a pairwise support vector machine. To solve the problem in such a situation, Lu et al. [156] proposed dividing the training data for each class into clusters and determining, like pairwise support vector machines, decision functions for cluster pairs. We call this a *cluster-based support vector machine*. In the following, we discuss the architecture of cluster-based support vector machines with minimum operators.

Assume that class i ($i = 1, \ldots, n$) is divided into N_i clusters and we denote the jth cluster for class i cluster ij. Let the decision function for cluster ij and cluster op be

$$D_{ij-op}(\mathbf{x}) = \mathbf{w}_{ij-op}^T \mathbf{g}(\mathbf{x}) + b_{ij-op}, \tag{3.39}$$

where $\mathbf{g}(\mathbf{x})$ is a mapping function from $\mathbf{x}$ to the l-dimensional feature space, $\mathbf{w}_{ij-op}$ is an l-dimensional vector, b_{ij-op} is a bias term, and $D_{ij-op}(\mathbf{x}) = -D_{op-ij}(\mathbf{x})$. For cluster ij, we determine the decision function $D_{ij}(\mathbf{x})$ using D_{ij-op} ($o \neq i, o = 1, \ldots, n, j = 1, \ldots, N_o$):

$$D_{ij}(\mathbf{x}) = \min_{\substack{o \neq i, o=1,\ldots,n,\\ p=1,\ldots,N_o}} D_{ij-op}(\mathbf{x}). \tag{3.40}$$

Now if $\mathbf{x}$ belongs to cluster ij, $\mathbf{x}$ needs to be classified into class i. Thus, we define the decision function for class i by

$$D_i(\mathbf{x}) = \max_{j=1,\ldots,N_i} D_{ij}(\mathbf{x}). \tag{3.41}$$

Then unknown $\mathbf{x}$ is classified into

$$\arg \max_{i=1,\ldots,n} D_i(\mathbf{x}). \tag{3.42}$$

For the special case where each class consists of one cluster, (3.42) reduces to (3.38), which is equivalent to a pairwise fuzzy support vector machine.

Example 3.1. Consider the case where each of two classes consists of two clusters as shown in Fig. 3.8. Cluster 11 is separated from Clusters 21 and 22 by $D_{11-21}(\mathbf{x}) = 0$ and $D_{11-22}(\mathbf{x}) = 0$, respectively. Thus the region for Cluster 11, where $D_{11}(\mathbf{x}) > 0$, is $\{\mathbf{x} \mid D_{11-21}(\mathbf{x}) > 0, D_{11-22}(\mathbf{x}) > 0\}$. Likewise, the region for Cluster 12, where $D_{12}(\mathbf{x}) > 0$, is $\{\mathbf{x} \mid D_{12-21}(\mathbf{x}) > 0, D_{12-22}(\mathbf{x}) > 0\}$. These regions are disjoint, and if $\mathbf{x}$ is in any of the regions, it is classified into Class 1. On the other hand, as seen from the figure, the regions for Clusters 21 and 22 are overlapped.

Similar to pairwise support vector machines, unclassifiable regions that may appear using discrete decision functions are resolved using the continuous decision functions given by (3.40). But in this example, unclassifiable regions do not exist, even if we use discrete decision functions.

3.2.5 Decision-Tree-Based Support Vector Machines

Similar to decision-tree-based support vector machines discussed for one-against-all formulation, we can formulate decision-tree-based support vector machines for pairwise classification [103, 134, 192, 194]. In this section, we discuss two types of decision-tree-based support vector machines and then we discuss how to optimize structures of decision trees [242].

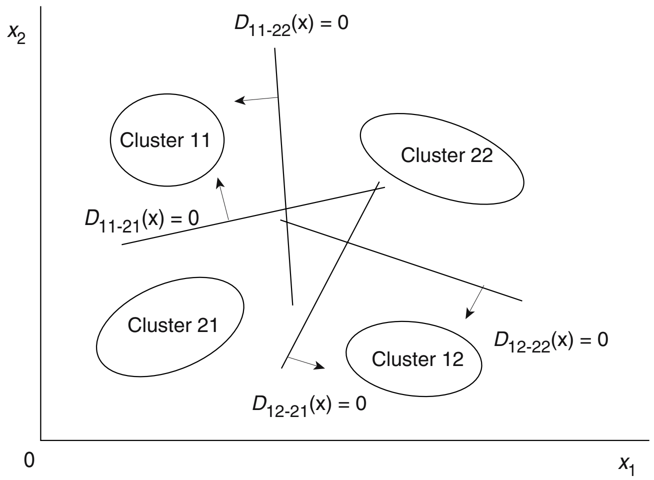

x_2
$D_{11\text{-}22}(x) = 0$
Cluster 11
Cluster 22
$D_{11\text{-}21}(x) = 0$
Cluster 21
Cluster 12
$D_{12\text{-}22}(x) = 0$
$D_{12\text{-}21}(x) = 0$
0
x_1

Fig. 3.8. Decision boundaries by cluster-based classification

Decision Directed Acyclic Graph Support Vector Machines

To resolve unclassifiable regions for pairwise support vector machines, Platt, Cristianini, and Shawe-Taylor [192] proposed decision-tree-based pairwise support vector machines called *decision directed acyclic graph (DDAG) support vector machines*. In the following we call them DDAGs for short. Figure 3.9 shows the decision tree for the three classes shown in Fig. 3.6. In the figure, $\bar{i}$ shows that $\mathbf{x}$ does not belong to class i. As the top-level classification, we can choose any pair of classes. And except for the leaf node if $D_{ij}(\mathbf{x}) > 0$, we consider that $\mathbf{x}$ does not belong to class j, and if $D_{ij}(\mathbf{x}) < 0$ not class i.[1] Thus if $D_{12}(\mathbf{x}) > 0$, $\mathbf{x}$ does not belong to Class 2. Thus it belongs to either Class 1 or Class 3, and the next classification pair is Classes 1 and 3. The generalization regions become as shown in Fig. 3.10. Unclassifiable regions are resolved, but clearly the generalization regions depend on the tree formation.

Figure 3.11 shows a DDAG for four classes. At the top level, Classes 1 and 2 are selected. At the second level, Classes 1 and 4, and 2 and 3, which cover all four classes, are selected. But we can select any pair from Classes 1, 3, and 4 at the left node and from 2, 3, and 4 at the right node. Thus we may select Classes 3 and 4 for both nodes as shown in Fig. 3.12. This is an extension of the DDAG originally defined.

Classification by an original DDAG is executed by list processing. Namely, first we generate a list with class numbers as elements. Then we calculate the decision function, for the input $\mathbf{x}$, corresponding to the first and last elements. Let these classes be i and j and $D_{ij}(\mathbf{x}) > 0$. We delete the element j from

[1]We may resolve the tie $D_{ij}(\mathbf{x}) = 0$ by $D_{ij}(\mathbf{x}) \geq 0$.

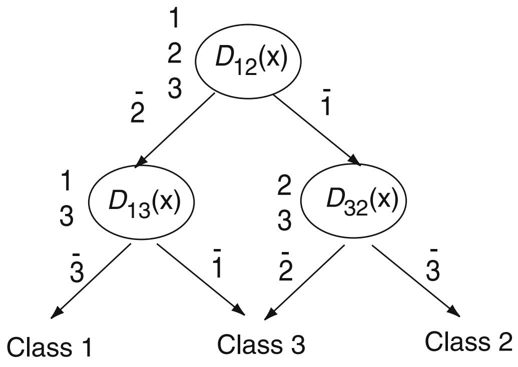


Fig. 3.9. Decision-tree-based pairwise classification

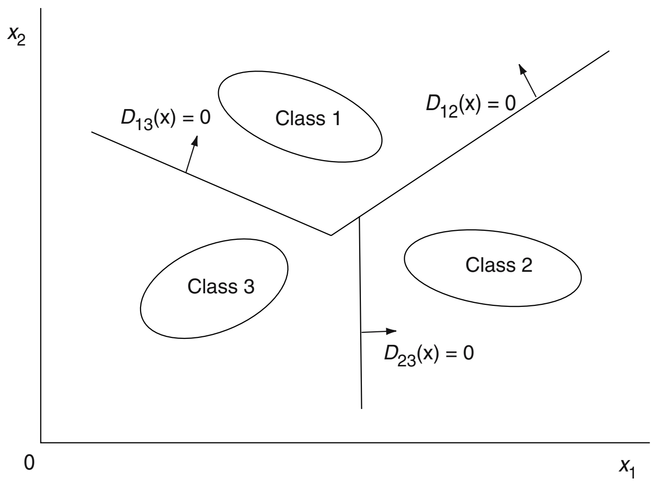


Fig. 3.10. Generalization region by decision-tree-based pairwise classification

the list. We repeat the procedure until one element is left. Then we classify $\mathbf{x}$ into the class that corresponds to the element number. For Fig. 3.11, we generate the list $\{1, 3, 4, 2\}$. If $D_{12}(\mathbf{x}) > 0$, we delete element 2 from the list; we obtain $\{1, 3, 4\}$. Then if $D_{14}(\mathbf{x}) > 0$, we delete element 4 from the list; $\{1, 3\}$. If $D_{13}(\mathbf{x}) > 0$, we delete element 3 from the list. Because only one remains in the list, we classify $\mathbf{x}$ into Class 1.

Training of a DDAG is the same as conventional pairwise support vector machines. Namely, we need to determine $n\,(n-1)/2$ decision functions for an n-class problem. The advantage of DDAGs is that classification is faster than by conventional pairwise support vector machines or pairwise fuzzy support

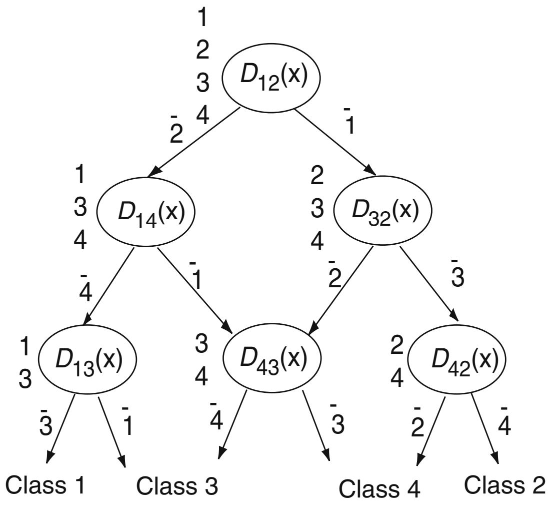


Fig. 3.11. A DDAG for four classes

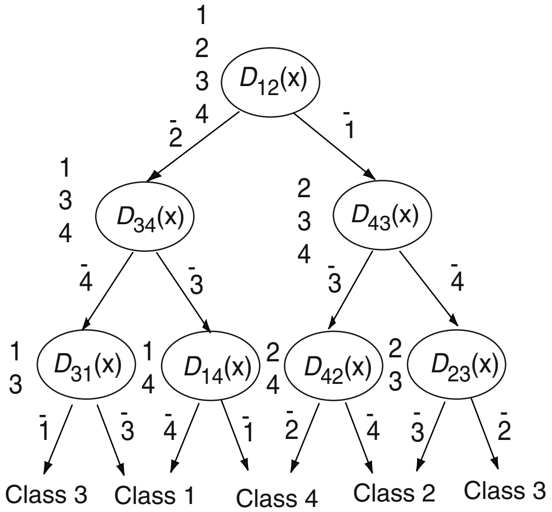


Fig. 3.12. An extended DDAG equivalent to the ADAG shown in Fig. 3.13

vector machines. In a DDAG, classification can be done by calculating $(n-1)$ decision functions.

Adaptive Directed Acyclic Graphs

Pontil and Verri [194] proposed using rules of a tennis tournament to resolve unclassified regions. Not knowing their work, Kijsirikul and Ussivakul [134] proposed the same method and called it *adaptive directed acyclic graph (ADAG)*. For three-class problems, the ADAG is equivalent to the DDAG. Reconsider the example shown in Fig. 3.6. Let the first-round matches be {Class 1, Class 2} and {Class 3}. Then for an input $\mathbf{x}$, in the first match, $\mathbf{x}$ is classified into Class 1 or Class 2, and in the second match $\mathbf{x}$ is classified into Class 3. Then the second-round match is either {Class 1, Class 3} or {Class 2, Class 3} according to the outcome of the first-round match. The resulting generalization regions for classes are the same as those shown in Fig. 3.10. Thus for a three-class problem there are three different ADAGs, each having an equivalent DDAG.

When there are more than three classes, the set of ADAGs is included in the set of extended DDAGs. Consider a four-class problem and let the ADAG be as shown in Fig. 3.13. Namely, the first round matches are {Class 1, Class 2} and {Class 3, Class 4}. An equivalent DDAG is shown in Fig. 3.12. The order of matches {Class 1, Class 2} and {Class 3, Class 4} is irrelevant; namely, they are independent. This can be realized in a DDAG by setting the match {Class 1, Class 2} at all the nodes of one level and the match {Class 3, Class 4} at all the nodes of another level of the tree. Here, we set the match {Class 1, Class 2} at the top of the tree. Thus we set the match {Class 3, Class 4} at the two nodes of the second level. The DDAG obtained by this method is an extension of the original DDAG. In this way, for an ADAG including n classes, we can generate an equivalent DDAG.

However, classification cannot be done by the list processing discussed previously. Namely, for the list $\{1, 3, 4, 2\}$ if $D_{12}(\mathbf{x}) > 0$, we delete element 2 and obtain $\{1, 3, 4\}$. Then we calculate $D_{13}(\mathbf{x})$, but this does not correspond to Fig. 3.12.

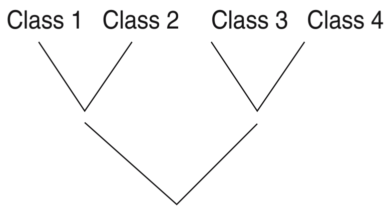


Fig. 3.13. An ADAG for a four-class problem

Any ADAG can be converted to an equivalent DDAG, but the reverse is not true for $n \geq 4$. The DDAG shown in Fig. 3.11 cannot be converted to an ADAG because the second-level decision functions are different and do not constitute matches. But according to the computer simulations [134, 175], classification performance of the two methods is almost identical.

Optimizing Decision Trees

Classification by DDAGs or ADAGs is faster than by pairwise fuzzy support vector machines. But the problem is that the generalization ability depends on the structure of decision trees. To solve this problem for ADAGs, in [189], ADAGs are reordered so that the sum of $\|\mathbf{w}_{ij}\|$ associated with the leaf nodes is minimized. Here we discuss optimizing structures of DDAGs and ADAGs according to [241].

In DDAGs, the unclassifiable regions are assigned to the classes associated with the leaf nodes [241]. By the DDAG for the three-class problem shown in Fig. 3.9, the unclassifiable region is assigned to Class 3, as shown in Fig. 3.10, which corresponds to the leaf node $D_{32}(\mathbf{x})$.

Suppose that the decision boundaries for a four-class problem are given by Fig. 3.14. Then the unclassifiable regions by the conventional pairwise support vector machine are as shown in Fig. 3.15. The shaded regions show the unclassifiable regions, and the thick lines show class boundaries.

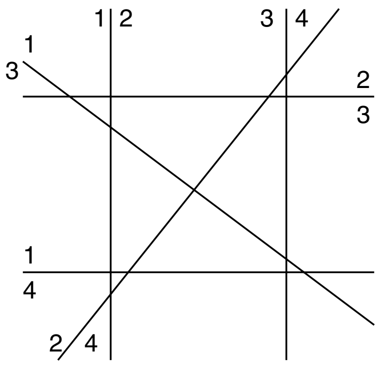


Fig. 3.14. Decision boundaries for a four-class problem. From [242, p. 124]

If the DDAG for the four-class problem is given by Fig. 3.11, the class boundaries are as shown in Fig. 3.16. Region A in the figure is classified into Class 3 by $D_{34}(\mathbf{x})$, which is at a leaf node of the DDAG.

Because any ADAG is converted into a DDAG, the preceding discussions hold for ADAGs. Namely, the unclassifiable regions are assigned to the classes associated with the leaf nodes of the equivalent DDAG.

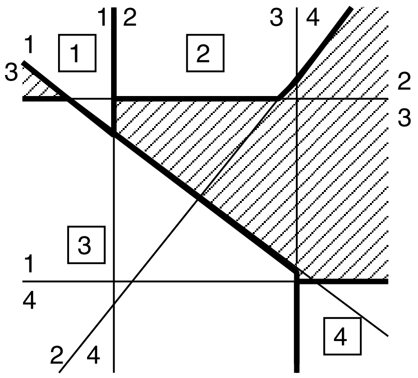


Fig. 3.15. Unclassifiable regions for Fig. 3.14. From [242, p. 125]

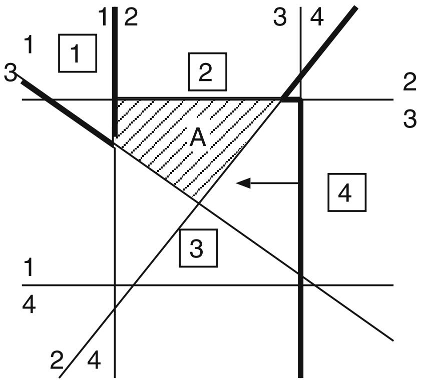


Fig. 3.16. Class regions for Fig. 3.14 using the DDAG given by Fig. 3.11. From [242, p. 125]

The unclassifiable regions caused by the conventional pairwise classification are assigned to the classes associated with the leaf nodes of a DDAG. Thus, if we put the class pairs that are easily separated in the upper nodes, unclassifiable regions are assigned to the classes that are difficult to separate. This means that the class pairs that are difficult to separate are classified by the decision boundaries that are determined by these pairs.

In forming a DDAG or an ADAG, we need to train support vector machines for all pairs of classes. Thus, to determine the optimal structure, we can use any of the measures that are developed for estimating the generalization ability (see Section 2.8.2).

Therefore, the algorithm to determine the DDAG structure for an n-class problem is as follows:

1. Generate the initial list: $\{1, \ldots, n\}$.
2. If there are no generated lists, terminate the algorithm. Otherwise, select a list and select the class pair (i, j) with the highest generalization ability from the list.
3. If the list selected at Step 2 has more than two elements, generate two lists deleting i or j from the list. Go to Step 2.

Figure 3.17 shows an example of a four-class problem. First we generate the list $\{1, 2, 3, 4\}$. At the top level we select a pair of classes that has the highest generalization ability from Classes 1 to 4. Let them be Classes 1 and 2. Then we generate the two lists $\{2, 3, 4\}$ and $\{1, 3, 4\}$. We iterate this procedure for the two lists.

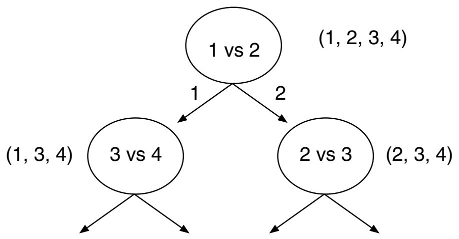


Fig. 3.17. Determination of a DDAG for a four-class problem. From [242, p. 126]

This procedure determines the structure off-line. We can determine the structure while classifying $\mathbf{x}$ as follows:

1. Generate the initial list: $\{1, \ldots, n\}$.
2. Select the class pair (i, j) with the highest generalization ability from the list. Let $\mathbf{x}$ be on the Class i side of the decision function. Delete j from the list. Otherwise, delete i.
3. If the list selected at Step 2 has more than one element, go to Step 2. Otherwise, classify $\mathbf{x}$ into the class associated with the element and terminate the algorithm.

Because any ADAG is converted into an equivalent DDAG, we can determine the tree structure selecting class pairs with the highest generalization ability. Figure 3.18 shows an example of an eight-class problem. At the first level, we select the class pair with the highest generalization ability; let them be Classes 1 and 4. We iterate the procedure for the remaining classes and let the class pairs be Classes 6 and 7, 2 and 5, and 3 and 8. In the second level, the pairs of classes are determined according to the input $\mathbf{x}$. Let the candidate classes be 1, 6, 5, and 3 for $\mathbf{x}$. Then, we choose the pair of classes with the highest generalization ability; let it be (1, 5). Then the remaining pair is (3, 6). We iterate the procedure until $\mathbf{x}$ is classified into a definite class.

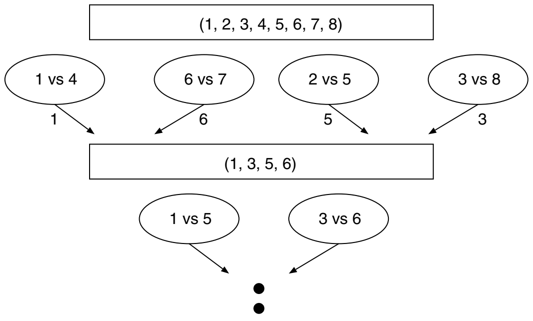


Fig. 3.18. Determination of an ADAG for a four-class problem. From [242, p. 126]

Performance Evaluation

Because the generalization abilities of the ADAGs and DDAGs do not differ very much, in the following we show only the results for DDAGs. As the measure for estimating the generalization ability, we evaluated the VC dimension given by (2.260), the LOO error estimator given by (2.261), and Joachims' $\xi\,\alpha$ LOO estimator given by (2.262), but there was not much difference [242]. Therefore, in the following we shows the results using the LOO error estimator given by (2.261), namely the number of support vectors divided by the number of training data.

We compared the maximum, minimum, and average recognition rates of the test data for DDAGs and the recognition rate of the pairwise fuzzy support vector machine with minimum operators, using the data sets listed in Table 1.1. We used the polynomial kernel with degree 3 and RBF kernel with $\gamma = 1$. The input range was normalized into $[0, 1]$. We trained support vector machines by the primal-dual interior-point method combined with the decomposition technique. For the thyroid and MNIST data sets we set $C = 10{,}000$ and for other data sets $C = 1000$. We used an Athron MP 2000 personal computer.

Table 3.4 shows the recognition rates of the test data for the conventional pairwise support vector machine (SVM), pairwise fuzzy support vector machine (FSVM), and DDAGs. Column "OPT" lists the recognition rate for the optimum DDAG. To compare the recognition rate of the DDAGs, the maximum, minimum, and average recognition rates for the DDAGs are also listed. In column "Kernel," for instance, $d3$ denotes the polynomial kernel with degree 3 and $\gamma 1$ denotes the RBF kernel with $\gamma = 1$.

The number of DDAGs for a three-class problem is 3, but it explodes as n increases. Thus, except for the iris and thyroid data, we randomly generated 10,000 DDAGs and calculated the maximum, minimum, and average recognition rates of the test data.

From the table, for the iris data, all the recognition rates are the same; this means that there are no test data in unclassifiable regions. The recognition rate of the optimum DDAG is comparable to the average recognition rate of DDAGs; in 9 cases out of 16, the recognition rates of the optimum DDAGs are better than or equal to the average recognition rates.

The recognition rates of FSVMs are equal to or better than the average recognition rates of DDAGs and comparable to the maximum recognition rates of DDAGs.

Table 3.4. Performance of pairwise SVMs

Data	Kernel	SVM	FSVM	DDAG			
				Max.	Min.	Ave	OPT
		(%)	(%)	(%)	(%)	(%)	(%)
Iris	$d3$	93.33	93.33	93.33	93.33	93.33	93.33
	$\gamma1$	97.33	97.33	97.33	97.33	97.33	97.33
Numeral	$d3$	99.76	100	100	99.76	99.84	99.76
	$\gamma1$	99.63	99.63	99.88	99.63	99.72	99.88
Thyroid	$d3$	97.81	97.87	97.89	97.81	97.86	97.81
	$\gamma1$	97.26	97.37	97.40	97.26	97.34	97.26
Blood cell	$d3$	92.07	92.77	93.00	92.32	92.47	92.65
	$\gamma1$	91.93	92.23	92.41	91.84	92.08	92.00
Hiragana-50	$d3$	98.57	98.89	99.37	98.52	98.75	98.61
	$\gamma1$	98.37	98.85	99.22	98.29	98.56	98.50
Hiragana-105	$d3$	100	100	100	100	100	100
	$\gamma1$	99.98	100	100	99.98	99.99	99.99
Hiragana-13	$d3$	99.63	99.66	99.72	99.58	99.64	99.66
	$\gamma1$	99.61	99.65	99.70	99.55	99.60	99.63
MNIST	$d3$	97.85	98.01	97.99	97.91	97.96	97.91
	$\gamma1$	97.18	97.78	97.49	97.31	97.41	97.54

For pairwise classification, training time is the same for fuzzy SVMs and DDAGs, but classification time of DDAGs is faster. Table 3.5 lists classification time of the blood cell and hiragna-50 test data sets. DDAGs are much faster than SVMs and FSVMs.

Table 3.5. Classification time comparison

Data	Kernel	DDAG (s)	SVM (s)	FSVM (s)
Blood cell	$d3$	0.9	1.2	2.4
	$\gamma 1$	0.7	2.6	5.2
Hiragana-50	$d3$	15	66	114
	$\gamma 1$	17	76	159

3.2.6 Pairwise Classification with Correcting Classifiers

In pairwise classification, let the classifier for classes i and j generate the estimate of the probability that the input $\mathbf{x}$ belongs to class i, $p_{ij}(\mathbf{x})$, where $p_{ji}(\mathbf{x}) = 1 - p_{ij}(\mathbf{x})$. Then we can estimate the probability that $\mathbf{x}$ belongs to class i, $p_i(\mathbf{x})$, by

$$p_i(\mathbf{x}) = \frac{2}{n\,(n-1)} \sum_{\substack{j \neq i,\\ j=1}}^{n} p_{ij}(\mathbf{x}), \tag{3.43}$$

where $2/(n\,(n-1))$ is to ensure $\sum_{i=1,\ldots,n} p_i(\mathbf{x}) = 1$, and we can classify $\mathbf{x}$ into

$$\arg\max_i p_i(\mathbf{x}). \tag{3.44}$$

This classification method is called *pairwise coupling classification* (Hastie and Tibshirani [107]). Because $p_i(\mathbf{x})$ is continuous, unclassifiable regions do not occur.

In pairwise coupling classification, $p_{ij}(\mathbf{x})$ is estimated using the training data belonging to classes i and j. Thus if $\mathbf{x}$ belongs to class k, a large value of $p_{ij}(\mathbf{x})$ or $p_{ji}(\mathbf{x})$ $(k \neq i,\, j)$ may mislead classification. To avoid this situation, Moreira and Mayoraz [169] introduced correcting classifiers. In addition to a pairwise classifier that separates class i from class j, we generate a classifier that separates classes i and j from the remaining classes. Let the output of the classifier be $q_{ij}(\mathbf{x})$, where $q_{ij}(\mathbf{x})$ is the probability that $\mathbf{x}$ belongs to class i or j. Then we calculate $p_i(\mathbf{x})$ by

$$p_i(\mathbf{x}) = \frac{2}{n\,(n-1)} \sum_{\substack{j \neq i,\\ j=1}}^{n} p_{ij}(\mathbf{x})\, q_{ij}(\mathbf{x}). \tag{3.45}$$

By multiplying $p_{ij}(\mathbf{x})$ by $q_{ij}(\mathbf{x})$, $p_i(\mathbf{x})$ is decreased when $\mathbf{x}$ does not belong to class i. Thus the problem with pairwise coupling classification can be avoided.

According to the computer experiments [169], the generalization ability of pairwise coupling with correcting classifiers was better than pairwise coupling. In addition, if correcting classifiers alone were used as classifiers, their performance was comparable with that of pairwise coupling with correcting classifiers.

In correcting classifiers, there are ${}_nC_2$ decision functions. For three-class classification, separating two classes from the remaining class is equivalent to separating one class from the remaining classes, namely one-against-all classification. The target values for four classes are shown in a matrix form as follows:

$$\left|\begin{array}{rrrr} 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \end{array}\right|, \tag{3.46}$$

where the ith row corresponds to the ith decision function and jth column, class j. Each decision function has a complementary counterpart. For example, the first and sixth rows are complementary. But for $n > 4$, each decision function is distinct. Let the target value of class i for the jth decision function be g_{ij} and the output of the jth decision function be $D_j(\mathbf{x})$. Then input $\mathbf{x}$ is classified into class

$$\arg\max_i \sum_{j=1}^{{}_nC_2} a(i,j), \tag{3.47}$$

where

$$a(i,j) = \begin{cases} D_j(\mathbf{x}) & \text{for } g_{ij} = 1, \\ 1 - D_j(\mathbf{x}) & \text{for } g_{ij} = -1. \end{cases} \tag{3.48}$$

To apply pairwise coupling to support vector machines we need to calculate $p_{ij}(\mathbf{x})$. In [107], $p_{ij}(\mathbf{x})$ and $p_{ji}(\mathbf{x})$ are determined by approximating the normal distributions in the direction orthogonal to the decision function. In [148], in addition to training support vector machines, sigmoid functions are trained [191] (see Section 4.7).

Pairwise coupling support vector machines and pairwise fuzzy support vector machines do not differ very much, either posteriori probabilities or degrees of membership are used. Thus, the idea of correcting classifiers can be readily introduced to pairwise fuzzy support vector machines. But we need to train correcting classifiers using all the training data. This leads to long training times compared to pairwise classification.

3.3 Error-Correcting Output Codes

Error-correcting codes, which detect and correct errors in data transmission channels, are used to encode classifier outputs to improve generalization ability. The codes are called *error-correcting output codes (ECOCs).* For support vector machines, in addition to generalization improvement they can be used to resolve unclassifiable regions. In this section, we first discuss how error-correcting codes can be used for pattern classification. Next, by introducing "don't care" outputs, we discuss a unified scheme for output coding that includes one-against-all and pairwise formulations. Then we show the equivalence of the error-correcting codes with the membership functions. Finally, we compare performance of ECOC support vector machines with one-against-all support vector machines.

3.3.1 Output Coding by Error-Correcting Codes

Dietterich and Bakiri [73] proposed using error-correcting output codes for multiclass problems. Let g_{ij} be the target value of the jth decision function $D_j(\mathbf{x})$ for class i $(i = 1, \ldots, n)$:

$$g_{ij} = \begin{cases} 1 & \text{if } D_j(\mathbf{x}) > 0 \text{ for class } i, \\ -1 & \text{otherwise.} \end{cases} \tag{3.49}$$

The jth column vector $\mathbf{g}_j = (g_{1j}, \ldots, g_{nj})^T$ is the target vector for the jth decision function. If all the elements of a column are 1 or -1, classification is not performed by this decision function and two column vectors with $\mathbf{g}_i = -\mathbf{g}_j$ result in the same decision function. Thus the maximum number of distinct decision functions is $2^{n-1} - 1$.

The ith row vector $(g_{i1}, \ldots, g_{ik})$ corresponds to a code word for class i, where k is the number of decision functions. In error-correcting codes, if the minimum Hamming distance between pairs of code words is h, the code can correct at least $\lfloor (h-1)/2 \rfloor$-bit errors. For three-class problems, there are three decision functions in maximum as shown in Table 3.6, which is equivalent to one-against-all formulation, and there is no error-correcting capability. Thus ECOC is considered to be a variant of one-against-all classification.

Table 3.6. Error-correcting codes for three classes (one-against-all)

Class	$\mathbf{g}_1$	$\mathbf{g}_2$	$\mathbf{g}_3$
1	1	-1	-1
2	-1	1	-1
3	-1	-1	1

3.3.2 Unified Scheme for Output Coding

Introducing "don't care" outputs, Allwein, Schapire, and Singer [14] unified output codes that include one-against-all, pairwise, and ECOC schemes. Denoting a "don't care" output by 0, pairwise classification for three classes can be shown as in Table 3.7.

Table 3.7. Extended error-correcting codes for pairwise classification with three classes

Class	$\mathbf{g}_1$	$\mathbf{g}_2$	$\mathbf{g}_3$
1	1	0	−1
2	−1	1	0
3	0	−1	1

To calculate the distance of $\mathbf{x}$ from the jth decision function for class i, we define the error $\varepsilon_{ij}(\mathbf{x})$ by

$$\varepsilon_{ij}(\mathbf{x}) = \begin{cases} 0 & \text{for } g_{ij} = 0, \\ \max(1 - g_{ij} D_j(\mathbf{x}), 0) & \text{otherwise.} \end{cases} \tag{3.50}$$

If $g_{ij} = 0$, we need to skip this case. Thus, $\varepsilon_{ij}(\mathbf{x}) = 0$. If $g_{ij} D_j(\mathbf{x}) \geq 1$, $\mathbf{x}$ is on the correct side of the jth decision function with more than or the same maximum margin. Thus, $\varepsilon_{ij}(\mathbf{x}) = 0$. If $g_{ij} D_j(\mathbf{x}) < 1$, $\mathbf{x}$ is on the wrong side or even if it is on the correct side, the margin is smaller than the maximum margin. We evaluate this disparity by $1 - g_{ij} D_i(\mathbf{x})$.

Then the distance of $\mathbf{x}$ from class i is given by

$$d_i(\mathbf{x}) = \sum_{j=1}^{k} \varepsilon_{ij}(\mathbf{x}). \tag{3.51}$$

Using (3.51), $\mathbf{x}$ is classified into

$$\arg \min_{i=1,\ldots,n} d_i(\mathbf{x}). \tag{3.52}$$

Instead of (3.50), if we use the discrete function:

$$\varepsilon_{ij}(\mathbf{x}) = \begin{cases} 0 & \text{for } g_{ij} = 0, \\ 0 & \text{for } g_{ij} = \pm 1,\ g_{ij} D_i(\mathbf{x}) \geq 1, \\ 1 & \text{otherwise;} \end{cases} \tag{3.53}$$

(3.51) gives the Hamming distance. But by this formulation, as seen in Sections 3.1 and 3.2, unclassifiable regions occur.

3.3.3 Equivalence of ECOC with Membership Functions

Here we discuss the relationship between ECOC and membership functions. For $g_{ij} = \pm 1$, the error $\varepsilon_{ij}(\mathbf{x})$ is expressed by the one-dimensional membership functions $m_{ij}(\mathbf{x})$:

$$\begin{aligned} m_{ij}(\mathbf{x}) &= \min(g_{ij} D_j(\mathbf{x}), 1) \\ &= 1 - \varepsilon_{ij}(\mathbf{x}). \end{aligned} \tag{3.54}$$

Thus, if we define the membership function for class i by

$$m_i(\mathbf{x}) = \frac{1}{\sum_{j=1}^{k} |g_{ij}|} \sum_{\substack{g_{ij} \neq 0, \\ j=1}}^{k} m_{ij}(\mathbf{x}) \tag{3.55}$$

and classify $\mathbf{x}$ into

$$\arg \max_{i=1,\ldots,n} m_i(\mathbf{x}), \tag{3.56}$$

we obtain the same recognition result as that by (3.52). This is equivalent to a fuzzy support vector machine with average operators.

Similarly, if, instead of (3.51) we use

$$d_i(\mathbf{x}) = \max_{j=1,\ldots,n} \varepsilon_{ij}(\mathbf{x}), \tag{3.57}$$

the resulting classifier is equivalent to a fuzzy support vector machine with minimum operators.

According to the discussions in Section 3.1.3, one-against-all fuzzy support vector machines with average and minimum operators give the same decision boundaries. For pairwise classification, they are different, but according to Section 3.2 the difference is small.

3.3.4 Performance Evaluation

We compared recognition performance of ECOC support vector machines with one-against-all support vector machines using the blood cell data and hiragana-50 data listed in Table 1.1 [135, 136]. As error-correcting codes we used the BCH (Bose-Chaudhuri-Hochquenghem) codes, which belong to one type of cyclic codes. We used four BCH codes with 15, 31, 63, and 127 word lengths, properly setting the minimum Hamming distances. For each word length we randomly assigned the class labels to generated codes 10 times, and using the assigned codes we trained 10 ECOC support vector machines with $C = 5000$.

Table 3.8 shows the results for the blood cell data with polynomial kernels with degree 3. In the "Code" column, e.g., (15, 5, 7) means that the word length is 15 bits, the number of information bits is 5, and the minimum Hamming distance is 7. The "Hamming," "Average," and "Minimum" columns list the average recognition rates of the test and the training data (in parentheses) using the Hamming distance, the average operator, and the minimum operator, respectively. The numeral in boldface shows the maximum recognition rate among the different codes.

Table 3.8. Performance of blood cell data with polynomial kernels ($d = 3$)

Code	Hamming (%)	Average (%)	Minimum (%)
One-against-all	87.13 (92.41)	**92.84** (96.09)	**92.84** (96.09)
(15, 5, 7)	90.17 (93.34)	91.56 (94.45)	91.19 (93.95)
(31, 11, 11)	90.86 (93.60)	91.90 (94.59)	91.80 (94.16)
(63, 7, 31)	**91.82** (94.64)	92.20 (94.98)	92.23 (94.32)
(127, 8, 63)	91.80 (94.58)	92.01 (94.82)	91.93 (96.09)

From the table, using the Hamming distance, the recognition rates of both training and test data improved as the word length was increased, and they reached the maximum at the word length of 63. But because by the Hamming distance unclassifiable regions existed, the recognition rates were lower than by average and minimum operators. By the average and minimum operators, however, the one-against-all support vector machines showed the best recognition rates. This may be caused by the lower recognition rates of the training data by the ECOC support vector machines than by the one-against-all support vector machines.

Thus, to improve the recognition rate of the test data, we used the RBF kernels. Table 3.9 shows the results for the RBF kernels with $\gamma = 1$. The ECOC support vector machines showed better recognition performance than the one-against-all support vector machines. In addition, the average operator showed better recognition performance than the minimum operator.

Table 3.10 shows the results of the hiragana-50 data for the polynomial kernels with degree 3. The ECOC support vector machine with the average operator and the word length of 127 showed the best recognition performance.

Table 3.9. Performance of blood cell data with RBF kernels ($\gamma = 1$)

Code	Hamming (%)	Average (%)	Minimum (%)
One-against-all	86.68 (98.58)	92.94 (99.29)	92.94 (99.29)
(15, 5, 7)	92.43 (98.27)	93.47 (98.49)	93.07 (98.18)
(31, 11, 11)	92.88 (98.36)	93.85 (98.59)	93.53 (98.13)
(63, 10, 27)	**93.68** (98.64)	**94.05** (98.68)	**93.75** (98.37)
(127, 15, 55)	**93.68** (98.60)	93.96 (98.61)	93.63 (97.94)

Table 3.10. Performance of hiragana-50 data with polynomial kernels ($d = 3$)

Code	Hamming (%)	Average (%)	Minimum (%)
One-against-all	97.55 (100)	99.28 (100)	**99.28** (100)
(15, 5, 7)	95.50 (99.96)	97.63 (99.85)	96.93 (99.97)
(31, 11, 11)	98.38 (99.99)	99.01 (100)	98.56 (99.97)
(63, 7, 31)	99.01 (100)	99.30 (100)	99.17 (99.97)
(127, 8, 63)	**99.31** (100)	**99.46** (100)	99.26 (99.97)

But some ECOC support vector machines showed lower recognition performance than the one-against-all support vector machine. Thus the performance of ECOC support vector machines was not stable.

From the preceding computer experiments, it is shown that ECOC support vector machines do not always perform better than one-against-all support vector machines. Thus, to obtain good recognition performance, we need to optimize the structure of ECOC support vector machines. To do this, Pérez-Cruz and Artés-Rodríguez [186] proposed an iterative pruning of worst classifier from redundant classifiers.

3.4 All-at-Once Support Vector Machines

In this section, we resolve unclassifiable regions of multiclass problems by determining all the decision functions simultaneously [20, 29, 36, 40, 65, 100, 257, 269, 270]. In [77, pp. 174–6], a multiclass problem is converted into a two-class problem by expanding the m-dimensional input data into $(m \times n)$-dimensional data.[2] Here, we do not use this method. We discuss two methods: the basic architecture that uses $(M \times n)$ slack variables [257] and the sophisticated architecture that uses M slack variables [65].

3.4.1 Basic Architecture

For an n-class problem, we define the decision function for class i by

$$\mathbf{w}_i^T\, \mathbf{g}(\mathbf{x}) + b_i > \mathbf{w}_j^T\, \mathbf{g}(\mathbf{x}) + b_j \qquad \text{for} \quad j \neq i, j = 1, \ldots, n, \tag{3.58}$$

where $\mathbf{w}_i$ is the weight vector for class i in the feature space, $\mathbf{g}(\mathbf{x})$ is the mapping function, and b_i is the bias term.

The L1 soft-margin support vector machine can be obtained by minimizing

[2] A similar method is discussed in [19].

$$Q(\mathbf{w}, \mathbf{b}, \boldsymbol{\xi}) = \frac{1}{2} \sum_{i=1}^{n} \|\mathbf{w}_i\|^2 + C \sum_{i=1}^{M} \sum_{\substack{j \neq y_i, \\ j=1}}^{n} \xi_{ij} \tag{3.59}$$

subject to the constraints

$$(\mathbf{w}_{y_i} - \mathbf{w}_j)^T \mathbf{g}(\mathbf{x}_i) + b_{y_i} - b_j \geq 1 - \xi_{ij}$$
$$\text{for} \quad j \neq y_i, j = 1, \ldots, n, i = 1, \ldots, M, \tag{3.60}$$

where $y_i \, (\in \{1, \ldots, n\})$ is the class label for $\mathbf{x}_i$, C is the margin parameter that determines the trade-off between the maximization of the margin and minimization of the classification error, ξ_{ij} (> 0) is the slack variable associated with $\mathbf{x}_i$ and class j, $\boldsymbol{\xi} = (\ldots, \xi_{ij}, \ldots)^T$, $\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_n)^T$, and $\mathbf{b} = (b_1, \ldots, b_n)^T$.

Introducing the nonnegative Lagrange multipliers α_{ij} and β_{ij}, we obtain

$$\begin{aligned}
Q(\mathbf{w}, \mathbf{b}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= \frac{1}{2} \sum_{i=1}^{n} \|\mathbf{w}_i\|^2 + C \sum_{i=1}^{M} \sum_{\substack{j \neq y_i, \\ j=1}}^{n} \xi_{ij} \\
&\quad - \sum_{i=1}^{M} \sum_{\substack{j \neq y_i, \\ j=1}}^{n} \alpha_{ij} \, \left((\mathbf{w}_{y_i} - \mathbf{w}_j)^T \mathbf{g}(\mathbf{x}_i) + b_{y_i} - b_j - 1 + \xi_{ij} \right) \\
&\quad - \sum_{i=1}^{M} \sum_{\substack{j \neq y_i, \\ j=1}}^{n} \beta_{ij} \, \xi_{ij} \\
&= \frac{1}{2} \sum_{i=1}^{n} \|\mathbf{w}_i\|^2 - \sum_{i=1}^{M} \sum_{j=1}^{n} z_{ij} \, \left(\mathbf{w}_j^T \mathbf{g}(\mathbf{x}_i) + b_j - 1 \right) \\
&\quad - \sum_{i=1}^{M} \sum_{\substack{j \neq y_i, \\ j=1}}^{n} (\alpha_{ij} + \beta_{ij} - C) \, \xi_{ij},
\end{aligned} \tag{3.61}$$

where

$$z_{ij} = \begin{cases} \displaystyle\sum_{\substack{k \neq y_i, \\ k=1}}^{n} \alpha_{ik} & \text{for } j = y_i, \\ -\alpha_{ij} & \text{otherwise.} \end{cases} \tag{3.62}$$

The conditions of optimality are given by

$$\frac{\partial Q(\mathbf{w}, \mathbf{b}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \mathbf{b}} = \mathbf{0}, \tag{3.63}$$

$$\frac{\partial Q(\mathbf{w}, \mathbf{b}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \mathbf{w}} = \mathbf{0}, \tag{3.64}$$

$$\frac{\partial Q(\mathbf{w}, \mathbf{b}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \boldsymbol{\xi}} = \mathbf{0}, \tag{3.65}$$

$$\alpha_{ij}\ \left((\mathbf{w}_{y_i}-\mathbf{w}_j)^T\mathbf{g}(\mathbf{x}_i)+b_{y_i}-b_j-1+\xi_{ij}\right)=0$$
$$\text{for} \quad j\neq y_i, j=1,\dots,n, i=1,\dots,M, \tag{3.66}$$
$$\beta_{ij}\,\xi_{ij}=0 \quad \text{for} \quad j\neq y_i, j=1,\dots,n, i=1,\dots,M, \tag{3.67}$$

where (3.66) and (3.67) are the KKT (complementarity) conditions.

Using (3.61), (3.63) to (3.65) reduce, respectively, to

$$\sum_{i=1}^{M} z_{ij}=0 \quad \text{for} \quad j=1,\dots,n, \tag{3.68}$$
$$\mathbf{w}_j=\sum_{i=1}^{M} z_{ij}\,\mathbf{g}(\mathbf{x}_i) \quad \text{for} \quad j=1,\dots,n, \tag{3.69}$$
$$\alpha_{ij}+\beta_{ij}=C, \quad \alpha_{ij}\geq 0, \quad \beta_{ij}\geq 0$$
$$\text{for} \quad i=1,\dots,M, \ j\neq y_i,\ j=1,\dots,n. \tag{3.70}$$

Thus we obtain the following dual problem. Maximize

$$Q(\boldsymbol{\alpha})=\sum_{i=1}^{M}\sum_{\substack{j\neq y_i,\\ j=1}}\alpha_{ij}-\frac{1}{2}\sum_{i,k=1}^{M}\sum_{j=1}^{n} z_{ij}\,z_{kj}\,H(\mathbf{x}_i,\mathbf{x}_j) \tag{3.71}$$

subject to the constraints

$$\sum_{i=1}^{M} z_{ij}=0 \quad \text{for} \quad j\neq y_i,\ j=1,\dots,n, \tag{3.72}$$
$$0\leq\alpha_{ij}\leq C \quad \text{for} \quad i=1,\dots,M,\ j\neq y_i,\ j=1,\dots,n. \tag{3.73}$$

The decision function is given by

$$D_i(\mathbf{x})=\sum_{j=1}^{M} z_{ji}\,H(\mathbf{x}_j,\mathbf{x})+b_i. \tag{3.74}$$

Because α_{ji} are nonzero for the support vectors, the summation in (3.74) is added only for nonzero z_{ji}.

Then the datum $\mathbf{x}$ is classified into the class

$$\arg\max_{i=1,\dots,n} D_i(\mathbf{x}). \tag{3.75}$$

If the maximum is reached for plural classes, the datum is on the class boundary and is unclassifiable.

3.4.2 Sophisticated Architecture

Crammer and Singer [65] proposed replacing the slack variables ξ_{ij} with $\xi_i=\max_j \xi_{ij}$. Because the bias term is not included in the original formulation, we restate their formulation according to [114, 154].

We minimize

$$\frac{1}{2}\sum_{j=1}^{n}\|\mathbf{w}_j\|^2 + C\sum_{i=1}^{M}\xi_i \tag{3.76}$$

subject to the constraints

$$(\mathbf{w}_{y_i}-\mathbf{w}_j)^T\mathbf{g}(\mathbf{x}_i) + b_{y_i} - b_j \geq 1-\xi_i \quad \text{for} \quad j \neq y_i, j=1,\dots,n, i=1,\dots,M. \tag{3.77}$$

Introducing the nonnegative Lagrange multipliers α_i and β_i, we obtain

$$\begin{aligned} Q(\mathbf{w},\mathbf{b},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta}) &= \frac{1}{2}\sum_{j=1}^{n}\|\mathbf{w}_j\|^2 + C\sum_{i=1}^{M}\xi_i \\ &\quad -\sum_{i=1}^{M}\sum_{\substack{j\neq y_i,\\ j=1}}^{n}\alpha_i\,\left((\mathbf{w}_{y_i}-\mathbf{w}_j)^T\mathbf{g}(\mathbf{x}_i)+b_{y_i}-b_j-1+\xi_i\right) \\ &\quad -\sum_{i=1}^{M}\beta_i\,\xi_i \\ &= \frac{1}{2}\sum_{j=1}^{n}\|\mathbf{w}_j\|^2 - \sum_{i=1}^{M}\sum_{j=1}^{n} z_i^j\,\alpha_i\left(\mathbf{w}_j^T\,\mathbf{g}(\mathbf{x}_i)+b_j\right) \\ &\quad +\sum_{i=1}^{M}\left(C\,\xi_i-(n-1)\,\alpha_i\,(-1+\xi_i)-\beta_i\,\xi_i\right), \end{aligned} \tag{3.78}$$

where

$$z_i^j = \begin{cases} n-1 & \text{for } j=y_i, \\ -1 & \text{otherwise.} \end{cases} \tag{3.79}$$

The conditions of optimality are given by

$$\frac{\partial Q(\mathbf{w},\mathbf{b},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta})}{\partial \mathbf{b}} = \mathbf{0}, \tag{3.80}$$

$$\frac{\partial Q(\mathbf{w},\mathbf{b},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta})}{\partial \mathbf{w}} = \mathbf{0}, \tag{3.81}$$

$$\frac{\partial Q(\mathbf{w},\mathbf{b},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta})}{\partial \boldsymbol{\xi}} = \mathbf{0}, \tag{3.82}$$

$$\alpha_i\,\left((\mathbf{w}_{y_i}-\mathbf{w}_j)^T\mathbf{g}(\mathbf{x}_i)+b_{y_i}-b_j-1+\xi_i\right)=0 \quad \text{for} \quad j\neq y_i, j=1,\dots,n, i=1,\dots,M, \tag{3.83}$$

$$\beta_i\,\xi_i = 0 \qquad \text{for} \qquad j\neq y_i, j=1,\dots,n, i=1,\dots,M, \tag{3.84}$$

where (3.83) and (3.84) are the KKT complementarity conditions.

Using (3.78), (3.80) to (3.82) reduce, respectively, to

$$\sum_{i=1}^{M} z_i^j \, \alpha_i = 0 \qquad \text{for} \quad j = 1, \ldots, n, \tag{3.85}$$

$$\mathbf{w}_j = \sum_{i=1}^{M} z_i^j \, \alpha_i \, \mathbf{g}(\mathbf{x}_i) \qquad \text{for} \quad j = 1, \ldots, n, \tag{3.86}$$

$$(n-1)\,\alpha_i + \beta_i = C, \quad \alpha_i \geq 0, \quad \beta_i \geq 0 \qquad \text{for} \quad i = 1, \ldots, M. \tag{3.87}$$

Thus we obtain the following dual problem. Maximize

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,k=1}^{M} \sum_{j=1}^{n} z_i^j \, z_k^j \, \alpha_i \, \alpha_k \, H(\mathbf{x}_i, \mathbf{x}_j) \tag{3.88}$$

subject to the constraints

$$\sum_{i=1}^{M} z_i^j \, \alpha_i = 0 \qquad \text{for} \quad j = 1, \ldots, n, \tag{3.89}$$

$$0 \leq (n-1)\,\alpha_i \leq C \qquad \text{for} \quad i = 1, \ldots, M. \tag{3.90}$$

The decision function is given by

$$D_i(\mathbf{x}) = \sum_{j=1}^{M} z_i^j \, \alpha_i \, H(\mathbf{x}_j, \mathbf{x}) + b_i. \tag{3.91}$$

Datum $\mathbf{x}$ is classified into the class

$$\arg \max_{i=1,\ldots,n} D_i(\mathbf{x}). \tag{3.92}$$

3.5 Comparisons of Architectures

We have discussed four types of support vector machines: one-against-all, pairwise, ECOC, and all-at-once support vector machines. In this section we summarize their characteristics and compare their trainability.

3.5.1 One-against-All Support Vector Machines

To resolve unclassifiable regions of the original one-against-all support vector machines with discrete decision functions, the following extensions have been discussed in this chapter:

1. Support vector machines with continuous decision functions.
2. Fuzzy support vector machines with minimum and average operators are proved to be equivalent to support vector machines with continuous decision functions.

3. Decision-tree-based support vector machines. Training of the preceding two types of support vector machines is the same as of support vector machines with discrete decision functions. But for an n-class problem, because a decision-tree-based support vector machine determines $n - 1$ decision functions using smaller numbers of training data, training is faster.

For the data sets tested, an improvement of generalization ability of the support vector machines over support vector machines with discrete decision functions was large.

3.5.2 Pairwise Support Vector Machines

By pairwise support vector machines unclassifiable regions are reduced but they still exist. Thus to resolve unclassifiable regions the following extensions have been discussed:

1. Fuzzy support vector machines with minimum and average operators. The improvement by introducing the membership functions was smaller than for one-against-all support vector machines because the size of unclassifiable regions is smaller. In addition, there is not much difference of recognition rates using minimum and average operators.
2. Decision-tree-based support vector machines: decision directed acyclic graphs (DDAGs) and adaptive directed acyclic graphs (ADAGs). DDAGs are more general than ADAGs but the difference of generalization ability is very small. The generalization ability of fuzzy support vector machines is better than the average generalization ability of DDAGs and ADAGs, but the difference is small for the tested data sets. DDAGs and ADAGs are suited for the problem where high-speed classification is necessary. Optimization of DDAGs and ADAGs was discussed, and the recognition rates of the optimized DDAGs were comparable or better than the average recognition rates of DDAGs.

3.5.3 ECOC Support Vector Machines

Error-correcting output codes are introduced to improve generalization ability of support vector machines. By ECOC, resolution of unclassifiable regions is also achieved. ECOC support vector machines are equivalent to fuzzy support vector machines with average operators.

ECOC support vector machines are an extension of one-against-all support vector machines and by introducing "don't care" outputs, any classification scheme including pairwise classification can be realized.

According to the performance evaluation, ECOC support vector machines did not always outperform one-against-all fuzzy support vector machines, and there was an optimal code word length. Thus, optimization of ECOC structures is necessary.

3.5.4 All-at-Once Support Vector Machines

By determining all the decision functions at once, unclassifiable regions are resolved. But by the original formulation the number of variables is $M \times (n-1)$, where M is the number of training data and n is the number of classes. Thus training becomes difficult as the number of training data becomes large. This problem was overcome by the new formulation of all-at-once support vector machines in which the number of variables is reduced to M.

3.5.5 Training Difficulty

Table 3.11 summarizes the characteristics of four types of support vector machines from the number of decision functions to be determined for n class problems, the number of variables solved simultaneously for the M training data, and the number of equalities in the dual optimization problem. In the table, M_i is the number of training data belonging to class i and k is the length of code words.

Table 3.11. Comparison of support vector machines

	One-against-all	Pairwise	ECOC	All-at-once
Decision functions	n	$n(n-1)/2$	k	n
Variables	M	$M_i + M_j$	M	M
Equalities	1	1	1	n

In the following we discuss the difference of support vector machines from the standpoint of separability. We say that training data are separable by a support vector machine if each decision function for the support vector machine separates training data in the feature space. This means that training data are separated correctly 100 percent by the support vector machine.

Example 3.2. Consider a one-dimensional case shown in Fig. 3.19, where Class 1 is in (a, b), Class 2 is in $(-\infty, a)$, and Class 3 is in (b, ∞). We consider linear decision functions for three classes [252].

Because Class 1 is not separated from Classes 2 and 3 by a linear decision function, the problem is not separable by one-against-all formulation. But by pairwise formulation, by setting

$$g_{12}(x) = x - a, \tag{3.93}$$

$$g_{13}(x) = -x + b, \tag{3.94}$$

$$g_{23}(x) = -x + c, \tag{3.95}$$

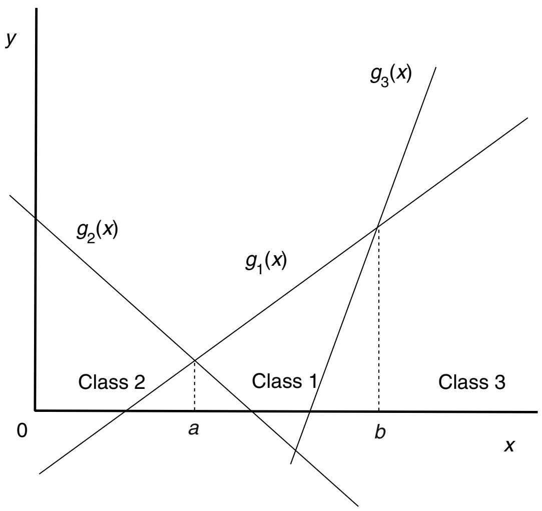


Fig. 3.19. Class boundaries for a one-dimensional case by all-at-once formulation

where $b > c > a$, the problem is separable.

By all-at-once formulation, the problem is also separable. Figure 3.19 shows an example of $g_i(x)$. Because $g_1(x) > g_2(x)$ and $g_1(x) > g_3(x)$ in the interval (a, b), the data in this interval are classified into Class 1.

Therefore, the separation power of one-against-all formulation is lower than pairwise or all-at-once formulation.

Theorem 3.3. *If training data are separable by a one-against-all support vector machine, the training data are separable by a pairwise support vector machine. But the reverse is not always true.*

Proof. Suppose for an n-class problem, training data are separable by a one-against-all support vector machine. Then class i is separated from the set of classes $\{j \mid j \neq i,\ j = 1, \ldots, n\}$. This means that class i is separated from class j $(j \neq i,\ j = 1, \ldots, n)$. Therefore, any pair of classes i and j is separable in the feature space.

We prove that the reverse does not hold by a counterexample. Example 3.2 shows one counterexample, but here we consider a three-class case shown in Fig. 3.20, where each pair of classes is separated linearly. Thus the training data are separable by a pairwise support vector machine. But as shown in Fig. 3.21, the training data are not separable by a one-against-all support vector machine. Thus the reverse is not always true. ■

Theorem 3.3 tells us that for the given kernel, training data are more difficult to separate by one-against-all support vector machines than by pairwise

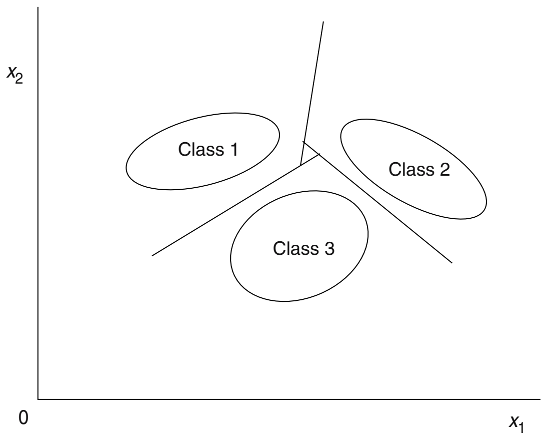


Fig. 3.20. Class boundaries by a pairwise support vector machine

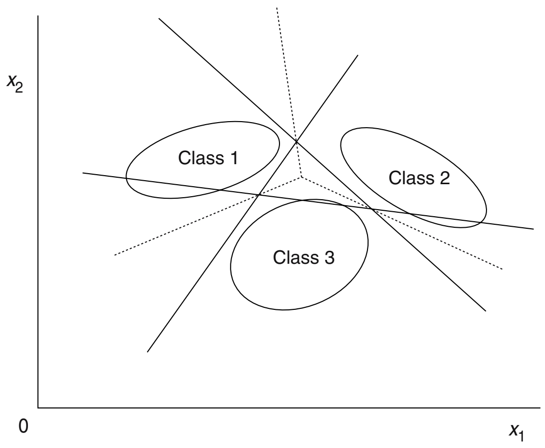


Fig. 3.21. Class boundaries by a one-against-all support vector machine

support vector machines. In addition, for one-against-all support vector machines, the number of training data for determining a decision function is the total number of training data compared to the sum of two-class training data for pairwise support vector machines. Thus, training of a one-against-all support vector machine is more difficult than that of a pairwise support vector machine.

Theorem 3.4. *If training data are separable by an ECOC support vector machine, the training data are separable by a pairwise support vector machine. But the reverse is not always true.*

Proof. Suppose for an n-class problem, training data are separable by an ECOC support vector machine. Then for the kth decision function a set of classes L_1 is separated from a set of classes L_2, where $L_1 \cap L_2 = \{1, \ldots, n\}$. Thus any class in L_1 is separated from any class in L_2. Assume that there is a pair of classes i and j that are not separable. This means that classes i and j are on the same sides of all the decision functions. This does not happen because the ECOC does not classify classes i and j.

Because ECOC support vector machines are an extension of one-against-all support vector machines, the reverse is not always true. ∎

Now compare one-against-all and all-at-once support vector machines when a classification problem is separable by the one-against-all support vector machine. In the one-against-all support vector machine, the decision function for class i is determined so that $D_i(\mathbf{x}) > 1$ for $\mathbf{x}$ belonging to class i and $D_i(\mathbf{x}) < -1$, otherwise. But by classification using continuous decision functions or fuzzy membership functions, if

$$D_i(\mathbf{x}) > D_j(\mathbf{x}) \quad \text{for} \quad j \neq i, j = 1, \ldots, n, \tag{3.96}$$

$\mathbf{x}$ is classified into class i. Equation (3.96) is the same constraint as that of the all-at-once support vector machine. Thus it is estimated that the decision boundaries obtained by one-against-all and all-at-once support vector machines are quite similar (compare the decision boundaries in Figs. 3.21 and 3.22).

As shown in Figs. 3.20 and 3.22, for three classes, training data that are separable by an all-at-once support vector machine are separable by a pairwise support vector machine, and vice versa. But in general, separation of a smaller number of data with a smaller number of constraints is easier than that by a larger number of data. Thus, training data are more separable by pairwise support vector machines than by all-at-once support vector machines.

Considering the fact that an all-at-once support vector machine has similar decision functions with a one-against-all support vector machine, and that its training is more difficult, an all-at-once support vector machine should be a last choice as a classifier.

3.5.6 Training Time Comparison

We evaluated the training time of support vector machines using the data sets listed in Table 1.1. We solved the optimization problem repeatedly, adding 200 data for the MNIST data set and 50 data except for the set at a time. We used the primal-dual interior-point method [254] combined with the decomposition technique to solve the quadratic programming problem. We set the value of the margin parameter C to 5000. We used an Athlon (1GHz) computer for the MNIST data set and a Pentium III (933MHz) computer for the other data sets.

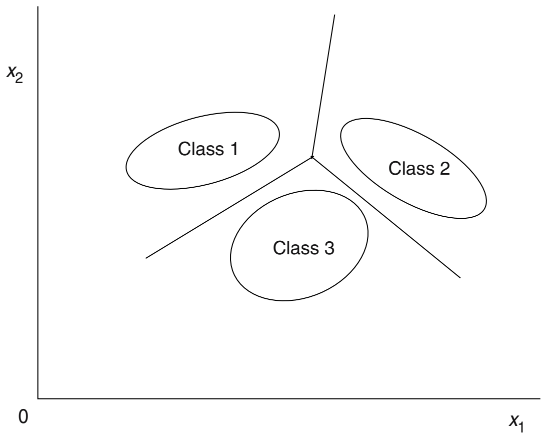


Fig. 3.22. Class boundaries by an all-at-once support vector machine

Table 3.12 shows the training time for one-against-all and pairwise support vector machines. For the hiragana-105 data with 38 classes, 703 decision functions need to be determined for pairwise classification, but training a pairwise support vector machine is 6 times faster than training a one-against-all support vector machine. For the data sets tried, training speedup by pairwise classification is 3 to 9.

Table 3.12. Training time comparison

Data	Kernel	One-against-all (s)	Pairwise (s)	Ratio
Thyroid	$d4$	44	5	8.8
Blood cell	$d4$	32	5	6.4
Hiragana-50	$d2$	128	40	3.2
Hiragana-105	$d2$	600	98	6.1
Hiragana-13	$d2$	271	43	6.3
MNIST	$d3$	3153	554	5.7

4

Variants of Support Vector Machines

Since the introduction of support vector machines, numerous variants have been developed. In this chapter, we discuss some of them: least squares support vector machines, linear programming support vector machines, robust support vector machines, Bayesian support vector machines, and committee machines. We also discuss incremental training, introduction of confidence level, and visualization of support vector machines.

4.1 Least Squares Support Vector Machines

Suykens et al. [23, 234, 236] proposed least squares (LS) support vector machines for two-class problems, in which the inequality constraints in L2 soft-margin support vector machines are converted into equality constraints. The training of the LS support vector machine is done by solving a set of linear equations, instead of a quadratic programming problem.

We can extend the two-class LS support vector machines to multiclass LS support vector machines [234, 238] in a way similar to that of support vector machines. In [162], the effect of coding methods for multiclass classification on generalization ability was evaluated by computer simulations. Three coding methods—one-against-all classification, a variant of pairwise classification, i.e., correcting classification, and classification based on error-correcting output codes were tested for two benchmark data sets and the last two outperformed the first one.

In the following, we first discuss two-class LS support vector machines. Then we discuss one-against-all, pairwise, and all-at-once LS support vector machines, and finally we compare their classification and training performance.

4.1.1 Two-Class Least Squares Support Vector Machines

The LS support vector machine is trained by minimizing

$$\frac{1}{2}\,\mathbf{w}^T\mathbf{w} + \frac{C}{2}\sum_{i=1}^{M}\xi_i^2 \tag{4.1}$$

subject to the equality constraints:[1]

$$y_i\,(\mathbf{w}^T\mathbf{g}(\mathbf{x}_i) + b) = 1 - \xi_i \quad \text{for} \quad i = 1,\dots,M, \tag{4.2}$$

where $y_i = 1$ and -1 if $\mathbf{x}_i$ belongs to Classes 1 and 2, respectively, $\mathbf{w}$ is the l-dimensional vector, b is the bias term, $\mathbf{g}(\mathbf{x})$ is the l-dimensional vector that maps m-dimensional vector $\mathbf{x}$ into the feature space, ξ_i is the slack variable for $\mathbf{x}_i$, and C is the margin parameter. Here, if $\xi_i \geq 1$, $\mathbf{x}_i$ is misclassified and if $\xi_i < 1$, $\mathbf{x}_i$ is correctly classified. If $1 > \xi_i > 0$, $\mathbf{x}_i$ is correctly classified but with a smaller margin. Unlike conventional support vector machines, ξ_i can be negative.

Introducing the Lagrange multipliers α_i into (4.1) and (4.2), we obtain the unconstrained objective function:

$$\begin{aligned} &Q(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}) \\ &\quad = \frac{1}{2}\,\mathbf{w}^T\mathbf{w} + \frac{C}{2}\sum_{i=1}^{M}\xi_i^2 - \sum_{i=1}^{M}\alpha_i\,\{y_i\,(\mathbf{w}^T\mathbf{g}(\mathbf{x}_i) + b) - 1 + \xi_i\}, \end{aligned} \tag{4.3}$$

where $\boldsymbol{\alpha} = (\alpha_1,\dots,\alpha_M)^T$ and $\boldsymbol{\xi} = (\xi_1,\dots,\xi_M)^T$.

Taking the partial derivatives of (4.3) with respect to $\mathbf{w}$, b, and $\boldsymbol{\xi}$ and equating them to zero, together with the equality constraint (4.2), we obtain the optimal conditions as follows:

$$\mathbf{w} = \sum_{i=1}^{M}\alpha_i\,y_i\,\mathbf{g}(\mathbf{x}_i), \tag{4.4}$$

$$\sum_{i=1}^{M}\alpha_i\,y_i = 0, \tag{4.5}$$

$$\alpha_i = C\,\xi_i \quad \text{for} \quad i = 1,\dots,M, \tag{4.6}$$

$$y_i\left(\mathbf{w}^T\,\mathbf{g}(\mathbf{x}_i) + b\right) - 1 + \xi_i = 0 \quad \text{for} \quad i = 1,\dots,M. \tag{4.7}$$

From (4.6), unlike conventional support vector machines, α_i can be negative.

Substituting (4.4) and (4.6) into (4.7) and expressing it and (4.5) in matrix form, we obtain

$$\begin{pmatrix} \Omega & \mathbf{y} \\ \mathbf{y}^T & 0 \end{pmatrix}\begin{pmatrix} \boldsymbol{\alpha} \\ b \end{pmatrix} = \begin{pmatrix} \mathbf{1} \\ 0 \end{pmatrix}, \tag{4.8}$$

or

[1]Multiplying y_i to both sides of (4.2), we obtain $y_i - \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) - b = y_i\,\xi_i$. Because ξ_i takes either a positive or a negative value and $(y_i\,\xi_i)^2 = \xi_i^2$, we can use $y_i - \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) - b = \xi_i$ instead of (4.2).

$$\Omega \boldsymbol{\alpha} + \mathbf{y} b = \mathbf{1}, \tag{4.9}$$

$$\mathbf{y}^T \boldsymbol{\alpha} = 0, \tag{4.10}$$

where $\mathbf{1}$ is the M-dimensional vector and

$$\Omega_{ij} = y_i \, y_j \, \mathbf{g}^T(\mathbf{x}_i) \, \mathbf{g}(\mathbf{x}_j) + \frac{\delta_{ij}}{C}, \tag{4.11}$$

$$\delta_{ij} = \begin{cases} 1 & i = j, \\ 0 & i \neq j, \end{cases} \tag{4.12}$$

$$\mathbf{y} = (y_1, \ldots, y_M)^T, \tag{4.13}$$

$$\mathbf{1} = (1, \ldots, 1)^T. \tag{4.14}$$

Like the support vector machine, setting $H(\mathbf{x}, \mathbf{x}') = \mathbf{g}^T(\mathbf{x}) \, \mathbf{g}(\mathbf{x}')$, we can avoid the explicit treatment of variables in the feature space.

The original minimization problem is solved by solving (4.8) for $\boldsymbol{\alpha}$ and b as follows. Because of $1/C \, (> 0)$ in the diagonal elements, Ω is positive definite. Therefore,

$$\boldsymbol{\alpha} = \Omega^{-1}(1 - \mathbf{y} \, b). \tag{4.15}$$

Substituting (4.15) into (4.10), we obtain

$$b = (\mathbf{y}^T \Omega^{-1} \mathbf{y})^{-1} \mathbf{y}^T \Omega^{-1} \mathbf{1}. \tag{4.16}$$

Thus, substituting (4.16) into (4.15), we obtain $\boldsymbol{\alpha}$.

By changing the inequality constraints into the equality constraints, training of support vector machines reduces to solving a set of linear equations instead of a quadratic programming problem. But by this formulation, sparsity of $\boldsymbol{\alpha}$ is not guaranteed. To avoid this, Suykens et al. [234, 235] proposed pruning the data whose associated α_i have small absolute values. Namely, first we solve (4.8) using all the training data. Next, we sort α_i according to their absolute values and delete a portion of the training data set (say 5 percent of the set) starting from the data with the minimum absolute value in order. Then we solve (4.8) using the reduced training data set and iterate this procedure while the user-defined performance index is not degraded.

Cawley and Talbot [54] proposed a greedy algorithm. By assuming that the weight vector is approximated by

$$\mathbf{w} = \sum_{i \in S} \alpha_i \, y_i \, \mathbf{g}(\mathbf{x}_i), \tag{4.17}$$

where $S \subset \{1, \ldots, M\}$, (4.1) becomes

$$\frac{1}{2} \sum_{i,j \in S} \alpha_i \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) + \frac{C}{2} \sum_{i=1}^{M} \left(\sum_{j \in S} y_i \, (y_j \, \alpha_j \, H(\mathbf{x}_i, \mathbf{x}_j) + b) - 1 \right)^2 . \tag{4.18}$$

Then starting with only a bias term, i.e., $S = \emptyset$, the training pattern that minimizes the objective function (4.18) is added to S until some convergence test is satisfied.

For the original LS support vector machine, all the training data become support vectors. Thus, classification becomes slow for a large problem. For linear kernels, calculating $\mathbf{w}$ once and saving it, we can avoid this problem [252]. Unfortunately, as discussed in Section 2.6.5, a similar method is not applicable to nonlinear kernels.

4.1.2 One-against-All Least Squares Support Vector Machines

For a one-against-all LS support vector machine, we determine n decision functions that separate one class from the remaining classes. The ith decision function,

$$D_i(\mathbf{x}) = \mathbf{w}_i^T \, \mathbf{g}(\mathbf{x}) + b_i, \tag{4.19}$$

separates class i from the remaining classes with the maximum margin, where $\mathbf{w}_i$ is the l-dimensional weight vector and b_i is the bias term.

The hyperplane $D_i(\mathbf{x}) = 0$ forms the optimal separating hyperplane and if the classification problem is separable, the training data belonging to class i satisfy $D_i(\mathbf{x}) \geq 1$, and those belonging to the remaining classes satisfy $D_i(\mathbf{x}) \leq -1$.

In classification, if for the input vector $\mathbf{x}$

$$D_i(\mathbf{x}) > 0 \tag{4.20}$$

is satisfied for one i, $\mathbf{x}$ is classified into class i. Because only the sign of the decision function is used, the decision is discrete.

If (4.20) is satisfied for plural is, or if there is no i that satisfies (4.20), $\mathbf{x}$ is unclassifiable.

To avoid this, instead of the discrete decision functions, continuous decision functions can be used. Namely, datum $\mathbf{x}$ is classified into the class

$$\arg \max_{i=1,\ldots,n} D_i(\mathbf{x}). \tag{4.21}$$

Another way of resolving unclassifiable regions is to introduce membership functions. For class i we define one-dimensional membership functions $m_{ij}(\mathbf{x})$ in the directions orthogonal to the optimal separating hyperplanes $D_j(\mathbf{x}) = 0$ as follows:

1. For $i = j$

$$m_{ii}(\mathbf{x}) = \begin{cases} 1 & \text{for} \quad D_i(\mathbf{x}) \geq 1, \\ D_i(\mathbf{x}) & \text{otherwise.} \end{cases} \tag{4.22}$$

2. For $i \neq j$

$$m_{ij}(\mathbf{x}) = \begin{cases} 1 & \text{for} \quad D_j(\mathbf{x}) \leq -1, \\ -D_j(\mathbf{x}) & \text{otherwise.} \end{cases} \tag{4.23}$$

For $i \neq j$, class i is on the negative side of $D_j(\mathbf{x}) = 0$.

We define the class i membership function of $\mathbf{x}$ by the minimum operation for $m_{ij}(\mathbf{x})\ (j = 1, \dots, n)$,

$$m_i(\mathbf{x}) = \min_{j=1,\dots,n} m_{ij}(\mathbf{x}), \tag{4.24}$$

or the average operation,

$$m_i(\mathbf{x}) = \frac{1}{n} \sum_{j=1,\dots,n} m_{ij}(\mathbf{x}). \tag{4.25}$$

The datum $\mathbf{x}$ is classified into the class

$$\arg \max_{i=1,\dots,n} m_i(\mathbf{x}). \tag{4.26}$$

In Section 3.1.3, one-against-all support vector machines with continuous decision functions and one-against-all fuzzy support vector machines with minimum or average operators are proved to be equivalent. This also holds for LS support vector machines.

4.1.3 Pairwise Least Squares Support Vector Machines

In pairwise classifications, unclassifiable regions exist. Thus, similar to Section 3.2.2, we introduce fuzzy membership functions to resolve unclassifiable regions in pairwise classification [250, 251].

In pairwise classification we require a binary classifier for each possible pair of classes and the number of the total pairs is $n\,(n-1)/2$ for an n-class problem. The decision function for the pair of classes i and j is given by

$$D_{ij}(\mathbf{x}) = \mathbf{w}_{ij}^T\, \mathbf{g}(\mathbf{x}) + b_{ij}, \tag{4.27}$$

where $\mathbf{w}_{ij}$ is the l-dimensional weight vector, $\mathbf{g}(\mathbf{x})$ maps $\mathbf{x}$ into the l-dimensional feature space, b_{ij} is the bias term, and $D_{ij}(\mathbf{x}) = -D_{ji}(\mathbf{x})$. Then for datum $\mathbf{x}$ we calculate

$$D_i(\mathbf{x}) = \sum_{j \neq i, j=1}^{n} \operatorname{sign}(D_{ij}(\mathbf{x})), \tag{4.28}$$

where

$$\operatorname{sign}(a) = \begin{cases} 1 & a \geq 0, \\ -1 & \text{otherwise}, \end{cases}$$

and this datum is classified into the class

$$\arg \max_{i=1,\dots,n} D_i(\mathbf{x}). \tag{4.29}$$

If (4.29) is satisfied for one i, $\mathbf{x}$ is classified into class i. But if (4.29) is satisfied for plural is, $\mathbf{x}$ is unclassifiable.

To avoid this, we introduce membership functions. First, we define the one-dimensional membership function, $m_{ij}(\mathbf{x})$, in the direction orthogonal to the optimal separating hyperplane $D_{ij}(\mathbf{x})$ as follows:

$$m_{ij} = \begin{cases} 1 & \text{for} \quad D_{ij}(\mathbf{x}) \geq 1, \\ D_{ij}(\mathbf{x}) & \text{otherwise.} \end{cases} \tag{4.30}$$

Here, we allow a negative degree of membership to make any data except those on the decision boundary be classified.

Using the minimum operator the membership function, $m_i(\mathbf{x})$, of $\mathbf{x}$ for class i is given by

$$m_i(\mathbf{x}) = \min_{j=1,\ldots,n} m_{ij}(\mathbf{x}). \tag{4.31}$$

The shape of the resulting membership function is a truncated polyhedral pyramid in the feature space, and the contour surface, in which the degree of membership is the same, is parallel to the decision function.

Using the average operator the membership function, $m_i(\mathbf{x})$, of $\mathbf{x}$ for class i is given by

$$m_i(\mathbf{x}) = \frac{1}{n-1} \sum_{j \neq i, j=1}^{n} m_{ij}(\mathbf{x}). \tag{4.32}$$

The shape of the resulting membership function is a truncated polyhedral pyramid but some part of the contour surface is not parallel to the decision function.

Using either (4.31) or (4.32), datum $\mathbf{x}$ is classified into the class

$$\arg \max_{i=1,\ldots,n} m_i(\mathbf{x}). \tag{4.33}$$

Comparing the minimum and average operators, the regions where $m_i(\mathbf{x}) = 1$ are the same, but the regions where $m_i(\mathbf{x}) < 1$ are different. We can show that the decision boundaries with the minimum operator are the same as those given by (4.28) for classifiable regions but the decision boundaries for the average operator are not. Thus the recognition rate using the minimum operator is always better than or equal to that by the conventional pairwise LS support vector machine. But this does not hold for the average operator.

4.1.4 All-at-Once Least Squares Support Vector Machines

Similar to all-at-once support vector machines discussed in Section 3.4, we can define all-at-once LS support vector machines.

To obtain the maximum margin classifier, we minimize

$$\frac{1}{2} \sum_{j=1}^{n} \|\mathbf{w}_j\|^2 + \frac{C}{2} \sum_{i=1}^{M} \sum_{\substack{j \neq y_i, \\ j=1}}^{n} \xi_{ij}^2 \tag{4.34}$$

subject to the equality constraints

$$(\mathbf{w}_{y_i} - \mathbf{w}_j)^T \mathbf{g}(\mathbf{x}_i) + b_{y_i} - b_j = 1 - \xi_{ij} \quad \text{for} \quad j \neq y_i, j = 1, \ldots, n, i = 1, \ldots, M, \tag{4.35}$$

where $\mathbf{w}_j$ is the l-dimensional weight vector for class j, b_j is the bias term for class j, ξ_{ij} are slack variables associated with $\mathbf{x}_i$ and class j, y_i ($\in \{1, \ldots, n\}$) is the class label for $\mathbf{x}_i$, and C is the margin parameter that determines the trade-off between the maximization of the margin and minimization of the classification error. Unlike all-at-once support vector machines discussed in Section 3.4.2, because of the equality constraints, we cannot reduce slack variables from ξ_{ij} to ξ_i.

Introducing the Lagrange multipliers α_{ij}, we obtain

$$Q(\mathbf{w}, \mathbf{b}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2} \sum_{j=1}^{n} \|\mathbf{w}_j\|^2 + \frac{C}{2} \sum_{i=1}^{M} \sum_{\substack{j \neq y_i, \\ j=1}}^{n} \xi_{ij}^2 - \sum_{i=1}^{M} \sum_{\substack{j \neq y_i, \\ j=1}}^{n} \alpha_{ij} \left((\mathbf{w}_{y_i} - \mathbf{w}_j)^T \mathbf{g}(\mathbf{x}_i) + b_{y_i} - b_j - 1 + \xi_{ij} \right). \tag{4.36}$$

Taking the partial derivatives of (4.36) with respect to $\mathbf{w}_j, b_j, \alpha_{ij}$, and ξ_{ij} and equating them to zero, we obtain the optimal conditions as follows:

$$\mathbf{w}_j = \sum_{i=1}^{M} z_{ij}\, \mathbf{g}(\mathbf{x}_i) \qquad \text{for} \quad j = 1, \ldots, n, \tag{4.37}$$

$$\sum_{i=1}^{M} z_{ij} = 0 \qquad \text{for} \quad j = 1, \ldots, n, \tag{4.38}$$

$$\alpha_{ij} = C\xi_{ij}, \quad \alpha_{ij} \geq 0 \quad \text{for} \quad i = 1, \ldots, M,\ j \neq y_i,\ j = 1, \ldots, n, \tag{4.39}$$

$$(\mathbf{w}_{y_i} - \mathbf{w}_j)^T \mathbf{g}(\mathbf{x}_i) + b_{y_i} - b_j - 1 + \xi_{ij} = 0 \quad \text{for} \quad i = 1, \ldots, M,\ j \neq y_i,\ j = 1, \ldots, n, \tag{4.40}$$

where

$$z_{ij} = \begin{cases} \displaystyle\sum_{\substack{k \neq y_i, \\ k=1}}^{n} \alpha_{ik} & \text{for } j = y_i, \\ -\alpha_{ij} & \text{otherwise.} \end{cases} \tag{4.41}$$

Similar to a two-class problem, substituting (4.37) and (4.39) into (4.40), we can solve the resulting equation and (4.38) for α_{ij} and b_i.

The decision function for class i is given by

$$D_i(\mathbf{x}) = \sum_{j=1}^{M} z_{ji}\, \mathbf{g}^T(\mathbf{x}_j)\, \mathbf{g}(\mathbf{x}) + b_i. \tag{4.42}$$

As usual, to avoid explicit treatment of variables in the feature space, we use kernel tricks: $H(\mathbf{x}, \mathbf{x}') = \mathbf{g}^T(\mathbf{x}_j)\,\mathbf{g}(\mathbf{x})$.

4.1.5 Performance Comparison

Condition of Experiments

Using the data sets listed in Table 1.1, we compared the performance of the fuzzy one-against-all LS support vector machine (LS SVM), the fuzzy pairwise LS SVM with minimum and average operators, and the all-at-once LS SVM. Each input variable was scaled into $[0, 1]$.

Among linear kernels, polynomial kernels with degree 2, 3, and 4, and RBF kernels with $\gamma = 0.1$, 1, and 10 we selected the optimum kernel and the value of C from 1 to 100,000 by five-fold cross-validation. The simulations were done on an AthlonMP 2GHz personal computer.

Classification Performance

Table 4.1 shows the recognition performance of the fuzzy one-against-all LS SVM, the fuzzy pairwise LS SVMs with minimum and average operators, and all-at-once LS SVM. The “Parm” rows list the kernel types and parameters and the values of C optimized by the five-fold cross-validation. The “Min” and “Avg.” rows show the recognition rates with the decision functions with minimum operators and with average operators, respectively. If the recognition rates of the training data were not 100 percent, we list the recognition rates in parentheses. For the all-at-once LS SVM, we list the recognition rates in “Min” rows. The highest recognition rates of the test data are shown in boldface.

We could not get the results of the all-at-once LS SVM other than for the iris, numeral, and thyroid data sets due to memory overflow. In the same reason, for the MNIST data set, we set $C = 10{,}000$ and used RBF kernels for the pairwise LS SVM.

In the following, the recognition rate means that of the test data.

For all the data sets, the fuzzy pairwise LS SVMs performed best, and except for the horagana-13 data set the average operator performed better than the minimum operator.

Now compare the recognition rates of LS SVMs with those of SVMs listed in Tables 3.2 and 3.3. For the SVMs, the performance depends on the data sets; no single architecture performed best. But for the LS SVMs, the pairwise fuzzy LS SVMs performed best. Except for the thyroid data set, the pairwise LS SVM with the average operators showed the same or higher recognition rates than the SVMs. Especially for the blood cell data set, the difference was large. But for the thyroid data set, the recognition rate of the LS SVMs was very poor. Therefore, the performance of LS SVMs is more dependable on the data sets than that of the SVMs.

Table 4.1. Performance comparison of LS support vector machines

Data	Item	One-against-all	Pairwise	All-at-once
Iris	Parm	$\gamma 1$, $C10^4$	$d1$, $C50$	$d3$, $C10^4$
	Min	96.00	97.33	92.00
	Avg.	96.00	**98.67**	—
Numeral	Parm	$\gamma 10$, $C50$	$d1$, $C10$	$\gamma 0.1$, $C500$
	Min	99.39	99.27 (99.75)	99.02 (99.75)
	Avg.	99.39	**99.76**	—
Thyroid	Parm	$\gamma 10$, $C10^5$	$\gamma 10$, $C10^5$	$\gamma 10$, $C10^5$
	Min	94.22 (97.38)	95.48 (98.49)	94.25 (97.91)
	Avg.	94.22 (97.38)	**95.57** (98.28)	—
Blood cell	Parm	$\gamma 10$, $C500$	$d2$, $C10^5$	—
	Min	93.58 (96.48)	93.55 (97.87)	—
	Avg.	93.58 (96.48)	**94.32** (98.10)	—
Hiragana-50	Parm	$\gamma 10$, $C10^4$	$\gamma 1$, $C3000$	—
	Min	99.22	99.02	—
	Avg.	99.22	**99.33**	—
Hiragana-13	Parm	$\gamma 10$, $C10^4$	$\gamma 10$, $C10^5$	—
	Min	99.64 (99.77)	**99.90**	—
	Avg.	99.64 (99.77)	99.88	—
Hiragana-105	Parm	$\gamma 10$, $C10^4$	$\gamma 10$, $C7000$	—
	Min	**100**	**100**	—
	Avg.	**100**	**100**	—
MNIST	Parm	—	$\gamma 10$, $C10^4$	—
	Min	—	**98.79**	—
	Avg.	—	**98.79**	—

Training Speed

Table 4.2 shows the training time of the one-against-all, pairwise, and all-at-once LS SVMs for the polynomial kernels with degree 2. In training the LS SVM, we used the Cholesky factorization to solve the set of linear equations. For all the cases training of the pairwise LS SVM was the fastest, and training of the all-at-once LS SVM was the slowest.[2] For the blood cell and hiragana

[2]For conventional SVMs, this fact was shown in [114].

data sets, we could not train the all-at-once LS SVM because of the memory overflow. Therefore, as indicated in [237], we need to use iterative methods for speedup and efficient memory use.

Table 4.2. Training time in seconds

Data	One-against-all	Pairwise	All-at-once
Numeral	25	1	2026
Thyroid	716	409	1565
Blood cell	1593	59	—
Hiragana-50	15130	129	—

Classification Speed

Table 4.3 lists the classification time for linear kernels with the values of C determined by the five-fold cross-validation for linear kernels. The conventional and calculated methods mean that the weights are calculated for each datum and the weights are calculated once before classification, respectively. Except for the iris data set, which is very small, speedup by the calculated method is evident.

Table 4.3. Classification time in seconds

Data	Method	One-against-all	Pairwise	All-at-once
Iris	Conventional	0.01	0.00	0.01
	Calculated	0.00	0.00	0.00
Numeral	Conventional	3.1	2.7	3.2
	Calculated	0.01	0.02	0.00
Thyroid	Conventional	43	28	43
	Calculated	0.02	0.02	0.02
Blood cell	Conventional	80	52	—
	Calculated	0.02	0.26	—

Influence of Outliers

Because LS SVMs use equality constraints instead of inequality constraints, they are vulnerable to outliers [237]. The only difference between LS SVMs and L2 SVMs is that the former uses equality constraints whereas the latter uses the inequality constraints. Thus, we compared their recognition performance when outliers were included.

For evaluation, we used the blood cell data belonging to Classes 2 and 3, which overlap heavily and are difficult to classify. As outliers, we added 10 data belonging to classes other than 2 and 3 to Class 2 training data. We used the polynomial kernel with degree 2.

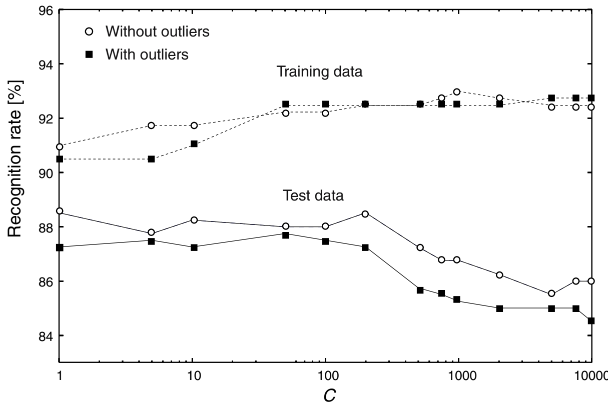


Fig. 4.1. Influence of outliers to LS SVM. Reprinted from [251, p. 791] with permission from Elsevier

Figures 4.1 and 4.2 show the recognition rates against the margin parameter C for the LS SVM and L2 SVM, respectively. In the figures, the dotted lines show the recognition rates of the training data and the solid lines show those of the test data. We calculated the recognition rate of the training data excluding outliers.

In Fig. 4.1, the recognition rates of the training data for the LS SVM did not change much for $100 < C < 10{,}000$ even if the outliers were included. But the recognition rate of the test data dropped rapidly, especially when outliers were included.

In Fig. 4.2, the recognition rates of the training data for the L2 SVM increased as the value of C was increased, and there is not much difference between the recognition rates with and those without outliers. In addition, the recognition rate of the test data with outliers was almost constant for the change of C, and for a large value of C it is better than without outliers.

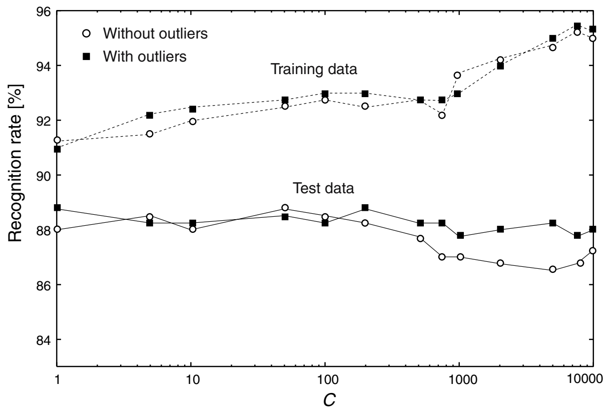


Fig. 4.2. Influence of outliers to L2 SVM. Reprinted from [251, p. 791] with permission from Elsevier

Comparing Figs. 4.1 and 4.2, we can see that the L2 SVM is more robust than the LS SVM for outliers.

4.2 Linear Programming Support Vector Machines

In the original formulation of support vector machines, quadratic programming problems need to be solved. But we can formulate classification problems by linear programming, replacing the quadratic objective function with a linear function [29, 129, 229, 285]. In this section, we discuss linear programming (LP) support vector machines, their characteristics, and their classification performance for benchmark data sets.

4.2.1 Architecture

In the L1 soft-margin support vector machine, replacing the L2 norm $\|\mathbf{w}\|_2^2 = w_1^2 + w_2^2 + \cdots + w_l^2$ in the objective function with an L1 norm $\|\mathbf{w}\|_1 = |w_1| + |w_2| + \cdots + |w_l|$, where $\mathbf{w}$ is a coefficient vector of the separating hyperplane, we obtain the following LP support vector machine. Minimize

$$Q(\mathbf{w}, \boldsymbol{\xi}) = \sum_{i}^{l} |w_i| + C \sum_{i=1}^{M} \xi_i \tag{4.43}$$

subject to

$$y_i \left(\mathbf{w}^T \mathbf{g}(\mathbf{x}_i) + b\right) \geq 1 - \xi_i \qquad \text{for} \quad i = 1, \ldots, M, \tag{4.44}$$

where C is the margin parameter, ξ_i are slack variables associated with the training data $\mathbf{x}_i$, y_i are class labels and are 1 if $\mathbf{x}_i$ belong to Class 1 and -1 otherwise, $\mathbf{g}(\mathbf{x})$ is the l-dimensional mapping function that maps $\mathbf{x}$ into the feature space, and b is the bias term.

By this formulation, for the linear kernel, i.e., $\mathbf{g}(\mathbf{x}) = \mathbf{x}$, we can solve the problem by linear programming. However, for the kernels other than linear kernels, we need to treat the feature space explicitly.

To formulate an LP support vector machine in the feature space, we define the decision function in the dual form as follows [215]:[3]

$$D(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i \, H(\mathbf{x}, \mathbf{x}_i) + b, \tag{4.45}$$

where α_i take on real values. Thus, unlike L1 support vector machines, in (4.45) we need not multiply $\alpha_i H(\mathbf{x}, \mathbf{x}_i)$ by y_i . Then we consider minimizing

$$Q(\boldsymbol{\alpha}, \boldsymbol{\xi}) = \sum_{i=1}^{M} (|\alpha_i| + C\,\xi_i) \tag{4.46}$$

subject to

$$y_j \left(\sum_{i=1}^{M} \alpha_i \, H(\mathbf{x}_j, \mathbf{x}_i) + b \right) \geq 1 - \xi_j \quad \text{for} \quad j = 1, \ldots, M. \tag{4.47}$$

Letting $\alpha_i = \alpha_i^+ - \alpha_i^-$ and $b = b^+ - b^-$, where $\alpha_i^+ \geq 0$, $\alpha_i^- \geq 0$, $b^+ \geq 0$, and $b^- \geq 0$, we can solve (4.46) and (4.47) for $\boldsymbol{\alpha}$, b, and $\boldsymbol{\xi}$ by linear programming. But because

$$\mathbf{w} = \sum_{i=1}^{M} \alpha_i \, \mathbf{g}(\mathbf{x}_i), \tag{4.48}$$

minimization of the sum of $|\alpha_i|$ does not lead to maximization of the margin measured in the L1 norm.

By this formulation, the number of variables is $3M + 2$ and the number of inequality constraints is M. Thus for a large number of training data, training becomes very slow even by linear programming. Therefore, we need to use decomposition techniques [38].

Example 4.1. Consider solving the problem in Example 2.17 by the LP support vector machine. Because the problem is linearly separable, (4.46) becomes

$$Q(\boldsymbol{\alpha}, \boldsymbol{\xi}) = |\alpha_1| + |\alpha_2| + |\alpha_3| + |\alpha_4| \tag{4.49}$$

[3]According to [98], by defining the mapping to the feature space by $\mathbf{g}(\mathbf{x}) = (H(\mathbf{x}_1, \mathbf{x}), \ldots, H(\mathbf{x}_M, \mathbf{x}))^T$, α_i are considered to be the weight coefficients in the feature space.

and (4.47) becomes

$$\alpha_1 - \alpha_4 + b \geq 1 \quad \text{for Datum 1,} \tag{4.50}$$
$$\alpha_2 - \alpha_3 + b \geq 1 \quad \text{for Datum 2,} \tag{4.51}$$
$$\alpha_2 - \alpha_3 - b \geq 1 \quad \text{for Datum 3,} \tag{4.52}$$
$$\alpha_1 - \alpha_4 - b \geq 1 \quad \text{for Datum 4.} \tag{4.53}$$

Because (4.49) is minimized, from (4.50) and (4.53) or (4.51) and (4.52), b needs to be 0. Thus, (4.50) to (4.53) reduce to

$$\alpha_1 - \alpha_4 \geq 1, \tag{4.54}$$
$$\alpha_2 - \alpha_3 \geq 1. \tag{4.55}$$

Therefore, (4.49) is minimized when

$$\alpha_1 = 1 - \beta_1, \qquad \alpha_4 = -\beta_1, \tag{4.56}$$
$$\alpha_2 = 1 - \beta_2, \qquad \alpha_3 = -\beta_2, \tag{4.57}$$

where $1 \geq \beta_1 \geq 0$ and $1 \geq \beta_2 \geq 0$.

Similar to the solution of the L1 support vector machine, the solution is nonunique and the decision boundary is given by

$$D(\mathbf{x}) = x_1 + x_2 = 0, \tag{4.58}$$

which is the same as that for the L1 support vector machine. Notice that for $\beta_1 = \beta_2 = 0$, α_1 and α_2, which belong to Class 1, are nonzero and α_3 and α_4 are zero. Thus, the definition of support vectors does not hold for LP support vector machines.

Similar to L1 and L2 support vector machines, LP support vector machines have degenerate solutions. Namely, α_i are all zero. The difference is that LP support vector machines have degenerate solutions when the value of C is small as the following theorem shows.

Theorem 4.2. *For the LP support vector machine, there exists a positive C_0 such that for $0 \leq C \leq C_0$ the solution is degenerate.*

Proof. Because of the slack variables ξ_i, (4.47) has a feasible solution. Thus, for some C, (4.46) and (4.47) have the optimal solution with some α_i being nonzero.

For $\boldsymbol{\alpha} = \mathbf{0}$, (4.47) reduces to

$$y_i\, b \geq 1 - \xi_i. \tag{4.59}$$

For $b = 0$, (4.59) is satisfied for $\xi_i = 1$. Then (4.46) is

$$Q(\boldsymbol{\alpha}, \boldsymbol{\xi}) = MC. \tag{4.60}$$

Thus, by decreasing the value of C from a large value, we can find a maximum value of C, C_0, in which (4.46) is minimized for $\boldsymbol{\alpha} = \mathbf{0}$. For $0 < C \leq C_0$, it is evident that $\boldsymbol{\alpha} = \mathbf{0}$ is the optimal solution for (4.46) and (4.47). ■

Zhou, Zhang, and Jiao [285] proposed a slightly different linear programming support vector machine as follows. Minimize

$$Q(r, \boldsymbol{\alpha}, \boldsymbol{\xi}) = -r + C \sum_{i=1}^{M} \xi_i \tag{4.61}$$

subject to

$$y_i \left(\sum_{j=1}^{M} \alpha_j \, H(\mathbf{x}_j, \mathbf{x}_i) + b \right) \geq r - \xi_i \qquad \text{for} \quad i = 1, \ldots, M, \tag{4.62}$$

$$r \geq 0, \tag{4.63}$$

$$-1 \leq \alpha_i \leq 1 \qquad \text{for} \quad i = 1, \ldots, M, \tag{4.64}$$

$$\xi_i \geq 0 \qquad \text{for} \quad i = 1, \ldots, M. \tag{4.65}$$

In [285], $\alpha_j \, y_j$ is used instead of α_j in (4.62). But this is not necessary because α_j may take on negative values. In conventional support vector machines r in (4.62) is 1, but here r is maximized under the constraints (4.64).

Similar to two-class support vector machines, we can easily extend two-class LP support vector machines to multiclass LP support vector machines, such as one-against-all, pairwise, and ECOC LP support vector machines. But because the extension is straightforward [7], we will not discuss the details here.

4.2.2 Performance Evaluation

In this section, we first show the performance of LP support vector machines (LP SVMs) for the two-class blood cell data used in Section 2.6.9. Then we show the performance for the multiclass data sets listed in Table 1.1.

Performance of Two-Class LP Support Vector Machines

Figure 4.3 shows the recognition rates and the number of support vectors against the polynomial degree with $C = 5000$. As the polynomial degree becomes higher, the recognition rate of the training data increases and reaches 100 percent for the polynomial degree of four. But the recognition rate of the test data reaches the maximum at the polynomial degree of two and decreases afterward.

Figure 4.4 shows the recognition rates and the number of support vectors against the RBF kernel parameter γ with $C = 5000$. As the value of γ becomes larger, the recognition rate of the training data increases. But the recognition rate of the test data reaches the maximum at $\gamma = 3$. The number of support vectors increases as the value of γ increases.

Figure 4.5 shows the recognition rates and the number of support vectors against C for RBF kernels with $\gamma = 1$. For $C = 0.1$, the recognition rates

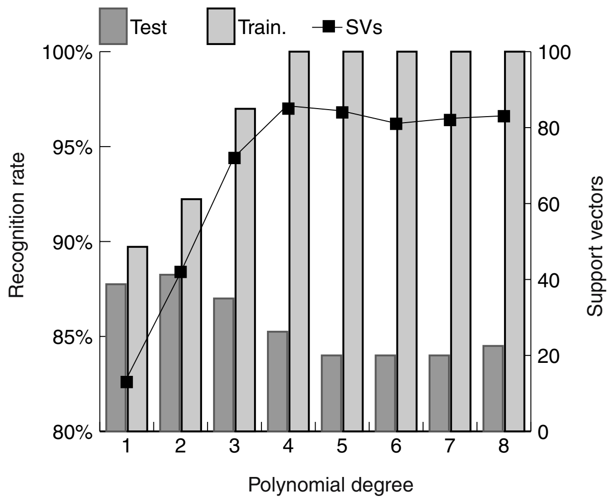


Fig. 4.3. Recognition rates and support vectors against a polynomial degree

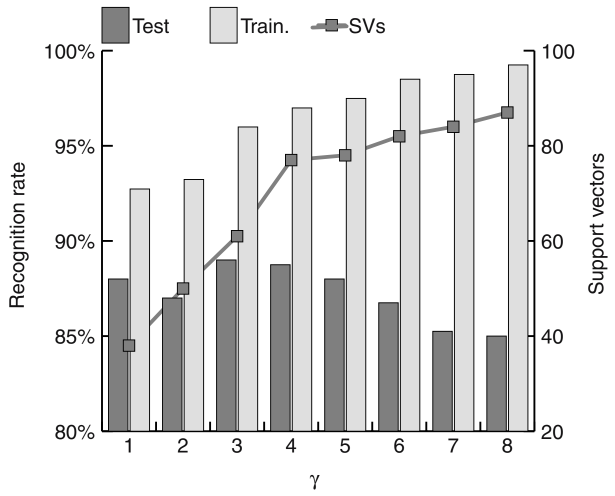


Fig. 4.4. Recognition rates and support vectors against γ

are very low due to the effect of the degenerate solution. For $C = 0.01$, the degenerate solution was obtained. The recognition rates of the test data do not vary very much for C larger than or equal to 1.

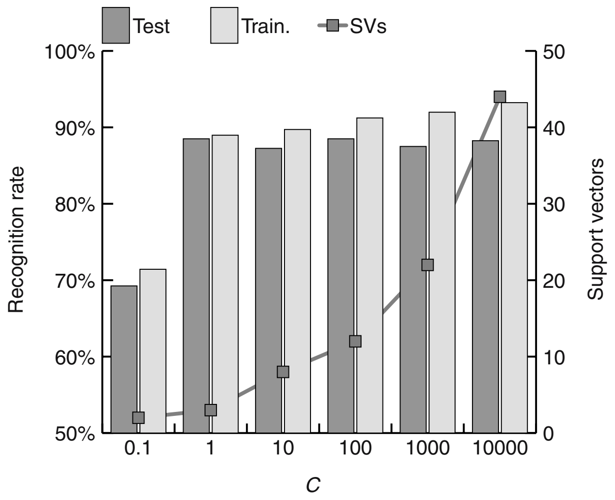


Fig. 4.5. Recognition rates and support vectors against C

Performance of Multiclass LP Support Vector Machines

Here we show the performance of one-against-all and pairwise fuzzy LP SVMs. We determined the kernels and parameters by five-fold cross-validation except for one-against-all fuzzy LP SVMs for the thyroid, blood cell, and hiragana data sets. For each following kernel, we set the value of C from 1, 10, 50, 100, 500, 1000, 2000, 3000, 5000, 7000, 10,000, and 100,000 and determined the optimal kernel and the parameters:

1. linear kernels and polynomial kernels with degrees 2, 3, and 4;
2. RBF kernels with $\gamma = 0.1$, 1, and 10; and
3. neural network kernels with $\nu = 0.2$ and $a = 0.1$.

Here, we used neural network kernels because kernels need not be positive semidefinite.

Because we used the simplex method, we could not determine the optimal parameters of the one-against-all LP SVM, by cross-validation, for data sets other than iris and numeral data sets. For these data sets we trained the LP SVM without cross-validation and selected the kernel and the value of C with the highest recognition rates of the test data. (For hiragana-105 data set, we set $C = 10{,}000$.) But because cross-validation worked well, this did not favor one-against-all LP SVM very much.

Tables 4.4 and 4.5 show the parameters, recognition rates of the one-against-all LP SVM and pairwise LP SVM, and the average number of support vectors per decision function. The recognition rates of the training data are shown in parentheses when they are not 100 percent. For each data set, the maximum recognition rate of the test data is shown in boldface. From the tables, the recognition rates of the test data of both methods are comparable.

Similar to the one-against-all fuzzy SVMs, the recognition rates of the fuzzy LP SVMs with minimum and average operators were the same. Comparing to Tables 3.2 and 3.3, the recognition rates of the test data are comparable but a little lower. But the numbers of support vectors of LP SVMs are smaller. This is because in LP SVMs, the sum of $|\alpha_i|$ is minimized. Thus it leads to fewer support vectors.

Similar to the pairwise fuzzy SVMs, the recognition rates of the fuzzy LP SVMs with minimum and average operators are almost the same. Comparing to Tables 3.2 and 3.3, the recognition rates of the test data by fuzzy LP SVMs are slightly lower but the numbers of support vectors are smaller.

4.3 Incremental Training

Support vector machines are suited for incremental training due to the fact that only support vectors are necessary for training [185, 198, 222, 273]. When new incremental training data are obtained we estimate the candidates of support vectors, and using only these data we train the support vector machine. In [273], the least-recently used (LRU) strategy, which is a page replacement algorithm for paged memory allocation of a computer, is used for discarding the least-recently used data for training the support vectors. In [222], initial value selection for incremental training is considered using all the data previously used for training, but selection of the training data is not considered. Namely, when new data are added, we "hot start" the training from the old solution plus the initial values for the added data with $\alpha_i = 0$ for $y_i\, D(\mathbf{x}_i) > 1$ and $\alpha_i = C$ for $y_i\, D(\mathbf{x}_i) \le 1$. In [198], when a new datum $\mathbf{x}_t$ with $y_t D(\mathbf{x}_t) \le 1$ is obtained, incremental training using the working set that includes $\mathbf{x}_t$ and the data near $\mathbf{x}_t$ is performed.

Here, exploiting the properties of support vector machines, we discuss an incremental training method, which is slightly different from [273], for a one-against-all support vector machine. First we explain how to estimate the candidates of support vectors using Fig. 4.6. In the figure, assume that the data shown in filled circles and rectangles are newly obtained training data. The optimal hyperplane $D_i(\mathbf{x}) = 0$ that separates class i from the remaining classes was determined using the data excluding those data. Then if we retrain the support vector machine, the data that satisfy $y(\mathbf{x})\, D_i(\mathbf{x}) \le 1$ are candidates for support vectors where $y(\mathbf{x}) = 1$ when $\mathbf{x}$ belongs to class i and $y(\mathbf{x}) = -1$, otherwise. In addition, the data that satisfy $y(\mathbf{x})\, D_i(\mathbf{x}) > 1$ but are near $y(\mathbf{x})\, D_i(\mathbf{x}) = 1$ can be support vectors. Thus we determine that the data are candidates for support vectors if

$$y(\mathbf{x})\, D_i(\mathbf{x}) \le \beta + 1 \tag{4.66}$$

Table 4.4. Performance comparison of LP support vector machines

Data	Item	One-against-all LP SVM	Pairwise LP SVM
Iris	Parm	γ0.1, C100	γ1, C5000
	Dis.	92.00 (94.67)	92.00
	Min	**94.67** (97.33)	92.00
	Avg.	**94.67** (97.33)	92.00
	SVs	3	3
Numeral	Parm	d2, C1	γ10, C5000
	Dis.	99.27 (99.63)	99.39
	Min	**99.51** (99.75)	99.39
	Avg.	**99.51** (99.75)	99.39
	SVs	7	4
Thyroid	Parm	d3, C500	Linear, C5000
	Dis.	95.36 (99.18)	97.35 (98.59)
	Min	97.58 (99.66)	97.64 (98.78)
	Avg.	97.58 (99.66)	**97.67** (98.78)
	SVs	76	14
Blood cell	Parm	γ10, C50	γ10, C50
	Dis.	88.10 (93.57)	91.90 (97.16)
	Min	**92.90** (96.51)	92.52 (97.26)
	Avg.	**92.90** (96.51)	92.65 (97.32)
	SVs	28	6
Hiragana-50	Parm	γ10, C1000	γ10, C50
	Dis.	94.69	97.44 (99.93)
	Min	**98.13**	97.87 (99.98)
	Avg.	**98.13**	97.79
	SVs	31	8
Hiragana-13	Parm	γ10, $C10^4$	nn, C5000
	Dis.	96.15	98.79 (99.81)
	Min	**99.23**	99.13 (99.87)
	Avg.	**99.23**	99.16 (99.93)
	SVs	29	4

Table 4.5. Performance comparison of LP support vector machines

Data	Item	One-against-all LP SVM	Pairwise LP SVM
Hiragana-105	Parm	$\gamma 10$, $C 10^4$	$\gamma 10$, $C 50$
	Dis.	99.84	**100**
	Min	**100**	**100**
	Avg.	**100**	**100**
	SVs	39	9

is satisfied, where $\beta\,(>0)$ is a user-defined parameter. But if all the new data satisfy

$$y(\mathbf{x})\,D_i(\mathbf{x}) \geq 1, \tag{4.67}$$

retraining of the support vector machine adding the new data will give the same support vector machine. Thus in this case, we only need to add the data that satisfy (4.66) to the training data for future training. (The same idea is discussed in "practical considerations" of [51].)

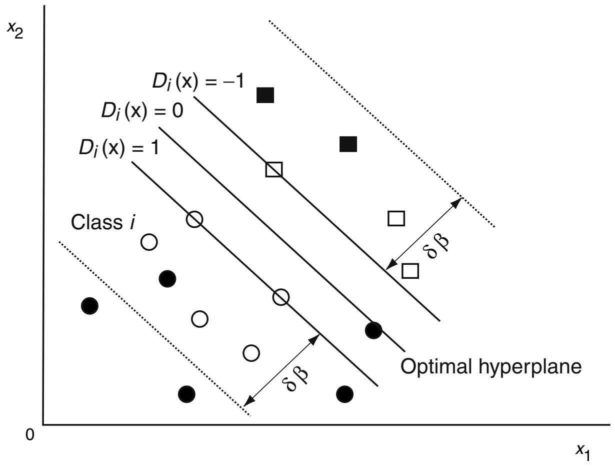


Fig. 4.6. Estimating the candidates for support vectors

Assume that we have a training set X_a in addition to the set X_c that was used for training the current classifier. The general procedure for incremental training for an n-class problem is as follows:

1. Initialize the set S, i.e., $S = \phi$.

2. If for $\mathbf{x} \in X_a$, $y(\mathbf{x})\, D_i(\mathbf{x}) < \beta + 1$ is satisfied for $i\, (i \in \{1, \ldots, n\})$, add $\mathbf{x}$ to the set S, i.e., $S = S \cup \{\mathbf{x}\}$.
3. Add S to X_c, i.e., $X_c = X_c \cup S$.
4. If all the data in X_a satisfy (4.67), we do not retrain the support vector machine. Otherwise, go to Step 5.
5. Using X_c we retrain the support vector machine. After the training, if for $\mathbf{x} \in X_c$, $y(\mathbf{x})\, D_i(\mathbf{x}) > \beta + 1$ is satisfied for all i $(i = 1, \ldots, n)$, we delete $\mathbf{x}$ from the set S, i.e., $S = S - \{\mathbf{x}\}$.

Because $\delta\, \|\mathbf{w}_i\| = 1$, (4.66) is rewritten as follows:

$$\frac{y(\mathbf{x})\, D_i(\mathbf{x})}{\|\mathbf{w}_i\|} \leq \delta\, (\beta + 1), \tag{4.68}$$

where δ is the margin. Namely, the distance of $\mathbf{x}$ from the hyperplane $y(\mathbf{x})\, D_i(\mathbf{x}) = 1$ is $\delta\, \beta$. As the new training data are added to the training data, the value of the margin δ decreases or remains the same. Thus, the regions that extract support vector candidates are shrunk as the incremental training proceeds even for the fixed value of β.

4.4 Robust Support Vector Machines

Compared to conventional classifiers, support vector machines are robust for outliers because of the margin parameter that controls the trade-off between the generalization ability and the training error. But there are several approaches to enhance robustness of support vector machines [57, 110, 139, 176, 265, 267, 284].

Herbrich and Weston [110, 265, 267] introduced an adaptive margin for each training datum so that the margins for outliers become large and hence the classification becomes robust.

When outliers are included, the median of data is known to be more robust than the center of data [3]. Kou, Xu, Zhang, and Ji [139] proposed median support vector machines (MSVM), an improved version of central support vector machines [284], to improve robustness of support vector machines. Instead of maximizing the separating margin of two classes, in MSVMs, the sum of distances of the class medians to the decision boundary is maximized.

4.5 Bayesian Support Vector Machines

To improve generalization ability of support vector machines, Burges and Schölkopf [47] introduced virtual support vectors. First, train a support vector machine using a training data set. Then obtain virtual support vectors by applying, to the obtained support vectors, linear transformations, such as translation, scaling, and rotation for character recognition. Finally, train another support vector machine using the virtual support vectors. By this method, the generalization ability for the handwritten numeral classification was improved but the classification time was increased due to the increase in the number of support vectors. To shorten the classification time, Burges and Schölkopf [47] proposed approximating the decision function with a smaller set of data.

To improve class separability, Amari and Wu [15, 16] proposed enlarging the regions around support vectors by replacing $H(\mathbf{x}, \mathbf{x}')$ with

$$\tilde{H}(\mathbf{x}, \mathbf{x}') = c(\mathbf{x})\, c(\mathbf{x}')\, H(\mathbf{x}, \mathbf{x}'), \tag{4.69}$$

where $c(\mathbf{x})$ is a positive scalar function, and one example is

$$c(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i \, \exp(-\|\mathbf{x} - \mathbf{x}_i\|/2\tau^2). \tag{4.70}$$

Here, τ is a parameter. It is shown that by (4.69) the volumes around support vectors are expanded in the feature space. Thus class separability is increased.

In LS support vector machines, after determination of the bias term of a hyperplane, it is further tuned to improve classification accuracy [157]. A similar technique is also discussed in [81]. In the following, we discuss the generalization improvement by optimizing the bias term based on the Bayes' theory [117]. We call this support vector machine the *Bayesian support vector machine.*

In support vector machines, the optimal hyperplane is placed in the middle of the support vectors belonging to different classes. This is because the training data are generated by a single unknown probability distribution. Thus, if the distribution of each class is known, we can improve the generalization ability by the Bayes' theory. In the following we assume that the distribution of the class data in the direction perpendicular to the optimal hyperplane is normal. Under the assumption, the optimal separating hyperplane is no longer optimal; the boundary determined by the Bayes' theory becomes optimal. Thus, by the parallel displacement of the optimal hyperplane to the position determined by the Bayes' theory, we can improve the generalization ability.

In the following, we first discuss the one-dimensional Bayesian decision function. Then we move the optimal hyperplane in parallel so that the class boundary becomes the boundary given by the Bayesian decision function, and we discuss how to test whether the class data are normal.

4.5.1 One-Dimensional Bayesian Decision Functions

Let two classes be $C_1,\ C_2$, where C_1 is on the positive side of the hyperplane and C_2 on the negative side. The posterior probability that the observed $\mathbf{x}$ belongs to class C_k, $P(C_k \,|\, \mathbf{x})$, is given by

$$P(C_k \,|\, \mathbf{x}) = \frac{P(C_k)\, p(\mathbf{x} \,|\, C_k)}{p(\mathbf{x})}, \tag{4.71}$$

where $P(C_k)$ is the a priori probability of class C_k, $p(\mathbf{x} \,|\, C_k)$ is the conditional probability density function when $\mathbf{x}$ belonging to class C_k is observed, and $p(\mathbf{x})$ is a probability density function given by

$$p(\mathbf{x}) = \sum_{k=1}^{2} P(C_k)\, p(\mathbf{x} \,|\, C_k), \qquad \int p(\mathbf{x})\, d\mathbf{x} = 1. \tag{4.72}$$

Equation (4.71) is called *Bayes' rule.*

We assume that one-dimensional data, which belong to Class C_k, obey the normal distribution with the mean μ_k and the variance σ_k^2 given by

$$p(x \,|\, C_k) = \frac{1}{\sqrt{2\,\pi\,\sigma_k^2}} \exp\left(-\frac{1}{2\,\sigma_k^2}\,(x-\mu_k)^2\right). \tag{4.73}$$

According to Bayes' theory, the optimal classification is to classify the datum to the class with the maximum posterior probability. Instead of comparing the posterior probabilities, we compare the logarithms of the posterior probabilities, deleting the term $p(x)$ common to the classes:

$$g_k(x) = \log P(C_k) - \frac{1}{2}\left(\log(2\,\pi\,\sigma_k^2) + \frac{1}{\sigma_k^2}\,(x-\mu_k)^2\right). \tag{4.74}$$

Thus the Bayesian decision function for the two classes is given by

$$\begin{aligned} D_{\text{bayes}}(x) &= g_1(x) - g_2(x) \\ &= \log \frac{P(C_1)}{P(C_2)} \\ &\quad -\frac{1}{2}\left(\log\frac{\sigma_1^2}{\sigma_2^2} + \frac{1}{\sigma_1^2}\,(x-\mu_1)^2 - \frac{1}{\sigma_2^2}\,(x-\mu_2)^2\right). \end{aligned} \tag{4.75}$$

If $D_{\text{bayes}}(x) > 0$, x is classified into class C_1, and if $D_{\text{bayes}}(x) < 0$, class C_2.

4.5.2 Parallel Displacement of a Hyperplane

We move the hyperplane obtained by training an L1 support vector machine, in parallel, by the Bayesian decision function.

Let the optimal hyperplane be

$$D(\mathbf{x}) = \mathbf{w}^T \mathbf{g}(\mathbf{x}) + b = 0, \tag{4.76}$$

where $\mathbf{w}$ is the l-dimensional weight vector, $\mathbf{g}(\mathbf{x})$ is the mapping function from $\mathbf{x}$ to the l-dimensional feature space, and b is the bias term.

First we calculate the component $(x_e)_j$ of datum $\mathbf{x}_j$ in the direction orthogonal to the separating hyperplane:

$$(x_e)_j = \frac{\mathbf{w}^T \mathbf{g}(\mathbf{x}_j)}{\|\mathbf{w}\|} = \sum_{i=1}^{M} \delta\, \alpha_i\, y_i\, H(\mathbf{x}_i, \mathbf{x}_j), \tag{4.77}$$

where δ is the margin and $\delta\|\mathbf{w}\| = 1$, $y_i = 1$ if $\mathbf{x}_i$ belongs to Class 1 and -1 otherwise, and α_i is a dual variable associated with $\mathbf{x}_i$. Then we calculate the center

$$\mu_k = \frac{1}{n_k} \sum_{j=1}^{n_k} (x_e)_j$$

and the variance

$$v_k = \frac{1}{n_k} \sum_{j=1}^{n_k} \left((x_e)_j - \mu_k\right)^2$$

for class C_k, where n_k is the number of class C_k data. Substituting these data into (4.75), we calculate the bias term of the Bayes' decision boundary, b_{bayes}. Namely, we solve

$$D_{\text{bayes}}(x) = 0 \tag{4.78}$$

for x and obtain x_{b_1}, $x_{b_2} (x_{b_1} > x_{b_2})$. Then because $b_{\text{bayes}} = -x_{b_1}$, we replace b with b_{bayes}. Here, we set $P(C_k)$ as follows:

$$P(C_k) = \frac{\text{Number of data belonging to } C_k}{\text{Total number of data}}. \tag{4.79}$$

If the recognition rate of the training data is decreased by the parallel displacement of the optimal hyperplane, we do not move the hyperplane so that the generalization ability is not worsened.

4.5.3 Normal Test

To move the optimal hyperplane, we first need to test if the distribution of training data in the direction orthogonal to the hyperplane is normal. The procedure is as follows.

First, using the number of data belonging to class C_k $(k = 1, 2)$ and the center μ_k, we determine the number of divisions and the width and generate a frequency distribution. Assume that we divided the interval into p divisions. Let the probability that division j occurs be p_j and the probability assuming the normal distribution be π_j. We set the hypothesis:

$$\begin{aligned} H_0 :&\quad p_j = \pi_j \quad \text{for} \quad j = 1, \dots, p, \\ H_1 :&\quad p_j \neq \pi_j \quad \text{for} \quad j \in \{1, \dots, p\}. \end{aligned}$$

If hypothesis H_0 holds, the normal distribution is assumed. Let among n_k observations, f_j observations belong to division j and the number of expected observations under the normal distribution be $e_j = n_k \pi_j$. Then the deviation from the expected observation

$$x^2 = \sum_{j=1}^{p} \frac{(f_j - e_j)^2}{e_j} = \sum_{j=1}^{p} \frac{f_j^2}{e_j} - n_k \tag{4.80}$$

obeys the χ-square distribution with freedom $\nu = p - r - 1$ under the hypothesis H_0, where r is the number of independent variables. Because we use the center and variances, $r = 2$. Therefore, we calculate the value of x^2 and that if it is not within the critical region with the level of significance s,

$$R = \{x^2 \,|\, x^2 > \chi^2_{p-3}(s)\}, \tag{4.81}$$

we judge that the hypothesis that the distribution is normal is correct. If not, it is not normal.

4.6 Committee Machines

A committee machine, which is based on the principle of divide and conquer, consists of a number of classifiers with different classification powers, whose decisions are combined into a single decision by some strategy such as voting [108, 271]. A boosting machine is one type of committee machine, and each machine is trained using a different subset of training data.

There is much work on committee machines using support vector machines. For example, Martinez and Millerioux [161] used support vector machines with different kernels. Committee machines can alleviate the long training time caused by large training data by splitting the training data into approximately equal-size subsets, each of which constitutes a training data for each committee machine [217].

4.7 Confidence Level

Conventional support vector machines give a prediction result but without its confidence level. One way to give this is the introduction of fuzzy membership functions as discussed in Chapter 3.

Another approach is to obtain the confidence level assuming that the training pairs are generated by an independent and identically distributed (i.i.d.) process [92, 263]. Platt [191] proposed, in addition to support vector machine training, to train the sigmoid function that maps the support vector machine outputs into posterior probabilities. This method is extended to multiclass problems by Frasconi, Passerini, and Vullo [85].

Fumera and Roli [91] modified the objective function of the support vector machine so that the rejection region is obtained.

4.8 Visualization

Interpretability of the classifier or explanation of the decision process is especially important for decision support systems such as medical diagnosis. Without it, a classifier would not be used even if it has high generalization ability.[4] Support vector machines have high generalization ability compared to other classifiers but interpretability is relatively low, especially when nonlinear kernels are used. Thus to improve interpretability [49, 89, 179] is very important for the support vector machines to be used in such a field.

[4]Discussions with Prof. H. Motoda.

5
Training Methods

In training an L1 or L2 support vector machine, we need to solve the quadratic programming problem with the number of variables equal to the number of training data. Thus when the number of training data is very large, training takes time. To speed up training, several methods have been proposed, which are classified into two approaches. One is to extract support vector candidates and then train the support vector machine using these data. The other is to accelerate training by decomposition techniques, in which a small problem is repeatedly solved.

In this chapter, we discuss preselection of support vector candidates and training methods using decomposition techniques. We explain widely used primal-dual interior-point training methods and steepest ascent training methods, which are extensions of the sequential minimal optimization technique (SMO). We also discuss training methods for LP support vector machines.

5.1 Preselecting Support Vector Candidates

According to the architecture of the support vector machine, only the training data near the boundaries are necessary. In addition, because the training time becomes longer as the number of training data increases, the training time is shortened if the data far from the boundary are deleted. Therefore, if we can delete unnecessary data from the training data efficiently prior to training, we can speed up training. Several approaches have been developed to preselect support vector candidates [9, 71, 225, 232, 233, 277, 283]. In [71], all the training data are first clustered by the k-means clustering algorithm. Then if a generated cluster includes data that belong to the same class, the data are discarded. But if a generated cluster includes data with different classes, these data are retained for training, assuming that they may include support vectors. The k-means clustering algorithm is not a requisite. It may be the fuzzy c means clustering algorithm or the Kohonen network. This method works when classes overlap, but if classes are well separated, it may not extract boundary data. In [225, 232], k nearest neighbors are used to detect the boundary data.

In [277], a superset of support vectors called *guard vectors* is extracted by linear programming. Assume that the set of training input-output pairs $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)\}$ is linearly separable, where y_i are either 1 or -1. Then there exists a separating hyperplane that includes a datum $\mathbf{x}_i$:

$$\begin{aligned} y_1 \left(\mathbf{w}^T \mathbf{x}_1 + b\right) &\geq 0, \\ &\cdots\cdots\cdots \\ y_{i-1} \left(\mathbf{w}^T \mathbf{x}_{i-1} + b\right) &\geq 0, \\ y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right) &= 0, \\ y_{i+1} \left(\mathbf{w}^T \mathbf{x}_{i+1} + b\right) &\geq 0, \\ &\cdots\cdots\cdots \\ y_M \left(\mathbf{w}^T \mathbf{x}_M + b\right) &\geq 0. \end{aligned} \tag{5.1}$$

We call the datum $\mathbf{x}_i$ that satisfies (5.1) the *guard vector*. Clearly the support vectors are guard vectors but the reverse is not true. Thus the set of guard vectors is a superset of the set of support vectors. To obtain the set of guard vectors, we solve (5.1) by linear programming, changing i from 1 to M. Unlike usual linear programming we need only to check whether $\mathbf{x}_i$ is a feasible solution of (5.1). If the training data set is not linearly separable, we need to map the original input space into the feature space. But because the kernel LP has not been developed yet, in [277], the original input space is extended to the $(n+M)$-dimensional space where the $(n+i)$th element is 1 for $\mathbf{x}_i$ and 0 otherwise.

In this section we discuss speedup of training by deleting unnecessary training data [9]. We estimate the data near the boundaries using the classifier based on the Mahalanobis distance and extracting the misclassified data and the data near the boundaries.

5.1.1 Approximation of Boundary Data

The decision boundaries of the classifier using the Mahalanobis distance are expressed by the polynomials of the input variables with degree 2. Therefore, the boundary data given by the classifier are supposed to approximate the boundary data for the support vector machine, especially with the polynomial kernels with degree 2.

For the class i data $\mathbf{x}$, the Mahalanobis distance $d_i(\mathbf{x})$ is given by

$$d_i^2(\mathbf{x}) = (\mathbf{c}_i - \mathbf{x})^T Q_i^{-1} (\mathbf{c}_i - \mathbf{x}), \tag{5.2}$$

where $\mathbf{c}_i$ and Q_i are the center vector and the covariance matrix for the data belonging to class i, respectively:

$$\mathbf{c}_i = \frac{1}{|X_i|} \sum_{\mathbf{x} \in X_i} \mathbf{x}, \tag{5.3}$$

$$Q_i = \frac{1}{|X_i|} \sum_{\mathbf{x} \in X_i} (\mathbf{x} - \mathbf{c}_i)\,(\mathbf{x} - \mathbf{c}_i)^T. \tag{5.4}$$

Here, X_i denotes the set of data belonging to class i and $|X_i|$ is the number of data in the set. The data $\mathbf{x}$ is classified into the class with the minimum Mahalanobis distance. The most important feature of the Mahalanobis distance is that it is invariant for linear transformation of input variables. Therefore, we do not worry about the scaling of each input variable.

For the datum belonging to class i, we check whether

$$r(\mathbf{x}) = \frac{\displaystyle\min_{j \neq i, j=1,\ldots,n} \quad d_j(\mathbf{x}) - d_i(\mathbf{x})}{d_i(\mathbf{x})} \leq \eta \tag{5.5}$$

is satisfied, where $r(\mathbf{x})$ is the relative difference of distances, $\eta\,(> 0)$ controls the nearness to the boundary, and $0 < \eta < 1$. If $r(\mathbf{x})$ is negative, the datum is misclassified. We assume that the misclassified data are near the decision boundary. Equation (5.5) is satisfied when the second minimum Mahalanobis distance is shorter than or equal to $(1 + \eta)\, d_i(\mathbf{x})$ for the correctly classified $\mathbf{x}$.

In extracting boundary data, we set some appropriate value to η and for each class we select the boundary data whose number is between $N_{\min}$ and $N_{\max}$. Here the minimum number is set so that the number of boundary data is not too small for some classes because the data that satisfy (5.5) are scarce. The maximum number is set not to allow too many data to be selected. A general procedure for extracting boundary data is as follows:

1. Calculate the centers and covariance matrices for all the classes using (5.3) and (5.4).
2. For the training datum $\mathbf{x}$ belonging to class i, calculate $r(\mathbf{x})$ and put the data into the stack for class i, S_i, whose elements are sorted in increasing order of the value of $r(\mathbf{x})$ and whose maximum length is $N_{\max}$. Iterate this for all the training data.
3. If stack S_i includes more than $N_{\min}$ data that satisfy (5.5), select these data as the boundary data for class i. Otherwise, select the first $N_{\min}$ data as the boundary data.

This procedure is refined according to the architecture of the multiclass support vector machine. If we use the pairwise classification, in determining the decision boundary for classes i and j, we calculate $r(\mathbf{x})$ only for classes i and j. If we use the one-against-all multiclass architecture, in determining the decision boundary for class i and the remaining classes, we assume that the remaining classes consist of $n - 1$ clusters and use (5.5) to extract boundary data.

5.1.2 Performance Evaluation

Although the performance varies as kernels vary, the polynomial kernels with degree 2 perform relatively well. Thus in the following study, we use the polynomials with degree 2 as the kernels. We evaluated the method using the iris data, blood cell data, and hiragana data listed in Table 1.1. Except for the iris data, we set $N_{\max}$ as the half of the maximum number of class data, namely 200. And we set $N_{\min} = 50$ and evaluated the performance changing η.

We ran the software developed by Royal Holloway, University of London [210], on a SUN UltraSPARC-IIi (335MHz) workstation. The software used pairwise classification. The training was done without decomposition. The data, which were originally scaled in [0, 1], were rescaled in [−1, 1].

Because the number of the iris data is small, we checked only the lowest rankings, in the relative difference of the Mahalanobis distances, of support vectors for the pairs of classes. Table 5.1 lists the results when the boundary data were extracted for each class. The numeral in the ith row and the jth column shows the lowest ranking of the support vectors, belonging to class i, for a pair of classes i and j. The diagonal elements show the number of training data for the associated class. The maximum value among the lowest rankings was 8, which was smaller than half the number of class data. Thus, the relative difference of the Mahalanobis distances well reflected the boundary data.

Table 5.1. The lowest rankings of support vectors for the iris data

Class	1	2	3
1	(25)	1	2
2	8	(25)	3
3	2	3	(25)

Table 5.2 lists the results for the blood cell and hiragana data sets. The column "Selected" lists the number of data selected and the first row in each data set shows the results when all the training data were used. The numerals in parentheses in the "Time" column show the time for extracting boundary data and the speedup ratios were calculated, including the time for extracting boundary data.

For $\eta = 2$ to 4, two to five times speedup was obtained without deteriorating the recognition rates very much.

Table 5.2. Performance for the blood cell and hiragana data ($N_{\max} = 200$ and $N_{\min} = 50$)

Data	η	Selected	Rate (%)	Time (s)	Speedup
Blood cell	—	3097	92.13 (99.32)	924	1
	2.0	2102	92.13 (99.29)	448 (2)	2.1
Hiragana-50	—	4610	98.91 (100)	2862	1
	2.0	2100	97.11 (98.68)	644 (28)	4.2
	4.0	3611	98.79 (100)	1690 (28)	1.7
Hiragana-105	—	8375	100 (100)	11,656	1
	2.0	2824	99.61 (99.65)	1908 (177)	5.6
	3.0	5231	99.99 (99.99)	5121 (187)	2.2
Hiragana-13	—	8375	99.57 (100)	9183	1
	2.0	4366	99.56 (100)	2219 (16)	4.1

5.2 Decomposition Techniques

To reduce the number of variables in training, Osuna, Freund, and Girosi [182] proposed decomposing the problem into two. Let the index set $\{1, \ldots, M\}$ be partitioned into two sets W and N, where $W \cup N = \{1, \ldots, M\}$ and $W \cap N = \phi$. Then decomposing $\{\alpha_i \,|\, i = 1, \ldots, M\}$ into $\boldsymbol{\alpha}_W = \{\alpha_i \,|\, i \in W\}$ and $\boldsymbol{\alpha}_N = \{\alpha_i \,|\, i \in N\}$, we define the following subproblem. Fixing $\boldsymbol{\alpha}_N$, maximize

$$\begin{aligned} Q(\boldsymbol{\alpha}_W) = & \sum_{i \in W} \alpha_i - \frac{1}{2} \sum_{i,j \in W} \alpha_i \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) \\ & - \sum_{\substack{i \in W, \\ j \in N}} \alpha_i \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) \\ & - \frac{1}{2} \sum_{i,j \in N} \alpha_i \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i \in N} \alpha_i \end{aligned} \tag{5.6}$$

subject to the constraints

$$\sum_{i \in W} y_i \, \alpha_i = - \sum_{i \in N} y_i \, \alpha_i, \qquad 0 \le \alpha_i \le C \quad \text{for} \quad i \in W. \tag{5.7}$$

Because the equality constraint in (5.7) is satisfied for $|W| \ge 2$, the minimum number of $|W|$ is two.

A decomposition technique is usually called *chunking*; there are variable-size chunking and fixed-size chunking [210]. A procedure for the variable-size chunking is as follows:

1. Set F points from the training data set to W, where F is a fixed positive integer.
2. Solve the subproblem for $\boldsymbol{\alpha}_W$.
3. Delete the variables except for the support vector candidates and add F points that do not satisfy the KKT conditions and go to Step 2. Otherwise, terminate the algorithm.

By this algorithm the working set size increases as the iteration proceeds. The procedure of the fixed-size chunking is as follows [182]:

1. Select $|W|$ points from the training data set.
2. Solve the subproblem for $\boldsymbol{\alpha}_W$.
3. If there exist such $\alpha_j \, (\in \boldsymbol{\alpha}_N)$ that do not satisfy the KKT conditions, replace any $\alpha_i \, (\in \boldsymbol{\alpha}_W)$ with α_j and go to Step 2. Otherwise, terminate the algorithm.

The convergence characteristics of decomposition techniques have been investigated [131, 152, 153]. Especially for the working set size of two, if the variables that violate the KKT conditions most are selected, the asymptotic convergence of the algorithm is guaranteed [131, 153].

In [207], to speed up the chunking algorithm, two methods for estimating the support vector candidates, which are used for the first chunk, are proposed. The procedure of the method that performed better for benchmark data sets is as follows:

1. For each datum belonging to Class 1, find the datum belonging to Class 2 with the minimum distance in the feature space (see Fig. 5.1 (a)).
2. Among the selected pairs with the same Class 2 datum, select the pair with the minimum distance as the support vector candidates (see Fig. 5.1 (b)).
3. Iterate Steps 1 and 2 exchanging Class 1 and Class 2.
4. If the obtained data are not sufficient, delete the obtained data from the training data and iterate Steps 1 to 3 several times.

For the thyroid data, for the chunk sizes from 10 to 480, the training time was reduced by 12.3 percent on the average (50 percent in maximum) over the random chunking algorithm.

This algorithm can be used for preselection of training data [137]. Figure 5.2 shows the results when the algorithm was used for preselecting the data. In Step 1, the distance was measured in the input space, instead of the feature space, and Step 2 was not executed. The figure shows the recognition rates of the thyroid test data when the training data were preselected and all the training data were used for training the one-against-all support vector machine. The rate of support vector selection shows the number of support

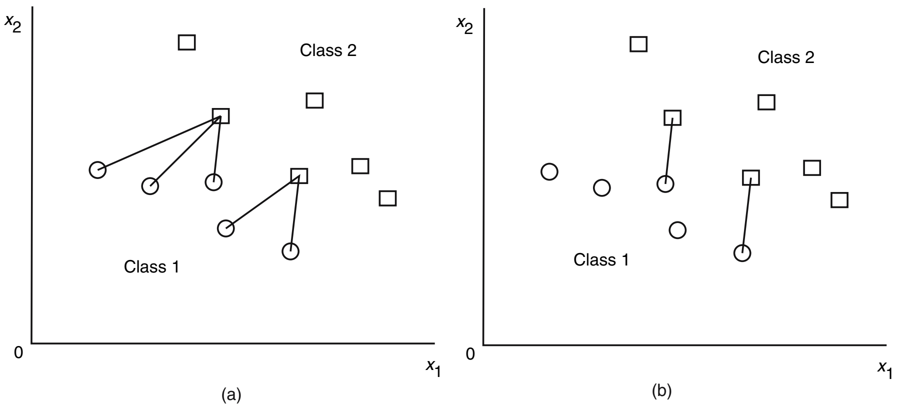


Fig. 5.1. Selection of support vector candidates: (**a**) For each Class 1 datum select a pair belonging to Class 2 with the minimum distance. (b) For the selected pairs with the same Class 2 datum, select the pair with the minimum distance

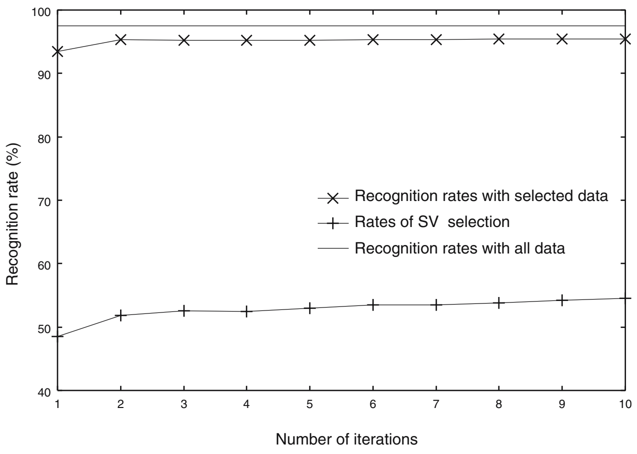


Fig. 5.2. Recognition rate by preselecting the thyroid training data ($d = 3, C =$ 10,000)

vectors, which are included in the selected data, divided by the number of support vectors for all the training data. By one iteration of the algorithm, the recognition rate was 4 percent lower than using all the training data, and for two to three iterations, the recognition rate was 2 percent lower. The training time with three iterations was 5 seconds by a Pentium III 1G personal computer, compared to the 40 seconds using all the training data. But even if we increased the iterations, the recognition rate did not reach the rate by using all the training data. This is reflected to the rate of support vector selection.

For the one-against-all support vector machine, one class has a smaller number of training data than the other. Thus if the selected data are deleted in Step 4, the training data for one class become extinct before the support vectors for the other class are selected. Thus, to prevent this, we stop deleting the training data for one class. Figure 5.3 shows the result for the modified algorithm. After nine iterations, the recognition rates are almost the same, and the training time including the selection time was 29 seconds, and the selected data was one third of the training data.

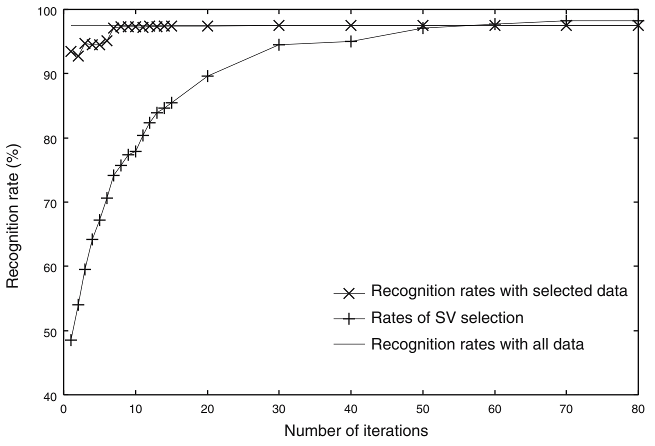


Fig. 5.3. Recognition rate by preselecting the thyroid training data without deleting the data for one class ($d = 3, C = 10{,}000$)

5.3 KKT Conditions Revisited

The KKT conditions for L1 support vector machines given by (2.63) to (2.65) are satisfied for the optimal solution. But because the bias term b is not included in the dual problem, if we detect the data that violate the KKT conditions using these equations, we need to estimate b. Thus, it will lead to incorrect detection. We call the KKT conditions including the bias term *inexact KKT conditions.* To avoid this, we need to redefine the KKT conditions for the dual problem that do not include b. We call them *exact KKT conditions.* In the following, we discuss the KKT conditions for the dual problem based on Keerthi and Gilbert [131].

We define F_i by

$$F_i = y_i - \sum_{j=1}^{M} y_j \, \alpha_j \, H(\mathbf{x}_i, \mathbf{x}_j). \tag{5.8}$$

We can classify the KKT conditions into the following three cases:

1. $\alpha_i = 0$
$$y_i\, b \geq y_i\, F_i,$$
2. $0 < \alpha_i < C$
$$b = F_i,$$
3. $\alpha_i = C$
$$y_i\, b \leq y_i\, F_i.$$

Then we define $\tilde{F}_i, \bar{F}_i$ as follows:

$$\begin{aligned} \tilde{F}_i = F_i \quad &\text{if} \quad (y_i = 1, \alpha_i = 0), \quad 0 < \alpha_i < C \\ &\quad\text{or} \quad (y_i = -1, \alpha_i = C), \end{aligned} \tag{5.9}$$

$$\begin{aligned} \bar{F}_i = F_i \quad &\text{if} \quad (y_i = -1, \alpha_i = 0), \quad 0 < \alpha_i < C \\ &\quad\text{or} \quad (y_i = 1, \alpha_i = C). \end{aligned} \tag{5.10}$$

Then the KKT conditions are simplified as follows:

$$\bar{F}_i \geq b \geq \tilde{F}_i \quad \text{for} \quad i = 1, \ldots, M. \tag{5.11}$$

Depending on the value of α_i, $\bar{F}_i$ or $\tilde{F}_i$ is not defined. For instance, if $y_i = 1$ and $\alpha_i = C$, $\tilde{F}_i$ is not defined.

To detect the violating variables, we define b_{low} and b_{up} as follows:

$$\begin{aligned} b_{\mathrm{low}} &= \max_i \tilde{F}_i, \\ b_{\mathrm{up}} &= \min_i \bar{F}_i. \end{aligned} \tag{5.12}$$

If the KKT conditions are satisfied,

$$b_{\mathrm{up}} \geq b_{\mathrm{low}}. \tag{5.13}$$

Figure 5.4 (a) illustrates the KKT conditions that are satisfied. The filled rectangles show $\bar{F}_i$ and the filled circles show $\tilde{F}_i$.

We can use (5.13) as a stopping condition for training. To lighten a computational burden, we loosen (5.13) as follows:

$$b_{\mathrm{up}} \geq b_{\mathrm{low}} - \tau, \tag{5.14}$$

where τ is a positive tolerance parameter.

Figure 5.4 (b) shows the case where the KKT conditions are not satisfied. The data between the two dotted lines are the data that violate the KKT conditions. We define the τ-violating set V_{KKT}:

$$\begin{aligned} V_{\mathrm{KKT}} = \{\mathbf{x}_i \,|\, b_{\mathrm{up}} < \tilde{F}_i - \tau \quad &\text{or} \quad b_{\mathrm{low}} > \bar{F}_i + \tau \\ &\text{for} \quad i \in \{1, \ldots, M\}\}. \end{aligned} \tag{5.15}$$

For $\boldsymbol{\alpha} = \mathbf{0}$, $b_{\mathrm{low}} = \tilde{F}_i = 1$ and $b_{\mathrm{up}} = \bar{F}_i = -1$. Thus, if we set $\boldsymbol{\alpha} = \mathbf{0}$ for the initial vector, all the data violate the KKT conditions.

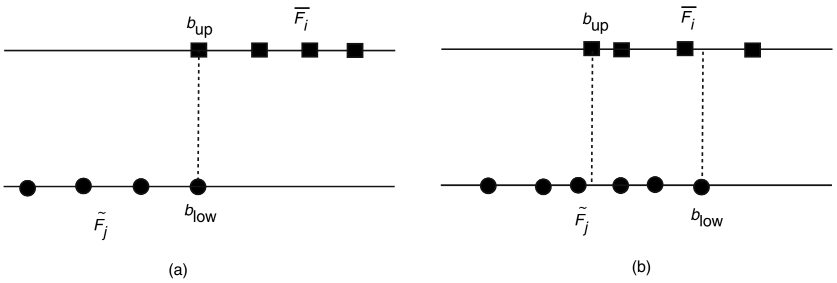


Fig. 5.4. KKT conditions for a dual problem: (a) KKT conditions satisfied, (b) KKT conditions violated

Example 5.1. We examine if the solution $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (1, 0, 1, 0)$ in Example 2.17 satisfies the KKT conditions. Because

$$\bar{F}_1 = \bar{F}_3 = \bar{F}_4 = \tilde{F}_1 = \tilde{F}_2 = \tilde{F}_3 = 0,$$

we obtain

$$b_{\text{up}} = b_{\text{low}}.$$

Thus the KKT conditions are satisfied.

Let $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (0.8, 0, 1, 0)$. Then because

$$\bar{F}_1 = \tilde{F}_1 = 0.2, \quad \tilde{F}_2 = \bar{F}_3 = \tilde{F}_3 = 0, \quad \tilde{F}_4 = -0.2,$$

we obtain

$$b_{\text{up}} = -0.2, \quad b_{\text{low}} = 0.2.$$

Thus the τ-violating set with $\tau = 0$ is

$$V_{\text{KKT}} = \{1, 2, 3\}.$$

The following example shows that $b_{\text{up}} > b_{\text{low}}$ when the primal solution is not unique.

Example 5.2. Reconsider Example 2.18. When $C < 1/2$, $\alpha_1 = \alpha_2 = C$. Thus,

$$\tilde{F}_1 = -1 + 2\,C,$$
$$\bar{F}_2 = 1 - 2\,C.$$

Thus the KKT conditions are satisfied and

$$1 - 2\,C \geq b \geq -1 + 2\,C.$$

Similar to L1 soft-margin support vector machines, we can derive the KKT conditions for L2 soft-margin support vector machines. For the L2 support vector machine, the KKT conditions are as follows:

1. $\alpha_i = 0$
$$y_i\, b \geq y_i\, F_i,$$
2. $\alpha_i > 0$
$$b = F_i - \frac{y_i\, \alpha_i}{C}.$$

Then $\tilde{F}_i$ and $\bar{F}_i$ are

$$\tilde{F}_i = F_i - \frac{y_i\, \alpha_i}{C} \quad \text{if} \quad (y_i = 1, \alpha_i = 0) \quad \text{or} \quad \alpha_i > 0, \tag{5.16}$$
$$\bar{F}_i = F_i - \frac{y_i\, \alpha_i}{C} \quad \text{if} \quad (y_i = -1, \alpha_i = 0) \quad \text{or} \quad \alpha_i > 0. \tag{5.17}$$

The KKT conditions are simplified as follows:

$$\bar{F}_i \geq b \geq \tilde{F}_i \quad \text{for} \quad i = 1, \ldots, M. \tag{5.18}$$

The remaining procedure is the same as that of the L1 support vector machine.

5.4 Overview of Training Methods

Training of a support vector machine results in solving a quadratic programming (QP) program. But commercially available QP solvers are not suited for training a support vector machine with a large number of training data due to long training time and large memory consumption. To overcome this problem, QP solvers have been combined with the decomposition techniques or other training methods have been developed. There are now many downloadable programs that train support vector machines.

Conventional learning algorithms developed for perceptrons are adapted to be used in the feature space [48, 86, 88, 104, 274]. The kernel-Adatron algorithm [48, 88] is extended from the Adatron algorithm [21]. Because the kernel-Adatron algorithm does not consider the equality constraint, the optimality for the bias term is not guaranteed. To compensate for this, in [69], a method to augment an additional input is discussed. Mangasarian and Musicant [160] added $b^2/2$ to the primal objective function of the L1 support vector machine. Then the dual problem becomes as follows. Maximize

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i\, \alpha_j\, y_i\, y_j\, \left(H(\mathbf{x}_i, \mathbf{x}_j) + 1\right) \tag{5.19}$$

subject to the constraints

$$0 \leq \alpha_i \leq C \qquad \text{for} \quad i = 1, \dots, M. \tag{5.20}$$

By this formulation, the linear constraint is vanished, and they applied the successive overrelaxation method for training.

Sequential minimal optimization is a training method developed by Platt [190]. It optimizes two data at a time. Thus it is equivalent to the decomposition method with $|W| = 2$ in Section 5.2. Vogt [262] extended SMO when the bias term is not used, i.e., there is no equality constraint in the dual problem. Instead of two variables, one variable is modified at a time. Kecman, Vogt, and Huang [130] showed that this algorithm is equivalent to the kernel-Adatron algorithm.

Some training algorithms are based on the geometrical properties of support vectors [132, 196, 203, 261]. Roobaert [203] proposed the direct SVM in which the optimal separating hyperplane is determined in a geometrical way. Initially, we find two nearest points of opposite classes and determine the hyperplane so that it contains the center of the two points and is orthogonal to the line connecting these points (see Fig. 5.5 (a)). Then we find the point that violates separation the most with respect to this hyperplane and rotate the hyperplane so that it goes through the center of the point and the initial point with the opposite class (see Fig. 5.5 (b)). According to the algorithm, without any proper explanation, the weight vector is updated so that the correction is orthogonal to the previous updates. The algorithm is extended to the feature space, again without an elaborate explanation.

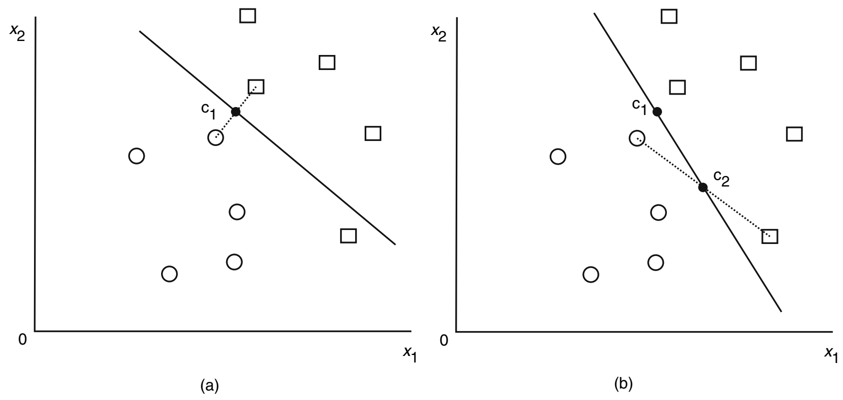


Fig. 5.5. Training by direct SVM: (a) Initialization of a hyperplane. (b) Modification of the hyperplane by the point that violates separation the most

Assuming that the problem is linearly separable in the feature space, Keerthi, Shevade, Bhattacharyya, and Murthy [132] showed the equivalence of training a support vector machine with finding the minimum distance between the two convex hulls for the two classes and derived an improved algorithm for finding the minimum distance.

Navia-Vázquez, Pérez-Cruz, Artés-Rodríguez, and Figueiras-Vidal [177] proposed an iterative least squares training algorithm. They first transform the objective function for the input space given by (2.41) into

$$Q = \frac{1}{2}\,\|\mathbf{w}\|^2 - \sum_{i=1}^{M} a_i^2\,\left(y_i - (\mathbf{w}^T\,\mathbf{x} + b)^2\right) + \sum_{i=1}^{M} \xi_i\,(C - \beta_i - \alpha_i), \quad (5.21)$$

where a_i are weights given by

$$\begin{aligned} a_i &= \left(\frac{\alpha_i\, y_i}{y_i - (\mathbf{w}^T\mathbf{x}_i + b)}\right)^{1/2} \\ &= \left(\frac{\alpha_i}{1 - y_i\,(\mathbf{w}^T\mathbf{x}_i + b)}\right)^{1/2} \\ &= \left(\frac{\alpha_i}{\xi_i}\right)^{1/2}. \end{aligned} \quad (5.22)$$

Assuming α_i, β_i, and ξ_i are constant, Q given by (5.21) is a square function of $\mathbf{w}$ and b. Thus by the least squares method we obtain $\mathbf{w}$ and b. Then using these $\mathbf{w}$ and b, we renew α_i, β_i, and ξ_i considering the inequality constraints. This procedure is iterated until the solution converges. Because this method is based on the primal problem, the extension to the feature space with an infinite dimension is not straightforward. Navia-Vázquez et al. proposed reducing the dimension by the kernel PCA and applying the preceding method.

5.5 Primal-Dual Interior-Point Methods

In training support vector machines, usually the primal-dual interior-point method is combined with the decomposition technique. In the following, first we briefly summarize the primal-dual interior-point method for linear programming based on [255, 272] and then we summarize the primal-dual interior-point method for quadratic programming. Last, we evaluate characteristics of support vector machine training by the primal-dual interior-point method combined with the decomposition technique.

5.5.1 Primal-Dual Interior-Point Methods for Linear Programming

Consider the following linear programming problem. Minimize

$$\mathbf{c}^T\mathbf{x} \quad (5.23)$$

subject to

$$A\mathbf{x} \geq \mathbf{b}, \qquad \mathbf{x} \geq \mathbf{0}, \tag{5.24}$$

where A is an $n \times m$ matrix, $\mathbf{x}$ and $\mathbf{c}$ are m-dimensional vectors, and $\mathbf{b}$ is an n-dimensional vector. Here, $\mathbf{x} \geq \mathbf{0}$ means that all the elements of $\mathbf{x}$ are nonnegative.

Consider that this problem is a primal problem. Then the dual problem of (5.23) and (5.24) is given as follows [63, 255]. Maximize

$$\mathbf{b}^T\mathbf{y} \tag{5.25}$$

subject to

$$A^T\mathbf{y} \leq \mathbf{c}, \qquad \mathbf{y} \geq \mathbf{0}, \tag{5.26}$$

where $\mathbf{y}$ is an n-dimensional dual variable vector.

Now consider the following linear programming problem with equality constraints. Minimize

$$\mathbf{c}^T\mathbf{x} \tag{5.27}$$

subject to

$$A\mathbf{x} = \mathbf{b}, \qquad \mathbf{x} \geq \mathbf{0}, \tag{5.28}$$

where A is an $n \times m$ matrix, $\mathbf{x}$ and $\mathbf{c}$ are m-dimensional vectors, and $\mathbf{b}$ is an n-dimensional vector.

Because the first equation in (5.28) is equivalent to

$$\begin{pmatrix} A \\ -A \end{pmatrix} \mathbf{x} \geq \begin{pmatrix} \mathbf{b} \\ -\mathbf{b} \end{pmatrix}, \tag{5.29}$$

the dual problem of (5.27) and (5.28) is given as follows. Maximize

$$\left(\mathbf{b}^T \;\; -\mathbf{b}^T\right)\mathbf{Y} \tag{5.30}$$

subject to

$$\left(A^T \;\; -A^T\right)\mathbf{Y} \leq \mathbf{c}, \qquad \mathbf{Y} \geq \mathbf{0}, \tag{5.31}$$

where $\mathbf{Y}$ is the $(2\,n)$-dimensional dual variable vector. Now we define

$$\mathbf{Y} = \begin{pmatrix} \mathbf{y}^+ \\ \mathbf{y}^- \end{pmatrix}, \quad \mathbf{y}^+ \geq \mathbf{0}, \quad \mathbf{y}^- \geq \mathbf{0}, \quad \mathbf{y} = \mathbf{y}^+ - \mathbf{y}^-. \tag{5.32}$$

Then the dual problem of (5.27) and (5.28) is given as follows. Maximize

$$\mathbf{b}^T\mathbf{y} \tag{5.33}$$

subject to

$$A^T\mathbf{y} + \mathbf{z} = \mathbf{c}, \quad \mathbf{z} \geq \mathbf{0}, \tag{5.34}$$

where $\mathbf{z}$ is the m-dimensional slack variable vector. Here we must notice that the elements of $\mathbf{y}$ need not be nonnegative.

The solution that satisfies the constraints is called the *feasible solution.* It is known that for feasible solutions $\mathbf{x}$ for (5.27) and (5.28) and $(\mathbf{y}, \mathbf{z})$ for (5.33) and (5.34),

$$\mathbf{c}^T\mathbf{x} \geq \mathbf{b}^T\mathbf{y} \tag{5.35}$$

is satisfied. The difference of the objective functions $\mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{y}$ is called the *duality gap.* It can be proved that if the primal solution has the optimal solution, the dual problem also has the optimal solution and the duality gap is zero. Namely, the strict equality holds in (5.35).

Then if the primal problem has the optimal solution,

$$\begin{aligned}\mathbf{x}^T\mathbf{z} &= \mathbf{x}^T(\mathbf{c} - A^T\mathbf{y})\\ &= \mathbf{x}^T\mathbf{c} - \mathbf{b}^T\mathbf{y}\\ &= 0\end{aligned} \tag{5.36}$$

is satisfied. Because $x_i \geq 0$ and $z_i \geq 0$, (5.36) is equivalent to

$$x_i\, z_i = 0 \qquad \text{for} \quad i = 1, \ldots, m. \tag{5.37}$$

This is called the *complementarity condition.*

Then if $\mathbf{x}$ and $(\mathbf{y}, \mathbf{z})$ satisfy the equality constraints for the primal and dual problems, respectively, and (5.37) is satisfied, $\mathbf{x}$ and $(\mathbf{y}, \mathbf{z})$ are the optimal solutions. Namely, solving the primal or dual problem is equivalent to solving

$$\begin{aligned} A\mathbf{x} &= \mathbf{b},\\ A^T\mathbf{y} + \mathbf{z} &= \mathbf{c},\\ x_i\, z_i &= 0, \quad x_i \geq 0, z_i \geq 0 \qquad \text{for} \quad i = 1, \ldots, m.\end{aligned} \tag{5.38}$$

This problem is called the *primal-dual problem.* In solving the primal-dual problem by the interior-point methods, the complementarity condition is changed to

$$x_i\, z_i = \mu \quad \text{for} \quad i = 1, \ldots, m, \tag{5.39}$$

where $\mu \geq 0$, and the value of μ is decreased to zero in solving the problem. The locus of the solutions when μ is decreased to zero is called the *central path.*

The primal-dual problem can be obtained by introducing the barrier (objective) function [255, pp. 277–85]. Consider the primal problem given by (5.27) and (5.28). We change the inequality constraints $x_i \geq 0$ to $\log x_i$ and subtract them from the objective function:

$$\mathbf{c}^T\mathbf{x} - \mu \sum_{i=1}^{m} \log x_i \tag{5.40}$$

where $\mu \geq 0$. Because $-\log x_i$ is finite for positive x_i and approaches positive infinity as x_i approaches 0, we can eliminate the inequality constraints. We call this the *barrier function* because $\log x_i$ works as a barrier and the solution cannot go into the infeasible region defined by the inequality constraints $x_i < 0$. We then introduce the Lagrange multipliers $\mathbf{y} = (y_1, \ldots, y_n)^T$:

$$L(\mathbf{x}, \boldsymbol{\beta}) = \mathbf{c}^T \mathbf{x} - \mu \sum_{i=1}^{m} \log x_i - \mathbf{y}^T (A\mathbf{x} - \mathbf{b}). \tag{5.41}$$

The optimality conditions satisfy

$$\frac{\partial L(\mathbf{x}, \mathbf{y})}{\partial \mathbf{x}} = \mathbf{c} - \mu \begin{pmatrix} 1/x_1 & & 0 \\ & \ddots & \\ 0 & & 1/x_n \end{pmatrix} - A^T \mathbf{y} = \mathbf{0}; \tag{5.42}$$

$$\frac{\partial L(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y}} = A\mathbf{x} - \mathbf{b} = \mathbf{0}. \tag{5.43}$$

By defining $x_i\, z_i = \mu$ for $j = 1, \ldots, m$, (5.42) becomes

$$A^T \mathbf{y} + \mathbf{z} = \mathbf{c}. \tag{5.44}$$

Thus, the primal-dual problem is obtained.

Primal-dual interior-point methods are classified according to whether a solution satisfies the constraints $x_i \geq 0, z_i \geq 0$ at every iteration step, into feasible and infeasible methods. The solution at each iteration step is generated by potential reduction methods, path-following methods, and predictor-corrector methods [272]. In the following we will discuss the path-following method [255].

We write the primal-dual interior-point method in a matrix form:

$$\begin{aligned} A\,\mathbf{x} &= \mathbf{b}, \\ A^T \mathbf{y} + \mathbf{z} &= \mathbf{c}, \\ X\,Z\,\mathbf{e} &= \mu\,\mathbf{e}, \end{aligned} \tag{5.45}$$

where $\mathbf{x} > \mathbf{0}$, $\mathbf{z} > \mathbf{0}$, $X = \mathrm{diag}(x_1, \ldots, x_m)$, $Z = \mathrm{diag}(z_1, \ldots, z_m)$, $\mathbf{e}$ is an m-dimensional vector, and $\mathbf{e} = (1, \ldots, 1)^T$. Here $\mathbf{x}$ and $\mathbf{z}$ are positive vectors because of positive μ.

In the path-following method, starting from the feasible solution $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and positive μ, we alternately calculate the corrections of $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and estimation of μ, and we stop calculations when the solution reaches the optimal solution.

For the given $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and μ, we calculate the corrections of $(\mathbf{x}, \mathbf{y}, \mathbf{z})$, $(\Delta\mathbf{x}, \Delta\mathbf{y}, \Delta\mathbf{z})$, by Newton's method:

$$\begin{aligned} A\,\Delta\mathbf{x} &= \mathbf{b} - A\,\mathbf{x}, \\ A^T \Delta\mathbf{y} + \Delta\mathbf{z} &= \mathbf{c} - A^T \mathbf{y} - \mathbf{z}, \\ Z\,\Delta\mathbf{x} + X\,\Delta\mathbf{z} &= \mu\,\mathbf{e} - X\,Z\,\mathbf{e}. \end{aligned} \tag{5.46}$$

Assuming that A is nonsingular, we can solve (5.46) for $(\Delta\mathbf{x}, \Delta\mathbf{y}, \Delta\mathbf{z})$. However, it is not guaranteed that the calculated corrections will satisfy the positive constraints on $\mathbf{x}$ and $\mathbf{z}$. Thus, introducing the parameter $\theta\,(\leq 1)$, we determine the maximum value of θ, which satisfies:

$$\mathbf{x} + \theta\Delta\mathbf{x} \geq \mathbf{0}, \tag{5.47}$$

$$\mathbf{z} + \theta\Delta\mathbf{z} \geq \mathbf{0}. \tag{5.48}$$

Then we set

$$\theta \leftarrow \min(r\,\theta, 1), \tag{5.49}$$

$$\mathbf{x} \leftarrow \mathbf{x} + \theta\,\Delta\mathbf{x}, \tag{5.50}$$

$$\mathbf{y} \leftarrow \mathbf{y} + \theta\,\Delta\mathbf{y}, \tag{5.51}$$

$$\mathbf{z} \leftarrow \mathbf{z} + \theta\,\Delta\mathbf{z}, \tag{5.52}$$

where r is smaller than but close to 1 to guarantee that $\mathbf{x}$ and $\mathbf{z}$ are positive vectors.

Because current μ is estimated by $\mu = x_i\,z_i$ for $i = 1, \ldots, m$, we update the value of μ by

$$\mu = \frac{\mathbf{z}^T\mathbf{x}}{m}\delta, \tag{5.53}$$

where $1 > \delta > 0$ and $\mathbf{z}^T\mathbf{x}/m$ is the average estimated value of μ.

We stop calculation if elements of $\mathbf{x}$ or $\mathbf{y}$ increase indefinitely or if the complementarity conditions are satisfied. Namely, $\mathbf{z}^T\mathbf{x}$ is within a specified value.

5.5.2 Primal-Dual Interior-Point Methods for Quadratic Programming

Consider the following quadratic programming problem. Minimize

$$\mathbf{c}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T H\,\mathbf{x} \tag{5.54}$$

subject to

$$A\,\mathbf{x} = \mathbf{b}, \qquad x_i \geq 0 \quad \text{for} \quad i = 1, \ldots, m, \tag{5.55}$$

where H is an $m \times m$ symmetric, positive semidefinite matrix, A is an $n \times m$ matrix with rank n, $\mathbf{x}$ and $\mathbf{c}$ are m-dimensional vectors, and $\mathbf{b}$ is an n-dimensional vector. Let this be a primal problem.

The dual problem of (5.54) and (5.55) is given as follows. Maximize

$$\mathbf{b}^T\mathbf{y} - \frac{1}{2}\mathbf{x}^T H\,\mathbf{x} \tag{5.56}$$

subject to[1]

[1]If the inequality constraint $A\mathbf{x} \geq \mathbf{b}$ is used, $\mathbf{y}$ needs to be a nonnegative vector.

$$A^T \mathbf{y} - H\,\mathbf{x} + \mathbf{z} = \mathbf{c}, \qquad x_i \geq 0, \quad z_i \geq 0 \quad \text{for} \quad i = 1\ldots, m, \tag{5.57}$$

where $\mathbf{y}$ is the n-dimensional dual variable vector and $\mathbf{z}$ is the n-dimensional slack variable vector.

If $\mathbf{x}$ is the optimal solution for the primal problem and $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is the optimal solution for the dual problem, the following conditions are satisfied:

$$\begin{aligned} &A\,\mathbf{x} = \mathbf{b}, \\ &A^T \mathbf{y} - H\,\mathbf{x} + \mathbf{z} = \mathbf{c}, \\ &x_i\, z_i = 0, \quad x_i \geq 0, \quad z_i \geq 0 \qquad \text{for} \quad i = 1, \ldots, m. \end{aligned} \tag{5.58}$$

This is called the *primal-dual problem.* If $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is the solution, $\mathbf{x}$ is the optimal solution of the primal problem and $(\mathbf{y}, \mathbf{z})$ is the optimal solution of the dual problem.

Because a primal-dual problem of quadratic programming is similar to a primal-dual problem of linear programming, there is not much difference in solving linear programming and quadratic programming problems by primal-dual interior-point methods.

Now derive the optimality conditions of the L1 support vector machine. Training of the L1 support vector machine is given as follows. Minimize

$$Q(\boldsymbol{\alpha}) = -\sum_{i=1}^{M} \alpha_i + \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i\, \alpha_j\, y_i\, y_j\, H(\mathbf{x}_i, \mathbf{x}_j) \tag{5.59}$$

subject to the constraints

$$\sum_{i=1}^{M} y_i\, \alpha_i = 0, \tag{5.60}$$

$$C \geq \alpha_i \geq 0 \qquad \text{for} \quad i = 1, \ldots, M. \tag{5.61}$$

Here we multiplied (2.61) by the minus sign to make the problem a minimization problem. Introducing slack variables $\beta_i\,(i = 1, \ldots, M)$, we convert the inequality constraints (5.61) into equality constraints:

$$\alpha_i + \beta_i = C, \quad \alpha_i \geq 0, \quad \beta_i \geq 0 \qquad \text{for} \quad i = 1, \ldots, M. \tag{5.62}$$

Then the dual problem of the problem given by (5.59), (5.60), and (5.62) is given as follows. Maximize

$$C \sum_{i=1}^{M} \delta_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i\, \alpha_j\, y_i\, y_j\, H(\mathbf{x}_i, \mathbf{x}_j) \tag{5.63}$$

subject to the constraints

$$\delta_i + y_i\,\delta_{M+1} - \sum_{j=1}^{M} y_i\,y_j\,\alpha_j H(\mathbf{x}_i,\mathbf{x}_j) + z_i = -1 \qquad \text{for} \quad i = 1,\dots,M, \tag{5.64}$$

$$\delta_i + z_{M+i} = 0 \qquad \text{for} \quad i = 1,\dots,M, \tag{5.65}$$

where $\boldsymbol{\delta} = (\delta_1,\dots,\delta_{M+1})^T$, $\boldsymbol{\delta}$ corresponds to $\mathbf{y}$ in (5.57), and $\mathbf{z} = (z_1,\dots, z_{2M})^T$.

Substituting $\delta_i = -z_{M+i}$ obtained from (5.65) into (5.64), we obtain

$$y_i\,\delta_{M+1} - \sum_{j=1}^{M} y_i\,y_j\,\alpha_j H(\mathbf{x}_i,\mathbf{x}_j) + z_i - z_{M+i} = -1 \qquad \text{for} \quad i = 1,\dots,M. \tag{5.66}$$

Then the optimality conditions for the L1 support vector machine are given as follows:

$$\sum_{i=1}^{M} y_i\,\alpha_i = 0, \tag{5.67}$$

$$y_i\,\delta_{M+1} - \sum_{j=1}^{M} y_i\,y_j\,\alpha_j H(\mathbf{x}_i,\mathbf{x}_j) + z_i - z_{M+i} = -1 \qquad \text{for} \quad i = 1,\dots,M, \tag{5.68}$$

$$\alpha_i\,z_i = 0 \qquad \text{for} \quad i = 1,\dots,M, \tag{5.69}$$

$$(C - \alpha_i)\,z_{M+i} = 0 \qquad \text{for} \quad i = 1,\dots,M, \tag{5.70}$$

$$C \geq \alpha_i \geq 0, \quad z_i \geq 0, \quad z_{M+i} \geq 0 \qquad \text{for} \quad i = 1,\dots,M. \tag{5.71}$$

Similarly, we can obtain the optimality conditions for the L2 support vector machine as follows:

$$\sum_{i=1}^{M} y_i\,\alpha_i = 0, \tag{5.72}$$

$$y_i\,\delta - \sum_{j=1}^{M} y_i\,y_j\,\alpha_j \left(H(\mathbf{x}_i,\mathbf{x}_j) + \frac{\delta_{ij}}{C}\right) + z_i = -1 \qquad \text{for} \quad i = 1,\dots,M, \tag{5.73}$$

$$\alpha_i\,z_i = 0 \qquad \text{for} \quad i = 1,\dots,M, \tag{5.74}$$

$$\alpha_i \geq 0, \quad z_i \geq 0 \qquad \text{for} \quad i = 1,\dots,M, \tag{5.75}$$

where δ is a scalar variable associated with the linear constraint (5.72).

5.5.3 Performance Evaluation

Using the primal-dual interior-point method [254] combined with the variable chunking technique, we compared L1 and L2 support vector machines from the standpoint of training time and the generalization ability [138] using the data sets listed in Table 1.1. We used one-against-all fuzzy SVMs to resolve unclassifiable regions. We used linear, polynomial, and RBF kernels and set some appropriate value to C. We ran the C program on an Athlon MP 2000+ personal computer.

We used the inexact and exact KKT conditions for working set selection. For the inexact KKT conditions, we randomly selected F points that violated the KKT conditions. For the exact KKT conditions, we set $\tau = 0.01$. Initially, we randomly selected the working set variables. Then we sorted $\tilde{F}_i$ and $\bar{F}_i$ in descending order of KKT violations. Then we alternately selected F points from the top of $\tilde{F}_i$ and $\bar{F}_i$. We set the initial working set size of 50 and added 50 variables at a time. Because the initial working set was randomly selected, for each training condition, we trained the SVM 100 times and calculated the average values of the recognition rates and training time.

Table 5.3 shows the results for L1 and L2 SVMs using the inexact KKT conditions. Numerals in parentheses show the recognition rates of the training data when they were not 100 percent. The recognition rates of the test data by the L2 SVM are higher than those by the L1 SVM for 16 cases out of 30. But those by the L1 SVM are higher for seven cases. Thus the L2 SVM performed better than the L1 SVM, but the difference in recognition rates is small.

For linear kernels, the maximum rank of the Hessian matrix for the L1 SVM is the number of input variables plus 1 (see Theorem 2.15). Thus, if the working set size exceeds this value, the Hessian matrix for the L1 SVM is positive semidefinite. But the Hessian matrix for L2 SVMs is always positive definite. But this fact is not reflected in the training time. From the table, excluding the iris data with nonlinear kernels, training of the L1 SVM was faster than that of the L2 SVM.

Table 5.4 shows the results when the exact KKT conditions were used. Due to the difference in the convergence tests, in some cases, the recognition rates were slightly different from those in Table 5.3. Similar to the inexact KKT conditions, in most cases, training time of the L1 SVM was shorter than that of the L2 SVM.

Comparing Tables 5.3 and 5.4, training time of the L1 SVM by the inexact KKT conditions was shorter than by the exact KKT conditions except for the iris data. The tendency was the same for the L2 SVM.

To investigate why the exact KKT conditions were not always better than the inexact KKT conditions, we studied the two-class problem that separates Class 2 from the remaining classes for the blood cell data. We trained the L1 SVM with $d = 3$ and $C = 2000$.

Figure 5.6 shows the working set sizes of the inexact and exact KKT conditions against the number of iterations. From the figure, the number of iterations for the exact KKT conditions is smaller than that of the inexact KKT conditions, but the working set size of the exact KKT conditions is larger after the second iteration. This means that the inexact KKT conditions

Table 5.3. Recognition rates and training time of L1 SVM and L2 SVM using inexact KKT conditions

Data	Kernel	L1 SVM (%)	Time (s)	L2 SVM (%)	Time (s)
Iris	Linear	96.00 (97.33)	**0.02**	**97.33** (98.67)	0.04
($C = 5000$)	$d2$	94.67	0.01	94.67	0.01
	$d3$	94.67	0.01	94.67	0.01
Numeral	Linear	99.63	**0.43**	99.63	0.65
($C = 50$)	$d2$	99.39	**0.40**	**99.63**	0.62
	$d3$	99.51	**0.40**	**99.63**	0.65
	$d4$	99.51	**0.45**	**99.63**	0.65
Thyroid	Linear	**96.70** (97.56)	**39**	94.22 (94.67)	4008
($C = 10{,}000$)	$d2$	**97.14** (98.75)	**60**	96.47 (98.38)	328
	$d3$	**97.49** (99.31)	**27**	97.26 (99.10)	151
	$d4$	**97.43** (99.34)	**19**	97.35 (99.23)	73
	$\gamma 1$	**96.79** (99.02)	**83**	96.50 (99.02)	57
	$\gamma 2$	96.53 (99.36)	**55**	96.53 (99.34)	212
Blood cell	Linear	87.23 (91.02)	**214**	**87.87** (90.64)	872
($C = 2000$)	$d2$	92.97 (96.67)	**27**	**93.52** (97.06)	44
	$d3$	93.19 (98.22)	**24**	**93.71** (98.55)	32
	$d4$	92.68 (98.93)	**24**	**93.42** (99.00)	30
	$\gamma 1$	93.35 (97.74)	**27**	**93.74** (98.06)	42
	$\gamma 2$	93.42 (98.84)	**26**	**93.58** (98.90)	38
Hiragana-50	Linear	92.60 (98.07)	**129**	**93.28** (98.79)	253
($C = 5000$)	$d2$	99.24	**109**	99.24	137
	$d3$	**99.31**	**112**	99.26	137
	$d4$	**99.33**	**112**	99.28	147
Hiragana-105	Linear	97.03 (97.50)	**806**	**97.45** (98.08)	1652
($C = 2000$)	$d2$	100	**430**	100	531
	$d3$	100	**434**	100	532
Hiragana-13	Linear	96.37 (97.92)	**360**	**96.43** (97.89)	705
($C = 5000$)	$d2$	99.27 (99.59)	**295**	**99.35** (99.67)	333
	$d3$	99.37 (99.64)	**289**	**99.39** (99.69)	342
	$d4$	99.34 (99.62)	**290**	**99.37** (99.68)	368

Table 5.4. Recognition rates and training time of L1 SVM and L2 SVM using exact KKT conditions

Data	Kernel	L1 SVM (%)	Time (s)	L2 SVM (%)	Time (s)
Iris	Linear	96.00 (97.33)	0.03	**97.33** (98.67)	0.03
($C = 5000$)	$d2$	94.67	0.01	94.67	0.01
	$d3$	94.67	0.01	94.67	0.01
Numeral	Linear	99.63	**0.53**	99.63	0.69
($C = 50$)	$d2$	99.39	**0.52**	**99.63**	0.69
	$d3$	99.51	**0.56**	**99.63**	0.75
	$d4$	99.51	**0.59**	**99.63**	0.79
Thyroid	Linear	**96.32** (97.40)	**98**	94.22 (94.70)	69,840
($C = 10{,}000$)	$d2$	**97.14** (98.81)	**162**	96.44 (98.33)	1867
	$d3$	**97.52** (99.26)	**69**	97.08 (99.07)	363
	$d4$	**97.46** (99.34)	**54**	97.32 (99.23)	150
	$\gamma1$	**96.82** (99.02)	**218**	96.50 (99.02)	1362
	$\gamma2$	96.53 (99.36)	**170**	96.53 (99.34)	410
Blood cell	Linear	87.77 (91.41)	**267**	**88.45**(91.12)	4180
($C = 2000$)	$d2$	93.00 (96.74)	51	**93.48**(97.06)	**49**
	$d3$	93.26 (98.22)	39	**93.71**(98.55)	**37**
	$d4$	92.65 (98.93)	34	**93.42**(99.00)	34
	$\gamma1$	93.35 (97.74)	**42**	**93.74**(98.06)	52
	$\gamma2$	93.42 (98.84)	**39**	**93.58**(98.90)	41
Hiragana-50	Linear	92.58 (98.09)	**187**	**93.34** (98.79)	366
($C = 5000$)	$d2$	99.24	**124**	99.24	135
	$d3$	**99.31**	**129**	99.26	135
	$d4$	**99.33**	**129**	99.28	146
Hiragana-105	Linear	97.04 (97.55)	**1323**	**97.47** (98.10)	2063
($C = 2000$)	$d2$	100	**508**	100	527
	$d3$	100	**524**	100	530
Hiragana-13	Linear	96.42 (97.93)	**563**	**96.47** (97.89)	1121
($C = 5000$)	$d2$	99.26 (99.59)	**356**	**99.37** (99.67)	364
	$d3$	99.39 (99.64)	**318**	99.39 (99.69)	344
	$d4$	99.34 (99.62)	**335**	**99.39** (99.69)	367

estimate the violating variables conservatively. Thus, with the smaller working set size, training by the inexact KKT conditions was faster.

Figure 5.7 shows the training time of the inexact and exact KKT conditions for the change of F, namely the number of variables added to the working set W. For the inexact KKT conditions, training was fastest when 100 variables were added, but for the exact KKT conditions, training was fastest when 20 variables were added. Because for the exact KKT conditions the training time was not monotonic for the increase of the added variables, the exact KKT conditions are less suitable as a variable selection strategy.

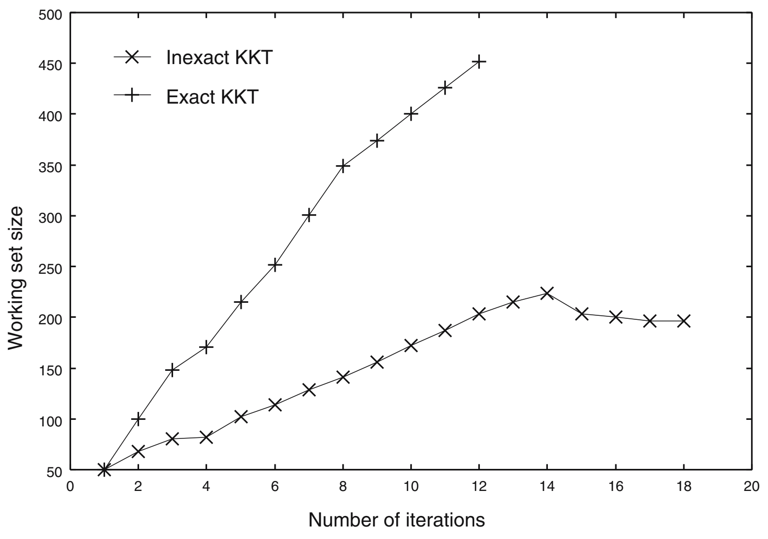


Fig. 5.6. Relationship between the working set size and number of iterations ($d = 3$, $C = 2000$)

According to the computer experiments, we have found that

1. training of L2 SVMs was not always faster than that of L1 SVMs,
2. the difference of the generalization abilities between the L1 and L2 SVMs was small, and
3. training by the exact KKT conditions was not always faster than by the inexact KKT conditions. This was due to the conservative estimation of violating variables by the inexact KKT conditions.

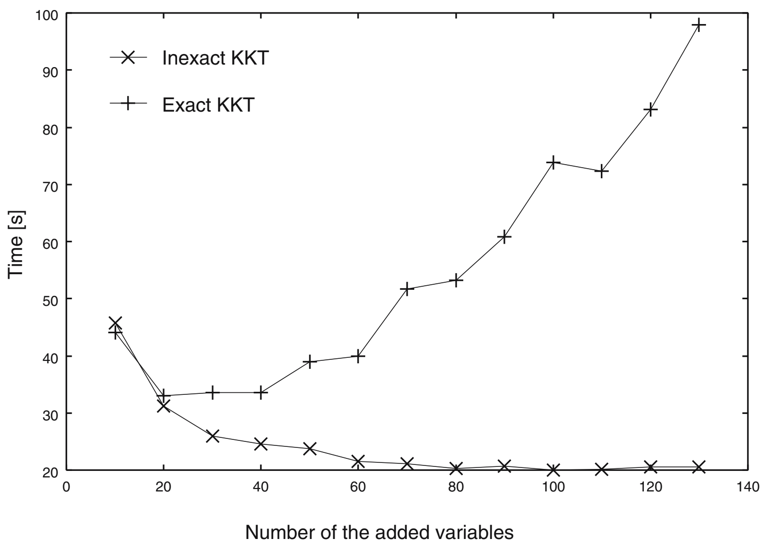


Fig. 5.7. Relationship between the number of variables added and training time ($d = 3$, $C = 2000$)

5.6 Steepest Ascent Methods

In the following, we discuss the training algorithm based on the steepest ascent method [8]. The algorithm reduces to SMO when two data are optimized simultaneously. Because it uses the Hessian matrix, it is a variant of Newton's method [31].

5.6.1 Training Algorithms

We decompose the set of variables $\{\alpha_i \,|\, i \ldots, M\}$ into a working set $\boldsymbol{\alpha}_W = \{\alpha_i \,|\, i \in W\}$ and a fixed set $\boldsymbol{\alpha}_N = \{\alpha_i \,|\, i \in N\}$, where W and N are the index sets, $W \cup N = \{1, \ldots, M\}$, and $W \cap N = \phi$. Then fixing $\boldsymbol{\alpha}_N$, we maximize

$$\begin{aligned} Q(\boldsymbol{\alpha}_W) = & \sum_{i \in W} \alpha_i - \frac{1}{2} \sum_{i,j \in W} \alpha_i \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) \\ & - \sum_{\substack{i \in W, \\ j \in N}} \alpha_i \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) \\ & - \frac{1}{2} \sum_{i,j \in N} \alpha_i \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i \in N} \alpha_i \end{aligned} \tag{5.76}$$

subject to the constraints

$$\sum_{i \in W} y_i\, \alpha_i = -\sum_{i \in N} y_i\, \alpha_i, \qquad 0 \le \alpha_i \le C \quad \text{for} \quad i \in W. \tag{5.77}$$

Now consider solving the subprogram for $\boldsymbol{\alpha}_W$. Solving the equality in (5.77) for $\alpha_s \in \boldsymbol{\alpha}_W$, we obtain

$$\alpha_s = -\sum_{\substack{i \neq s, \\ i=1}}^{M} y_s\, y_i\, \alpha_i. \tag{5.78}$$

Substituting (5.78) into (5.76), we can eliminate the equality constraint. Let $\boldsymbol{\alpha}_{W'} = \{\alpha_i \,|\, i \neq s, i \in W\}$. Now because $Q(\boldsymbol{\alpha}_{W'})$ is quadratic, we can express the change of $Q(\boldsymbol{\alpha}_{W'})$, $\Delta Q(\boldsymbol{\alpha}_{W'})$, as a function of the change of $\boldsymbol{\alpha}_{W'}$, $\Delta\boldsymbol{\alpha}_{W'}$, by

$$\Delta Q(\boldsymbol{\alpha}_{W'}) = \frac{\partial Q(\boldsymbol{\alpha}_{W'})}{\partial \boldsymbol{\alpha}_{W'}} \Delta\boldsymbol{\alpha}_{W'} + \frac{1}{2} \Delta\boldsymbol{\alpha}_{W'}^T \frac{\partial^2 Q(\boldsymbol{\alpha}_{W'})}{\partial \boldsymbol{\alpha}_{W'}^2} \Delta\boldsymbol{\alpha}_{W'}. \tag{5.79}$$

Considering that

$$\frac{\partial \alpha_s}{\partial \alpha_i} = -y_s\, y_i, \tag{5.80}$$

we derive the partial derivatives of $Q(\boldsymbol{\alpha}_{W'})$ with respect to α_i $(i \neq s, i \in W')$:

$$\begin{aligned} \frac{\partial Q(\boldsymbol{\alpha}_{W'})}{\partial \alpha_i} &= 1 + \frac{\partial \alpha_s}{\partial \alpha_i} - \sum_{j=1}^{M} \alpha_j\, y_i\, y_j\, H(\mathbf{x}_i, \mathbf{x}_j) \\ &\quad - \sum_{j=1}^{M} \alpha_j\, y_s\, y_j\, \frac{\partial \alpha_s}{\partial \alpha_i}\, H(\mathbf{x}_s, \mathbf{x}_j) \\ &= 1 - y_s\, y_i - \sum_{j=1}^{M} \alpha_j\, y_i\, y_j\, H(\mathbf{x}_i, \mathbf{x}_j) \\ &\quad + \sum_{j=1}^{M} \alpha_j\, y_j\, y_i\, H(\mathbf{x}_s, \mathbf{x}_j). \end{aligned} \tag{5.81}$$

Using (5.81), the second partial derivatives of $Q(\boldsymbol{\alpha}_{W'})$ with respect to α_i and α_j $(i,\, j \neq s,\, i,\, j \in W')$ are

$$\frac{\partial^2 Q(\boldsymbol{\alpha}_{W'})}{\partial \alpha_i^2} = -H(\mathbf{x}_i, \mathbf{x}_i) + 2\, H(\mathbf{x}_i, \mathbf{x}_s) - H(\mathbf{x}_s, \mathbf{x}_s), \tag{5.82}$$

$$\begin{aligned} \frac{\partial^2 Q(\boldsymbol{\alpha}_{W'})}{\partial \alpha_i \partial \alpha_j} &= y_i\, y_j\, \left(-H(\mathbf{x}_i, \mathbf{x}_j) + H(\mathbf{x}_i, \mathbf{x}_s) + H(\mathbf{x}_s, \mathbf{x}_j)\right. \\ &\quad \left. - H(\mathbf{x}_s, \mathbf{x}_s)\right). \end{aligned} \tag{5.83}$$

From (2.59),

$$\frac{\partial^2 Q(\boldsymbol{\alpha}_{W'})}{\partial \boldsymbol{\alpha}_{W'}^2} = -\left(\cdots y_i(\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s))\cdots\right)^T \times \left(\cdots y_j(\mathbf{g}(\mathbf{x}_j) - \mathbf{g}(\mathbf{x}_s))\cdots\right). \tag{5.84}$$

Thus, if $\mathbf{x}_i = \mathbf{x}_s$ $(i, s \in W)$, $\partial^2 Q(\boldsymbol{\alpha})/\partial \boldsymbol{\alpha}_{W'}^2$ is singular. Therefore, we need to avoid choosing $\mathbf{x}_i$ that is the same as $\mathbf{x}_s$, if included. Because $-\partial^2 Q(\boldsymbol{\alpha})/\partial \boldsymbol{\alpha}_{W'}^2$ is expressed by the product of a transposed matrix and the matrix, it is positive semidefinite. Assuming that it is positive definite and neglecting the bounds, $\Delta Q(\boldsymbol{\alpha}_{W'})$ has the maximum at

$$\Delta \boldsymbol{\alpha}_{W'} = -\left(\frac{\partial^2 Q(\boldsymbol{\alpha}_{W'})}{\partial \boldsymbol{\alpha}_{W'}^2}\right)^{-1} \frac{\partial Q(\boldsymbol{\alpha}_{W'})}{\partial \boldsymbol{\alpha}_{W'}}. \tag{5.85}$$

Assume that α_i $(i = 1, \ldots, M)$ satisfy (5.77). Then from (5.77) and (5.85), we obtain the correction of α_s:

$$\Delta \alpha_s = -\sum_{i \in W'} y_s\, y_i\, \Delta \alpha_i. \tag{5.86}$$

Now the correction of $\boldsymbol{\alpha}_W$, $\Delta \boldsymbol{\alpha}_W$, is obtained by (5.85) and (5.86). For α_i $(i \in W)$, if

$$\alpha_i = 0, \quad \Delta \alpha_i < 0, \quad \text{or} \tag{5.87}$$

$$\alpha_i = C, \quad \Delta \alpha_i > 0, \tag{5.88}$$

we cannot modify $\boldsymbol{\alpha}_W$, because this will violate the lower or upper bound. If this happens, we delete these variables from the working set and repeat the procedure for the reduced working set.

If (5.87) or (5.88) does not hold for any variable in the working set, we can modify the variables. Let $\Delta \alpha_i'$ be the maximum or minimum correction of α_i that is within the bounds. Then if $\alpha_i + \Delta \alpha_i < 0$, $\Delta \alpha_i' = -\alpha_i$. And if $\alpha_i + \Delta \alpha_i > C$, $\Delta \alpha_i' = C - \alpha_i$. Otherwise $\Delta \alpha_i' = \Delta \alpha_i$. Now we calculate

$$r = \min_{i \in W} \frac{\Delta \alpha_i'}{\Delta \alpha_i}, \tag{5.89}$$

where $0 < r \le 1$.

We modify $\boldsymbol{\alpha}_W$ by

$$\boldsymbol{\alpha}_W^{\text{new}} = \boldsymbol{\alpha}_W^{\text{old}} + r\, \Delta \boldsymbol{\alpha}_W. \tag{5.90}$$

It is clear that the obtained $\boldsymbol{\alpha}_W^{\text{new}}$ satisfies the equality constraint. In addition, $Q(\boldsymbol{\alpha}_W^{\text{new}}) \ge Q(\boldsymbol{\alpha}_W^{\text{old}})$ is satisfied.

If $\partial^2 Q(\boldsymbol{\alpha}_{W'})/\partial \boldsymbol{\alpha}_{W'}^2$ is singular, we can use the pseudo-inverse [96]. But because the calculation is time-consuming, we delete the variables that cause singularity from the working set, and if the number of reduced variables is more than 1, do the procedure for the reduced working set.

Consider deleting the variables using the symmetric Cholesky factorization in calculating (5.85). In factorizing the ith column of $\partial^2 Q(\boldsymbol{\alpha}_{W'})/\partial \boldsymbol{\alpha}_{W'}^2$, if the value of the ith diagonal element of the lower triangular matrix is smaller than the prescribed small value, we discard the ith column and row of $\partial^2 Q(\boldsymbol{\alpha}_{W'})/\partial \boldsymbol{\alpha}_{W'}^2$ and proceed to the $(i+1)$st column. To simplify this procedure, in the following simulations, we stop factorization and delete the variables corresponding to the ith to the $(|W'|)$th diagonal elements.

Let V be the index set of support vector candidates. At the start of training, V includes all the indices, i.e., $V = \{1, \ldots, M\}$. Assume that the maximum working set size $|W|_{\max}$ is given. We change the working set size according to the change of $|V|$ as follows:

$$|W| = \begin{cases} |W|_{\max} & \text{for} \quad |V| \geq |W|_{\max}, \\ \max\{2, |V|\} & \text{for} \quad |V| < |W|_{\max}. \end{cases} \tag{5.91}$$

We call the corrections of all the variables $\alpha_i \, (i \in V)$ *one epoch of training.* At the beginning of a training epoch, we select one variable $\alpha_i \, (i \in V)$ as α_s and set $W = \{i\}$ and $V = V - \{i\}$. Then we randomly select $(|W|-1)$ variables $\alpha_i \, (i \in V)$, where $|W|$ is given by (5.91), and delete and add the associated indices from V and to W, respectively. We calculate the variable vector $\boldsymbol{\alpha}_W^{\text{new}}$ and iterate the procedure until V is empty.

For each calculation of $\boldsymbol{\alpha}_W^{\text{new}}$, we check if

$$|r \, \Delta\alpha_i| < \varepsilon_{\text{var}} \tag{5.92}$$

holds for all $i \, (i \in W)$, where ε_{var} is a tolerance of convergence for the variables. At the end of a training epoch, we calculate the bias term b_i for $\alpha_i \, (i \in W)$ that is within the bounds, i.e., $0 < \alpha_i < C$ by

$$b_i = y_i - \sum_{j \in W} \alpha_j \, y_j \, H(\mathbf{x}_j, \mathbf{x}_i), \tag{5.93}$$

and their average b_{ave} by

$$b_{\text{ave}} = \frac{1}{N_{\text{b}}} \sum_{\substack{i \in W, \\ 0 < \alpha_i < C}} b_i, \tag{5.94}$$

where N_{b} is the number of $\alpha_i \, (\in V)$ that satisfies $0 < \alpha_i < C$.

To accelerate training, for N_{ite} consecutive iterations, where N_{ite} is the positive integer, we confine calculations within the support vector candidates [190]. Namely, at the end of the training epoch we delete the index i with $\alpha_i = 0$ from V, and proceed with training.

If this does not happen and (5.92) holds for all $\boldsymbol{\alpha}_W$s in the training epoch and if

$$\frac{|b_{\text{ave}} - b_i|}{|b_{\text{ave}}|} \leq \varepsilon_{\text{b}} \tag{5.95}$$

holds for all $i\,(i \in V, 0 < \alpha_i < C)$, we check whether there is a new candidate $\mathbf{x}_i\,(i \notin V)$ of support vectors that satisfy

$$y_j \left(\sum_{i \in V} \alpha_i \, y_i \, H(\mathbf{x}_i, \mathbf{x}_j) + b_{\text{ave}} \right) < 1. \tag{5.96}$$

If there is none, we stop training. Otherwise we add the indices to V and proceed with training. At the end of every $(I_{\text{ite}} + 1)$st training epoch, we check (5.96) and add the support vector candidates if there are some.

To further speed training, we impose the following condition:

$$\frac{Q^{(n+1)}(\boldsymbol{\alpha}) - Q^{(n)}(\boldsymbol{\alpha})}{Q^{(n)}(\boldsymbol{\alpha})} \le \varepsilon_{\text{q}}, \tag{5.97}$$

where ε_{q} is a small positive parameter. If this criterion is applied at the early stage of convergence, the calculation is terminated before the solution reaches the optimal solution. Therefore, we apply the criterion after $N_{\text{q}}\,(> 1)$ epoch of training.

To reduce the number of calculations of $H(\mathbf{x}_i, \mathbf{x}_j)$, we prepare an $N_M \times N_M$ matrix H for $H(\mathbf{x}_i, \mathbf{x}_j)$. When we calculate $H(\mathbf{x}_i, \mathbf{x}_j)$, where α_i and α_j are support vector candidates, i.e., i, $j \in V$, we store it into H in a way similar to how the cache memory does. Namely, if $i = j$, $H(\mathbf{x}_i, \mathbf{x}_i)$ is not stored in H, and the kth row and column of H are empty, we assign i to the kth column (row) and store $H(\mathbf{x}_i, \mathbf{x}_i)$ in H_{kk}. If $i \neq j$, and i and j are not assigned, we assign i and j to the empty columns (rows) k_1 and k_2, respectively, if they exist, and store $H(\mathbf{x}_i, \mathbf{x}_j)$ to $H_{k_1 k_2}$. If either of i and j is assigned already, we assign the one that is not and store $H(\mathbf{x}_i, \mathbf{x}_j)$ in H. When α_i is deleted from V and i is assigned to a column (row) of H, we clear the column (row). We define the hit ratio of H by

$$\frac{\text{Number of successful accesses}}{\text{Total number of accesses}} \times 100(\%), \tag{5.98}$$

where a successful access means that $H(\mathbf{x}_i, \mathbf{x}_j)$ is stored in H, and its calculation is not necessary.

5.6.2 Sequential Minimal Optimization

In the following, we discuss correction of variables for SMO, which is the steepest ascent method with $|W| = 2$. We restate the optimization problem. Maximize

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \, \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) \tag{5.99}$$

subject to the constraints

$$\sum_{i=1}^{M} y_i \, \alpha_i = 0, \qquad 0 \le \alpha_i \le C \quad \text{for} \quad i = 1, \dots, M. \tag{5.100}$$

Solving the equality in (5.100) for α_s, we get

$$\alpha_s = - \sum_{\substack{i \ne s, \\ i=1}}^{M} y_s \, y_i \, \alpha_i. \tag{5.101}$$

Substituting (5.101) into (5.100), we can eliminate the equality constraint. Now because $Q(\boldsymbol{\alpha})$ is quadratic, we can express the change of $Q(\boldsymbol{\alpha})$, $\Delta Q(\boldsymbol{\alpha})$, for the change of α_i, $\Delta \alpha_i$, by

$$\Delta Q(\boldsymbol{\alpha}) = \frac{\partial Q(\boldsymbol{\alpha})}{\partial \alpha_i} \Delta \alpha_i + \frac{1}{2} \frac{\partial^2 Q(\boldsymbol{\alpha})}{\partial \alpha_i^2} (\Delta \alpha_i)^2. \tag{5.102}$$

Considering that

$$\frac{\partial \alpha_s}{\partial \alpha_i} = - y_s \, y_i, \tag{5.103}$$

we derive the partial derivatives of $Q(\boldsymbol{\alpha})$ with respect to α_i $(i \ne s, i = 1, \dots, M)$,

$$\begin{aligned}
\frac{\partial Q(\boldsymbol{\alpha})}{\partial \alpha_i} &= 1 + \frac{\partial \alpha_s}{\partial \alpha_i} - \sum_{j=1}^{M} \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) \\
&\quad - \sum_{j=1}^{M} \alpha_j \, y_s \, y_j \, \frac{\partial \alpha_s}{\partial \alpha_i} \, H(\mathbf{x}_i, \mathbf{x}_j) \\
&= 1 - y_s \, y_i - \sum_{j=1}^{M} \alpha_j \, y_i \, y_j \, H(\mathbf{x}_i, \mathbf{x}_j) \\
&\quad + \sum_{j=1}^{M} \alpha_j \, y_j \, y_i \, H(\mathbf{x}_s, \mathbf{x}_j).
\end{aligned} \tag{5.104}$$

Using (5.104), the second partial derivatives of $Q(\boldsymbol{\alpha})$ with respect to α_i $(i \ne s, i = 1, \dots, M)$ are

$$\frac{\partial^2 Q(\boldsymbol{\alpha})}{\partial \alpha_i^2} = -H(\mathbf{x}_i, \mathbf{x}_i) + 2 \, H(\mathbf{x}_i, \mathbf{x}_s) - H(\mathbf{x}_s, \mathbf{x}_s). \tag{5.105}$$

From (2.59),

$$\begin{aligned}
\frac{\partial^2 Q(\boldsymbol{\alpha})}{\partial \alpha_i^2} &= -H(\mathbf{x}_i, \mathbf{x}_i) + 2 \, H(\mathbf{x}_i, \mathbf{x}_s) - H(\mathbf{x}_s, \mathbf{x}_s) \\
&= -\mathbf{g}(\mathbf{x}_i)^T \mathbf{g}(\mathbf{x}_s) + 2 \, \mathbf{g}(\mathbf{x}_i)^T \mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s)^T \mathbf{g}(\mathbf{x}_s) \\
&= - \left(\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s) \right)^T \left(\mathbf{g}(\mathbf{x}_i) - \mathbf{g}(\mathbf{x}_s) \right) \le 0.
\end{aligned} \tag{5.106}$$

Thus, if $\partial^2 Q(\boldsymbol{\alpha})/\partial \alpha_i^2 < 0$, $\Delta Q(\boldsymbol{\alpha})$ has the maximum at

$$\Delta\alpha_i = -\frac{\dfrac{\partial Q(\boldsymbol{\alpha})}{\partial \alpha_i}}{\dfrac{\partial^2 Q(\boldsymbol{\alpha})}{\partial \alpha_i^2}}. \tag{5.107}$$

As initial values, we set $\alpha_i = 0$ so that they satisfy (5.100). For $i \neq s$, $i = 1, \ldots, M$, if $\partial^2 Q(\boldsymbol{\alpha})/\partial \alpha_i^2 > 0$, we modify α_i by

$$\alpha_i^{\text{new}} = \alpha_i^{\text{old}} + r\,\Delta\alpha_i, \tag{5.108}$$

where $0 < r \leq 1$ and is determined so that α_i^{new} and α_s^{new} satisfy the upper and lower bounds (see Fig. 5.8).

Then α_s is given as follows:

1. if $y_i \neq y_s$,
$$\alpha_s^{\text{new}} = \alpha_s^{\text{old}} + r\,\Delta\alpha_i; \tag{5.109}$$
2. if $y_i = y_s$,
$$\alpha_s^{\text{new}} = \alpha_s^{\text{old}} - r\,\Delta\alpha_i. \tag{5.110}$$

Because $Q(\boldsymbol{\alpha})$ is monotonic for the change of $\alpha_i + r\,\Delta\alpha_i$ ($r \in (0, 1]$), by this modification the resultant $Q(\boldsymbol{\alpha})$ is nondecreasing. Thus the steepest ascent is guaranteed.

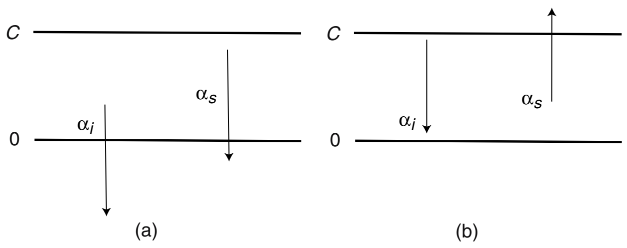


Fig. 5.8. Satisfying the upper and lower bounds: (**a**) $y_i \neq y_s$. Because α_i violates the constraint more than α_s, r is determined so that $\alpha_i^{\text{new}} = 0$, i.e., $r\,\Delta\alpha_i = -\alpha_i^{\text{old}}$. (**b**) $y_i = y_s$. Because α_s violates the constraint, r is adjusted so that $\alpha_s^{\text{new}} = C$, i.e., $r\,\Delta\alpha_i = \alpha_i^{\text{old}} - C$

5.6.3 Training of L2 Soft-Margin Support Vector Machines

The steepest ascent method obtained for the L1 soft-margin support vector machine can be extended to the L2 soft-margin support vector machine by replacing $H(\mathbf{x}_i, \mathbf{x}_j)$ with $H(\mathbf{x}_i, \mathbf{x}_j) + \delta_{ij}/C$ except for calculating the decision function (see Section 2.4). In the following performance evaluation, we compare these two support vector machine architectures.

5.6.4 Performance Evaluation

We used a Sun UltraSPARC-IIi (335MHz) workstation to train fuzzy pairwise support vector machines for the thyroid, blood cell, and hiragana data sets listed in Table 1.1. The input ranges were scaled into $[-1, 1]$. We used polynomial kernels and RBF kernels. The parameters used for simulations for L1 and L2 SVMs are as follows: $C = 5000$, $\varepsilon_{\mathrm{b}} = 10^{-2}$, $\varepsilon_{\mathrm{var}} = 10^{-6}$, $\varepsilon_{\mathrm{q}} = 10^{-7}$, $N_{\mathrm{ite}} = 10$, and $N_{\mathrm{q}} = 50$. Because for the thyroid data and blood cell data, $N_{\mathrm{q}} = 50$ was too small, we set as follows: for the thyroid data except for $d = 4$ and the blood cell data, we set $N_{\mathrm{q}} = 100$; for the thyroid data with $d = 4$, we set $N_{\mathrm{q}} = 600$.

To compare the training time with the quadratic programming technique, we used the software developed by London University[2] [210]. We used the primal-dual interior-point method (PDIP) as the optimization technique. The working set size was set to be 50.

Table 5.5 shows the results for the different training methods. In the table, columns PDIP, SMO, SAM (steepest ascent method for L1 SVM), L2 SMO (for L2 SVM), and L2 SAM (for L2 SVM) show the training time and in parentheses the speedup ratios to SMO. For the thyroid data and blood cell data with linear kernels, training by the L2 SAM is much faster than that by the SAM. But except for this, there is not much difference between SAM and the L2 SAM. Because the implementation of SMO is different from that in [190], only comparison of the training time with that of SAM is meaningful. In [190], SMO was shown to be faster than the projected conjugate gradient method with the working set size of 500. But, comparing PDIP and SMO, PDIP outperformed SMO for all the data sets evaluated. For PDIP also the working set size affected the training time considerably. For instance, without the decomposition technique, the training time by PDIP for blood cell data with $d = 4$ was 936 seconds. Thus the optimal selection of the working set size for the primal-dual interior-point method is important.

Comparing SMO and SAM, there is not much difference for the hiragana data but for the thyroid and blood cell data training by SAM is much faster for most cases. SAM is comparable with PDIP for hiragana data, but for thyroid data and blood cell data with $d = 1$, PDIP is much faster. L2 SAM is comparable to PDIP for the blood cell data, but for the thyroid data with $d = 1$, PDIP is faster.

According to the simulations, there is not much difference between L1 SVMs and L2 SVMs. The difference occurred for the thyroid data and blood cell data with $d = 1$. For larger $|W|$, training by L2 SVMs was faster. This is because the thyroid data and blood cell data were difficult to classify with $d = 1$ and L2 SVMs exploited the positive definiteness of the Hessian matrix.

[2]http://svm.cs.rhbnc.ac.uk/

Table 5.5. Performance comparison of training methods

Data	Parm	PDIP (s)	SMO (s)	SAM (s)	L2 SMO (s)	L2 SAM (s)
Thyroid	Linear	**38 (40)**	1531 (1)	931 (1.6)	3650 (0.42)	154 (9.9)
	$d4$	**14 (145)**	2032 (1)	109 (19)	2362 (0.86)	134 (15)
	$\gamma10$	**108 (37)**	4044 (1)	424 (9.5)	4309 (0.94)	390 (10)
Blood cell	Linear	**20 (23)**	463 (1)	150 (3.1)	932 (0.50)	28 (17)
	$d4$	**19 (18)**	338 (1)	31 (11)	353 (0.96)	31 (11)
	$\gamma10$	185 (1.3)	238 (1)	**165 (1.4)**	235 (1.0)	**165 (1.4)**
Hiragana-50	$d2$	143 (1.4)	202 (1)	**117 (1.7)**	215 (0.94)	124 (1.6)
	$\gamma0.1$	288 (1.1)	321 (1)	**194 (1.6)**	333 (0.96)	199 (1.6)
Hiragana-105	$d2$	**318 (2.1)**	671 (1)	394 (1.7)	698 (0.96)	414 (1.6)
	$\gamma0.1$	1889 (1.2)	2262 (1)	**1479 (1.5)**	2316 (0.98)	1525 (1.5)
Hiragana-13	$d2$	99 (1.4)	136 (1)	**93 (1.5)**	147 (0.93)	104 (1.3)
	$\gamma1$	**181 (1.8)**	325 (1)	190 (1.7)	333 (0.98)	195 (1.7)

5.7 Training of Linear Programming Support Vector Machines

In this section we discuss training of LP support vector machines by linear programming combined with the decomposition technique.

5.7.1 Primal-Dual Problems

Letting $\alpha_i = \alpha_i^+ - \alpha_i^-$ and $b = b^+ - b^-$, where $\alpha_i^+ \geq 0$, $\alpha_i^- \geq 0$, $b^+ \geq 0$, $b^- \geq 0$, in (4.46) and (4.47) we obtain the following LP support vector machine. Minimize

$$Q(\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-, \boldsymbol{\xi}) = \sum_{i=1}^{M} (\alpha_i^+ + \alpha_i^- + C\,\xi_i) \tag{5.111}$$

subject to

$$y_j \left(\sum_{i=1}^{M} (\alpha_i^+ - \alpha_i^-) H(\mathbf{x}_j, \mathbf{x}_i) + b^+ - b^- \right) + \xi_j \geq 1 \quad \text{for} \quad j = 1, \ldots, M. \tag{5.112}$$

Let (5.111) and (5.112) be a primal problem. Because there are M inequality constraints, the number of dual variables is M. Then the dual problem is given as follows. Maximize

$$\sum_{i=1}^{M} z_i \tag{5.113}$$

subject to

$$\sum_{i=1}^{M} y_i\, H(\mathbf{x}_i, \mathbf{x}_j)\, z_i \le 1 \quad \text{for} \quad j = 1, \dots, M, \tag{5.114}$$

$$\sum_{i=1}^{M} y_i\, H(\mathbf{x}_i, \mathbf{x}_j)\, z_i \ge -1 \quad \text{for} \quad j = 1, \dots, M, \tag{5.115}$$

$$\sum_{i=1}^{M} y_i\, z_i = 0, \tag{5.116}$$

$$C \ge z_i \ge 0 \quad \text{for} \quad j = 1, \dots, M, \tag{5.117}$$

where $z_i\,(i = 1, \dots, M)$ are dual variables.

For the primal problem, for $\alpha_i^+ = \alpha_i^- = 0$, $\xi_i = 1\,(i = 1, \dots, M)$, and $b^+ = b^- = 0$, the inequality constraints are satisfied. Also for the dual problem, for $z_i = 0\,(i = 1, \dots, M)$, the inequality and equality constraints are satisfied. Thus, the primal and dual problems have the optimal solutions and the values of the objective functions at the optimal solutions are the same.

Introducing the slack variables $u_i\,(i = 1, \dots, M)$ into (5.112), we obtain

$$y_j \left(\sum_{i=1}^{M} (\alpha_i^+ - \alpha_i^-) H(\mathbf{x}_j, \mathbf{x}_i) + b^+ - b^- \right) + \xi_j = 1 + u_j, \quad u_j \ge 0 \quad \text{for} \quad j = 1, \dots, M. \tag{5.118}$$

Likewise, introducing the slack variables v_i^+, v_i^-, and $w_i^+\,(i = 1, \dots, M)$ into (5.114), (5.115), and (5.117), respectively, we obtain

$$\sum_{i=1}^{M} y_i\, H(\mathbf{x}_i, \mathbf{x}_j)\, z_i + v_j^+ = 1 \qquad \text{for} \quad j = 1, \dots, M, \tag{5.119}$$

$$\sum_{i=1}^{M} y_i\, H(\mathbf{x}_i, \mathbf{x}_j)\, z_i = v_j^- - 1 \qquad \text{for} \quad j = 1, \dots, M, \tag{5.120}$$

$$z_i + w_i = C, \quad v_i^+ \ge 0 \quad v_i^- \ge 0, \quad w_i \ge 0 \qquad \text{for} \quad i = 1, \dots, M. \tag{5.121}$$

The complementarity conditions, which give the optimality of the solution are given by

$$\alpha_i^+\, v_i^+ = 0 \quad \text{for} \quad i = 1, \dots, M, \tag{5.122}$$

$$\alpha_i^-\, v_i^- = 0 \quad \text{for} \quad i = 1, \dots, M, \tag{5.123}$$

$$\xi_i\, w_i = 0 \quad \text{for} \quad i = 1, \dots, M, \tag{5.124}$$

$$u_i\, z_i = 0 \quad \text{for} \quad i = 1, \dots, M. \tag{5.125}$$

Therefore, if

$$\alpha_i = \alpha_i^+ - \alpha_i^- = 0, \tag{5.126}$$

$$\xi_i = 0, \tag{5.127}$$

$$z_i = 0, \tag{5.128}$$

it is not necessary to check v_i^+, v_i^-, w_i, and u_i. Thus, the training data $\mathbf{x}_i$ that satisfy (5.126) to (5.128) do not affect the solution even if they are removed. Namely, for the training data set $\{\mathbf{x}, \ldots, \mathbf{x}_M\}$, the smallest set of data that does not satisfy (5.126) to (5.128) constitutes a set of "support vectors," in that it forms the smallest subset of the training data set that gives the same solution as that of the original training data set. This means that, unlike conventional support vector machines, only retaining the training data $\mathbf{x}_i$ associated with the nonzero α_i does not guarantee giving the same optimal solution with that given by the original training data set.

5.7.2 Training by Decomposition

If the number of training data is very large, decomposition of the problem becomes a necessity. In such a situation, we can combine either the simplex method or the primal-dual interior-point method with the decomposition technique. If we use the simplex method, we need only to solve the primal or dual problem. This is because if we solve one, the other is also solved [63, 255].

The complementarity conditions given by (5.122) to (5.125) play an important role in decomposition; for the variables in the fixed set, we detect the variables that do not satisfy the complementarity conditions. If they exist, we add them to the working set and delete the variables that satisfy (5.126) to (5.128) and iterate the training.

6

Feature Selection and Extraction

Conventional classifiers do not have a mechanism to control class boundaries. Thus if the number of input variables is large compared to the number of training data, class boundaries may not overlap. In such a situation, the generalization ability of the conventional classifiers may not be good. Therefore, to improve the generalization ability, we usually generate a small set of features from the original input variables by either feature selection or feature extraction.

Because support vector machines directly determine the class boundaries by training, the generalization ability does not degrade greatly even when the number of input variables is large. Vapnik [257] even claims that feature selection or feature extraction is not necessary for support vector machines. But it is important, even using support vector machines.

In this chapter, we first survey feature selection methods using support vector machines and show how feature selection affects generalization ability of a support vector machine for some benchmark data sets. Then we discuss one of the feature extraction methods, kernel discriminant analysis, which extracts, in the feature space, a feature that maximizes the distance between class centers and minimizes the sum of class variances.

6.1 Procedure for Feature Selection

In selecting features, we first need to determine some appropriate selection criterion. A recognition rate can be a selection criterion. But because training a classifier takes time, we use a more simplified criterion. One of the criteria is the exception ratios that approximate the overlap between classes [3, pp. 217–23]. In the following section, we discuss the criteria that are suitable for support vector machines.

The forward or backward selection method using a selection criterion is widely used. In backward selection, we start from all the features and delete one feature at a time, which deteriorates the selection criterion the least. We delete features until the selection criterion reaches a specified value. In forward selection, we start from an empty set of features and add one feature at a time, which improves the selection criterion the most. We iterate this procedure until the selection criterion reaches a specified value. Because forward or backward selection is slow, we may add or delete more than one feature at a time based on feature ranking, or we may combine backward and forward selection.

Because these selection methods are local optimization techniques, global optimality of feature selection is not guaranteed. Usually, backward selection is slower but rather is stabler in selecting optimal features than forward selection [3]. If a selection criterion is monotonic for deletion or addition of a feature, we can use optimization techniques such as the branch-and-bound technique.

Another way is to combine training and features selection; Because training of support vector machines results in solving a quadratic optimization problem, feature selection can be done by modifying the objective function.

6.2 Feature Selection Using Support Vector Machines

In this section, we discuss some of the feature selection methods based on support vector machines. The methods are classified into two: backward or forward feature selection based on some selection criterion [79, 105, 172] and SVM-based feature selection, in which a feature selection criterion is added to the objective function [37] or forward feature selection is done by changing the margin parameter [41, 42].

6.2.1 Backward or Forward Feature Selection

Selection Criteria

The selection criterion used in the literature is, except for some cases [79, 172], the margin [34, 105, 188, 197]. In addition, in most cases, a linear support vector machine is used. In [266], selection of features in support vector machines with polynomial kernels is discussed, but this is for deletion of feature space variables, not input variables.

Assume that a classification problem is linearly separable in the feature space. Then training the support vector machine with the associated kernel results in maximizing the margin δ or minimizing $\|\mathbf{w}\|$.

Now we show that the margin is nonincreasing for the deletion of the input variable so long as the classification problem is separable in the feature space for the reduced input variables. Namely, the margin remains the same or decreases. First, we show that the margin is nonincreasing when the problem is linearly separable in the input space. Figure 6.1 shows the two-dimensional case where the margin decreases if x_2 is deleted.

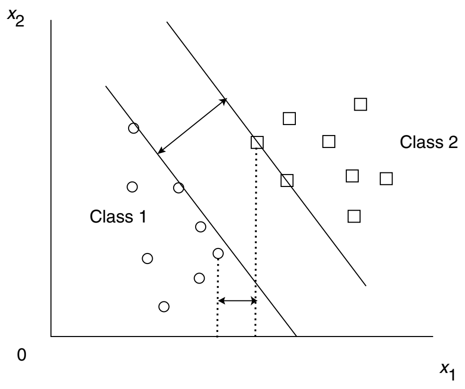


Fig. 6.1. Decrease of the margin by deleting x_2

Figure 6.2 shows the two-dimensional case where the optimal separating line is parallel to x_2. In this case, x_2 does not contribute in classification, and the margin remains the same even if x_2 is deleted.

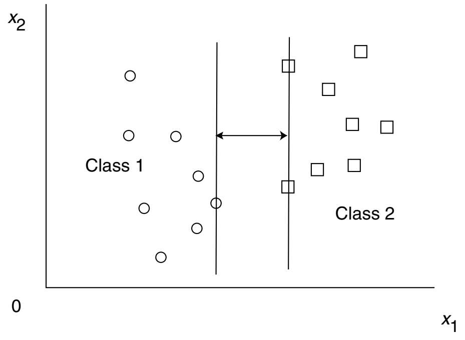


Fig. 6.2. The margin remains the same by deleting x_2

In general, if the optimal separating hyperplane is not parallel to an input variable, the deletion of the variable results in the decrease in the margin. But if the optimal hyperplane is parallel to input variables, the margin does not change for the deletion of these variables.

This is paraphrased as follows. If some elements of **w** are zero, the deletion of the associated input variables does not change the optimal hyperplane for the remaining variables. But if we delete variables associated with nonzero elements, the optimal solution changes. Thus the magnitude of the margin decreases.

We can extend this discussion to the feature space. If an input variable is deleted, some of the variables that span the feature space are deleted. Thus, the margin is nonincreasing for the deletion of input variables.

Therefore, in this situation, it is natural to delete input variables as far as possible under the constraint that the classification problem is linearly separable for the reduced input variables and under the constraint that

$$\frac{\delta - \delta'}{\delta} < \varepsilon, \tag{6.1}$$

where δ' is the margin for the reduced variables and ε is a small positive value.

Feature Ranking

The change of $\|\mathbf{w}\|^2$ for the deletion of the kth input variable needs to be calculated by training the SVM with the deleted input variable. But this is time-consuming. Thus, we consider how to choose the deletion candidate from the input variables. For the linear kernel, if $w_k^2 = 0$, the optimal hyperplane is the same for the deletion of the kth input variable. Even if w_k^2 is not zero, if the value is small, the deletion of the kth input variable does not affect very much for the optimal hyperplane. Thus, we can choose the input variable with the minimum w_k^2. This is the same measure as that proposed in [105].

For the kernel other than the linear kernel, multiple variables in the feature space are deleted for the deletion of an input variable. In this case, we also choose the input variable with the minimum square sum of weights associated with the input variable. The square sum of weights in the feature space that correspond to the kth input variable, $\Delta_k \|\mathbf{w}\|^2$, is estimated by

$$\Delta^{(k)} \|\mathbf{w}\|^2 = \sum_{i,j \in S} \alpha_i \, y_i \alpha_j \, y_j \, (H(\mathbf{x}_i, \mathbf{x}_j) - H(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)})), \tag{6.2}$$

where $\mathbf{x}^{(k)}$ is the vector with the kth element of $\mathbf{x}$ set to zero.

If more than one $\Delta^{(k)} \|\mathbf{w}\|^2$ are zero, we can safely delete the associated variables, but this may be rare. We may choose plural variables with $\Delta^{(k)} \|\mathbf{w}\|^2$ smaller than a threshold. But how to determine the threshold is difficult. In [34], three random variables are added to a regression problem, and the average value of the three weights associated with the variables is used for the threshold.

Backward Feature Selection

In backward feature selection we first train the support vector machine using all the input variables. Then we delete the input variable ranked first in the feature ranking and train the support vector machine with the reduced input variables. We iterate the deletion procedure until the stopping criterion is satisfied.

The procedure of backward feature selection is as follows:

1. Train the SVM using all the input variables. Let the margin be δ_0.
2. Let $\Delta^{(k)}\|\mathbf{w}\|^2$ be minimum. Then delete the kth input variable. (To speed up the deletion procedure, we may delete more than one variable by feature ranking.)
3. Train the SVM with the reduced input variables. If nonseparable or $(\delta_0 - \delta_c)/\delta_0 < \varepsilon$, terminate the algorithm, where δ_c is a current margin and $0 < \varepsilon < 1$. Otherwise, go to Step 2.

We can extend this method for forward selection if we calculate the margin change for the variable addition by training, which is time-consuming. But without training, estimation of the change is difficult.

6.2.2 Support Vector Machine–Based Feature Selection

Instead of backward or forward feature selection, feature selection can be done while training by reformulating training [37, 99, 188, 266, 268]. The idea of feature selection developed by Bradley and Mangasarian [37] is quite similar to that by Guyon, Weston, Barnhill, and Vapnik [105]. In the former, feature selection is done by minimizing the classification error and the number of nonzero components of $\mathbf{w}$. The problem is formulated as follows. Minimize

$$(1-\lambda)\,\frac{1}{M}\sum_{i=1}^{M}\xi_i + \lambda\sum_{i=1}^{n} w_i^* \tag{6.3}$$

subject to

$$y_i\,(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for} \quad i = 1,\ldots,M, \tag{6.4}$$

where $\lambda\,(1 > \lambda \geq 0)$ is a regularization parameter and

$$w_i^* = \begin{cases} 1 & \text{for } w_i \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{6.5}$$

In (6.3), the second term minimizes the number of nonzero components of $\mathbf{w}$. The variables with zero w_i are regarded as redundant.

This optimization problem is written by introducing a positive vector $\mathbf{v}$. Minimize

$$(1-\lambda)\,\frac{1}{M}\sum_{i=1}^{M}\xi_i + \lambda\sum_{i=1}^{n} v_i^* \tag{6.6}$$

subject to

$$y_i\,(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for} \quad i = 1,\ldots,M, \tag{6.7}$$

$$-v_i \leq w_i \leq v_i \quad \text{for} \quad i = 1,\ldots,n, \tag{6.8}$$

where v_i^* is defined as in (6.5).

Because v_i^* is a step function, it is approximated by

$$1 - \exp(-\gamma \, v_i), \tag{6.9}$$

where γ is a positive parameter. Then the optimization problem is solved by concave minimization technique [37].

Let the objective function be

$$Q(\mathbf{w}, \boldsymbol{\xi}) = \lambda \sum_{i=1}^{m} |w_i| + \sum_{i=1}^{M} \xi_2^2, \tag{6.10}$$

where $\lambda = 1/C$ is the regularization parameter. For $\lambda = \infty$, $w_i = 0$, which means that all the input features are not used. Brown's idea [41, 42] is to decrease the value of λ toward zero and select the classifier with the optimal features. For $\lambda \in [0, \infty)$, the set of support vectors changes at finite points (see Section 2.6.2) and so does the set of input features. To facilitate searching the points where the set of support vectors changes, Brown proposed an iterative linear programming technique.

6.2.3 Feature Selection by Cross-Validation

Usually we select a feature selection criterion other than the recognition rate, because it is time-consuming to evaluate the recognition rate. But if the number of training data is small, the recognition rate can be used as a feature selection criterion.[1]

In the following we discuss backward feature selection estimating the generalization ability of the classifier by cross-validation of the training data.

Let the initial set of selected features be F^m, where m is the number of input variables, and the recognition rate of the validation set by cross-validation be R^m.

We delete the ith ($i = 1, \ldots, m$) feature temporally from F^m and estimate the generalization ability by cross-validation. Let the recognition rate of the validation set be R_i^m. We check whether the maximum R_i^m ($i \in \{1, \ldots, m\}$) is larger than or equal to R_m:

$$\max_{i=1,\ldots,m} R_i^m \geq R^m. \tag{6.11}$$

If (6.11) is not satisfied, we assume that the deletion of one feature results in degrading the generalization ability. Thus we cannot delete any feature from the original set of features F_m.

Assume that (6.11) is satisfied for k. Then we set

$$F^{m-1} = F^m - \{k\}. \tag{6.12}$$

[1]Professor N. Kasabov's lecture at Kobe University on June 1, 2004, showed the usefulness of this criterion.

Namely, we assume that the set of features F^{m-1} can realize the same generalization ability as F^m. To speed up selecting the features from F^{m-1}, we consider that the features that satisfy

$$R_i^m < R^m \tag{6.13}$$

are indispensable for classification and thus they cannot be deleted. Thus, we set the set of features that are candidates for deletion

$$S^{m-1} = \{i \,|\, R_i^m \geq R^m,\, i \neq k\}. \tag{6.14}$$

If S^{m-1} is empty, we stop deleting the feature. If it is not empty, we iterate the preceding backward selection procedure.

We evaluated the method using some of the data sets listed in Table 1.1. We estimated the generalization ability by five-fold cross-validation for a given kernel changing the value of C. Table 6.1 shows the results. The "Deleted" column lists the features deleted according to the algorithm and the "Validation" and "Test" columns show the recognition rates of the validation sets and test data sets, respectively. If the recognition rate of the training data is not 100 percent, it is shown in parentheses. For the iris and numeral data sets we used a polynomial kernel with degree 2 and for the blood cell and thyroid data sets we used polynomial kernels with degree 4. For the iris and numeral data sets, the recognition rates of the test data with deleted input variables are equal to or higher than those with all the input variables. For the blood cell data, the recognition rates with deleted input variables are equal to or lower, but the differences are small.

For the thyroid data set, many redundant features are included. Based on the analysis of class regions approximated by ellipsoids, five features, i.e., the third, eighth, eleventh, seventeenth, and twenty-first features, were selected as important features by the forward feature selection method [3]. Thus we started from these five features. The "Deleted" column in the table lists the remaining features. Three features, i.e., the third, eighth, and seventeenth were selected, and the recognition rate for the test data was higher than that for using all the features.

6.3 Feature Extraction

Principal component analysis is widely used for feature extraction. As a variant of PCA, kernel PCA, which will be discussed in Section 8.2, has been gaining wide acceptance. In [216], the simulation study showed that the combination of KPCA and the linear support vector machine gave better generalization ability than the nonlinear support vector machine. In [204, 205], KPCA is combined with least squares, which is a variant of kernel least squares discussed in Section 8.1.

Table 6.1. Feature selection by cross-validation. For the thyroid data, the "Deleted" column lists the remaining features

Data	Deleted	C	Validation (%)	Test (%)
Iris	None	5000	94.67	93.33
	3	500	96.00 (99.00)	96.00 (98.67)
	3, 1	5000	94.67	96.00
	3, 4	5000	94.67	93.33
Numeral	None	1	99.51 (99.97)	99.63
	4	1	99.75	99.63
	4, 10	1	99.75	99.76
	4, 10, 3	1	99.63	99.63
	4, 10, 3, 12	1	99.51 (99.94)	99.76
Blood cell	None	1	93.77 (96.23)	93.23 (96.51)
	1	1	94.38 (96.65)	93.06 (96.51)
	1, 13	1	94.51 (96.56)	93.03 (96.84)
	1, 13, 8	1	94.35 (96.41)	93.23 (96.67)
	1, 13, 8, 10	1	94.45 (96.48)	93.16 (96.71)
	1, 13, 8, 10, 9	1	94.54 (96.67)	92.97 (96.38)
	1, 13, 8, 10, 9, 6	1	94.25 (96.00)	92.45 (95.93)
Thyroid	None	10^5	97.96	97.93
	(3, 8, 11, 17, 21)	10^5	98.44 (99.85)	98.37 (99.81)
	(3, 8, 11, 21)	10^5	98.52 (99.77)	98.45 (99.81)
	(3, 8, 17)	10^4	98.52 (99.76)	98.48 (99.81)

Principal component analysis does not use class information. Thus, the first principal component is not necessarily useful for class separation. On the other hand, linear discriminant analysis, defined for a two-class problem, finds the component that maximally separates two classes [77, pp. 118–21]. Likewise, kernel discriminant analysis finds the component that maximally separates two classes in the feature space [26, 166], [215, pp. 457–68]. It is extended to multiclass problems, and there are also variants [146, 184]. In the following, we discuss kernel discriminant analysis for two-class problems.

Let the sets of m-dimensional data belonging to Class i ($i = 1, 2$) be $\{\mathbf{x}^i_1, \ldots, \mathbf{x}^i_{M_i}\}$, where M_i is the number of data belonging to Class i, and data $\mathbf{x}$ be mapped into the l-dimensional feature space by the mapping function

$\mathbf{g}(\mathbf{x})$. Now we find the l-dimensional vector $\mathbf{w}$ in which the two classes are separated maximally in the direction of $\mathbf{w}$ in the feature space.

The projection of $\mathbf{g}(\mathbf{x})$ on $\mathbf{w}$ is $\mathbf{w}^T\,\mathbf{g}(\mathbf{x})/\|\mathbf{w}\|$. In the following we assume that $\|\mathbf{w}\| = 1$, but this is not necessary. We find such $\mathbf{w}$ that maximizes the difference of the centers and minimizes the variances of the projected data.

The square difference of the centers of the projected data, d^2, is

$$\begin{aligned} d^2 &= (\mathbf{w}^T(\mathbf{c}_1 - \mathbf{c}_2))^2 \\ &= \mathbf{w}^T(\mathbf{c}_1 - \mathbf{c}_2)\,(\mathbf{c}_1 - \mathbf{c}_2)^T\,\mathbf{w}, \end{aligned} \tag{6.15}$$

where $\mathbf{c}_i$ are the centers of class i data:

$$\begin{aligned} \mathbf{c}_i &= \frac{1}{M_i}\sum_{j=1}^{M_i}\mathbf{g}(\mathbf{x}_j^i) \\ &= (\mathbf{g}(\mathbf{x}_1^i),\ldots,\mathbf{g}(\mathbf{x}_{M_i}^i))\begin{pmatrix}\frac{1}{M_i}\\ \vdots\\ \frac{1}{M_i}\end{pmatrix} \qquad \text{for} \quad i = 1, 2. \end{aligned} \tag{6.16}$$

We define

$$Q_B = (\mathbf{c}_1 - \mathbf{c}_2)\,(\mathbf{c}_1 - \mathbf{c}_2)^T \tag{6.17}$$

and call Q_B the *between-class scatter matrix.*

The variances of the projected data, s_i^2, are

$$s_i^2 = \mathbf{w}^T\,Q_i\,\mathbf{w} \qquad \text{for} \quad i = 1, 2, \tag{6.18}$$

where

$$\begin{aligned} Q_i &= \frac{1}{M_i}\sum_{j=1}^{M_i}(\mathbf{g}(\mathbf{x}_j^i) - \mathbf{c}_i)\,(\mathbf{g}(\mathbf{x}_j^i) - \mathbf{c}_i)^T \\ &= \frac{1}{M_i}\sum_{j=1}^{M}\mathbf{g}(\mathbf{x}_j^i)\,\mathbf{g}(\mathbf{x}_j^i)^T - \mathbf{c}_i\,\mathbf{c}_i^T \\ &= \frac{1}{M_i}\,(\mathbf{g}(\mathbf{x}_1^i),\ldots,\mathbf{g}(\mathbf{x}_{M_i}^i))\,(I_{M_i} - \mathbf{1}_{M_i})\begin{pmatrix}\mathbf{g}^T(\mathbf{x}_1^i)\\ \vdots\\ \mathbf{g}^T(\mathbf{x}_{M_i}^i)\end{pmatrix} \qquad \text{for} \quad i = 1, 2. \end{aligned} \tag{6.19}$$

Here, I_{M_i} is the $M_i \times M_i$ unit matrix and $\mathbf{1}_{M_i}$ is the $M_i \times M_i$ matrix with all elements being $1/M_i$. We define

$$Q_W = Q_1 + Q_2 \tag{6.20}$$

and call Q_W the *within-class scatter matrix.*

Now, we want to maximize

$$\begin{aligned} J(\mathbf{w}) &= \frac{d^2}{s_1^2 + s_2^2} \\ &= \frac{\mathbf{w}^T\, Q_B\, \mathbf{w}}{\mathbf{w}^T\, Q_W\, \mathbf{w}}, \end{aligned} \tag{6.21}$$

but because $\mathbf{w}$, Q_B, and Q_W are defined in the feature space, we need to use kernel tricks. Assume that a set of M' vectors $\{\mathbf{g}(\mathbf{y}_1), \ldots, \mathbf{g}(\mathbf{y}_{M'})\}$ spans the space generated by $\{\mathbf{g}(\mathbf{x}_1^1), \ldots, \mathbf{g}(\mathbf{x}_{M_1}^1), \mathbf{g}(\mathbf{x}_1^2), \ldots, \mathbf{g}(\mathbf{x}_{M_2}^2)\}$, where $\{\mathbf{y}_1, \ldots, \mathbf{y}_{M'}\} \subset \{\mathbf{x}_1^1, \ldots, \mathbf{x}_{M_1}^1, \mathbf{x}_1^2, \ldots, \mathbf{x}_{M_2}^2\}$ and $M' \leq M_1 + M_2$. Then $\mathbf{w}$ is expressed as

$$\mathbf{w} = (\mathbf{g}(\mathbf{y}_1), \ldots, \mathbf{g}(\mathbf{y}_{M'}))\, \boldsymbol{\alpha}, \tag{6.22}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{M'})^T$ and $\alpha_1, \ldots, \alpha_{M'}$ are scalars. Substituting (6.22) into (6.21), we obtain

$$J(\boldsymbol{\alpha}) = \frac{\boldsymbol{\alpha}^T\, K_B\, \boldsymbol{\alpha}}{\boldsymbol{\alpha}^T\, K_W\, \boldsymbol{\alpha}}, \tag{6.23}$$

where

$$K_B = (\mathbf{k}_{B_1} - \mathbf{k}_{B_2})\, (\mathbf{k}_{B_1} - \mathbf{k}_{B_2})^T, \tag{6.24}$$

$$\mathbf{k}_{Bi} = \begin{pmatrix} \displaystyle\frac{1}{M_i} \sum_{j=1}^{M_i} H(\mathbf{y}_1, \mathbf{x}_j^i) \\ \cdots \\ \displaystyle\frac{1}{M_i} \sum_{j=1}^{M_i} H(\mathbf{y}_{M'}, \mathbf{x}_j^i) \end{pmatrix} \quad \text{for} \quad i = 1, 2, \tag{6.25}$$

$$K_W = K_{W_1} + K_{W_2}, \tag{6.26}$$

$$\begin{aligned} K_{W_i} &= \frac{1}{M_i} \begin{pmatrix} H(\mathbf{y}_1, \mathbf{x}_1^i) \cdots H(\mathbf{y}_1, \mathbf{x}_{M_i}^i) \\ \cdots \\ H(\mathbf{y}_{M'}, \mathbf{x}_1^i) \cdots H(\mathbf{y}_{M'}, \mathbf{x}_{M_i}^i) \end{pmatrix} (I_{M_i} - \mathbf{1}_{M_i}) \\ &\quad \times \begin{pmatrix} H(\mathbf{y}_1, \mathbf{x}_1^i) \cdots H(\mathbf{y}_1, \mathbf{x}_{M_i}^i) \\ \cdots \\ H(\mathbf{y}_{M'}, \mathbf{x}_1^i) \cdots H(\mathbf{y}_{M'}, \mathbf{x}_{M_i}^i) \end{pmatrix}^T \quad \text{for} \quad i = 1, 2. \end{aligned} \tag{6.27}$$

Taking the partial derivative of (6.23) with respect to $\mathbf{w}$ and equating the resulting equation to zero, we obtain the following generalized eigenvalue problem:

$$K_B\, \boldsymbol{\alpha} = \lambda\, K_W\, \boldsymbol{\alpha}, \tag{6.28}$$

where λ is a generalized eigenvalue.

Substituting

$$K_W\, \boldsymbol{\alpha} = \mathbf{k}_{B_1} - \mathbf{k}_{B_2} \tag{6.29}$$

into the left-hand side of (6.28), we obtain

$$(\boldsymbol{\alpha}^T K_W \boldsymbol{\alpha}) K_W \boldsymbol{\alpha}. \tag{6.30}$$

Thus, by letting $\lambda = \boldsymbol{\alpha}^T K_W \boldsymbol{\alpha}$, (6.29) is a solution of (6.28).

Because K_{W_1} and K_{W_2} are positive semidefinite, K_W is positive semidefinite. If K_W is positive definite, $\boldsymbol{\alpha}$ is given by

$$\boldsymbol{\alpha} = K_W^{-1} (\mathbf{k}_{B_1} - \mathbf{k}_{B_2}). \tag{6.31}$$

Even if we choose independent vectors $\mathbf{y}_1, \ldots, \mathbf{y}_{M'}$, for nonlinear kernels, K_W may be positive semidefinite, i.e., singular. One way to overcome singularity is to add positive values to the diagonal elements [166]:

$$\boldsymbol{\alpha} = (K_W + \varepsilon I)^{-1} (\mathbf{k}_{B_1} - \mathbf{k}_{B_2}), \tag{6.32}$$

where ε is a small positive parameter. Another way is to use the pseudo-inverse:

$$\boldsymbol{\alpha} = K_W^{+} (\mathbf{k}_{B_1} - \mathbf{k}_{B_2}), \tag{6.33}$$

which gives the solution in a least squares sense.

7

Clustering

Unlike multilayer neural networks, support vector machines can be formulated for one-class problems. This technique is called *domain description* or *one-class classification* and is applied to clustering and detection of outliers for both pattern classification and function approximation [158]. In this chapter, we first discuss domain description, in which a region for a single class is approximated by a hypersphere in the feature space. Then we discuss an extension of domain description to clustering.

7.1 Domain Description

In pattern classification, we consider more than one class. And, if data with only one class are available as training data, we cannot use multilayer neural networks. In such a situation, if we approximate the class region by some method and if we test whether new data are outside the region, we can detect outliers. The approximation of the class region is called the *domain description.* Tax and Duin [246, 247] extended the support vector method to domain description. In the following we discuss their method.

We consider approximating the class region by the minimum hypersphere with center $\mathbf{a} = (a_1, \ldots, a_M)^T$ and radius R in the input space, excluding the outliers. Let $\mathbf{x}_i, (i = 1, \ldots, M)$ be data belonging to one class. Then the problem is to minimize

$$Q_p(R, \mathbf{a}, \boldsymbol{\xi}) = R^2 + C \sum_{i=1}^{M} \xi_i \tag{7.1}$$

subject to

$$\|\mathbf{x}_i - \mathbf{a}\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0 \qquad \text{for} \quad i = 1, \ldots, M, \tag{7.2}$$

where $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_M)^T$ is the slack variable vector and C determines the trade-off between the hypersphere volume and outliers.

Introducing the nonnegative Lagrange multipliers α_i and γ_i, we obtain

$$\begin{aligned} Q_d(R, \mathbf{a}, \boldsymbol{\alpha}, \boldsymbol{\xi}, \boldsymbol{\gamma}) = R^2 + C \sum_{i=1}^{M} \xi_i & \\ - \sum_{i=1}^{M} \alpha_i \, (R^2 + \xi_i - \mathbf{x}_i^T \mathbf{x}_i + 2\mathbf{a}^T \mathbf{x}_i - \mathbf{a}^T \mathbf{a}) & \\ - \sum_{i=1}^{M} \gamma_i \, \xi_i, & \qquad (7.3) \end{aligned}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)^T$ and $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_M)^T$.

Setting the partial derivatives of $Q_d(R, \mathbf{a}, \boldsymbol{\alpha}, \boldsymbol{\xi}, \boldsymbol{\gamma})$ with respect to R, $\mathbf{a}$, and $\boldsymbol{\xi}$ to zero, we obtain

$$\sum_{i}^{M} \alpha_i = 1, \qquad (7.4)$$

$$\mathbf{a} = \sum_{i=1}^{M} \alpha_i \, \mathbf{x}_i, \qquad (7.5)$$

$$C - \alpha_i - \gamma_i = 0 \quad \text{for} \quad i = 1, \ldots, M. \qquad (7.6)$$

Substituting (7.4) to (7.6) into (7.3) gives the following maximization problem. Maximize

$$Q_d(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i \, \mathbf{x}_i^T \mathbf{x}_i - \sum_{i,j=1}^{M} \alpha_i \, \alpha_j \, \mathbf{x}_i^T \mathbf{x}_j \qquad (7.7)$$

subject to

$$\sum_{i}^{M} \alpha_i = 1, \qquad (7.8)$$

$$0 \le \alpha_i \le C \quad \text{for} \quad i = 1, \ldots, M. \qquad (7.9)$$

The KKT conditions are as follows:

$$\alpha_i \, (R^2 + \xi_i - \mathbf{x}_i^T \mathbf{x}_i + 2\mathbf{a}^T \mathbf{x}_i - \mathbf{a}^T \mathbf{a}) = 0 \quad \text{for} \quad i = 1, \ldots, M, \qquad (7.10)$$

$$\gamma_i \, \xi_i = 0 \quad \text{for} \quad i = 1, \ldots, M. \qquad (7.11)$$

If $0 < \alpha_i < C$, from (7.6), $\gamma_i \neq 0$. Thus $\xi_i = 0$ and

$$R = \|\mathbf{x}_i - \mathbf{a}\|, \qquad (7.12)$$

where $\mathbf{a}$ is given by (7.5) and $\mathbf{x}_i$ is a support vector. Namely, the unbounded support vectors form the surface of the hypersphere.

If $\alpha_i = C$, $\gamma_i = 0$. Thus if $\xi_i > 0$, the bounded support vectors are outside of the hypersphere and thus are outliers. Notice that, from (7.4), $1 \geq \alpha_i \geq 0$ is satisfied. And because at least two support vectors are necessary for defining a hypersphere, if $C \geq 1$, there are no bounded support vectors.

Thus the unknown datum $\mathbf{x}$ is inside the hypersphere if

$$\mathbf{x}^T\mathbf{x} - 2\sum_{i \in S} \alpha_i \mathbf{x}^T \mathbf{x}_i + \sum_{\substack{i \in S, \\ j \in S}} \alpha_i \, \alpha_j \, \mathbf{x}_i^T \mathbf{x}_j \leq R^2, \tag{7.13}$$

where S is the set of support vectors.

If we want to determine the minimum volume of the hypersphere in the feature space, we change (7.2) to

$$\|\mathbf{g}(\mathbf{x}_i) - \mathbf{a}\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0 \qquad \text{for} \quad i = 1, \ldots, M, \tag{7.14}$$

where $\mathbf{g}(\mathbf{x})$ is the mapping function that maps $\mathbf{x}$ into the l-dimensional feature space and $\mathbf{a}$ is the center of the hypersphere in the feature space.

Introducing the Lagrange multipliers α_i and γ_i, we obtain

$$\begin{aligned} Q_d(R, \mathbf{a}, \boldsymbol{\alpha}, \boldsymbol{\xi}, \boldsymbol{\gamma}) = R^2 + C \sum_{i=1}^{M} \xi_i \\ - \sum_{i=1}^{M} \alpha_i \left(R^2 + \xi_i - H(\mathbf{x}_i, \mathbf{x}_i) + 2\mathbf{a}^T \mathbf{g}(\mathbf{x}_i) - \mathbf{a}^T \mathbf{a}\right) \\ - \sum_{i=1}^{M} \gamma_i \, \xi_i, \end{aligned} \tag{7.15}$$

where $H(\mathbf{x}_i, \mathbf{x}_i) = \mathbf{g}^T(\mathbf{x}_i) \, \mathbf{g}(\mathbf{x}_i)$.

Setting the partial derivatives of $Q_d(R, \mathbf{a}, \boldsymbol{\alpha}, \boldsymbol{\xi}, \boldsymbol{\gamma})$ with respect to R, $\mathbf{a}$, and $\boldsymbol{\xi}$ to zero, we obtain

$$\sum_{i}^{M} \alpha_i = 1, \tag{7.16}$$

$$\mathbf{a} = \sum_{i=1}^{M} \alpha_i \, \mathbf{g}(\mathbf{x}_i), \tag{7.17}$$

$$C - \alpha_i - \gamma_i = 0 \qquad \text{for} \quad i = 1, \ldots, M. \tag{7.18}$$

Thus substituting (7.16) to (7.18) into (7.15) gives the following maximization problem:[1] Maximize

[1]When kernels, such as RBF kernels, that depend only on $\mathbf{x} - \mathbf{x}'$ are used, the linear term in (7.19) is constant from (7.20). Then it is shown that the problem is equivalent to maximizing the margin in separating data from the origin [215, pp. 230–4].

$$Q_d(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i H(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,j=1}^{M} \alpha_i\, \alpha_j H(\mathbf{x}_i, \mathbf{x}_j) \tag{7.19}$$

subject to

$$\sum_{i}^{M} \alpha_i = 1, \tag{7.20}$$

$$0 \le \alpha_i \le C \qquad \text{for} \quad i = 1, \ldots, M. \tag{7.21}$$

For the unbounded support vectors $\mathbf{x}_i$,

$$R = \|\mathbf{g}(\mathbf{x}_i) - \mathbf{a}\|, \tag{7.22}$$

where $\mathbf{a}$ is given by (7.17). Thus the center is not implicitly expressed by kernels. The bounded support vectors with $\xi_i > 0$ are outside of the hypersphere and thus are outliers.

The unknown datum $\mathbf{x}$ is inside the hypersphere if

$$H(\mathbf{x}, \mathbf{x}) - 2 \sum_{i \in S} \alpha_i H(\mathbf{x}, \mathbf{x}_i) + \sum_{\substack{i \in S, \\ j \in S}} \alpha_i\, \alpha_j H(\mathbf{x}_i, \mathbf{x}_j) \le R^2, \tag{7.23}$$

where S is the set of support vectors.

According to the simulation with polynomial kernels, an approximated region mapped into the input space included a redundant space that did not include training data. But the use of RBF kernels showed better results [246]. To reduce the redundant space, Tax and Juszczak [248] used the kernel PCA to rescale the data in the feature space to the unit variance before one-class classification.

Example 7.1. In a one-dimensional problem, assume that we have two data: $x_1 = -1$ and $x_2 = 1$. For linear kernels, the one-class classification is to maximize

$$Q_d(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 - (\alpha_1 - \alpha_2)^2 \tag{7.24}$$

subject to

$$\alpha_1 + \alpha_2 = 1, \quad 0 \le \alpha_1 \le C, \quad 0 \le \alpha_2 \le C. \tag{7.25}$$

Let $C > 1$. Then the optimum solution is given by $\alpha_1 = \alpha_2 = 0.5$, and $a = 0$ and $R = 1$. Thus, the hypersphere is given by $x^2 = 1$, which is the minimum hypersphere that includes the two data in the input space.

For the polynomial kernel with degree 2, the one-class classification is to maximize

$$Q_d(\boldsymbol{\alpha}) = 4\alpha_1 + 4\alpha_2 - 4(\alpha_1^2 + \alpha_2^2) \tag{7.26}$$

subject to

$$\alpha_1 + \alpha_2 = 1, \quad 0 \le \alpha_1 \le C, \quad 0 \le \alpha_2 \le C. \tag{7.27}$$

Let $C > 1$. Then the optimum solution is given by $\alpha_1 = \alpha_2 = 0.5$. The center vector is $\mathbf{a} = 0.5\,(\mathbf{g}(x_1) + \mathbf{g}(x_2)) = (1, 0, 1)^T$ for the coordinates of $(x^2, \sqrt{2}x, 1)^T$, $R = \sqrt{2}$, and the hypersphere is given by $x^4 = 1$, which is equivalent to $x^2 = 1$.

Yuan and Casasent [280] proposed a support vector representation machine (SVRM), using the fact that with the RBF kernel data are on the surface of the unit hypersphere centered at the origin of the feature space because $\mathbf{g}^T(\mathbf{x})\,\mathbf{g}(\mathbf{x}) = \exp(-\gamma\,\|\mathbf{x} - \mathbf{x}\|^2) = 1$. In the SVRM, we determine vector $\mathbf{h}$, in the feature space with the minimum Euclidean norm, that satisfies $\mathbf{h}^T\mathbf{g}(\mathbf{x}_i) \geq 1$ for $i = 1, \dots, M$. Minimize

$$Q'_p(\mathbf{h}) = \frac{1}{2}\|\mathbf{h}\|^2 \tag{7.28}$$

subject to

$$\mathbf{h}^T\,\mathbf{g}(\mathbf{x}_i) \geq 1 \qquad \text{for} \quad i = 1, \dots, M. \tag{7.29}$$

The dimension of the feature space associated with the RBF kernel is infinite, but for simplicity the example of an SVRM shown in Fig. 7.1 assumes a two-dimensional feature space. In the figure, let training data lie on the arc between $\mathbf{g}(\mathbf{x}_1)$ and $\mathbf{g}(\mathbf{x}_2)$ and θ be the angle between $\mathbf{g}(\mathbf{x}_1)$ (or $\mathbf{g}(\mathbf{x}_2)$) and $\mathbf{h}$. Then

$$\mathbf{h}^T\mathbf{g}(\mathbf{x}_1) = \|\mathbf{h}\|\cos\theta = 1, \quad \mathbf{h}^T\mathbf{g}(\mathbf{x}_2) = 1 \tag{7.30}$$

must be satisfied. Therefore $\mathbf{x}_1$ and $\mathbf{x}_2$ are support vectors and $\mathbf{g}(\mathbf{x}_1)$ (or $\mathbf{g}(\mathbf{x}_2)$) and $\mathbf{h} - \mathbf{g}(\mathbf{x}_1)$ (or $\mathbf{h} - \mathbf{g}(\mathbf{x}_2)$) are orthogonal.

Any datum $\mathbf{x}$ on the arc satisfies $\mathbf{h}^T\mathbf{g}(\mathbf{x}) \geq 1$, and for any datum $\mathbf{x}$ that is not, $\mathbf{h}^T\mathbf{g}(\mathbf{x}) < 1$.

A soft-margin SVRM is given as follows. Minimize

$$Q'_p(\mathbf{h}, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{h}\|^2 + C\sum_{i=1}^{M}\xi_i \tag{7.31}$$

subject to

$$\mathbf{h}^T\,\mathbf{g}(\mathbf{x}_i) \geq 1 - \xi_i, \quad \xi_i \geq 0 \qquad \text{for} \quad i = 1, \dots, M, \tag{7.32}$$

where $\xi_i\ (\geq 0)$ are slack variables associated with $\mathbf{x}_i$, $\boldsymbol{\xi} = (\xi_1, \dots, \xi_M)^T$, and C is the margin parameter.

Introducing the nonnegative Lagrange multipliers α_i and β_i, we obtain

$$\begin{aligned} Q'_d(\mathbf{h}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = {} & \frac{1}{2}\,\|\mathbf{h}\|^2 + C\sum_{i\,=\,1}^{M}\xi_i \\ & - \sum_{i\,=\,1}^{M}\alpha_i\,\left(\mathbf{h}^T\,\mathbf{g}(\mathbf{x}_i) - 1 + \xi_i\right) - \sum_{i\,=\,1}^{M}\beta_i\,\xi_i, \end{aligned} \tag{7.33}$$

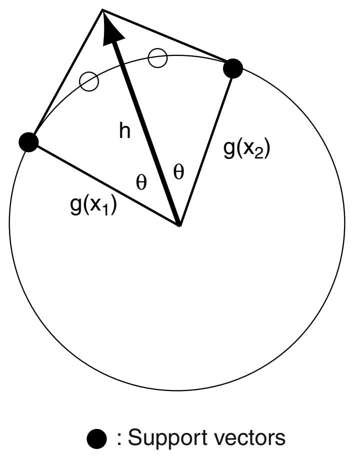


Fig. 7.1. Concept of an SVRM

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)^T$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_M)^T$.

The following conditions must be satisfied for the optimal solution:

$$\frac{\partial Q(\mathbf{h}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \mathbf{h}} = \mathbf{0}, \tag{7.34}$$

$$\frac{\partial Q(\mathbf{h}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \boldsymbol{\xi}} = \mathbf{0}. \tag{7.35}$$

Using (7.33), (7.34) and (7.35) reduce, respectively, to

$$\mathbf{h} = \sum_{i=1}^{M} \alpha_i \, \mathbf{g}(\mathbf{x}_i), \tag{7.36}$$

$$\alpha_i + \beta_i = C, \quad \alpha_i \geq 0, \quad \beta_i \geq 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{7.37}$$

Thus substituting (7.36) and (7.37) into (7.33), we obtain the following dual problem. Maximize

$$Q'_d(\boldsymbol{\alpha}) = \sum_{i}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \, \alpha_j \, H(\mathbf{x}_i, \mathbf{x}_j) \tag{7.38}$$

subject to

$$0 \leq \alpha_i \leq C \qquad \text{for} \quad i = 1, \ldots, M, \tag{7.39}$$

where $H(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \, \|\mathbf{x}_i - \mathbf{x}_j\|)$.

This problem is very similar to an L1 support vector machine. In the L1 support vector machine, deleting the equality constraint that corresponds to optimization of the bias term and setting $y_i = 1$ for $i = 1, \ldots, M$, we obtain the associated SVRM.

7.2 Extension to Clustering

Conventional clustering methods such as k-means clustering algorithm and fuzzy c-means clustering algorithms can be extended to feature space [95, 168].

The domain description discussed in Section 7.1 defines the region of data by a hypersphere in the feature space. The hypersphere in the feature space corresponds to clustered regions in the input space. Thus domain description can be used for clustering (Ben-Hur, Horn, Siegelmann, and Vapnik [28]). In the following we discuss how domain description can be extended to clustering.

We assume that there are no outliers. Namely, all the data are in or on the hypersphere. The problem is how to determine the clusters in the input space. Using Fig. 7.2, we explain the idea discussed in [28]. In the figure, two clusters are generated in the input space by approximating the region of data by a hypersphere in the feature space. The insides of the two regions correspond to the inside of the hypersphere in the feature space. In the figure, Data 1, 2, and 3 are in Cluster 1 but Datum 4 is in Cluster 2. If we move along the line segment connecting Data 1 and 4 from Datum 1, we go out of Cluster 1 and into Cluster 2. Namely, in the feature space, we go out of and then back to the hypersphere. Therefore, if part of the line segment connecting two data is out of the hypersphere, the two data may belong to different clusters. But this is not always true, as the line segment connecting Data 1 and 3 shows.

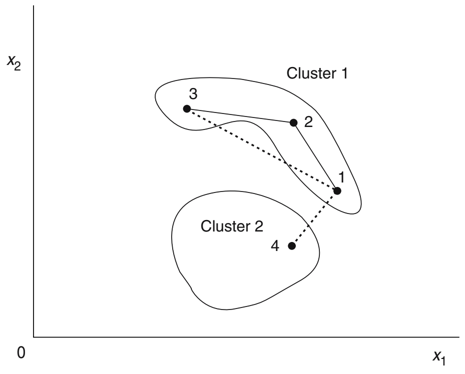


Fig. 7.2. Cluster assignment

To avoid this, for all the data pairs we check if the associated line segments are in the hypersphere. If a line segment is included, we consider that there is a path between the two data. Then we generate sets of data that are connected by paths. Each set constitutes a cluster. In [28] the line segments are considered to be in the hypersphere if sampled data on the line segments satisfy (7.23).

In Fig. 7.2, there are paths between Data 1 and 2 and between Data 2 and 3. But there is no path connecting Datum 4 and the remaining data. Thus Data 1 to 3 constitute a cluster, and so does Datum 4.

If cluster regions in the input space are convex, by the preceding method, all the data in a cluster are selected and no other data are selected. But if a cluster region is concave, data in a cluster may be separated into more than one set. For example, in Fig. 7.2 if part of the line segment connecting Data 2 and 3 is out of the cluster region, the line segment does not form a path. Thus if there are only Data 1 to 3, they are divided into two sets: $\{1, 2\}$ and $\{3\}$.

If $C \geq 1$, no outliers, namely no bounded support vectors, are detected. Thus, if clusters are considered to be well separated and no outliers are considered to be included, we set $C = 1$. If clusters are considered to overlap or outliers are considered to be included, we set C smaller than 1. From (7.4) and $0 \leq \alpha_i \leq C$,

$$|B| \leq 1/C, \tag{7.40}$$

where $|B|$ is the number of bounded support vectors.

According to this procedure for defining clusters, bounded support vectors do not belong to any cluster. But if bounded support vectors are caused by overlapping of clusters, we may include them to the nearest clusters [28].

The most crucial part of clustering is to check the paths for all pairs of data. To speed this up, in [28], only the pairs of data with one datum being an unbounded support vector are checked. But if many support vectors are bounded, this may lead to false cluster generation. To avoid this situation, Yang, Estivill-Castro, and Chalup [276] proposed using a proximity graph, in which a node corresponds to a datum, to model the nearness of data and to check pairs of data that are directly connected in the graph. They used the following proximity graphs:

1. a complete graph, in which all the data are connected;
2. a support vector graph, in which at least one of the data connected to a branch is an unbounded support vector;
3. the Delaunay diagram, which is the dual of the Voronoi diagram and which is composed of adjacent triangles whose edges are the data;
4. the minimum spanning tree with the minimum sum of distances; and
5. k nearest neighbors, in which a datum is connected to k nearest data.

According to the experiments, Delaunay diagrams and k nearest neighbors with k greater than 4 showed comparable clustering performance with complete graphs with much faster clustering.

In [24], training data are spatially chunked to speed training and to optimize RBF parameters. In [62], to speed training, an initial ball is generated by training a one-class classifier for randomly selected data and the ball is iteratively updated, adding the data outside the ball.

8 Kernel-Based Methods

Inspired by the success of support vector machines, to improve generalization and classification abilities, conventional pattern classification techniques have been extended to incorporate maximizing margins and mapping to a feature space. For example, perceptron algorithms [67, 86, 93, 147], neural networks (Chapter 9), and fuzzy systems (Chapter 10) have incorporated maximizing margins and/or mapping to a feature space.

There are numerous conventional techniques that are extended to be used in the high-dimensional feature space, e.g., kernel least squares [27, 206], kernel principal component analysis [215, 216, 230], kernel discriminant analysis [215, pp. 457–68], the kernel Mahalanobis distance [206], the kernel k-means clustering algorithm, the kernel self-organizing feature map, and other kernel-based methods [159, 184, 282].

In this chapter, we discuss some of the kernel-based methods: kernel least squares, kernel principal component analysis, and the kernel Mahalanobis distance.

8.1 Kernel Least Squares

8.1.1 Algorithm

Least squares methods in the input space can be readily extended to the feature space using kernel techniques [27, 206].

Assume that we have training input-output pairs $\{\mathbf{x}_i, y_i\}$ for $i = 1, \ldots, M$. We approximate the output y by

$$y = \mathbf{a}^T \mathbf{g}(\mathbf{x}), \tag{8.1}$$

where $\mathbf{g}(\mathbf{x})$ is the mapping function that maps $\mathbf{x}$ into the l-dimensional feature space and $\mathbf{a}$ is the l-dimensional vector. Without loss of generality, we can assume that the last element of $\mathbf{g}(\mathbf{x})$ is 1. By this assumption, we need not add a constant term in (8.1).

We determine vector $\mathbf{a}$ so that

$$J = \sum_{i=1}^{M} \left(y_i - \mathbf{a}^T \mathbf{g}(\mathbf{x}_i)\right)^2 \tag{8.2}$$

is minimized.

Without loss of generality we can assume that the set of M' vectors, $\{\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_{M'})\}$ $(M' \leq M)$, spans the space generated by $\{\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_M)\}$. Then, because $\mathbf{a}$ is in the space spanned by $\{\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_{M'})\}$, $\mathbf{a}$ is expressed by

$$\mathbf{a} = \sum_{i=1}^{M'} \alpha_i \, \mathbf{g}(\mathbf{x}_i), \tag{8.3}$$

where α_i are parameters. Then (8.1) becomes

$$\begin{aligned} y &= \sum_{i=1}^{M'} \alpha_i \, \mathbf{g}^T(\mathbf{x}_i) \, \mathbf{g}(\mathbf{x}) \\ &= \sum_{i=1}^{M'} \alpha_i \, H(\mathbf{x}, \mathbf{x}_i), \end{aligned} \tag{8.4}$$

where $H(\mathbf{x}, \mathbf{x}_i) = \mathbf{g}^T(\mathbf{x}) \, \mathbf{g}(\mathbf{x}_i) = \mathbf{g}^T(\mathbf{x}_i) \, \mathbf{g}(\mathbf{x})$.

When RBF kernels are used, (8.4) is equivalent to radial basis function neural networks with $\mathbf{x}_i$ $(i = 1, \ldots, M')$ being the centers of radial bases [27].

Substituting (8.3) into (8.2), we obtain

$$\begin{aligned} J &= \frac{1}{2} \sum_{i=1}^{M} \left(y_i - \sum_{j=1}^{M'} \alpha_j H(\mathbf{x}_i, \mathbf{x}_j) \right)^2 \\ &= \frac{1}{2} \, (\mathbf{y} - H\boldsymbol{\alpha})^T (\mathbf{y} - H\boldsymbol{\alpha}), \end{aligned} \tag{8.5}$$

where $\mathbf{y} = (y_1, \ldots, y_M)^T$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{M'})^T$, H is an $M \times M'$ matrix, and $H = \{H(\mathbf{x}_i, \mathbf{x}_j)\}$ $(i = 1, \ldots, M, j = 1, \ldots, M')$.

Taking the partial derivative of J with respect to $\boldsymbol{\alpha}$ and setting it to zero, we obtain

$$\frac{\partial J}{\partial \boldsymbol{\alpha}} = -H^T (\mathbf{y} - H\boldsymbol{\alpha}) = \mathbf{0}. \tag{8.6}$$

Because $\{\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_{M'})\}$ are linearly independent, the rank of H is M'. Thus, $H^T H$ is positive definite. Then, from (8.6),

$$\boldsymbol{\alpha} = (H^T H)^{-1} H^T \mathbf{y}. \tag{8.7}$$

If $\{\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_M)\}$ are linearly independent, H is a square matrix and positive definite. Then $\boldsymbol{\alpha}$ is given by

$$\boldsymbol{\alpha} = H^{-1}\,\mathbf{y}. \tag{8.8}$$

If we use RBF kernels, $M' = M$ and H is positive definite. Thus kernel least squares are solved by (8.8). But in general $M' < M$. Thus, to use (8.7) we need to select independent $\{\mathbf{g}(\mathbf{x}_1), \dots, \mathbf{g}(\mathbf{x}_{M'})\}$. Otherwise, we need to solve

$$\boldsymbol{\alpha} = H^{+}\,\mathbf{y}, \tag{8.9}$$

where H is an $M \times M$ matrix and H^{+} is the pseudo-inverse of H. If H is singular, H^{+} is calculated by singular value decomposition (see Section B.2). This is a time-consuming task, and in estimating $\mathbf{y}$ we need to use all M training data. Thus it is favorable to select independent vectors in advance.

Baudat and Anouar [27] proposed selecting independent vectors by sequential forward selection, in which starting from an empty set, a vector that maximizes the objective function is selected. Cawley and Talbot [53] proposed speeding up basis selection by using the matrix inversion lemma, deleting data that make the kernel matrix singular when added to the kernel matrix, sampling of the candidate basis vectors, and stochastic approximation of the objective function for basis vector selection.

Here, we consider using the Cholesky factorization to select independent vectors [83]. Let H be positive definite. Then H is decomposed by the Cholesky factorization into

$$H = L\,L^T, \tag{8.10}$$

where L is the regular lower triangular matrix and each element L_{ij} is given by

$$L_{op} = \frac{H_{op} - \sum\limits_{n=1}^{p-1} L_{pn}\,L_{on}}{L_{pp}} \quad \text{for} \quad o = 1, \dots, M, \quad p = 1, \dots, o-1, \tag{8.11}$$

$$L_{aa} = \sqrt{H_{aa} - \sum_{n=1}^{a-1} L_{an}^2} \quad \text{for} \quad a = 1, 2, \dots, M. \tag{8.12}$$

Here, $H_{ij} = H(\mathbf{x}_i, \mathbf{x}_j)$.

Then during the Cholesky factorization, if the diagonal element is smaller than the prescribed value $\eta\,(> 0)$:

$$H_{aa} - \sum_{n=1}^{a-1} L_{an}^2 \le \eta, \tag{8.13}$$

we delete the associated row and column and continue decomposing the matrix. The training data that are not deleted in the Cholesky factorization are independent. If no training data are deleted, the training data are all independent in the feature space.

After factorization, $\boldsymbol{\alpha}$ is obtained by solving

$$L\,\mathbf{c} = \mathbf{y}, \tag{8.14}$$

$$L^T \boldsymbol{\alpha} = \mathbf{c}, \tag{8.15}$$

where $\mathbf{c}$ is an M'-dimensional vector.

Selection of the value of η influences the generalization ability. For instance, for RBF kernels, a small value of η may result in selecting all the training data. Thus, there is the optimal value of η, which is determined by model selection.

If we use linear kernels, kernel least squares result in conventional least squares. The advantage of using kernel least squares is that we avoid using singular value decomposition (see Section B.2), when the dimension of the space spanned by the training data is lower than that of the input space.

In the kernel least squares, the regularization term is not included. Thus overfitting may occur. To avoid overfitting, in [80] the regularized least squares methods were proposed, in which the solution is obtained by solving

$$(H + D)\,\boldsymbol{\alpha} = \mathbf{y}, \tag{8.16}$$

where H is the kernel matrix and D is a diagonal matrix associated with the regularization term. Because $H + D$ is positive definite, (8.16) is solved without singular value decomposition. According to [281], the generalization abilities of support vector machines and regularized least squares methods are shown to be comparable for several benchmark data sets. But, unlike support vector machines, the solution is not sparse. This is the same situation with the least squares support vector machines, and the techniques for increasing sparsity discussed in Section 4.1 can be used.

8.1.2 Performance Evaluation

We evaluate the kernel least squares method using pattern classification and function approximation problems listed in Table 1.1 [170].

We measured the training time of our method using a personal computer (Xeon 2.4GHz dual, memory 4GB) with the Linux operating system.

For an n-class classification problem, we determined n hyperplanes, with each hyperplane separating one class from the others, setting the target value of 1 for the class and -1 for the remaining classes. In classification, we classified unknown datum to the class with the maximum output. This is a one-against-all classification strategy. When RBF kernels are used, the kernel least squares are equivalent to radial basis function neural networks.

Because the generalization ability depends on the value of η, we determined the value by five-fold cross-validation for linear kernels, polynomial kernels with $d = [2, 3, 4]$, and RBF kernels with $\gamma = [1, 5, 10]$ with $\eta = [10^{-8}, 10^{-7}, \ldots, 10^{-2}]$, and we selected the value with the maximum recognition rate.

If the same recognition rate was obtained for the validation data set, we broke the tie by selecting the simplest structure as follows:

1. Select the kernel and parameters with the highest recognition rate for the training data.
2. Select polynomial kernels from polynomial and RBF kernels.
3. Select the polynomial kernel with the smallest degree from polynomial kernels.
4. Select the RBF kernel with the smallest value of γ from RBF kernels.
5. Select the largest value of η.

Table 8.1 shows the cross-validation results for the blood cell data. In the table, the "Test," "Train.," and "Valid." columns list the recognition rates of the test data, training data, and cross-validation data, respectively. The "Num." and "Time" columns list the number of selected independent data and training and classification time, respectively. By selecting the value of η by cross-validation, the number of independent data was reduced to less than one-fifteenth of that of the training data. In addition, training and classification time was short.

Because the recognition rate of the cross-validation data with RBF kernels with $\gamma = 5$ was the highest, we selected RBF kernels with $\gamma = 5$ and $\eta = 10^{-4}$. In this case the recognition rate of the test data was the second highest.

Table 8.1. Recognition rates for the blood cell data

Kernel	η	Test (%)	Train. (%)	Valid. (%)	Num.	Time (s)
Linear	10^{-3}	67.71	69.78	74.15	10	25
$d2$	10^{-5}	90.29	91.12	91.44	49	28
$d3$	10^{-5}	91.16	93.51	92.37	79	30
$d4$	10^{-5}	90.74	92.64	91.15	70	30
$\gamma 1$	10^{-5}	82.90	84.99	92.37	119	37
$\gamma 5$	10^{-4}	93.03	95.96	**93.86**	253	58
$\gamma 10$	10^{-3}	93.32	95.87	93.63	227	52

Table 8.2 lists the cross-validation results for the Mackey-Glass data. Under the same conditions as pattern classification, we performed five-fold cross-validation for each kernel. Approximation performance was measured by the normalized root-mean-square error (NRMSE). Because the Mackey-Glass data did not include noise, NRMSEs for the training and test data did not vary very much. Because the NRMSE of the cross-validation data with RBF kernels with $\gamma = 10$ was the smallest, we selected RBF kernels with $\gamma = 10$ and $\eta = 10^{-4}$. For this case, the independent data were reduced to less than half

Table 8.2. Approximation errors of the Mackey-Glass data

Kernel	η	Test (NRMSE)	Train. (NRMSE)	Valid. (NRMSE)	Num.	Time (s)
Linear	10^{-2}	0.0586	0.0584	0.0603	1	0
$d2$	10^{-7}	0.0096	0.0098	0.0138	15	12
$d3$	10^{-5}	0.0062	0.0063	0.0093	15	0
$d4$	10^{-5}	0.0063	0.0064	0.0064	15	0
$\gamma1$	10^{-5}	0.0026	0.0026	0.0022	51	1
$\gamma5$	10^{-4}	0.0021	0.0021	0.0011	122	1
$\gamma10$	10^{-4}	0.0015	0.0015	**0.0008**	190	1

of the training data and the training time was very short. If the training time was shorter than 0.5 second, it is listed as 0 in the "Time" column.

Table 8.3 lists the results for the benchmark data sets. For comparison, for pattern classification problems we also list the recognition rates of the one-against-all fuzzy L1 SVM in Tables 3.2 and 3.3. For function approximation we used the performance of the SVM listed in Tables 11.6 and 11.8. Higher or equal recognition rates (lower or equal approximation errors) are shown in boldface. For the water purification data set, the "Test," "Train.," and

Table 8.3. Recognition performance

Data	Kernel	η	Test (%)	Train. (%)	Num.	Time (s)	SVM (%)
Iris	$d2$	10^{-5}	**96.00**	100	15	0	94.67
Numeral	$d3$	10^{-5}	99.15	100	199	2	**99.27**
Thyroid	$\gamma5$	10^{-4}	93.99	95.97	471	163	**97.93**
Blood cell	$\gamma5$	10^{-4}	93.03	95.96	253	57	**93.16**
Hiragana-50	$\gamma10$	10^{-4}	99.13	100	3511	1641	**99.26**
Hiragana-13	$\gamma10$	10^{-4}	99.44	99.61	860	1715	**99.63**
Hiragana-105	$\gamma10$	10^{-3}	**100**	100	7197	92495	**100**
Mackey Glass	$\gamma10$	10^{-4}	**0.002**^{1}	0.002^1	190	1	0.003^1
Water Purif.	$\gamma0.1$	10^{-4}	1.08^2	0.72^2	62	0	**1.03**^{2}

1: NRMSE, 2: mg/l

"SVM" columns list the average errors in mg/l. And for this data set, the cross-validation for the RBF kernels, we used $\gamma = [0.1, 0.5, 1.0]$.

Except for the hiragana-50 and hiragana-105 data sets, by five-fold cross-validation, small numbers of data were selected and training time was relatively short.

Except for the thyroid data, the generalization abilities of the kernel least squares and the SVM are comparable. For the thyroid test data, the recognition rate of the kernel least squares is much lower than that of the L1 SVM. This is the same tendency as that of the least squares SVM.

8.2 Kernel Principal Component Analysis

Principal component analysis (PCA) is a well-known feature extraction method, in which the principal components of the input vector relative to the mean vector are extracted by orthogonal transformation. Similarly kernel PCA extracts principal components in the feature space [215, 216, 230]. We call them *kernel principal components.*

Unlike the approach discussed in [216], here we consider that the mean of the data is nonzero from the beginning. Consider extracting kernel principal components of a set of data $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$. The covariance matrix Q of the data in the feature space is calculated by

$$\begin{aligned} Q &= \frac{1}{M} \sum_{i=1}^{M} (\mathbf{g}(\mathbf{x}_i) - \mathbf{c}) \, (\mathbf{g}(\mathbf{x}_i) - \mathbf{c})^T \\ &= \frac{1}{M} \sum_{i=1}^{M} \mathbf{g}(\mathbf{x}_i) \, \mathbf{g}^T(\mathbf{x}_i) - \mathbf{c}\, \mathbf{c}^T, \end{aligned} \tag{8.17}$$

where $\mathbf{g}(\mathbf{x})$ is the mapping function that maps $\mathbf{x}$ into the l-dimensional feature space and $\mathbf{c}$ is the mean vector of the mapped training data and is calculated by

$$\begin{aligned} \mathbf{c} &= \frac{1}{M} \sum_{i=1}^{M} \mathbf{g}(\mathbf{x}_i) \\ &= (\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_M)) \begin{pmatrix} \frac{1}{M} \\ \vdots \\ \frac{1}{M} \end{pmatrix}. \end{aligned} \tag{8.18}$$

Substituting (8.18) into (8.17) and rewriting it in a matrix form, we obtain

$$Q = \frac{1}{M} (\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_M)) \, (I_M - \mathbf{1}_M) \begin{pmatrix} \mathbf{g}^T(\mathbf{x}_1) \\ \vdots \\ \mathbf{g}^T(\mathbf{x}_M) \end{pmatrix}, \tag{8.19}$$

where I_M is the $M \times M$ unit matrix and $\mathbf{1}_M$ is the $M \times M$ matrix with all elements being $1/M$.

Let λ and $\mathbf{z}$ be the eigenvalue and the associated eigenvector of Q:

$$Q\,\mathbf{z} = \lambda\,\mathbf{z}. \tag{8.20}$$

Substituting (8.19) into (8.20),

$$\frac{1}{M}(\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_M))\,(I_M - \mathbf{1}_M)\begin{pmatrix} \mathbf{g}^T(\mathbf{x}_1)\,\mathbf{z} \\ \vdots \\ \mathbf{g}^T(\mathbf{x}_M)\,\mathbf{z} \end{pmatrix} = \lambda\,\mathbf{z}. \tag{8.21}$$

Thus $\mathbf{z}$ is expressed by a linear sum of $\{\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_M)\}$.

According to the formulation of [216], all the training data are used to represent $\mathbf{z}$. But by this method we need to retain all the training data after training. To overcome this problem, in [215, 230], sparsity is introduced into KPCA by imposing some restriction to the parameter range. Here, instead of restricting the parameter range, we select a set of linearly independent data from the training data as discussed in Section 8.1. By this method, we can retain only the selected data after training.

Without loss of generality, we can assume that a set of vectors, $\{\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_{M'})\}$ $(M' \le M)$, spans the space generated by $\{\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_M)\}$. Then $\mathbf{z}$ is expressed by

$$\mathbf{z} = (\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_{M'}))\begin{pmatrix} \rho_1 \\ \vdots \\ \rho_{M'} \end{pmatrix}, \tag{8.22}$$

where $\rho_1, \ldots, \rho_{M'}$ are scalars.

In (8.21) we are only interested in the components for $\mathbf{g}(\mathbf{x}_i)$ $(i = 1, \ldots, M')$. Multiplying both terms of (8.21) by $\mathbf{g}^T(\mathbf{x}_i)$ from the left and substituting (8.22) into (8.21), we obtain

$$\frac{1}{M}\,(H_{i1}, \ldots, H_{iM})\,(I_M - \mathbf{1}_M)\,H\,\boldsymbol{\rho}' = \lambda\,(H_{i1}, \ldots, H_{iM'})\,\boldsymbol{\rho}', \tag{8.23}$$

where $H_{ij} = H(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{g}^T(\mathbf{x}_i)\,\mathbf{g}(\mathbf{x}_j)$, $\boldsymbol{\rho}' = (\rho_1, \ldots, \rho_{M'})^T$, $H = \{H_{ij}\}$ $(i = 1, \ldots, M,\ j = 1, \ldots, M')$. Thus, combining (8.23) for $i = 1, \ldots, M'$,

$$\frac{1}{M}\,H^T(I_M - \mathbf{1}_M)\,H\boldsymbol{\rho}' = \lambda\,H^s\boldsymbol{\rho}', \tag{8.24}$$

where $H^s = \{H_{ij}\}$ $(i = 1, \ldots, M',\ j = 1, \ldots, M')$.[1]

Equation (8.24) has the form $A\,\mathbf{x} = \lambda\,B\,\mathbf{x}$, where A and B are $n \times n$ matrices. Solving the equation for λ and $\mathbf{x}$ is called the *generalized eigenvalue*

[1]If $M' = M$, (8.24) is the same as equation (20.13) in [216].

problem [96, pp. 375–6]. And if B is nonsingular, there are n eigenvalues. Because $\{\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_{M'})\}$ are linearly independent, H^s is positive definite. Thus by the Cholesky factorization, $H^s = L\,L^T$, where L is a lower triangular matrix. Multiplying both terms of (8.24) by L^{-1} from the left [195, pp. 462–3], we obtain

$$\frac{1}{M} L^{-1} H^T (I_M - \mathbf{1}_M)\, H (L^{-1})^T (L^T \boldsymbol{\rho}') = \lambda\, (L^T \boldsymbol{\rho}'). \tag{8.25}$$

Here, λ is the eigenvalue and the $L^T \boldsymbol{\rho}'$ is the eigenvector. Because the coefficient matrix is a symmetric, positive semidefinite matrix, the eigenvalues are nonnegative. Thus solving the eigenvalue λ and the eigenvector $L^T \boldsymbol{\rho}'$ of (8.25), we obtain the generalized eigenvalue λ and the eigenvector $\boldsymbol{\rho}'$ of (8.24).

Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{M'}$ be the eigenvalues and $\mathbf{z}_1, \ldots, \mathbf{z}_{M'}$ be the associated eigenvectors for (8.20) and

$$\mathbf{z}_i = (\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_{M'})) \begin{pmatrix} \rho_{i1} \\ \vdots \\ \rho_{iM'} \end{pmatrix} \quad \text{for} \quad i = 1, \ldots, M', \tag{8.26}$$

where $\rho_{i1}, \ldots, \rho_{iM'}$ are scalars. We normalize $\mathbf{z}_i$ for $i = 1, \ldots, M'$. Namely,

$$\mathbf{z}_i^T \mathbf{z}_i = 1. \tag{8.27}$$

Let the adjusted ρ_{ij} be ρ'_{ij}.

For $\mathbf{x}$ we call

$$\mathbf{z}_i^T (\mathbf{g}(\mathbf{x}) - \mathbf{c}) = \sum_{j=1}^{M'} \rho'_{ij}\, H(\mathbf{x}_j, \mathbf{x}) - \mathbf{z}_i^T \mathbf{c} \tag{8.28}$$

the ith *kernel principal component* of $\mathbf{x}$. Here, $\mathbf{z}_i^T \mathbf{c}$ is calculated by (8.18), (8.26), and (8.24). Thus, after training we can discard $(\mathbf{x}_{M'+1}, \ldots, \mathbf{x}_M)$.

The eigenvalue λ_i is the variance in the $\mathbf{z}_i$ direction. The trace of Q is defined as the sum of the diagonal elements of Q:

$$\operatorname{tr}(Q) = \sum_{i=1}^{M'} Q_{ii}. \tag{8.29}$$

Then $\operatorname{tr}(Q) = \lambda_1 + \cdots + \lambda_{M'}$ [96, p. 310]. Thus the sum of the variances of $\mathbf{x}$ is the same as the sum of the variances of $\mathbf{z}$. Suppose we select the first d principal components. We define the accumulation of d eigenvalues as follows:

$$A_c(d) = \frac{\displaystyle\sum_{i=1}^{d} \lambda_i}{\displaystyle\sum_{i=1}^{M'} \lambda_i} \times 100\,(\%). \tag{8.30}$$

The accumulation of eigenvalues shows how well the reduced feature vector reflects the characteristics of the original feature vector in the feature space.

Kernel PCA can be used for feature extraction for pattern classification and noise filtering of image data called *denoising*. In denoising by kernel PCA we choose the principal components and discard the remaining components. Then we restore the image called the *preimage*. Because usually, inverse mapping does not exist, this is done by minimizing

$$\|\Phi - \mathbf{g}(\mathbf{x})\|, \tag{8.31}$$

where Φ is the reduced image in the feature space and $\mathbf{x}$ is the preimage in the input space. Mika et al. [167] proposed an iterative method for RBF kernels. To suppress the effect of outliers, Takahashi and Kurita [243] proposed modifying the principal components during iterations applying robust statistical techniques.

8.3 Kernel Mahalanobis Distance

The Mahalanobis distance for a class is a distance, from the center of the class, normalized by the inverse of the covariance matrix for that class. The Mahalanobis distance is linear transformation invariant and is widely used for pattern classification. In this section, we discuss two methods to calculate the kernel version of the Mahalanobis distance: (1) the kernel Mahalanobis distance calculated by singular value decomposition (SVD) [206] and (2) the kernel Mahalanobis distance calculated by KPCA. In the former method, all the training data are used for calculating the pseudo-inverse. But in the latter method, independent vectors in the feature space are selected by Cholesky factorization. Thus, usually, the latter method takes less time in calculation.

8.3.1 SVD-Based Kernel Mahalanobis Distance

The kernel Mahalanobis distance $d_{\mathbf{g}_i}(\mathbf{x})$ for class i is given by

$$d^2_{\mathbf{g}_i}(\mathbf{x}) = (\mathbf{g}(\mathbf{x}) - \mathbf{c}_i)^T Q^{-1}_{\mathbf{g}_i} (\mathbf{g}(\mathbf{x}) - \mathbf{c}_i), \tag{8.32}$$

where $\mathbf{g}(\mathbf{x})$ is the mapping function from the input space to the l-dimensional feature space and $\mathbf{c}_i$ is the center of class i in the feature space:

$$\mathbf{c}_i = \frac{1}{M_i} \sum_{j=1}^{M_i} \mathbf{g}(\mathbf{x}_{ij}). \tag{8.33}$$

Here $\mathbf{x}_{ij} = (x_{ij1} \cdots x_{ijm})^T$ is the jth training datum for class i and M_i is the number of training data for class i.

The covariance matrix for class i in the feature space, $Q_{\mathbf{g}_i}$, is expressed in a matrix form as follows:

$$
\begin{aligned}
Q_{\mathbf{g}_i} &= \frac{1}{M_i}\sum_{j=1}^{M_i}(\mathbf{g}(\mathbf{x}_{ij}) - \mathbf{c}_i)\,(\mathbf{g}(\mathbf{x}_{ij}) - \mathbf{c}_i)^T \\
&= \mathbf{g}^T(X_i)\left(\frac{1}{M_i}(I_{M_i} - \mathbf{1}_{M_i})\right)\mathbf{g}(X_i), \qquad (8.34)
\end{aligned}
$$

where the second equation is derived by substituting (8.33) into the right-hand side of the first equation, I_{M_i} is the $M_i \times M_i$ unit matrix, $\mathbf{1}_{M_i}$ is the $M_i \times M_i$ matrix with all the components equal to $1/M_i$, and $\mathbf{g}(X_i)$ is an $M_i \times l$ matrix:

$$
\mathbf{g}(X_i) = \begin{pmatrix} \mathbf{g}^T(\mathbf{x}_{i1}) \\ \vdots \\ \mathbf{g}^T(\mathbf{x}_{iM_i}) \end{pmatrix}. \qquad (8.35)
$$

To calculate the kernel Mahalanobis distance given by (8.32) without explicitly treating the variables in the feature space, we need to use the kernel trick. Namely, we transform (8.32) so that only the dot-products $\mathbf{g}^T(\mathbf{x})\,\mathbf{g}(\mathbf{x})$ appear in (8.32) [206].

Because $I_{M_i} - \mathbf{1}_{M_i}$ is a symmetric, positive semidefinite matrix, we can define the square root of the matrix, Z_i by

$$
Z_i = \left(\frac{1}{M_i}(I_{M_i} - \mathbf{1}_{M_i})\right)^{\frac{1}{2}}. \qquad (8.36)
$$

Substituting (8.36) into (8.34), we obtain

$$
Q_{\mathbf{g}_i} = \mathbf{g}^T(X_i)\,Z_i^2\,\mathbf{g}(X_i). \qquad (8.37)
$$

Substituting (8.37) into (8.32) gives

$$
d^2_{\mathbf{g}_i}(\mathbf{x}) = (\mathbf{g}(\mathbf{x}) - \mathbf{c}_i)^T\left(\mathbf{g}^T(X_i)\,Z_i^2\,\mathbf{g}(X_i)\right)^{-1}(\mathbf{g}(\mathbf{x}) - \mathbf{c}_i)\,. \qquad (8.38)
$$

Now the following equation is valid for any integer n, symmetric, positive semidefinite matrix A, and any vectors $\mathbf{t}$ and $\mathbf{u}$ [206]:

$$
\mathbf{t}^T(X^T A\,X)^n\,\mathbf{u} = \mathbf{t}^T X^T(A^{\frac{1}{2}}(A^{\frac{1}{2}}HA^{\frac{1}{2}})^{n-1}A^{\frac{1}{2}})\,X\,\mathbf{u}, \qquad (8.39)
$$

where $H = XX^T$. If n is negative, it means pseudo-inverse. We calculate the pseudo-inverse using the singular value decomposition.

Here because Z_i^2 in (8.38) is a symmetric, positive semidefinite matrix, we can apply (8.39) to (8.38):

$$
\begin{aligned}
d^2_{\mathbf{g}_i}(\mathbf{x}) = {} & (\mathbf{g}(X_i)\,\mathbf{g}(\mathbf{x}) - \mathbf{g}(X_i)\,\mathbf{c}_i)^T\left(Z_i\,(Z_i\,\mathbf{g}(X_i)\,\mathbf{g}^T(X_i)\,Z_i)^{-2}Z_i\right) \\
& \times (\mathbf{g}(X_i)\,\mathbf{g}(\mathbf{x}) - \mathbf{g}(X_i)\,\mathbf{c}_i)\,. \qquad (8.40)
\end{aligned}
$$

Because (8.40) consists of only dot-products in the feature space, we can replace them with kernels:

$$H(\mathbf{x}, \mathbf{y}) = \mathbf{g}^T(\mathbf{x})\, \mathbf{g}(\mathbf{y}). \tag{8.41}$$

Using (8.41), we can rewrite the dot-products in (8.40) as follows:

$$\begin{aligned}
\mathbf{g}(X_i)\, \mathbf{g}(\mathbf{x}) &= \begin{pmatrix} \mathbf{g}^T(\mathbf{x}_{i1})\, \mathbf{g}(\mathbf{x}) \\ \vdots \\ \mathbf{g}^T(\mathbf{x}_{iM_i})\, \mathbf{g}(\mathbf{x}) \end{pmatrix} \\
&= \begin{pmatrix} H(\mathbf{x}_{i1}, \mathbf{x}) \\ \vdots \\ H(\mathbf{x}_{iM_i}, \mathbf{x}) \end{pmatrix} = H(X_i, \mathbf{x}), \qquad (8.42) \\
\mathbf{g}(X_i)\, \mathbf{c}_i &= \frac{1}{M_i}\, \mathbf{g}(X_i) \sum_{j=1}^{M_i} \mathbf{g}(\mathbf{x}_{ij}) \\
&= \frac{1}{M_i} \sum_{j=1}^{M_i} H(X_i, \mathbf{x}_{ij}), \qquad (8.43) \\
\mathbf{g}(X_i)\, \mathbf{g}^T(X_i) &= \begin{pmatrix} \mathbf{g}^T(\mathbf{x}_{i1}) \\ \vdots \\ \mathbf{g}^T(\mathbf{x}_{iM_i}) \end{pmatrix} \begin{pmatrix} \mathbf{g}(\mathbf{x}_{i1}) \cdots \mathbf{g}(\mathbf{x}_{iM_i}) \end{pmatrix} \\
&= \begin{pmatrix} H(\mathbf{x}_{i1}, \mathbf{x}_{i1}) & \cdots & H(\mathbf{x}_{i1}, \mathbf{x}_{iM_i}) \\ \vdots & \ddots & \vdots \\ H(\mathbf{x}_{iM_i}, \mathbf{x}_{i1}) & \cdots & H(\mathbf{x}_{iM_i}, \mathbf{x}_{iM_i}) \end{pmatrix} \\
&= H(X_i, X_i^T). \qquad (8.44)
\end{aligned}$$

Substituting (8.42), (8.43), and (8.44) into (8.40), we obtain

$$\begin{aligned}
d^2_{\mathbf{g}_i}(\mathbf{x}) = &\left(H(X_i, \mathbf{x}) - \frac{1}{M_i} \sum_{j=1}^{M_i} H(X_i, \mathbf{x}_{ij}) \right)^T \left(Z_i \left(Z_i\, H(X_i, X_i^T)\, Z_i \right)^{-2} Z_i \right) \\
&\times \left(H(X_i, \mathbf{x}) - \frac{1}{M_i} \sum_{j=1}^{M_i} H(X_i, \mathbf{x}_{ij}) \right). \qquad (8.45)
\end{aligned}$$

Using (8.45) we can calculate the kernel Mahalanobis distance without treating variables in the feature space. But $(Z_i\, H(X_i, X_i^T)\, Z_i)^{-2}$ needs to be calculated by singular value decomposition. In the following, we discuss the singular value decomposition and its variant to improve generalization ability when the number of data is small.

Any matrix A is decomposed into $A = S\, \Lambda\, U^T$ by singular value decomposition, where S and U are orthogonal matrices ($S\, S^T = I, U\, U^T = I$, where I is a unit matrix) and Λ is a diagonal matrix. If A is an $m \times m$ positive semidefinite matrix, the singular value decomposition is equivalent to the diagonalization of the matrix. Namely, S, Λ, U are $m \times m$ square matrices and

$S = U$. Because $Z_i\,H(X_i, X_i^T)\,Z_i$ is a symmetric, positive semidefinite matrix, in the following discussion we assume that A is symmetric and positive semidefinite.

If A is positive definite, the inverse of A is expressed as follows:

$$\begin{aligned} A^{-1} = (U\,\Lambda\,U^T)^{-1} &= (U^T)^{-1}\Lambda^{-1}\,U^{-1} \\ &= U\Lambda^{-1}U^T. \end{aligned} \tag{8.46}$$

Assume that A is positive semidefinite with rank r ($m > r$). In this case the pseudo-inverse of A, A^+, is used. In the following we discuss two methods to calculate the pseudo-inverse: the conventional and improved methods.

Conventional Method

Because $m > r$ holds, the $(r+1)$st to mth diagonal elements of Λ are zero. In the conventional pseudo-inverse, if a diagonal element λ_i is larger than or equal to σ, where σ is a predefined threshold, we set $1/\lambda_i$ to the ith element of Λ^+. But if it is smaller, we set 0.

Improved Method

In the conventional method, if a diagonal element is smaller than σ, the associated diagonal element of the pseudo-inverse is set to 0. This means that all the components with small singular values are neglected. Namely, the subspace corresponding to zero diagonal elements is neglected. This leads to decreasing the approximation ability of the subspace. To avoid this we set $1/\sigma$ instead of 0. Namely, we calculate the pseudo-inverse as follows:

$$\begin{aligned} A^+ &= U\,\Lambda^+\,U^T \\ &= U \begin{pmatrix} \lambda_1^{-1} & & & & & \\ & \ddots & & & & \\ & & \lambda_r^{-1} & & & \\ & & & \frac{1}{\sigma} & & \\ & & & & \ddots & \\ & & & & & \frac{1}{\sigma} \end{pmatrix} U^T. \end{aligned} \tag{8.47}$$

8.3.2 KPCA-Based Mahalanobis Distance

In this section we discuss a KPCA-based method to calculate a Mahalanobis distance in the feature space. This method is equivalent to the kernel Mahalanobis distance discussed in Section 8.3.1. But in this method because we can discard the redundant input vectors, calculation time can be shortened compared with that of the conventional kernel Mahalanobis distance given by (8.45).

Let λ_j and $\mathbf{z}_j$ be the jth eigenvalue and the associated eigenvector of the covariance matrix $Q_{\mathbf{g}_i}$ given by (8.34). Then

$$Q_{\mathbf{g}_i} \mathbf{z}_j = \lambda_j \, \mathbf{z}_j \qquad \text{for} \quad j = 1, \ldots, l, \tag{8.48}$$

where l is the dimension of the feature space. Then for the pseudo-inverse of $Q_{\mathbf{g}_i}$, $Q^+_{\mathbf{g}_i}$, the following equation holds:

$$Q^+_{\mathbf{g}_i} \, \mathbf{z}_j = \begin{cases} \lambda_j^{-1} \, \mathbf{z}_j & \lambda_j > 0, \\ 0 & \lambda_j = 0. \end{cases} \tag{8.49}$$

Then the jth kernel principal component of input $\mathbf{x}$ is defined by

$$y_j = \begin{cases} \mathbf{z}_j^T \, (\mathbf{g}(\mathbf{x}) - \mathbf{c}_i) & \lambda_j > 0, \\ 0 & \lambda_j = 0. \end{cases} \tag{8.50}$$

From (8.50), the Mahalanobis distance in that space can be calculated as follows:

$$d^2_{\mathbf{g}_i}(\mathbf{x}) = \frac{y_1^2}{\lambda_1} + \cdots + \frac{y^2_{M'_i}}{\lambda_{M'_i}}, \tag{8.51}$$

where $M_i^{'}$ is the number of nonzero eigenvalues.

To calculate the component given by (8.50) efficiently, we use the KPCA discussed in Section 8.2.

9

Maximum-Margin Multilayer Neural Networks

Three-layer (one-hidden-layer) neural networks are known to be universal approximators, in that they can approximate any continuous functions with any accuracy. However, training a multilayer neural network by the back-propagation algorithm is slow and the generalization ability depends on the initial weights. As for improving the generalization ability, there are several approaches. One way is to add a regularization term, e.g., the square sum of the weights, in the objective function of back-propagation training. Or we may train a support vector machine with sigmoid functions as kernels. But because sigmoid functions satisfy Mercer's condition only for specific parameter values, several approaches have been proposed to overcome this problem (e.g., [239]).

Instead of using support vector machines, we may maximize margins. For example, Jayadeva, Deb, and Chandra [120] formulated a decision tree by linear programming and maximized margins by support vector machine–based training. Based on the network synthesis theory [3, 279], Nishikawa and Abe [178] trained a three-layer neural network layer by layer, maximizing margins.

In this chapter, based on [178], we discuss how to maximize margins of a three-layer neural network classifier that is trained layer by layer and compare the recognition performance with that of support vector machines.

9.1 Approach

The CARVE (Constructive Algorithm for Real-Valued Examples) algorithm [3, 279] guarantees that any pattern classification problem can be synthesized in three layers if we train the hidden layer in the following way. First, we separate a part of the (or the whole) data belonging to a class from the remaining data by a hyperplane. Then we remove the separated data from the training data. We repeat this procedure until only the data belonging to one class remain.

In the following, we discuss a method for training neural network classifiers based on the CARVE algorithm. To improve generalization ability, we maximize margins of hidden-layer hyperplanes and output-layer hyperplanes. Because the training data on one side of the hidden-layer hyperplane need to belong to one class, we cannot apply the quadratic optimization technique used for training support vector machines. Therefore, we extend the heuristic training method called *direct SVMs* [203], which sequentially searches support vectors. For the output layer, because there is no such restriction, we use the quadratic optimization technique.

9.2 Three-Layer Neural Networks

Figure 9.1 shows a three-layer neural network with one hidden layer. In the figure, the input is fed from the left and each layer is called, from left to right, *input layer*, *hidden layer*, and *output layer*. We may have more than one hidden layer. The input layer consists of input neurons and a bias neuron whose input is constant (usually 1). The hidden layer consists of hidden neurons and a bias neuron, and the output layer consists of output neurons. The number of output neurons is the number of classes for pattern classification and the number of outputs to be synthesized for function approximation. The input neurons and the hidden neurons, and the hidden neurons and the output neurons are fully connected by weights. The input and output of the ith neuron of the kth layer are denoted by $x_i(k)$ and $z_i(k)$, respectively, and the weight between the ith neuron of the kth layer and the jth neuron of the $(k+1)$st layer is denoted by $w_{ji}(k)$.

Inputs to the input layer are output by the input neuron without change, and the output of the bias neuron is 1. Namely,

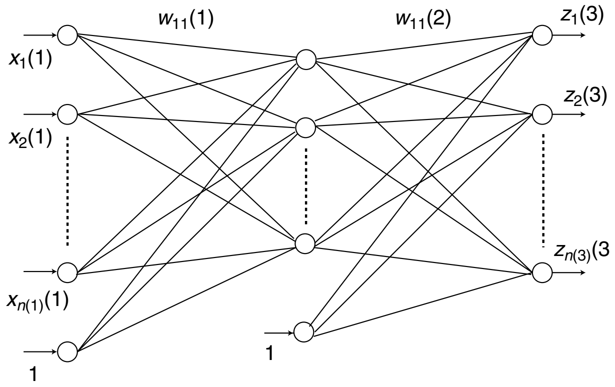


Fig. 9.1. Structure of a three-layer neural network

$$z_j(1) = \begin{cases} x_j(1) & \text{for } j = 1, \ldots, n(1), \\ \\ 1 & \text{for } j = n(1) + 1, \end{cases} \tag{9.1}$$

where $n(1)$ is the number of input neurons.

Outputs of the input (hidden) neurons and the bias neuron are multiplied by the weights and their sums are input to the hidden (output) neurons as follows:

$$x_i(k+1) = \mathbf{w}_i^T(k)\,\mathbf{z}(k) \qquad \text{for} \quad i = 1, \ldots, n(k+1), \quad k = 1, 2, \tag{9.2}$$

where $n(k+1)$ is the number of the $(k+1)$st-layer neurons and

$$\begin{aligned} \mathbf{x}(k) &= \big(x_1(k), \ldots, x_{n(k)}(k)\big)^T, \\ \mathbf{z}(k) &= \big(z_1(k), \ldots, z_{n(k)}(k), z_{n(k)+1}(k)\big)^T, \\ \mathbf{w}_i(k) &= \big(w_{i1}(k), \ldots, w_{i,n(k)}(k), w_{i,n(k)+1}(k)\big)^T \\ &\qquad \text{for} \quad i = 1, \ldots, n(k+1). \end{aligned} \tag{9.3}$$

The output function of the hidden (output) neurons is given by the sigmoid function shown in Fig. 9.2. Namely, their outputs are given by

$$z_i(k+1) = \frac{1}{1 + \exp\left(-\dfrac{x_i(k+1)}{T}\right)} \qquad \text{for} \quad i = 1, \ldots, n(k+1), \tag{9.4}$$

where T is a positive parameter that controls the slope of the sigmoid function, and usually $T = 1$.

Let for the training inputs $\mathbf{x}_1, \ldots, \mathbf{x}_M$, where M is the number of training data, the desired outputs be $\mathbf{s}_1, \ldots, \mathbf{s}_M$, respectively. Then the training of the

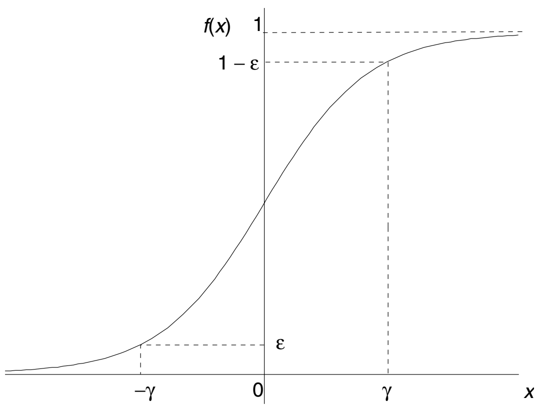


Fig. 9.2. Sigmoid function

network is to determine all the weights $\mathbf{w}_i(k)$ so that for the training input $\mathbf{x}_i$, the output becomes $\mathbf{s}_i$. Thus we want to determine the weights $\mathbf{w}_i(k)$ so that the sum-of-squares error between the target values and the network outputs:

$$E = \frac{1}{2} \sum_{l=1}^{M} (\mathbf{z}_l(3) - \mathbf{s}_l)^T (\mathbf{z}_l(3) - \mathbf{s}_l) \tag{9.5}$$

is minimized, where $\mathbf{z}_l(3)$ is the network output for $\mathbf{x}_l$.

Or for the input-output pairs $(\mathbf{x}_l, \mathbf{s}_l)$, we determine the weights $\mathbf{w}_i(k)$ so that

$$|z_{lj}(3) - s_{lj}| \le \varepsilon(3) \quad \text{for} \quad j = 1, \dots, n(3), \quad l = 1, \dots, M \tag{9.6}$$

are satisfied, where $\varepsilon(3)\,(> 0)$ is the tolerance of convergence for the output neuron outputs.

When the network is used for pattern classification, the ith output neuron corresponds to class i, and for the training input $\mathbf{x}_l$ belonging to class i, the target values of the output neurons j $(j = 1, \dots, n(3))$ are assigned as follows:

$$s_{lj} = \begin{cases} 1 & \text{for } j = i, \\ 0 & \text{for } j \ne i. \end{cases} \tag{9.7}$$

Because E given by (9.5) shows how well the neural network memorizes the training data, too small a value of E, i.e., overfitting may result in worsening the generalization ability of the neural network. To avoid this, the validation data set is prepared as well as the training data set, and training is stopped when the value of E for the validation data set starts to increase. Or, we may add a regularization term such as

$$\lambda \sum_{k=1}^{2} \sum_{i=1}^{n(k+1)} \mathbf{w}_i^T(k) \mathbf{w}_i(k) \tag{9.8}$$

to (9.5) to prevent overfitting, where $\lambda\,(> 0)$ is the regularization constant.

In the back-propagation algorithm, for each training datum, weights in each layer are corrected by the steepest descent method so that the output errors are reduced:

$$w_{ij}^{\text{new}}(k) = w_{ij}^{\text{old}}(k) - \alpha_{\text{lr}} \frac{\partial E_l}{\partial w_{ij}^{\text{old}}(k)}, \tag{9.9}$$

where $\alpha_{\text{lr}}\,(> 0)$ is the learning rate and E_l is the square error for the lth training datum:

$$E_l = \frac{1}{2} (\mathbf{z}_l(3) - \mathbf{s}_l)^T (\mathbf{z}_l(3) - \mathbf{s}_l). \tag{9.10}$$

Because one datum is processed at a time, training by the back-propagation algorithm is slow for a large problem.

In the following, we discuss training of three-layer networks layer by layer, maximizing margins.

9.3 CARVE Algorithm

According to the CARVE algorithm [3, 279], any pattern classification problem can be synthesized in three layers. In the hidden layer, we determine the hyperplane so that the data of a single class exist on one side of the hyperplane. Then the separated data are removed from the training data. The hidden layer training is completed when only data of a single class remain.

We explain the procedure using the example shown in Fig. 9.3. The class data shown in triangles include a datum, which is the farthest from the center of the training data shown in the asterisk. Thus we separate the data of this class from the remaining data.

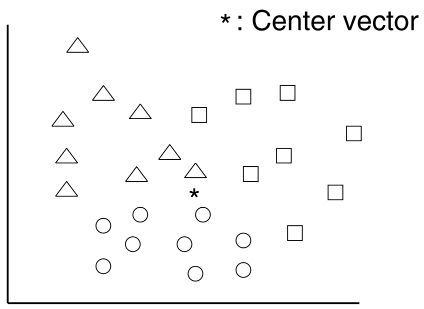


Fig. 9.3. Sample data. From [178, p. 323, ©IEEE 2002]

As shown in Fig. 9.4, a hyperplane is determined so that the data of this class are on one side of the hyperplane. Then the separated data in gray are removed from the training data. We call the data that are used for determining hyperplane *active data* and the deleted data *inactive data*. For the reduced training data, the class data shown in squares include a datum that is farthest from the center.

Thus we determine a hyperplane so that only the data of this class is on one side of the hyperplane, as shown in Fig. 9.5.

We repeat this procedure until the remaining training data belong to a single class (see Fig. 9.6). We say that a hyperplane satisfies the CARVE condition if on one side of the hyperplane data of only one class exist.

In the output layer, for class i we determine a hyperplane so that class i data are separated from the remaining data. In this way we can construct a three-layer neural network for an n-class problem.

9.4 Determination of Hidden-Layer Hyperplanes

According to the CARVE algorithm, on one side of the hyperplane, data of different classes cannot coexist. We may realize this constraint, as discussed

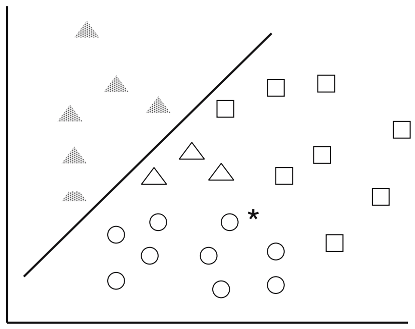


Fig. 9.4. Create the first hyperplane. From [178, p. 323, ©IEEE 2002]

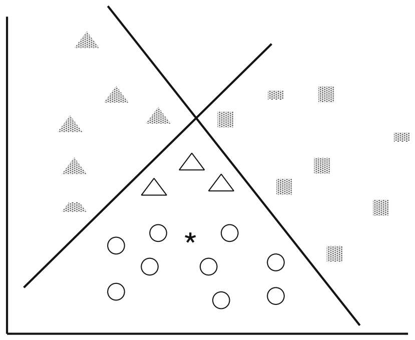


Fig. 9.5. Create the second hyperplane. From [178, p. 323, ©IEEE 2002]

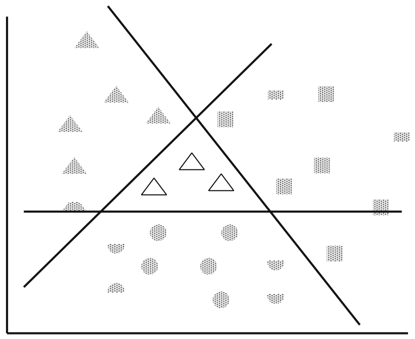


Fig. 9.6. All active data belong to one class. From [178, p. 323, ©IEEE 2002]

in Section 2.6.8, by training a support vector machine with different margin parameters. Because a hidden-layer hyperplane is determined in the input space, linear kernels need to be used. Therefore, the problem may be inseparable. However, if the problem is inseparable, there are cases where degenerate solutions are obtained. Thus, we cannot use a support vector machine.

To overcome this problem, we extend the direct SVM [203], which determines the optimal hyperplane geometrically. Let the farthest datum from the center of the training data belong to class i. Initially, we consider separating class i data from the remaining data, and we set the target values of the class i data to $+1$ and those of the remaining data to -1. We call the side of the hyperplane where the data with the positive targets reside the *positive side of the hyperplane.*

First, as initial support vectors we choose the data pair that has the minimum distance among the data pairs with opposite target values. The initial hyperplane is determined so that it goes through the center of the data pair and orthogonal to the line segment that connects the data pair. If there are data with negative targets on the positive side of the hyperplane, these data violate the CARVE condition. Thus, to satisfy the CARVE condition, we rotate the hyperplane until no data violate the condition. The previously violating data are added to the support vectors. In the following, we describe the procedure in more detail.

9.4.1 Rotation of Hyperplanes

Let the initial support vectors be $\mathbf{x}_0^+$ and $\mathbf{x}_0^-$. Then, the weight vector of the initial hyperplane is given by

$$\mathbf{w}_0 = \mathbf{x}_0^+ - \mathbf{x}_0^-, \tag{9.11}$$

and the hyperplane is written by

$$\mathbf{w}_0^T \mathbf{x} - \mathbf{w}_0^T \mathbf{c}_0 = 0, \tag{9.12}$$

where $\mathbf{c}_0$ is the center of the data pair $\{\mathbf{x}_0^+, \mathbf{x}_0^-\}$. If there are no data that violate the CARVE condition in the remaining data, we finish training. If there are data of the negative target, we add the most-violating datum to the support vectors. Let the obtained support vector be $\mathbf{x}_1^-$ and the center of the data pair $\{\mathbf{x}_0^+, \mathbf{x}_1^-\}$ be $\mathbf{c}_1$. The hyperplane is updated so that it passes through the two points $\mathbf{c}_0$ and $\mathbf{c}_1$. If there are still violating data, we repeat updating.

First, we show the updating method in the two-dimensional space using Fig. 9.7, which shows the initial hyperplane.

Because there are data that violate the CARVE condition, we rotate the hyperplane as shown in Fig. 9.8.

Define $\mathbf{r}_1$ by $\mathbf{r}_1 = \mathbf{c}_1 - \mathbf{c}_0$, where $\mathbf{r}_1$ is a vector that the rotated hyperplane includes (see Fig. 9.9). We calculate the weight vector $\mathbf{w}_1$ that is orthogonal to $\mathbf{r}_1$:

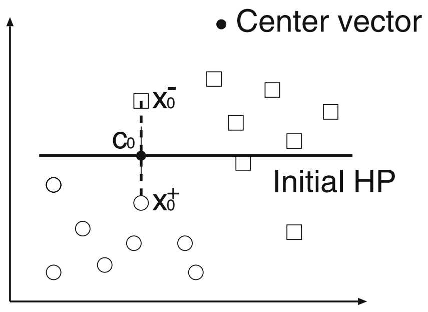


Fig. 9.7. Initial hyperplane. From [178, p. 324, ©IEEE 2002]

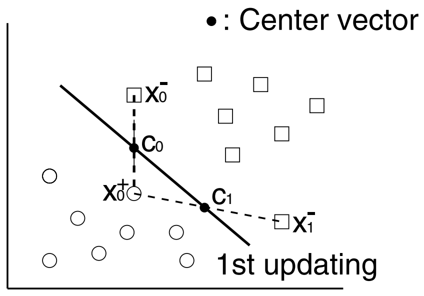


Fig. 9.8. First updating of the hyperplane. From [178, p. 324, ©IEEE 2002]

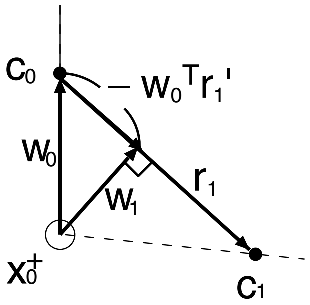


Fig. 9.9. Detail of updating. From [178, p. 324, ©IEEE 2002]

$$\mathbf{w}_1 = \mathbf{w}_0 - \mathbf{w}_0^T \, \mathbf{r'}_1 \, \mathbf{r'}_1, \tag{9.13}$$

where $\mathbf{r}'_1$ is the normalized vector of $\mathbf{r}_1$. The resulting hyperplane satisfies the CARVE condition.

Now consider the mth updating in the n-dimensional space, which requires the hyperplane to pass through the centers of support vector pairs, $\mathbf{c}_0, \ldots, \mathbf{c}_m$. Thus the maximum number of updatings is $(n - 1)$. This is because, if we update the hyperplane $(n-1)$ times, the hyperplane must go through n centers, and no further rotation is possible. For the mth updating, we find the most-violating datum with the negative target. Let the datum be $\mathbf{x}_m^-$. Then, because $\mathbf{c}_m$ is the center of $\{\mathbf{x}_0^+, \mathbf{x}_m^-\}$, $\mathbf{r}_m$, which is included in the rotated hyperplane, is given by $\mathbf{r}_m = \mathbf{c}_m - \mathbf{c}_0$.

The hyperplane with the coefficient vector $\mathbf{w}_{m-1}$ includes the vectors $\{\mathbf{r}_1, \ldots, \mathbf{r}_{m-1}\}$. Let the vectors $\{\mathbf{p}_1, \ldots, \mathbf{p}_{m-1}\}$ be the orthogonal system of $\{\mathbf{r}_1, \ldots, \mathbf{r}_{m-1}\}$. Obviously, the hyperplane with the weight vector $\mathbf{w}_{m-1}$ includes $\{\mathbf{p}_1, \ldots, \mathbf{p}_{m-1}\}$.

We use the Gram-Schmidt orthogonalization to calculate $\mathbf{p}_m$:

$$\mathbf{p}_m = \mathbf{r}_m - \sum_{k=1}^{m-1} \mathbf{r}_m^T \, \mathbf{p}_k' \, \mathbf{p}_k', \tag{9.14}$$

where $\mathbf{p}_k'$ is the normalized vector of $\mathbf{p}_k$. The weight vector $\mathbf{w}_{m-1}$ of the hyperplane is determined so that it is orthogonal to the orthogonal system $\{\mathbf{p}_1, \ldots, \mathbf{p}_{m-1}\}$. For the mth updating, it is updated to be orthogonal to $\mathbf{p}_m$. Namely, $\mathbf{w}_m$ is obtained by rotating the hyperplane in the direction of $\mathbf{p}_m$ with $\mathbf{c}_0$ as the center:

$$\mathbf{w}_m = \mathbf{w}_{m-1} - \mathbf{w}_{m-1}^T \, \mathbf{p}_m' \, \mathbf{p}_m'. \tag{9.15}$$

Then the updated hyperplane includes the vectors $\{\mathbf{r}_1, \ldots, \mathbf{r}_m\}$.

9.4.2 Training Algorithm

A procedure for determining hidden-layer hyperplanes is as follows:

1. Calculate the center of the active data, where initially all the training data are active.
2. Find the datum that is the farthest from the center. Set the targets of the data that belong to the same class with the farthest datum to 1 and those of the remaining data to -1.
3. Find the nearest data pair of opposite targets among all active data pairs. Let them be $(\mathbf{x}_0^+, \mathbf{x}_0^-)$. The center $\mathbf{c}_0$ and the weight $\mathbf{w}_0$ are given by

$$\mathbf{c}_0 = \frac{1}{2}(\mathbf{x}_0^+ + \mathbf{x}_0^-), \tag{9.16}$$

$$\mathbf{w}_0 = \mathbf{x}_0^+ - \mathbf{x}_0^-. \tag{9.17}$$

4. Calculate the values of the decision function:

$$D(\mathbf{x}_i) = \mathbf{w}_m^T \, \mathbf{x}_i - \mathbf{w}_m^T \, \mathbf{c}_0 \quad \text{for} \quad i = 1, \ldots, M. \tag{9.18}$$

5. If all the data with negative targets satisfy

$$-D(\mathbf{x}_i^-) \geq C(2) \, D(\mathbf{x}_0^+) \quad \text{for} \quad i \in B, \tag{9.19}$$

the hyperplane that satisfies the CARVE condition is found; go to Step 8, where B is the indices of data with negative targets and $C(2)$ plays the role of the soft margin. Otherwise, go to Step 5. Usually, $C(2)$ is set to $C(2) < 1$. If $C(2)$ is negative, the data with a negative target may exist on the positive side of the hyperplane. Thus for negative $C(2)$, the CARVE condition is violated.

6. In the mth updating, we include the most-violating datum x_m^- in the support vectors. Then we calculate $\mathbf{c}_m = (\mathbf{x}_0^+ + \mathbf{x}_m^-)/2$ and $\mathbf{r}_m = \mathbf{c}_m - \mathbf{c}_0$. From $\{\mathbf{r}_1, \ldots, \mathbf{r}_m\}$, the mth component of the orthogonal system, $\mathbf{p}_m$, is
$$\mathbf{p}_m = \mathbf{r}_m - \sum_{k=1}^{m-1} \mathbf{r}_m^T \, \mathbf{p}'_k \, \mathbf{p}'_k. \tag{9.20}$$
The weight vector $\mathbf{w}_m$ is written as follows:
$$\mathbf{w}_m = \mathbf{w}_{m-1} - \mathbf{w}_{m-1}^T \, \mathbf{p}'{}_m \, \mathbf{p}'{}_m. \tag{9.21}$$
We can obtain the orthogonal vector $\mathbf{w}_m$ for the hyperplane that passes through $\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_m$ by updating in this way.
7. If updating was repeated n or M times, it is failed for the current data pair. Return to Step 4, setting the next nearest data pair as $(\mathbf{x}_0^+, \mathbf{x}_0^-)$ and recalculating the associated weight vector. Otherwise, return to Step 4 and repeat training.
8. If the active data belong to one class, terminate the algorithm. Otherwise, make the data on the positive side of the hyperplane inactive and return to Step 2.

9.5 Determination of Output-Layer Hyperplanes

We determine the output-layer hyperplanes using the same technique that trains support vector machines. The hyperplane is obtained by solving the following dual problem. Maximize
$$Q(\boldsymbol{\rho}) = \sum_{i=1}^{M} \rho_i - \frac{1}{2} \sum_{i,j=1}^{M} \rho_i \, \rho_j \, y_i \, y_j \, \mathbf{z}_i^T \, \mathbf{z}_j \tag{9.22}$$
subject to
$$\sum_{i=1}^{M} y_i \, \rho_i = 0, \quad 0 \le \rho_i \le C(3) \quad \text{for} \quad i = 1, \ldots, M, \tag{9.23}$$
where $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_M)^T$ and ρ_i are the Lagrange multipliers, $C(3)$ is the margin parameter for the output layer, $\mathbf{z}_i$ are the output vectors of the hidden neurons, and y_i are the associated labels. For training the output layer for class i, we set $y_i = 1$ for the class i data and -1 for the data belonging to the remaining classes.

9.6 Determination of Parameter Values

Usually, the value of T in the sigmoid function is set to 1. Here, we determine the value of T so that the outputs of the unbounded support vectors are the same for hidden and output layers.

From (9.4), the jth output of the ith layer for the unbounded support vector is given by

$$z_j^*(i+1) = \frac{1}{1+\exp(-(\mathbf{w}_j^T(i)\,\mathbf{x}_j^*(i) + b_j(i))/T_j(i))}, \tag{9.24}$$

where $\mathbf{x}_j^*(i)$ and $T_j(i)$ are the support vector and the slope parameter of the jth hyperplane of the ith layer, respectively. We set the value of $\varepsilon(i+1)$ so that $0 < \varepsilon(i+1) < 0.5$. Then, setting $z_j^*(i+1) = 1 - \varepsilon(i+1)$, we get

$$T_j(i) = \frac{-\mathbf{x}_j^{*T}(i)\,\mathbf{w}_j(i) - b_j(i)}{\log\left(\frac{\varepsilon(i+1)}{1-\varepsilon(i+1)}\right)}. \tag{9.25}$$

9.7 Performance Evaluation

We compared the performance of maximum-margin neural networks (MM NNs) against one-against-all fuzzy support vector machines (FSVMs) and multilayer neural networks trained by the back-propagation algorithm (BP) using the data sets listed in Table 1.1. We used a Pentium III 1GHz personal computer.

In training FSVM and the output layer of MM NN, we used the primal-dual interior-point method combined with variable chunking (50 data were added). For FSVM we used polynomial kernels.

In training neural networks by the back-propagation algorithm, we set $T(2) = T(3) = 1$, the learning rate to be 1, and the maximum number of epochs to be $15,000$. The number of hidden units was set to be equal to or smaller than the number generated by MM NN. The neural network was trained 10 times (3 times for hiragana data) changing the initial weights.

For MM NN, we set $C(3) = 500$, $\varepsilon(2) = 0.4$, $\varepsilon(3) = 0.2$, and we evaluated the performance changing the value of $C(2)$.

Table 9.1 lists the best performance obtained by FSVM, the neural network trained by the back-propagation algorithm (BP), and MM NN and the training time. The highest recognition rate of the test data is shown in boldface. The column "Parm" for BP shows the number of hidden neurons.

From the table, recognition performance of MM NN and FSVM is comparable, but for the thyroid data, the recognition rate of the test data for MM NN is higher than for FSVM by about 1 percent, although the recognition rates of the training data are almost the same. Except for the numeral data, BP showed inferior recognition performance than MM NN. Especially for the

Table 9.1. Performance comparison of maximum-margin neural networks

Data	Classifier	Parm	Rate (%)	Time (s)
Numeral	FSVM	$d3$	99.51 (100)	1
	BP	18	**99.76** (100)	3
	MM NN	$C(2) = 0.2$	99.63 (100)	1
Thyroid	FSVM	$d3$	97.55 (99.26)	64
	BP	5	98.37 (99.58)	280
	MM NN	$C(2) = -0.2$	**98.51** (99.20)	29
Blood cell	FSVM	$d3$	**93.26** (98.22)	30
	BP	85	91.61 (98.29)	3091
	MM NN	$C(2) = -0.5$	93.00(98.84)	244
Hiragana-50	FSVM	$d5$	**99.46** (100)	122
	BP	100	97.25 (98.91)	11262
	MM NN	$C(2) = 0.6$	99.02 (100)	919
Hiragana-105	FSVM	$d2$	**100** (100)	564
	BP	100	99.84 (99.99)	28801
	MM NN	$C(2) = -0.3$	**100** (100)	2711
Hiragana-13	FSVM	$d2$	99.51(99.92)	292
	BP	100	99.17 (99.51)	13983
	MM NN	$C(2) = -0.1$	**99.58** (100)	2283

hirgana-50 data, the recognition rate of the test data was lower. This is because the number of the hiragana-50 training data per class is small and there is no overlap between classes. Thus, without the mechanism of maximizing margins, the high recognition rate was not obtained.

Training of MM NN was faster than that of BP, but except for the thyroid data, slower than that of FSVM. The reason for slow training compared to that of FSVM is that by MM NN a large number of hidden neurons were generated. For example, for hiragana-13 data, 261 hidden neurons were generated.

9.8 Summary

In this chapter, we discussed training multilayer neural networks layer by layer, maximizing the margins of separating hyperplanes. In training the input-to-hidden-layer weights, we discussed the geometrical training method, and in training the hidden-to-output-layer weights, we used the conventional quadratic training method. According to the computer experiments, classification performance of the proposed method was comparable to that of the support vector machine and better than that of the multilayer neural network trained by the back-propagation algorithm.

10

Maximum-Margin Fuzzy Classifiers

In conventional fuzzy classifiers, fuzzy rules are defined by experts. First, we divide the ranges of input variables into several nonoverlapping intervals. And for each interval, we define a membership function, which defines the degree to which the input value belongs to the interval. Now the input space is covered with nonoverlapping hyperrectangles. We define, for each hyperrectangle, a fuzzy rule. Suppose we have two variables, and each range is divided into three: small, medium, and large. An example of a fuzzy rule is as follows:

$$\text{If } x_1 \text{ is small and } x_2 \text{ is large, then } \mathbf{x} \text{ belongs to Class 2.}$$

One of the advantages of fuzzy classifiers over multilayer neural networks or support vector machines is that we can easily understand how they work. But because the fuzzy rules need to be defined by experts, development of classifiers is difficult.

To overcome this problem, many fuzzy classifiers that can be trained using numerical data have been developed [3]. Trainable fuzzy classifiers are classified, from the shape of the approximated class regions, into

1. fuzzy classifiers with hyperbox regions [3, 11, 226],
2. fuzzy classifiers with polyhedral regions [3, 224, 245, 253], and
3. fuzzy classifiers with ellipsoidal regions [2, 3, 13].

For these fuzzy classifiers, fuzzy rules are defined either by preclustering or postclustering the training data. In preclustering, the training data for each class are divided into clusters in advance, and using the training data in the cluster a fuzzy rule is defined. Then we tune slopes and/or locations of the membership functions so that the recognition rate of the training data is maximized allowing the previously correctly classified data to be misclassified. In postclustering, one fuzzy rule is defined using the training data included in each class and the membership functions are tuned. Then if the recognition rate of the training data is not sufficient, we define the fuzzy rule for the misclassified training data.

These fuzzy classifiers are trained so that the recognition rate of the training data is maximized. Thus, if the recognition rate reaches 100 percent, the training is terminated and no further tuning for the class boundaries is done. Therefore, the generalization ability of the conventional fuzzy classifiers degrades especially when the number of training data is small or the number of the input variables is large.

In this chapter we first discuss an architecture of a kernel version of a fuzzy classifier with ellipsoidal regions and improvement of generalization ability by transductive training, in which unlabeled data are used [126, 127], and by maximizing margins [12]. Then we discuss the architecture of a fuzzy classifier with polyhedral regions and an efficient rule-generation method [245].

10.1 Kernel Fuzzy Classifiers with Ellipsoidal Regions

In this section, we first summarize the architecture of a fuzzy classifier with ellipsoidal regions and then discuss its kernel version [126] and improvement of generalization ability by transductive training [127].

10.1.1 Conventional Fuzzy Classifiers with Ellipsoidal Regions

We summarize the conventional fuzzy classifier with ellipsoidal regions, which is generated in the input space [3, 12, 13]. Here we consider classification of the m-dimensional input vector $\mathbf{x}$ into one of n classes. We can define more than one fuzzy rule for each class, but to make discussions simple, we assume that we define one fuzzy rule for each class:

$$R_i\text{: if } \mathbf{x} \text{ is } \mathbf{c}_i \text{ then } \mathbf{x} \text{ belongs to class } i, \tag{10.1}$$

where $\mathbf{c}_i$ is the center vector of class i:

$$\mathbf{c}_i = \frac{1}{M_i} \sum_{j=1}^{M_i} \mathbf{x}_{ij}. \tag{10.2}$$

Here, M_i is the number of training data for class i and $\mathbf{x}_{ij}$ is the jth training datum for class i.

We define a membership function $m_i(\mathbf{x})$ $(i = 1, \ldots, n)$ for input $\mathbf{x}$ by

$$m_i(\mathbf{x}) = \exp(-h_i^2(\mathbf{x})), \tag{10.3}$$

$$\begin{aligned} h_i^2(\mathbf{x}) &= \frac{d_i^2(\mathbf{x})}{\alpha_i} \\ &= \frac{1}{\alpha_i} (\mathbf{x} - \mathbf{c}_i)^T Q_i^{-1} (\mathbf{x} - \mathbf{c}_i), \end{aligned} \tag{10.4}$$

where $d_i(\mathbf{x})$ is the Mahalanobis distance between $\mathbf{x}$ and $\mathbf{c}_i$; $h_i(\mathbf{x})$ is the tuned distance; α_i is the tuning parameter for class i; and Q_i is the covariance matrix for class i in the input space.

The covariance matrix for class i in the input space is given by

$$Q_i = \frac{1}{M_i} \sum_{j=1}^{M_i} (\mathbf{x}_{ij} - \mathbf{c}_i)(\mathbf{x}_{ij} - \mathbf{c}_i)^T. \tag{10.5}$$

We calculate the membership function of input $\mathbf{x}$ for each class. If the degree of membership for class j is maximum, the input is classified into class j. When α_i in (10.4) is equal to 1, this is equivalent to finding the minimum Mahalanobis distance. Function (10.3) makes the output range of (10.3) lie in [0,1], and if $m_j(\mathbf{x})$ is equal to 1, the input $\mathbf{x}$ corresponds to the center of class j, $\mathbf{c}_j$. We tune the membership function using α_i in (10.4).

When Q_i is positive definite, the Mahalanobis distance given by (10.4) can be calculated using the symmetric Cholesky factorization [96]. Thus Q_i can be decomposed into

$$Q_i = L_i \, L_i^T, \tag{10.6}$$

where L_i is the real valued regular lower triangular matrix.

But when Q is positive semidefinite, the value in the square root of diagonal element of L_i is nonpositive. To avoid this, during the Cholesky factorization, if the value in the square root is smaller than ζ, where ζ is a predefined threshold, the element is replaced by $\sqrt{\zeta}$ [3, 12]. This means that when the covariance matrix is positive semidefinite, the principal components in the subspace that the training data do not span, are taken into consideration to calculate the Mahalanobis distance. Thus by this method, the generalization ability does not decrease even when the training data are degenerate, namely, the training data do not span the input space.

10.1.2 Extension to a Feature Space

In a kernel fuzzy classifier with ellipsoidal regions [126], the input space is mapped into the feature space by a nonlinear mapping function. Because we map the input space into the feature space, we assume that each class consists of one cluster and we define a fuzzy rule for each class.

We define the following fuzzy rule for class i:

$$R_i : \text{if } \; \mathbf{g}(\mathbf{x}) \text{ is } \mathbf{c}_i \;\text{ then } \mathbf{x} \text{ belongs to class } i, \tag{10.7}$$

where $\mathbf{g}(\mathbf{x})$ is a mapping function that maps the input space into the l-dimensional feature space, $\mathbf{c}_i$ is the center of class i in the feature space and is calculated by the training data included in class i:

$$\mathbf{c}_i = \frac{1}{|X_i|} \sum_{\mathbf{x} \in X_i} \mathbf{g}(\mathbf{x}), \tag{10.8}$$

where X_i is the set of training data included in class i, and $|X_i|$ is the number of data included in X_i.

For center $\mathbf{c}_i$, we define the membership function $m_{\mathbf{g}_i}(\mathbf{x})$ that defines the degree to which $\mathbf{x}$ belongs to $\mathbf{c}_i$:

$$m_{\mathbf{g}_i}(\mathbf{x}) = \exp(-h^2_{\mathbf{g}_i}(\mathbf{x})), \tag{10.9}$$

$$h^2_{\mathbf{g}_i}(\mathbf{x}) = \frac{d^2_{\mathbf{g}_i}(\mathbf{x})}{\alpha_i}, \tag{10.10}$$

$$d^2_{\mathbf{g}_i}(\mathbf{x}) = (\mathbf{g}(\mathbf{x}) - \mathbf{c}_i)^T Q^+_{\mathbf{g}_i} (\mathbf{g}(\mathbf{x}) - \mathbf{c}_i), \tag{10.11}$$

where $h_{\mathbf{g}_i}(\mathbf{x})$ is a tuning distance, $d_{\mathbf{g}_i}(\mathbf{x})$ is a kernel Mahalanobis distance between $\mathbf{g}(\mathbf{x})$ and $\mathbf{c}_i$, α_i is a tuning parameter for class i, $Q_{\mathbf{g}_i}$ is the $l \times l$ covariance matrix for class i. And $Q^+_{\mathbf{g}_i}$ denotes the pseudo-inverse of the covariance matrix $Q_{\mathbf{g}_i}$. Here we calculate the covariance matrix $Q_{\mathbf{g}_i}$ using the data belonging to class i as follows:

$$Q_{\mathbf{g}_i} = \frac{1}{|X_i|} \sum_{\mathbf{x} \in X_i} (\mathbf{g}(\mathbf{x}) - \mathbf{c}_i)(\mathbf{g}(\mathbf{x}) - \mathbf{c}_i)^T. \tag{10.12}$$

In calculating the kernel Mahalanobis distance, we use either of the methods discussed in Section 8.3.

For an input vector $\mathbf{x}$ we calculate the degrees of membership for all the classes. If $m_{\mathbf{g}_k}(\mathbf{x})$ is the maximum, we classify the input vector into class k.

10.1.3 Transductive Training

Concept

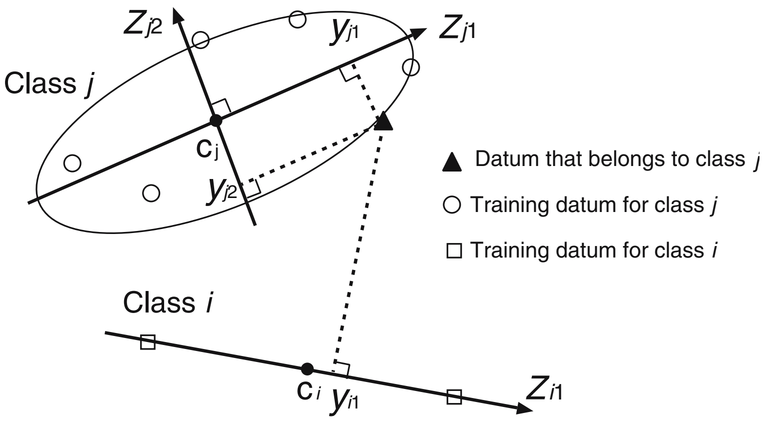
We discussed the SVD-based Mahalanobis distance in Section 8.3.1 and the KPCA-based Mahalanobis distance in Section 8.3.2. But with those methods, when the training data are degenerate, i.e., the space spanned by the mapped training data is a proper subspace in the feature space, the generalization ability of the fuzzy classifier is decreased [126]. It is because the principal components are zero in the subspace complementary to that spanned by the mapped training data.

We now explain why using the example shown in Fig. 10.1. In the figure, training data for class j are in the two-dimensional space, but those of class i are in one dimension.

Assume that we have a datum $\mathbf{x}$ belonging to class j. This datum is in the subspace spanned by the mapped training data belonging to class j, but not in the subspace spanned by class i. Then the kernel Mahalanobis distance between $\mathbf{x}$ and $\mathbf{c}_j$ is correctly calculated by

$$d^2_{\mathbf{g}_j}(\mathbf{g}(\mathbf{x})) = \frac{y^2_{j1}}{\lambda_{j1}} + \frac{y^2_{j2}}{\lambda_{j2}}. \tag{10.13}$$

But for class i, because the second eigenvalue is zero due to the degeneracy of the training data, $d^2_{\mathbf{g}_j}(\mathbf{x})$ is erroneously given by



z_{j2}
y_{j1}
z_{j1}
Class j
c_j
y_{j2}
▲ Datum that belongs to class j
○ Training datum for class j
□ Training datum for class i
Class i
c_i
y_{i1}
z_{i1}

Fig. 10.1. The training data of class i are degenerate. Because the second eigenvalue of class i is zero, the kernel Mahalanobis distance for class i, $d_{\mathbf{g}_i}(\mathbf{x})$, cannot be calculated correctly. Reprinted from [127, p. 201] with permission from Elsevier

$$d^2_{\mathbf{g}_j}(\mathbf{x}) = \frac{y^2_{i1}}{\lambda_{i1}}. \tag{10.14}$$

Because the kernel Mahalanobis distance for class i does not change even if $\mathbf{x}$ moves in the direction orthogonal to the eigenvector $\mathbf{z}_{i1}$, classification by the Mahalanobis distance becomes erroneous.

Similar to (8.47), we can overcome this problem, adding the vector that is orthogonal to the existing eigenvectors for the covariance matrix with a small positive eigenvalue. The next problem is how to select the appropriate vectors for addition. If the linear kernel is used and the training data are degenerate, i.e., the training data do not span the input space, we can add the basis vectors in the input space that are in the complementary subspace of the training data. But if the nonlinear kernel is used, the mapped training data may not span the feature space even though the training data are not degenerate.

To solve this problem, we use transductive training of the classifier. In training, we add the basis vectors of the input space as the unlabeled training data, and if the mappings of these data are in the complementary subspace associated with the covariance matrix, we generate associated eigenvectors with small positive eigenvalues. Namely, first we map the basis vectors into the feature space, and then we perform the Cholesky factorization to judge whether each mapped basis vector is included in the subspace associated with the covariance matrix. If it is not included in the subspace, we calculate the vector that is orthogonal to the subspace and modify the covariance matrix.

By this method, however, the complementary subspace may remain. Thus, in classification, whenever an unknown datum is given, we judge whether the mapped datum is included in the subspace associated with the covariance matrix by the Cholesky factorization. If it is not included, we calculate the vector orthogonal to the previously selected vectors by the Gram-Schmidt orthogonalization and modify the covariance matrix. We can speed up factorization, storing the previously factorized matrices and factorizing the row and column associated with only the unknown datum.

Transductive Training Using Basis Vectors

In this section, we discuss how to improve approximation of the feature space using the basis vectors $\{\mathbf{e}_1, \ldots, \mathbf{e}_m\}$ in the input space in addition to the vector of the mapped training data $\mathbf{g}(X_i)$. Because the class labels of $\{\mathbf{e}_1, \ldots, \mathbf{e}_m\}$ are not known, these data should not affect the principal components associated with $\mathbf{g}(X_i)$. Therefore, first we calculate the eigenvalues $\{\lambda_1, \ldots, \lambda_{M'_i}\}$ and eigenvectors $\{\mathbf{z}_1, \ldots, \mathbf{z}_{M'_i}\}$ of $Q_{\mathbf{g}_i}$ and add eigenvectors with small positive eigenvalues that are orthogonal to $\{\mathbf{z}_1, \ldots, \mathbf{z}_{M'_i}\}$ using some of the $\{\mathbf{g}(\mathbf{e}_1), \ldots, \mathbf{g}(\mathbf{e}_m)\}$.

Now we will explain the procedure in more detail. In a similar way as discussed in Section 8.1, we can select independent vectors by the Cholesky factorization of the kernel matrix for $\{\mathbf{x}_1, \ldots, \mathbf{x}_{M_i}\}$ and $\{\mathbf{e}_1, \ldots, \mathbf{e}_m\}$:

$$H(X'_i, X'^T_i) = \mathbf{g}(X'_i)\, \mathbf{g}^T(X'_i), \tag{10.15}$$

where

$$\mathbf{g}(X'_i) = \big(\mathbf{g}(\mathbf{x}_1) \cdots \mathbf{g}(\mathbf{x}_{M_i})\; \mathbf{g}(\mathbf{e}_1) \cdots \mathbf{g}(\mathbf{e}_m)\big)^T. \tag{10.16}$$

Placing $\{\mathbf{g}(\mathbf{e}_1), \ldots, \mathbf{g}(\mathbf{e}_m)\}$ after $\mathbf{g}(\mathbf{x}_{M_i})$, we can select independent vectors from $\{\mathbf{e}_1, \ldots, \mathbf{e}_m\}$ that are not included in the subspace spanned by $\mathbf{g}(X_i)$. Assume that m' basis vectors, $\mathbf{g}(\mathbf{e}_r)$ $(r = 1, \ldots, m')$ are selected as the independent vectors in addition to M'_i independent training vectors. From the M'_i independent training data, the orthogonal system $\{\mathbf{z}_1, \ldots, \mathbf{z}_{M'_i}\}$ is calculated by the method discussed in Section 8.3.2.

Next, using $\mathbf{g}(\mathbf{e}_r)$ we generate the $(M'_i + 1)$st extra eigenvector $\mathbf{z}_{M'_i+1}$ by the Gram-Schmidt orthogonalization:

$$\mathbf{z}_{M'_i+1} = \frac{\mathbf{p}_r}{\|\mathbf{p}_r\|}, \tag{10.17}$$

where $\|\mathbf{p}_r\|$ is the norm of $\mathbf{p}_r$:

$$\begin{aligned} \mathbf{p}_r &= \mathbf{g}(\mathbf{e}_r) - \sum_{i=1}^{M'_i} (\mathbf{g}^T(\mathbf{e}_r)\, \mathbf{z}_i)\, \mathbf{z}_i \\ &= \mathbf{g}(\mathbf{e}_r) - \sum_{i=1}^{M'_i} \omega_i\, \mathbf{z}_i. \end{aligned} \tag{10.18}$$

Here, $\omega_i = \mathbf{g}^T(\mathbf{e}_r)\, \mathbf{z}_i$.

By substituting (8.26) into (10.18), $\mathbf{z}_{M_i'+1}$ is expressed by the linear sum of $\{\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_{M_i'}), \mathbf{g}(\mathbf{e}_r)\}$:

$$\mathbf{z}_{M_i'+1} = \left(\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_{M_i'}), \mathbf{g}(\mathbf{e}_r)\right) \begin{pmatrix} \rho_{M_i'+1,1} \\ \vdots \\ \rho_{M_i'+1,M_i'} \\ \frac{1}{\|\mathbf{p}_r\|} \end{pmatrix}, \tag{10.19}$$

where

$$\rho_{M_i'+1,j} = -\frac{1}{\|\mathbf{p}_r\|} \sum_{i=1}^{M_i'} \omega_i \, \rho_{ij} \quad \text{for} \quad j = 1, \ldots, M_i'. \tag{10.20}$$

Using (10.19), we can calculate the $(M_i' + 1)$st principal component as follows:

$$\begin{aligned} y_{M_i'+1} &= \mathbf{z}_{M_i'+1}^T (\mathbf{g}(\mathbf{x}) - \mathbf{c}_i) \\ &= \sum_{i=1}^{M_i'} \rho_{M_i'+1,i} \, H(\mathbf{x}, \mathbf{x}_i) + \frac{1}{\|\mathbf{p}_r\|} H(\mathbf{e}_r, \mathbf{x}) - \mathbf{z}_{M_i'+1}^T \, \mathbf{c}_i. \end{aligned} \tag{10.21}$$

We iterate this procedure until the rest of the principal components y_i $(i = M_i' + 1, \ldots, M_i' + m')$ are calculated.

We add these new principal components to the right-hand side of (8.51), and finally the Mahalanobis distance becomes:

$$\delta_i^2(\mathbf{g}(\mathbf{x})) = \frac{y_1^2}{\lambda_1} + \cdots + \frac{y_{M_i'}^2}{\lambda_{M_i'}} + \frac{y_{M_i'+1}^2}{\varepsilon} + \cdots + \frac{y_{M_i'+m'}^2}{\varepsilon}, \tag{10.22}$$

where for the added eigenvectors the associated eigenvalues are assumed to be small and we set a small value to ε. We call (10.22) the *KPCA-based Mahalanobis distance with Gram-Schmidt orthogonalization.*

Transductive Training Using Unknown Data

In the previous section, we discussed approximation of a subspace using the mapped basis vectors $\mathbf{g}(\mathbf{e}_1), \ldots, \mathbf{g}(\mathbf{e}_m)$. With this method, the input space is spanned if linear kernels are used, but for nonlinear kernels the whole feature space cannot be covered. Thus approximation with the basis vectors may not be enough to prevent generalization ability from decreasing.

To overcome this problem, we need to make unknown data lie in the subspace spanned by the eigenvectors associated with the covariance matrix. To do so, when an unknown datum $\mathbf{t}$ is given, we judge whether this datum is included in the space spanned by $\mathbf{g}(X_i')$. If the datum is not in the space, we generate the extra eigenvector in a similar way as discussed previously. Namely, for the kernel matrix

$$H(X_i'', X_i''^T) = \mathbf{g}(X_i'')\mathbf{g}^T(X_i''), \tag{10.23}$$

where

$$\mathbf{g}(X_i'') = (\mathbf{g}(X_i'), \, \mathbf{g}(\mathbf{t}))^T, \tag{10.24}$$

we perform the Cholesky factorization. But because the factorization of $H(X_i'$, $X_i'^T)$ can be done off-line, we only have to calculate the $(M_i + m + 1)$st row and column of $H(X_i'', X_i''^T)$ (see (8.11) and (8.12)). If the value in the square root in (8.12) is larger than $\eta\,(> 0)$ (see (8.13)), the unknown datum $\mathbf{t}$ is not included in the space spanned by $\mathbf{g}(X_i')$. Thus we generate the extra eigenvector to span the subspace.

This method is not time-consuming because we do not need to calculate the whole elements. When we are not given enough training data, this online approximation may be effective.

10.1.4 Maximizing Margins

Concept

In the fuzzy classifier with ellipsoidal regions, if there are overlaps between classes, the overlaps are resolved by tuning the membership functions, i.e., by tuning α_i one at a time. When α_i is increased, the slope of $m_{\mathbf{g}_i}(\mathbf{x})$ is decreased and the degree of membership is increased. Then misclassified data may be correctly classified and correctly classified data may be misclassified. Based on this, we calculate the net increase of the correctly classified data. Likewise, by increasing the slope, we calculate the net increase of the correctly classified data. Then allowing new misclassification, we tune the slope so that the recognition rate is maximized. In this way we tune fuzzy rules successively until the recognition rate of the training data is not improved [3, pp. 121–9].

But if there is no overlap, the membership functions are not tuned. When the number of training data is small, usually the overlaps are scarce. Thus, the generalization ability is degraded. To tune membership functions even when the overlaps are scarce, we use the idea used in training support vector machines. In training support vector machines for a two-class problem, the separating margin between the two classes is maximized to improve the generalization ability. In the kernel fuzzy classifier with ellipsoidal regions, we tune the slopes of the membership functions so that the slope margins are maximized.

Initially, we set the values of α_i to be 1. Namely, the kernel fuzzy classifier with ellipsoidal regions is equivalent to the classifier based on the kernel Mahalanobis distance. Then we tune α_i so that the slope margins are maximized. Here we maximize the margins without causing new misclassification. When the recognition rate of the training data is not 100 percent after tuning, we tune α_i so that the recognition rate is maximized as discussed in [3]. Here we discuss how to maximize slope margins.

Unlike the support vector machines, tuning of α_i is not restricted to two classes, but for ease of illustration we explain the concept of tuning using two classes. In Fig. 10.2 the filled rectangle and circle show the training data, belonging to classes i and j, that are nearest to classes j and i, respectively. The class boundary of the two classes is somewhere between the two curves shown in the figure. We assume that the generalization ability is maximized when it is in the middle of the two.

In Fig. 10.3, if the datum belongs to class i, it is correctly classified because the degree of membership for class i is larger. This datum remains correctly classified until the degree of membership for class i is decreased, as shown in the dotted curve. Similarly, in Fig. 10.4, if the datum belongs to class j, it is correctly classified because the degree of membership for class j is larger. This datum remains correctly classified until the degree of membership for class i is increased as shown in the dotted curve. Thus, for each α_i, there is an interval of α_i that makes correctly classified data remain correctly classified. Therefore, if we change the value of α_i so that it is in the middle of the interval, the slope margins are maximized.

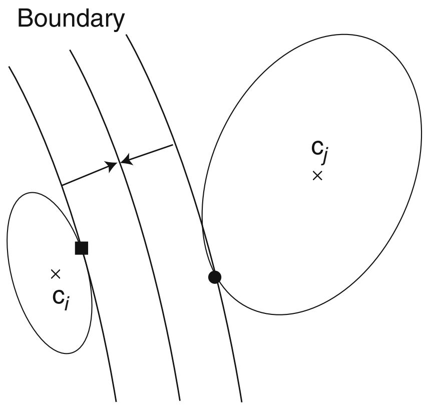


Fig. 10.2. Concept of maximizing margins. From [12, p. 209, ©IEEE 2001]

In the following we discuss how to tune α_i.

Upper and Lower Bounds of $\boldsymbol{\alpha_i}$

Let X be the set of training data that are correctly classified for the initial α_i. Let $\mathbf{x}\,(\in X)$ belong to class i.

If $m_{\mathbf{g}_i}(\mathbf{x})$ is the largest, there is a lower bound of α_i, $L_i(\mathbf{x})$, to keep $\mathbf{x}$ correctly classified:

$$L_i(\mathbf{x}) = \frac{d^2_{\mathbf{g}_i}(\mathbf{x})}{\min\limits_{j\neq i} h^2_{\mathbf{g}_j}(\mathbf{x})}. \tag{10.25}$$

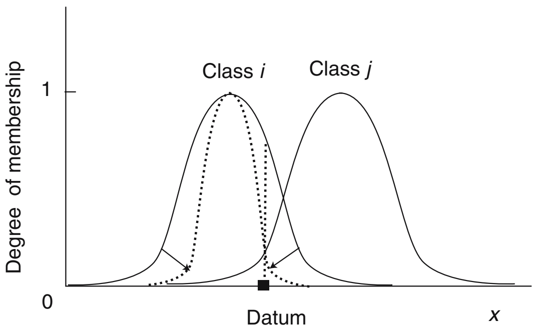


Fig. 10.3. Upper bound of α_i that does not cause misclassification. From [12, p. 209, ©IEEE 2001]

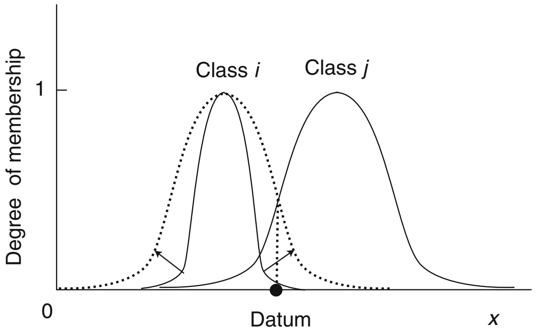


Fig. 10.4. Lower bound of α_i that does not cause misclassification. From [12, p. 209, ©IEEE 2001]

Then the lower bound $L_i(1)$ that does not cause new misclassification is given by

$$L_i(1) = \max_{\mathbf{x} \in X} L_i(\mathbf{x}). \tag{10.26}$$

Similarly, for $\mathbf{x}\,(\in X)$ belonging to a class other than class i, we can calculate the upper bound of α_i. Let $m_{\mathbf{g}_j}(\mathbf{x})\,(j \neq i)$ be the largest. Then the upper bound $U_i(\mathbf{x})$ of α_i that does not cause misclassification of $\mathbf{x}$ is given by

$$U_i(\mathbf{x}) = \frac{d^2_{\mathbf{g}_i}(\mathbf{x})}{\min\limits_{j} h^2_{\mathbf{g}_j}(\mathbf{x})}. \tag{10.27}$$

The upper bound $U_i(1)$ that does not make new misclassification is given by

$$U_i(1) = \min_{\mathbf{x} \in X} U_i(\mathbf{x}). \tag{10.28}$$

In [3, p. 122], $L_i(l)$ and $U_i(l)$ are defined as the lower and upper bounds in which $l-1$ correctly classified data are misclassified, respectively. Thus, $L_i(1)$ and $U_i(1)$ are the special cases of $L_i(l)$ and $U_i(l)$.

Tuning Procedure

The correctly classified data remain correctly classified even when α_i is set to some value in the interval $(L_i(1), U_i(1))$. The tuning procedure of α_i becomes as follows. For α_i we calculate $L_i(1)$ and $U_i(1)$ and set the value of α_i with the middle point of $L_i(1)$ and $U_i(1)$:

$$\alpha_i = \frac{1}{2}(L_i(1) + U_i(1)). \tag{10.29}$$

We successively tune one α_i after another. Tuning results depend on the order of tuning α_i, but usually tuning from α_1 to α_n leads to a good result.

10.1.5 Performance Evaluation

We evaluated our methods using two groups of data sets: (1) multiclass data sets listed in Table 1.1 and (2) two-class data sets[1] used in [173, 199]. In the first group, each data set consists of one set of training and test data, and in the second group, each data set consists of 100 sets of training and test data. We used a Pentium III 1GHz personal computer to evaluate the method.

Multiclass Data Sets

For the basic fuzzy classifier with ellipsoidal regions, we need to set the value of ζ, which detects the singularity of the covariance matrix and the value of l_M, in which $l_M - 1$ is the maximum number of misclassification allowed for tuning. We selected the value of ζ so that the recognition rate of the test data was maximized. And we set $l_M = 10$, which usually gives good generalization ability.

We compared performance of the four kinds of the kernel Mahalanobis distance:

1. the singular value decomposition-based kernel Mahalanobis distance discussed in Section 8.3.1 (denoted hereafter as "SVD");
2. the KPCA-based Mahalanobis distance discussed in Section 8.3.2 in which the kernel matrix is used to select the independent vectors ("KPCA I");
3. the KPCA-based Mahalanobis distance given by (10.22) by transductive training using the basis vectors ("KPCA II"); and
4. the KPCA-based Mahalanobis distance by transductive training using the basis vectors and test data ("KPCA III").

[1] http://ida.first.fraunhofer.de/projects/bench/benchmarks.htm

Because SVD-based training took a long time, we determined the value of σ in (8.47) so that the recognition rates of the test data were the highest. Table 10.1 shows the difference between the conventional method, in which the subspace associated with the small eigenvalues is neglected, and the improved method (8.47) to perform singular value decomposition for the numeral data. "Initial" and "Final" denote the initial recognition rates of the test (training) data with the tuning parameters $\alpha_i = 1$ and after tuning, respectively. "Num." denotes the number of selected diagonal elements. From the table, the improved method of singular value decomposition is effective to prevent generalization ability from decreasing, especially for initial recognition rates for linear kernels. Thus in the following experiments, we use this method to calculate the pseudo-inverse when the training data are degenerate (numeral, thyroid, and hiragana-50 data sets).

Table 10.1. Comparison of singular value decomposition. Reprinted from [127, p. 209] with permission from Elsevier

Type	Kernel	Initial (%)	Final (%)	Num.
Conventional	Linear	88.78 (90.00)	98.78 (98.40)	100
($\sigma = 10^{-8}$)	γ0.01	98.90 (99.75)	**99.02** (100)	206
Improved	Linear	98.54 (97.28)	98.78 (98.52)	810
($\sigma = 10^{-8}$)	γ0.01	99.39 (99.88)	**99.27** (100)	810

To calculate a KPCA-based Mahalanobis distance, we need to set two parameters, ε, which determines the minimum value of the eigenvalues and η, which determines how strictly we select the independent vectors. From our computational experiments, we know that the generalization ability is more sensitive to the value of ε than that of η. In addition, the best value of ε depends on the training data sets and kernel functions.

Thus we determined the value of η by five-fold cross-validation for the blood cell data and made the value of η common to all training data sets and kernel functions. Then we performed five-fold cross-validation to determine the value of ε for each training data set and kernel function.

For linear kernels, polynomial kernels with $d = [2, 3]$, or RBF kernels with $\gamma = [0.001, 0.01, 0.1, 1, 10]$, we performed five-fold cross-validation for $\varepsilon = [10^{-8}, 10^{-7}, \ldots, 10^{-1}]$. We iterated cross-validation five times to decrease the influence of random selection and determined the value of ε with the highest average recognition rate. For KPCA I, II, and III, we determined the parameters for KPCA II and used the values for KPCA I and III.

Table 10.2 shows the recognition rates of the test data. In the "Type" column, (1), (2), and (3) denote that linear, polynomial, and RBF kernels were used, respectively. For each data set the highest recognition rate is shown in boldface. The table also includes the recognition rates for the pairwise L1 fuzzy support vector machine evaluated in Section 3.2.3, and "Basic" denotes the conventional fuzzy classifier with ellipsoidal regions.

If the training data set is not degenerate and KPCA I (1) selects independent vectors that span the input space, the basic fuzzy classifier, SVD (1), and KPCA I (1) will give the same recognition rate. Except for the numeral, thyroid, and hiragana-50 data sets, the training data sets are nondegenerate. Thus, for these sets, the recognition rates are almost the same.

For the degenerate training data sets, the recognition rates of SVD (1) and KPCA I (1) are inferior to that of the basic fuzzy classifier. This is especially evident for the thyroid data set. Using nonlinear kernels, the recognition rates were improved but still lower for the thyroid data set. In KPCA II (1), because the basis vectors in the input space were added, the degeneracy was resolved and the recognition rates are comparable with those of the basic fuzzy classifier. For the linear kernels there were no online additions of the basis vectors. Thus KPCA II (1) and KPCA III (1) give the same results. For nonlinear

Table 10.2. Recognition rates of the test data in %. Reprinted from [127, p. 212] with permission from Elsevier

Type		Blood	Numeral	Iris	Thyroid	H-50	H-105	H-13
SVM		92.03	**99.63**	**97.33**	**97.61**	**99.11**	99.95	99.74
Basic		91.32	99.27	**97.33**	95.62	98.85	**100**	99.34
SVD	(1)	91.32	98.78	**97.33**	89.53	96.38	99.99	99.25
	(2)	92.35	97.68	94.67	90.58	95.99	99.90	98.96
	(3)	92.45	99.27	96.00	94.40	93.51	**100**	99.74
KPCA I	(1)	91.23	99.02	96.00	87.25	97.53	**100**	99.43
	(2)	92.87	99.27	**97.33**	90.14	97.25	99.99	99.86
	(3)	91.35	99.39	94.67	95.01	97.57	**100**	99.86
KPCA II	(1)	91.23	99.51	96.00	94.75	98.31	**100**	99.43
	(2)	93.16	99.51	**97.33**	96.82	97.61	**100**	99.86
	(3)	91.32	99.39	94.67	95.01	98.52	**100**	**99.88**
KPCA III	(1)	91.23	99.51	96.00	94.75	98.31	**100**	99.43
	(2)	**93.23**	99.51	**97.33**	96.76	97.55	**100**	99.86
	(3)	91.03	99.39	94.67	95.71	98.52	**100**	**99.88**

Table 10.3. Recognition rates for test data by the Mahalanobis distance in %. Reprinted from [127, p. 212] with permission from Elsevier

Type		Blood	Numeral	Iris	Thyroid	H-50	H-105	H-13
Basic		87.45	99.63	**98.67**	86.41	98.79	100	98.36
SVD	(1)	87.45	98.54	**98.67**	74.77	80.89	99.98	98.36
	(2)	**92.42**	95.98	**97.33**	86.03	67.53	97.75	**99.63**
	(3)	91.58	**99.39**	**98.67**	83.46	82.43	99.99	**99.86**
KPCA I	(1)	88.65	96.34	**98.67**	72.40	94.84	100	99.15
	(2)	92.29	97.93	**98.67**	83.72	85.36	98.90	**99.90**
	(3)	89.52	97.20	**98.67**	80.25	81.13	99.96	99.84
KPCA II	(1)	88.65	99.27	**98.67**	84.92	97.85	100	99.15
	(2)	92.26	99.15	**98.67**	95.60	**97.96**	99.89	**99.90**
	(3)	89.48	99.39	**98.67**	87.95	98.50	100	99.86
KPCA III	(1)	88.65	99.27	**98.67**	84.92	97.85	100	99.15
	(2)	92.16	99.27	**98.67**	95.68	**98.42**	100	**99.89**
	(3)	89.58	99.39	**98.67**	90.32	98.48	100	99.87

kernels KPCA II shows better performance than SVD. But there is not much difference between KPCA II and KPCA III. This means that the training data and the basis vectors in the input space are sufficient to represent the space spanned by the test data.

Performance of the SVM and that of the KPCA-based methods are comparable.

Table 10.3 shows the recognition rates of the test data without tuning the tuning parameters. This means that the classifier classifies a datum according to the (kernel) Mahalanobis distance. The recognition rates that are higher than those by tuning are shown in boldface.

From Tables 10.2 and 10.3, in most cases by tuning the tuning parameters, the recognition rates of the test data are improved. The effect of tuning is especially evident for blood cell and thyroid data sets. But for the iris data set, the recognition rates were not improved by tuning for all the cases tried.

Each classifier has a different number of parameters to optimize. Thus, it is difficult to compare the training time of each classifier fairly. Therefore, here we only compare the training time of fuzzy classifiers for the parameters determined by model selection. Table 10.4 lists the training time of each method for the given conditions. In the table, 0 means that the calculation time is shorter than 0.5 second. Because training of the classifiers for the iris data set was very short, we do not include the result in the table.

We do not include the training time of KPCA III because all the test data were added at once. For KPCA III, we measured the time to process one unknown datum for classification by the fast online method, but it was too short to be measured correctly.

Training of the basic fuzzy classifier is the fastest, and training of KPCA I and II is the second fastest. Training of SVD is the slowest due to singular value decomposition. From this table the effectiveness of the KPCA-based methods is evident.

Table 10.4. Training time comparison in seconds. Reprinted from [127, p. 214] with permission from Elsevier

Type		Blood	Numeral	Thyroid	H-50	H-105	H-13
Basic		0	0	0	2	8	1
SVD	(1)	228	2	96,660	98	906	991
	(2)	218	2	96,120	101	933	885
	(3)	222	2	94,440	100	841	983
KPCA I	(1)	2	0	31	7	52	5
	(2)	10	1	194	33	239	31
	(3)	5	0	364	21	40	21
KPCA II	(1)	2	0	31	8	69	6
	(2)	12	1	201	50	395	35
	(3)	6	0	384	33	72	23

Two-Class Data Sets

We compared recognition rates of the SVM, the conventional fuzzy classifier with ellipsoidal regions, KPCA I, and KPCA II using the two-class benchmark data sets. As a reference we used recognition performance of the SVM listed in the home page.[2]

Throughout the experiment, we set $\eta = 10^{-5}$ for independent data selection. As for the determination of the values of γ and ε, first we fixed ε to 10^{-7}, and performed five-fold cross-validation of the first five training data sets for $\gamma = [0.001, 0.01, 0.1, 1, 10]$ and selected the median of the best value for each data set. Then fixing the value of γ with the median we performed five-fold cross-validation for $\varepsilon = [10^{-8}, 10^{-7}, \ldots, 10^{-1}, 0.5]$. We also determine

[2]http://ida.first.fraunhofer.de/projects/bench/benchmarks.htm

the value of ζ by five-fold cross-validation. For the ringnorm and titanic data sets we performed cross-validation, including $d = [2, 3]$.

Table 10.5 lists the mean classification errors and standard deviations with the $\pm$ symbol. Comparing Basic and KPCA I, except for the breast cancer and flare-solar data sets, KPCA I is better than or comparable to the conventional fuzzy classifier with ellipsoidal regions (Basic). KPCA II is better than the conventional fuzzy classifier with ellipsoidal regions and better than or comparable to KPCA I. Thus, the generalization improvement of KPCA II over the conventional fuzzy classifier with ellipsoidal regions is clear. As for KPCA II and the SVM, except for the ringnorm and splice data sets, they are comparable. To improve the recognition performance for the ringnorm and splice data sets we need to do more extensive parameter survey. We tested the performance of KPCA III for these data sets, but the generalization performance was not improved. This may mean that the transductive training using the basis vectors was enough for these data sets.

Table 10.5. Comparison among the four methods. Reprinted from [127, p. 215] with permission from Elsevier

Data	SVM	Basic	KPCA I	KPCA II
Banana	11.5±0.7	35.8±4.2	**10.9**±0.6	**10.9**±0.6
Breast cancer	**26.0**±4.7	28.9±5.3	33.5±4.9	26.5±4.4
Diabetes	**23.5**±1.7	25.8±2.2	25.3±2.0	25.3±2.0
German	**23.6**±2.1	27.3±2.6	25.2±2.5	25.2±2.4
Heart	**16.0**±3.3	20.2±3.7	16.5±3.6	16.5±3.6
Image	3.0±0.6	11.4±1.2	3.1±0.9	**2.9**±0.7
Ringnorm	**1.7**±0.1	27.6±1.7	3.6±0.4	3.2±0.3
Flare-solar	**32.4**±1.8	34.6±1.8	47.5±2.0	34.4±2.3
Splice	**10.9**±0.7	15.9±1.1	15.2±1.0	15.2±1.0
Thyroid	**4.8**±2.2	8.9±2.8	4.9±2.4	5.0±2.2
Titanic	**22.4**±1.0	23.0±1.1	23.0±1.3	22.5±1.2
Twonorm	3.0±0.2	3.6±0.4	**2.6**±0.2	**2.6**±0.3
Waveform	**9.9**±0.4	19.5±1.9	12.0±0.9	11.9±0.9

10.1.6 Summary

In this section we discussed a kernel fuzzy classifier with ellipsoidal regions, in which the input space is mapped into a high-dimensional feature space and a fuzzy classifier is generated in the feature space.

To speed up training, we used the KPCA-based method to calculate a Mahalanobis distance in the feature space. To improve generalization ability when training data are degenerate, we discussed transductive training of the classifier. Namely, using the basis vectors in the input space as unlabeled data, we span the space by the Gram-Schmidt orthogonalization. We extended this method to online training.

Using the multiclass and two-class classification benchmark data sets, we confirmed the training speedup and improvement of the generalization ability over the conventional fuzzy classifier with ellipsoidal regions.

10.2 Fuzzy Classifiers with Polyhedral Regions

The generalization of a classifier depends on how well we approximate the input region for each class. Approximation by polyhedrons is one way to improve the approximation accuracy. In [3, 224], starting from an initial convex hull for a class, the convex hull is expanded for the training data that are not in the convex hull. By this method, however, the number of generated facets explodes as the numbers of input variables and training data increase. To overcome this problem, in this section we discuss a different approach. We start from a hyperbox that includes all the class data and cut the hyperbox in the region where class data overlap [245]. We evaluate the performance of the method for some benchmark data sets.

10.2.1 Training Methods

Concept

In the following we discuss how to approximate a region for class i by a convex hull:

1. As shown in Fig. 10.5, we first approximate the class region by a hyperbox calculating the minimum and maximum values of the training inputs belonging to class i.
2. We sequentially read data belonging to classes other than class i. Let $\mathbf{q}$ be the current datum. If $\mathbf{q}$ is in the convex hull (initially the hyperbox), we cut it so that class i data are on the facet or inside the convex hull and that the facet is far from $\mathbf{q}$. This is the same idea with support vector machines that maximize margins. In Fig. 10.5, let $\mathbf{c}_i$ be the center of class i. To make class i data in the convex hull or on the facet, we project class i data in the direction orthogonal to $\mathbf{q} - \mathbf{c}_i$. Then, let $\mathbf{p}'$ be the datum nearest to $\mathbf{q}$ on the line that includes $\mathbf{q} - \mathbf{c}_i$. If we cut the convex hull by the facet that goes through $\mathbf{p}'$ and that is orthogonal to $\mathbf{q} - \mathbf{c}_i$, the distance between $\mathbf{q}$ and the facet is not maximized. Thus as shown in Fig. 10.6, we cut the convex hull by the facet that goes through $\mathbf{p}'$ and that is orthogonal to $\mathbf{q} - \mathbf{p}'$.

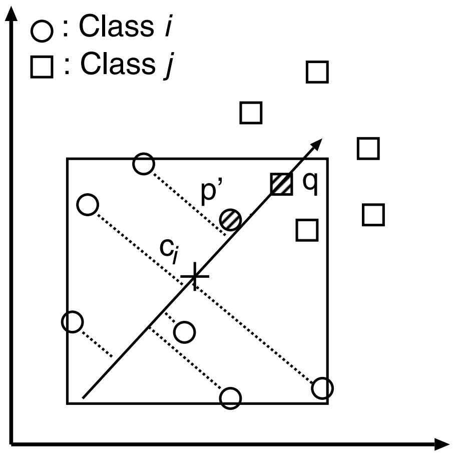


Fig. 10.5. Extracting a boundary datum. From [245, p. 674]

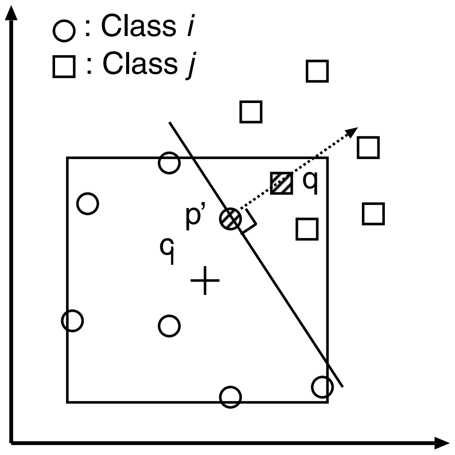


Fig. 10.6. Class i data exist on the negative side of the hyperplane. From [245, p. 674]

Then, as shown in Figs. 10.7 and 10.8, if datum $\mathbf{q}$ and class i data are outside of facet h1, we rotate the orthogonal vector of the facet from $\mathbf{q} - \mathbf{c}_i$ to $\mathbf{q} - \mathbf{p}'$ until the facet touches the class i data (the facet h2 in each figure). By this method, however, two convex hulls may overlap as shown in Fig. 10.8, but we do not consider this case here.

According to this procedure, if the data of other classes are in the convex hull, the convex hull is cut by a facet. Then in the case shown in Fig. 10.9, the class i convex hull is not cut by a facet. In this case, although the data in

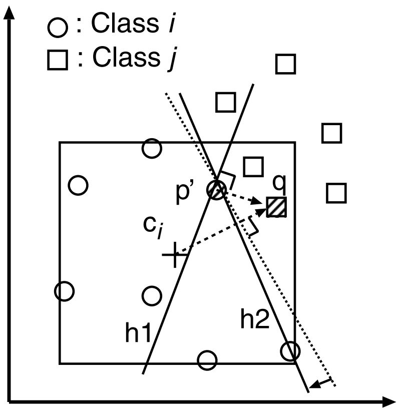


Fig. 10.7. Class i data exist on both sides of the hyperplane. From [245, p. 674]

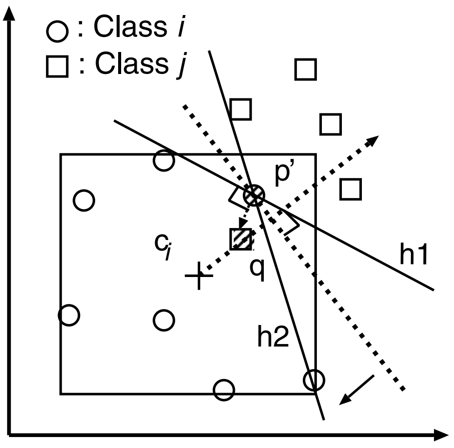


Fig. 10.8. Class i data exist on the positive side of the hyperplane. From [245, p. 674]

the shaded region are nearer to class j than to class i, they are classified into class i. To avoid this we use data that are near the convex hull (initially the hyperbox). Namely, as shown in Fig. 10.10, we expand the convex hull and use the data in the expanded convex hull.

In the following, we discuss the detailed procedure of convex hull generation, definition of membership functions, and tuning of membership functions.

Generation of Hyperbox

Let the set of m-dimensional data for class i be X_i. Then for each input variable we calculate the minimum and maximum values:

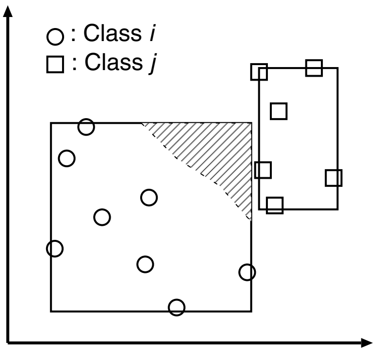


Fig. 10.9. Degradation of generalization ability. From [245, p. 674]

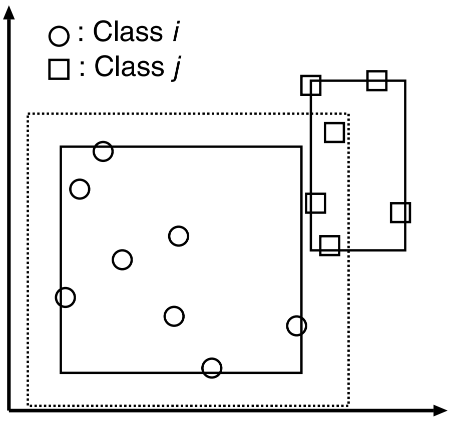


Fig. 10.10. Generalization improvement by expansion of a polyhedron. From [245, p. 675]

$$V_{ij} = \max_{\mathbf{p} \in X_i} p_j \quad \text{for} \quad j = 1, \ldots, m, \tag{10.30}$$

$$v_{ij} = \min_{\mathbf{p} \in X_i} p_j \quad \text{for} \quad j = 1, \ldots, m, \tag{10.31}$$

and we generate a hyperbox.

Generation of Polyhedral Regions

In the following we discuss in detail how to cut the hyperbox for class i by facets:

1. We define the center of class i by the average of class i data:

$$\mathbf{c}_i = \frac{1}{|X_i|} \sum_{\mathbf{p} \in X_i} \mathbf{p}. \tag{10.32}$$

2. As shown in Fig. 10.11, we expand the convex hull by $s\,d$ where d is the distance from the center to a facet and s is an expansion parameter. We call this the *expanded convex hull.* Then we read a datum not belonging to class i, $\mathbf{q}\,(\mathbf{q} \in X_k, k \neq i)$, sequentially and check if it is in the expanded convex hull. The expansion parameter s specifies how much the convex hull is expanded, and for $s = 0$, we use only the data in the convex hull. As s increases, the number of facets increases.

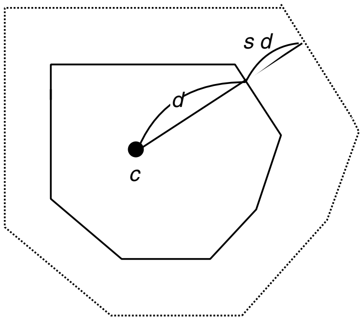


Fig. 10.11. An expanded polyhedron. From [245, p. 675]

We first check if the datum is in the expanded hyperbox. If it is, we further check if it is in the convex hull. Namely, if for $q_j\ (j = 1, \ldots, m)$

$$v_{ij} - d'_j\, s < q_j < V_{ij} + d''_j\, s \tag{10.33}$$

is satisfied, $\mathbf{q}$ is in the hyperbox, where d'_j, d''_j are the one-dimensional distances given by

$$d'_j = c_j - v_{ij}, \tag{10.34}$$

$$d''_j = V_{ij} - c_j, \tag{10.35}$$

respectively.

If $\mathbf{q}$ is in the hyperbox, we further check if it is in the convex hull. Let the facets that form the convex hull for class i be $F_{ij} (j = 1, \ldots, f)$, where f is the number of facets. Let F_{ij} go through $\mathbf{p}^j$ and its outer orthogonal vector be $\mathbf{n}_{ij}$. As shown in Fig. 10.12, we project $\mathbf{q} - \mathbf{p}^j$ and $\mathbf{p}^j - \mathbf{c}_i$ in the direction of $\mathbf{n}_{ij}$. If

$$\mathbf{n}_{ij}^T\,(\mathbf{q} - \mathbf{p}^j) \leq s\,\mathbf{n}_{ij}^T(\mathbf{p}^j - \mathbf{c}_i) \qquad \text{for} \quad j = 1, \ldots, f$$

is satisfied, $\mathbf{q}$ is in the convex hull. If the point is in the convex hull, go to Step 3. Otherwise, read the next datum. If all the data have been read, we finish training.

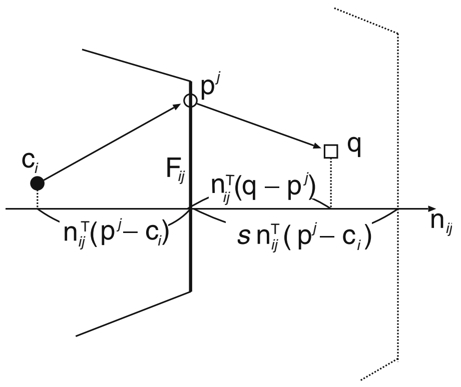


Fig. 10.12. Check if a datum is in the expanded polyhedron. From [245, p. 675]

3. We find the class i datum $\mathbf{p}'$, which is farthest in the direction from the class i center $\mathbf{c}_i$ to the datum $\mathbf{q}$:

$$\mathbf{p}' = \arg\max_{\mathbf{p}\in X_i} (\mathbf{q} - \mathbf{c}_i)^T (\mathbf{p} - \mathbf{c}_i). \tag{10.36}$$

   Go to Step 4.
4. We generate a facet that goes through $\mathbf{p}'$, which includes class i data on the facet or in the convex hull, and from which $\mathbf{q}$ is the farthest in the outer direction of the facet. Here, the orthogonal vector of the facet, which goes through $\mathbf{p}'$ and in which $\mathbf{q}$ is the farthest in the outer direction of the facet, is $\mathbf{q} - \mathbf{p}'$.

   If we generate a facet with $\mathbf{q} - \mathbf{p}'$ as the orthogonal vector, there may be cases where class i data are outside of the facet. To avoid this, we check if the data are outside of the facet with $\mathbf{q} - \mathbf{p}'$ as the orthogonal vector. If there are no data (see Fig. 10.6), we generate the facet and go back to Step 2. If there are (see Fig. 10.7), go to Step 5. Here, for $\mathbf{p} \in X_i$, if

$$(\mathbf{q} - \mathbf{p}')^T (\mathbf{p}' - \mathbf{p}) \le 0 \tag{10.37}$$

   is satisfied, class i datum $\mathbf{p}$ is within the facet, and if

$$(\mathbf{q} - \mathbf{p}')^T (\mathbf{p}' - \mathbf{p}) > 0 \tag{10.38}$$

   is satisfied, it is outside of the facet.
5. We rotate the orthogonal vector of the facet from $\mathbf{q} - \mathbf{c}_i$ to $\mathbf{q} - \mathbf{p}'$ and stop rotation at the point where class i data are included on the facet. As shown in Fig. 10.13 (a), for the orthogonal vector $\mathbf{q} - \mathbf{p}'$, let there be o data outside of the facet, and let $\mathbf{p}''_k (k = 1, 2, \ldots, o)$ and $\mathbf{a}$ and $\mathbf{b}$ be

$$\mathbf{a} = \mathbf{q} - \mathbf{c}_i, \tag{10.39}$$
$$\mathbf{b} = \mathbf{q} - \mathbf{p}'. \tag{10.40}$$

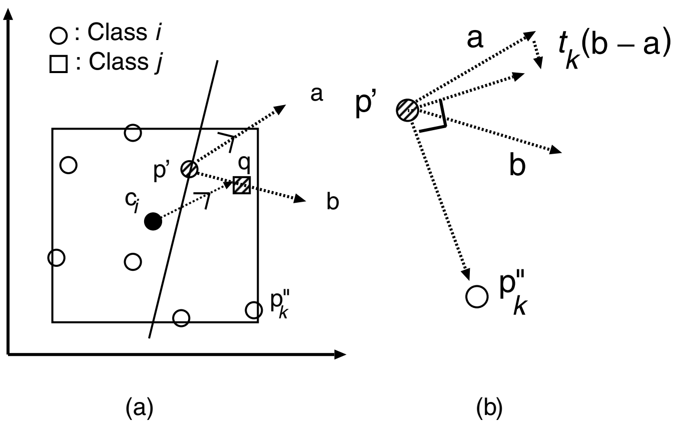


Fig. 10.13. Rotation of a hyperplane. From [245, p. 676]

As shown in Fig. 10.13 (b), let $\mathbf{p}''_k$ be on the facet and $\mathbf{p}' - \mathbf{p}''_k$ be the orthogonal vector. We define the parameter t_k for rotation by

$$(\mathbf{a} + t_k(\mathbf{b} - \mathbf{a}))^T(\mathbf{p}' - \mathbf{p}''_k) = 0. \tag{10.41}$$

We denote the minimum t_k by t:

$$t = \min_{k=1,\ldots,o} t_k. \tag{10.42}$$

To determine the facet with a minimum rotation, we determine the orthogonal vector $\mathbf{n}$ by

$$\mathbf{n} = \mathbf{a} + t(\mathbf{b} - \mathbf{a}). \tag{10.43}$$

Next, we generate the facet that goes through $\mathbf{p}'$ with the orthogonal vector $\mathbf{n}$ and return to Step 2.

Membership Functions

We define a membership function in which the degree of membership is 1 at the center of the convex hull and it decreases as the location moves away from the center.

We define a one-dimensional membership function, $m_i(\mathbf{p})$, of datum $\mathbf{p}$ for the facet F_i by

$$m_i(\mathbf{p}) = \exp\left(h_i(\mathbf{p})\right) = \exp\left(-\frac{d_i(\mathbf{p})}{\alpha_c}\right), \tag{10.44}$$

where $h_i(\mathbf{p})$ is a tuning distance, α_c is a tuning parameter for class c, and $d_i(\mathbf{p})$ is given by

$$d_i(\mathbf{p}) = \begin{cases} \dfrac{\mathbf{a}_i^T(\mathbf{p}-\mathbf{c})}{|\mathbf{a}_i| w_i} & \mathbf{a}_i^T(\mathbf{p}-\mathbf{c}) \geq 0, \\ 0 & \mathbf{a}_i^T(\mathbf{p}-\mathbf{c}) < 0, \end{cases} \tag{10.45}$$

where $\mathbf{a}_i$ is the outer orthogonal vector for the facet and w_i is the distance from the center to the facet F_i.

Then using (10.44), the membership function of $\mathbf{p}$ for the convex hull, $m(\mathbf{p})$, is given by

$$m(\mathbf{p}) = \min_i m_i(\mathbf{p}) = \exp\left(-\max_i h_i(\mathbf{p})\right). \tag{10.46}$$

Tuning of Membership Functions

We can improve the recognition performance by tuning membership functions. In the following, we explain the idea [3].

Now suppose we tune the tuning parameter α_i. Up to some value we can increase or decrease α_i without making the correctly classified datum $\mathbf{x}$ be misclassified. Thus we can calculate the upper bound $U_i(\mathbf{x})$ or lower bound $L_i(\mathbf{x})$ of α_i that causes no misclassification of $\mathbf{x}$. Now let $U_i(1)$ and $L_i(1)$ denote the upper and lower bounds that do not make the correctly classified data be misclassified, respectively. Likewise, $U_i(l)$ and $L_i(l)$ denote the upper and lower bounds in which $l-1$ correctly classified data are misclassified, respectively.

Similarly, if we increase or decrease α_i, misclassified data may be correctly classified. Let $\beta_i(l)$ denote the upper bound of α_i that is smaller than $U_i(l)$ and that resolves misclassification. Let $\gamma_i(l)$ denote the lower bound of α_i that is larger than $L_i(l)$ and that resolves misclassification.

Then the next task is to find which interval among $(L_i(l), \gamma_i(l))$ and $(\beta_i(l), U_i(l))$ $(l = 1, \ldots)$ gives the maximum recognition rate. To limit the search space, we introduce the maximum l, i.e., l_M. Let $(L_i(l), \gamma_i(l))$ be the interval that gives the maximum recognition rate of the training data among $(L_i(k), \gamma_i(k))$ and $(\beta_i(k), U_i(k))$ for $k = 1, \ldots, l_M$. Then even if we set any value in the interval to α_i, the recognition rate of the training data does not change but the recognition rate of the test data may change. To control the generalization ability, we set α_i as follows:

$$\alpha_i = \beta_i(l) + \delta(U_i(l) - \beta_i(l)) \tag{10.47}$$

for $(\beta_i(l), U_i(l))$, where δ satisfies $0 < \delta < 1$ and

$$\alpha_i = \gamma_i(l) - \delta(\gamma_i(l) - L_i(l)) \tag{10.48}$$

for $(L_i(l), \gamma_i(l))$.

In the following performance evaluation, we set $l_M = \infty$. The value of δ does not affect the recognition rate of the test data very much, and we set $\delta = 0.1$.

10.2.2 Performance Evaluation

Using the numeral data, thyroid data, blood cell data, hiragana-13 data, hiragana-50 data, and hiragan-105 data listed in Table 1.1, we compared the performance of the fuzzy classifier with polyhedral regions (FCPR) discussed in this section with that of the conventional fuzzy classifier with polyhedral regions based on the dynamic convex hull generation method (C-FCPR) [224]. The performance of the FCPR was measured with $s = 0$ and the best recognition rate of the test data obtained by changing s. We used a Pentium III (1GHz) personal computer.

If the number of inputs is large, the C-FCPR takes a long time generating polyhedral regions. Thus, in this simulation, we bounded the number of facets to be generated by 5000.

Table 10.6 shows the results for the FCPR and C-FCPR. For comparison, we also include the results of the fuzzy classifier with ellipsoidal regions (FCER), and the one-against-all support vector machine (SVM). In the table, the "Parm" column shows the parameter value and for example, $s5$ means that 5 is set to s and $d2$ and $\gamma 0.5$ mean that polynomial kernels with degree 2 and RBF kernels with $\gamma = 0.5$ are used for the support vector machine, respectively. "Initial" means the recognition rate before tuning membership functions, and "Final" means the recognition rates after tuning membership functions. If the recognition rates of the training data were not 100 percent, they are shown in parentheses.

The training time of the FCPR with $s = 0$ is much shorter than that of C-FCPR, and the recognition rates of the test data are better. For the FCPR with nonzero s, training was slowed, but the recognition rates of the test data were improved and comparable with those of the fuzzy classifier with ellipsoidal regions and the one-against-all support vector machine.

In evaluating the thyroid data, for the conventional FCPR, the number of variables was decreased from 21 to 5 so that the convex hulls could be generated. For the FCPR, it was not necessary to reduce the number.

Figure 10.14 shows the recognition rates of the hiragana-50 data for the change of s. For $0 \leq s < 2$, the recognition rate of the test data increases, but for s larger than 2, the recognition rate decreases.

Summary

In this section, we discussed a fuzzy classifier with polyhedral regions (FCPR). Initially, we generate the hyperbox that includes training data belonging to a class. Then we generate the convex hull that approximates the class region, cutting the hyperbox by hyperplanes so that class separability is maximized.

For all the data sets used in the study, training by the FCPR with $s = 0$ was very fast but the recognition rates of the test data were lower for some data sets than those by other methods. Thus we need to set positive s, which controls the expansion of convex hulls. From the simulation results (only the

Table 10.6. Comparison of recognition rates

Data	Method	Parm	Initial (%)	Final (%)	Time (s)
Numeral	FCPR	$s5$	99.63	99.63	0.10
	FCPR	$s0$	99.51	99.51	0.10
	C-FCPR		99.39	99.39	251
	FCER		99.63 (99.63)	99.39 (99.88)	0.14
	SVM	$d2$	—	99.88	1.93
Thyroid	FCPR	$s0.004$	99.21 (99.92)	99.18 (99.97)	6
	FCPR	$s0$	99.18 (99.92)	99.18 (99.97)	5
	C-FCPR		98.16 (98.25)	98.22 (99.81)	72
	FCER		86.41 (86.77)	97.29 (99.02)	9
	SVM	$d2$	—	98.02 (99.34)	7
Blood cell	FCPR	$s2$	88.29 (94.51)	90.23 (96.45)	101
	FCPR	$s0$	88.81 (94.12)	89.29 (95.83)	15
	C-FCPR		82.13 (89.09)	86.39 (93.51)	1514
	FCER		87.45 (92.64)	92.03 (96.29)	3
	SVM	$\gamma0.5$	—	92.87(99.23)	10
Hiragana-13	FCPR	$s5$	98.56 (99.51)	98.92 (99.84)	4380
	FCPR	$s0$	97.34 (98.97)	97.57 (99.31)	17
	C-FCPR		94.78 (99.45)	97.74 (99.94)	8161
	FCER		99.41 (99.84)	99.59 (99.96)	9
	SVM	$\gamma1$	—	99.77	131
Hiragana-50	FCPR	$s2$	96.07(99.87)	96.23(99.98)	840
	FCPR	$s0$	90.69 (99.50)	90.78 (99.89)	11
	FCER		98.83 (99.87)	98.85	107
	SVM	$d2$	—	98.89	238
Hiragana-105	FCPR	$s0.6$	99.90	99.90	203
	FCPR	$s0$	97.82	97.82	25
	FCER		99.94 (99.92)	100	512
	SVM	$d2$	—	100	836

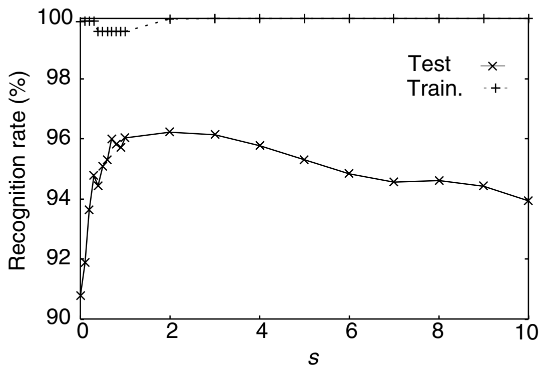


Fig. 10.14. Recognition rates of the hiragana-50 data for the different values of *s*. From [245, p. 679]

result of the hiragana-50 data was included in this book), the optimal value of *s* was around the point where the recognition rate of the training data reached a peak value.

11

Function Approximation

Support vector regressors, which are extensions of support vector machines, have shown good generalization ability for various function approximation and time series prediction problems (e.g., [171, 174]). In this chapter, we discuss extensions of various support vector machines for pattern classification to function approximation: L1 and L2 support vector regressors, LP support vector regressors, LS support vector regressors, and so on. Then we show performance comparison of L1 and L2 support vector regressors including the training characteristics [111, 112].

11.1 Optimal Hyperplanes

In function approximation, we determine the input-output relation using the input-output pairs $(\mathbf{x}_i, y_i)\,(i = 1, \ldots, M)$, where $\mathbf{x}_i$ is the ith m-dimensional input vector, y_i is the ith scalar output, and M is the number of training data.

In support vector regression, we map the input space into the high-dimensional feature space, and in the feature space we determine the optimal hyperplane given by

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{g}(\mathbf{x}) + b, \tag{11.1}$$

where $\mathbf{w}$ is the l-dimensional weight vector, $\mathbf{g}(\mathbf{x})$ is the mapping function that maps $\mathbf{x}$ into the l-dimensional feature space, and b is the bias term.

In linear regression usually the square error function shown in Fig. 11.1 (a) is used, where r is a residual and $r = y - f(\mathbf{x})$. Using this error function, however, the large residuals caused by outliers worsen the estimation accuracy significantly. To avoid this, in support vector regressors, the piecewise linear function, shown in Fig. 11.1 (b), which is originally used for robust function approximation, is used:

$$E(r) = \begin{cases} 0 & \text{for } |r| \le \varepsilon, \\ |r| - \varepsilon & \text{otherwise,} \end{cases} \tag{11.2}$$

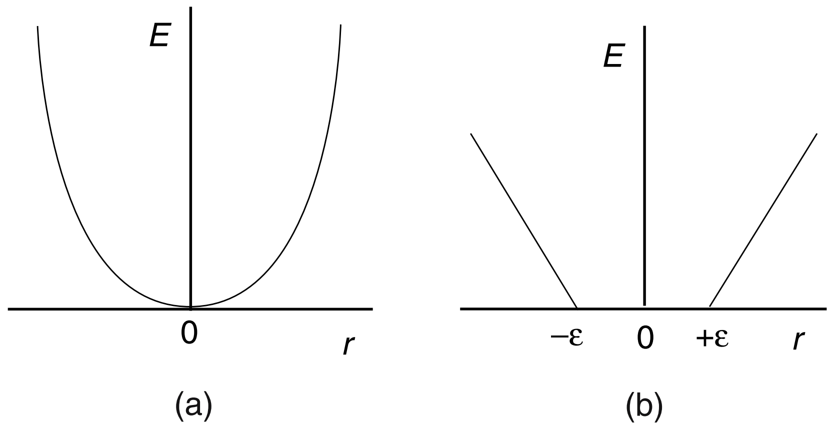


Fig. 11.1. Error functions: (a) A square function; (b) a piecewise linear function

where ε is a small positive value.

Now define the residual of output y and the estimate $f(\mathbf{x})$ by

$$D(\mathbf{x}, y) = y - f(\mathbf{x}). \tag{11.3}$$

According to (11.2), the ideal estimation is realized when all the absolute residuals are within ε, namely

$$-\varepsilon \leq D(\mathbf{x}, y) \leq +\varepsilon, \tag{11.4}$$

which is rewritten as follows:

$$|D(\mathbf{x}, y)| \leq \varepsilon. \tag{11.5}$$

Figure 11.2 illustrates this; in the original input-output space, if all the training data are within the zone with radius ε shown in Fig. 11.2 (a), the ideal estimation is realized. We call this *ε-insensitive zone.*[1] In the feature-input-output space, this tube corresponds to the straight tube shown in Fig. 11.2 (b).

There are infinite possibilities of solutions that satisfy (11.5). Here, we consider obtaining a solution with the maximum generalization ability. Assuming that all the training data satisfy (11.5), the datum that satisfies $D(\mathbf{x}, y) = \pm\varepsilon$ is the farthest from the hyperplane. We call the associated distance the *margin.*

[1] Pérez-Cruz et al. [187] extended a one-dimensional output to a multidimensional one, where data are constrained in the hyperspherical insensitive zone with radius ε.

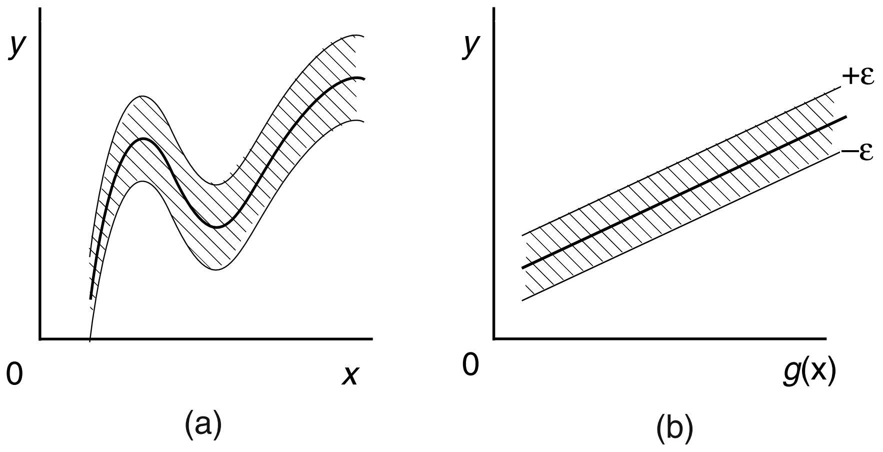


Fig. 11.2. Insensitive zone: (a) In the original input space, and (b) in the feature space

Maximizing the margin leads to maximizing the possibility that the unknown data get into this tube. Thus, it leads to maximizing the generalization ability.

The distance from the hyperplane $D(\mathbf{x}, y) = 0$ to a datum $(\mathbf{x}, y)$ is given by $|D(\mathbf{x}, y)|/\|\mathbf{w}^*\|$, where $\mathbf{w}^*$ is given by

$$\mathbf{w}^* = (1, -\mathbf{w}^T)^T. \tag{11.6}$$

Assuming that the maximum distance of data from the hyperplane is δ, all the training data satisfy

$$\frac{|D(\mathbf{x}, y)|}{\|\mathbf{w}^*\|} \le \delta. \tag{11.7}$$

Namely,

$$|D(\mathbf{x}, y)| \le \delta \|\mathbf{w}^*\|. \tag{11.8}$$

From (11.5) and (11.8), the data that are farthest from the hyperplane satisfy $|D(\mathbf{x}, y)| = \varepsilon$. Thus

$$\delta \|\mathbf{w}^*\| = \varepsilon. \tag{11.9}$$

Therefore, to maximize the margin δ, we need to minimize $\|\mathbf{w}^*\|$. Because $\|\mathbf{w}^*\|^2 = \|\mathbf{w}\|^2 + 1$ holds, minimizing $\|\mathbf{w}\|$ leads to maximizing the margin.

Now the regression problem is solved by minimizing

$$\frac{1}{2}\|\mathbf{w}\|^2 \tag{11.10}$$

subject to the constraints

$$y_i - \mathbf{w}^T \mathbf{g}(\mathbf{x}_i) - b \le \varepsilon \quad \text{for} \quad i = 1, \dots, M, \tag{11.11}$$
$$\mathbf{w}^T \mathbf{g}(\mathbf{x}_i) + b - y_i \le \varepsilon \quad \text{for} \quad i = 1, \dots, M. \tag{11.12}$$

In the formulation, we assumed that all the training data are within the tube with radius ε. To allow the data that are outside of the tube to exist, we introduce the nonnegative slack variables ξ_i and ξ_i^* as shown in Fig. 11.3, where

$$\xi_i = \begin{cases} 0 & \text{for } D(\mathbf{x}_i, y_i) - \varepsilon \le 0, \\ D(\mathbf{x}_i, y_i) - \varepsilon & \text{otherwise,} \end{cases} \tag{11.13}$$
$$\xi_i^* = \begin{cases} 0 & \text{for } \varepsilon - D(\mathbf{x}_i, y_i) \le 0, \\ \varepsilon - D(\mathbf{x}_i, y_i) & \text{otherwise.} \end{cases} \tag{11.14}$$

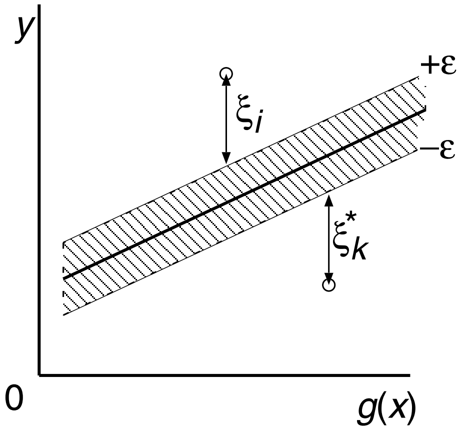


Fig. 11.3. Slack variables

Now the regression problem is solved by minimizing

$$Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{M} (\xi_i^p + \xi_i^{*p}) \tag{11.15}$$

subject to the constraints

$$\begin{aligned} & y_i - \mathbf{w}^T \mathbf{g}(\mathbf{x}_i) - b \le \varepsilon + \xi_i \quad \text{for} \quad i = 1, \dots, M, \\ & \mathbf{w}^T \mathbf{g}(\mathbf{x}_i) + b - y_i \le \varepsilon + \xi_i^* \quad \text{for} \quad i = 1, \dots, M, \\ & \xi_i \ge 0, \quad \xi_i^* \ge 0 \quad \text{for} \quad i = 1, \dots, M, \end{aligned} \tag{11.16}$$

where C is the margin parameter that determines the trade-off between the magnitude of the margin and the estimation error of the training data and p is either 1 or 2. If $p = 1$, we call the support vector regressor the *L1 soft-margin support vector regressor (L1 SVR)* and $p = 2$, the *L2 soft-margin support vector regressor (L2 SVR)*.

The optimization problem given by (11.15) and (11.16) is solved by the quadratic programming technique, by converting (11.15) and (11.16) into the dual problem.

11.2 L1 Soft-Margin Support Vector Regressors

In this section we derive the dual problem of the L1 soft-margin support vector regressors. Introducing the Lagrange multipliers $\alpha_i, \alpha_i^*, \eta_i$, and η_i^* (≥ 0), we convert the original constrained problem into an unconstrained one:

$$\begin{aligned} &Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*, \boldsymbol{\eta}, \boldsymbol{\eta}^*) \\ &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{M}(\xi_i + \xi_i^*) - \sum_{i=1}^{M}\alpha_i\,(\varepsilon + \xi_i - y_i + \mathbf{w}^T\mathbf{g}(\mathbf{x}) + b) \\ &-\sum_{i=1}^{M}\alpha_i^*\,(\varepsilon + \xi_i^* + y_i - \mathbf{w}^T\mathbf{g}(\mathbf{x}) - b) - \sum_{i=1}^{M}(\eta_i\,\xi_i + \eta_i^*\,\xi_i^*). \end{aligned} \tag{11.17}$$

This function has the saddle point that corresponds to the optimal solution for the original problem. At the optimal solution, the partial derivatives of Q with respect to $\mathbf{w}, b, \boldsymbol{\xi}$, and $\boldsymbol{\xi}^*$ vanish. Namely,

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*, \boldsymbol{\eta}, \boldsymbol{\eta}^*)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)\,\mathbf{g}(\mathbf{x}_i) = \mathbf{0}, \tag{11.18}$$

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*, \boldsymbol{\eta}, \boldsymbol{\eta}^*)}{\partial b} = \sum_{i=1}^{M}(\alpha_i^* - \alpha_i) = 0, \tag{11.19}$$

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*, \boldsymbol{\eta}, \boldsymbol{\eta}^*)}{\partial \xi_i} = C - \alpha_i - \eta_i = 0 \quad \text{for} \quad i = 1, \dots, M, \tag{11.20}$$

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*, \boldsymbol{\eta}, \boldsymbol{\eta}^*)}{\partial \xi_i^*} = C - \alpha_i^* - \eta_i^* = 0 \quad \text{for} \quad i = 1, \dots, M. \tag{11.21}$$

From (11.18),

$$\mathbf{w} = \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)\,\mathbf{g}(\mathbf{x}_i). \tag{11.22}$$

Therefore, using α_i and α_i^*, $f(\mathbf{x})$ is expressed by

$$f(\mathbf{x}) = \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)\,\mathbf{g}^T(\mathbf{x}_i)\,\mathbf{g}(\mathbf{x}) + b. \tag{11.23}$$

Substituting (11.18) to (11.21) into (11.17), we obtain the following dual problem. Maximize

$$Q(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = -\frac{1}{2} \sum_{i,j=1}^{M} (\alpha_i - \alpha_i^*)\,(\alpha_j - \alpha_j^*)\,\mathbf{g}^T(\mathbf{x}_i)\,\mathbf{g}(\mathbf{x}_j) \\ -\varepsilon \sum_{i=1}^{M} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{M} y_i\,(\alpha_i - \alpha_i^*) \tag{11.24}$$

subject to the constraints:

$$\sum_{i=1}^{M} (\alpha_i - \alpha_i^*) = 0, \tag{11.25}$$

$$0 \le \alpha_i \le C, \quad 0 \le \alpha_i^* \le C \quad \text{for} \quad i = 1, \ldots, M. \tag{11.26}$$

The optimal solution must satisfy the following KKT complementarity conditions:

$$\alpha_i\,(\varepsilon + \xi_i - y_i + \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) + b) = 0 \quad \text{for} \quad i = 1, \ldots, M, \tag{11.27}$$

$$\alpha_i^*\,(\varepsilon + \xi_i^* + y_i - \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) - b) = 0 \quad \text{for} \quad i = 1, \ldots, M, \tag{11.28}$$

$$\eta_i\,\xi_i = (C - \alpha_i)\,\xi_i = 0 \quad \text{for} \quad i = 1, \ldots, M, \tag{11.29}$$

$$\eta_i^*\,\xi_i^* = (C - \alpha_i^*)\,\xi_i^* = 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{11.30}$$

From (11.29), when $0 < \alpha_i < C$, $\xi_i = 0$ holds. Likewise, from (11.30), when $0 < \alpha_i^* < C$, $\xi_i^* = 0$ holds. When either of them is satisfied, in (11.27) or (11.28) the equation in the parentheses vanishes. Namely, either of the following equation holds:

$$\varepsilon - y_i + \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) + b = 0 \quad \text{for} \quad 0 < \alpha_i < C, \tag{11.31}$$

$$\varepsilon + y_i - \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) - b = 0 \quad \text{for} \quad 0 < \alpha_i^* < C. \tag{11.32}$$

This means that for the datum with the residual $y - f(\mathbf{x}) = +\varepsilon$, α_i satisfies $0 < \alpha_i < C$, and the datum with the residual $y - f(\mathbf{x}) = -\varepsilon$, $0 < \alpha_i^* < C$. Thus, b satisfies

$$b = y_i - \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) - \varepsilon \quad \text{for} \quad 0 < \alpha_i < C, \tag{11.33}$$

$$b = y_i - \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) + \varepsilon \quad \text{for} \quad 0 < \alpha_i^* < C. \tag{11.34}$$

In calculating b, to avoid calculation errors, we average bs that satisfy (11.33) and (11.34).

From (11.29) and (11.30), if either ξ_i or ξ_i^* is not zero, namely, the datum is outside of the tube of radius ε, α_i equals C when the datum is above the tube and α_i^* equals C when the datum is under the tube.

From (11.27) and (11.28), when data satisfy $|y - f(\mathbf{x})| < \varepsilon$, both α_i and α_i^* are zero, and these data do not contribute in constructing the function given by (11.23). Conversely, for the data that satisfy $|y - f(\mathbf{x})| \geq \varepsilon$, α_i and α_i^* are not zero, and they contribute in constructing the function. The training data $\mathbf{x}_i$ with $0 < \alpha_i \leq C$ or $0 < \alpha_i^* \leq C$ are called *support vectors*, especially those with $0 < \alpha_i < C$ or $0 < \alpha_i^* < C$, *unbounded support vectors*, and those with $\alpha_i = C$ or $\alpha_i^* = C$, *bounded support vectors.*

Define

$$H(\mathbf{x}, \mathbf{x}') = \mathbf{g}^T(\mathbf{x})\,\mathbf{g}(\mathbf{x}). \tag{11.35}$$

Then $H(\mathbf{x}, \mathbf{x}')$ satisfies Mercer's condition and is called a *kernel.* Using kernels, we need not treat the high-dimensional feature space explicitly.

11.3 L2 Soft-Margin Support Vector Regressors

In this section we derive the dual problem of the L2 soft-margin support vector regressors. Introducing the Lagrange multipliers α_i and α_i^*, we convert the original constrained problem into an unconstrained one:

$$\begin{aligned} & Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*) \\ & = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{2}\sum_{i=1}^{M}(\xi_i^2 + \xi_i^{*2}) - \sum_{i=1}^{M}\alpha_i\,(\varepsilon + \xi_i - y_i + \mathbf{w}^T\mathbf{g}(\mathbf{x}) + b) \\ & -\sum_{i=1}^{M}\alpha_i^*\,(\varepsilon + \xi_i^* + y_i - \mathbf{w}^T\mathbf{g}(\mathbf{x}) - b). \end{aligned} \tag{11.36}$$

Here, unlike the L1 support vector regressor, we do not need to introduce the Lagrange multipliers associated with ξ_i and ξ_i^*.

This function has the saddle point that corresponds to the optimal solution for the original problem. For the optimal solution, the partial derivatives of Q with respect to $\mathbf{w}, b, \boldsymbol{\xi}$, and $\boldsymbol{\xi}^*$ vanish. Namely,

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)\,\mathbf{g}(\mathbf{x}_i) = \mathbf{0}, \tag{11.37}$$

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*)}{\partial b} = \sum_{i=1}^{M}(\alpha_i^* - \alpha_i) = 0, \tag{11.38}$$

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*)}{\partial \xi_i} = C\,\xi_i - \alpha_i = 0 \quad \text{for} \quad i = 1, \dots, M, \tag{11.39}$$

$$\frac{\partial Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*)}{\partial \xi_i^*} = C\,\xi_i^* - \alpha_i^* = 0 \quad \text{for} \quad i = 1, \dots, M. \tag{11.40}$$

From (11.37),

$$\mathbf{w} = \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)\,\mathbf{g}(\mathbf{x}_i). \tag{11.41}$$

Therefore, using α_i and α_i^*, $f(\mathbf{x})$ is expressed by

$$f(\mathbf{x}) = \sum_{i=1}^{M} (\alpha_i - \alpha_i^*)\, H(\mathbf{x}_i, \mathbf{x}) + b, \tag{11.42}$$

where $H(\mathbf{x}_i, \mathbf{x}) = \mathbf{g}^T(\mathbf{x}_i)\, \mathbf{g}(\mathbf{x})$.

Substituting (11.37) to (11.40) into (11.36), we obtain the following dual problem. Maximize

$$\begin{aligned} Q(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = & -\frac{1}{2} \sum_{i,j=1}^{M} (\alpha_i - \alpha_i^*)\, (\alpha_j - \alpha_j^*) \left(H(\mathbf{x}_i, \mathbf{x}_j) + \frac{\delta_{ij}}{C} \right) \\ & -\varepsilon \sum_{i=1}^{M} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{M} y_i\, (\alpha_i - \alpha_i^*) \end{aligned} \tag{11.43}$$

subject to the constraints:

$$\sum_{i=1}^{M} (\alpha_i - \alpha_i^*) = 0, \tag{11.44}$$

$$\alpha_i \geq 0, \quad \alpha_i^* \geq 0 \quad \text{for} \quad i = 1, \ldots, M, \tag{11.45}$$

where δ_{ij} is Kronecker's delta function.

The optimal solution must satisfy the following KKT complementarity conditions:

$$\alpha_i\, (\varepsilon + \xi_i - y_i + \mathbf{w}^T \mathbf{g}(\mathbf{x}_i) + b) = 0 \quad \text{for} \quad i = 1, \ldots, M, \tag{11.46}$$

$$\alpha_i^*\, (\varepsilon + \xi_i^* + y_i - \mathbf{w}^T \mathbf{g}(\mathbf{x}_i) - b) = 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{11.47}$$

Thus, $\alpha_i = 0$, $\alpha_i^* = 0$ or

$$b = y_i - \mathbf{w}^T \mathbf{g}(\mathbf{x}_i) - \varepsilon - \frac{\alpha_i}{C} \quad \text{for} \quad \alpha_i > 0, \tag{11.48}$$

$$b = y_i - \mathbf{w}^T \mathbf{g}(\mathbf{x}_i) + \varepsilon + \frac{\alpha_i}{C} \quad \text{for} \quad \alpha_i^* > 0. \tag{11.49}$$

From (11.46) and (11.47), when data satisfy $|y - f(\mathbf{x})| < \varepsilon$, both α_i and α_i^* are zero, and these data do not contribute in constructing the function given by (11.42). The data $\mathbf{x}_i$ with nonzero α_i are called *support vectors.*

Comparing L1 and L2 support vector regressors, the latter does not have bounded support vectors and the Hessian matrix is positive definite.

Model selection, i.e., selection of optimal parameters: ε, C, and γ for RBF kernels, is a difficult task for support vector regressors. Several methods have been proposed. Ito and Nakano [118] proposed determining these parameters by alternating training of a support vector regressor (determination of α_i and α_i^*) and optimizing the parameters by steepest descent for the cross-validation data. This method works for L2 support vector regressors but not for L1 support vector regressors due to the nonsmooth output for the parameter ε.

11.4 Training Speedup

Support vector regressors have all the advantages and disadvantages that support vector machines have. Because a support vector regressor is expressed by a quadratic optimization problem, the solution is globally optimal. However, because we usually use nonlinear kernels, we need to solve the dual optimization problem whose number of variables is twice the number of training data. Therefore, if the number of training data is very large, training becomes difficult. To solve this problem, as discussed in Section 5.2, we can use the decomposition technique. Then, in selecting a working set, should we select α_i and α_i^* simultaneously? Liao, Lin, and Lin [149] showed that there is not much difference in convergence in selecting both or either.

Usually, we combine the primal-dual interior-point method with the decomposition technique. But, similar to support vector machines, many methods that do not use QP solvers have been developed. Mattera, Palmieri, and Haykin [163] redefined variables α_i and α_i^* in (11.24) to (11.26) by $u_i = \alpha_i^* - \alpha_i$ and $|u_i| = \alpha_i^* + \alpha_i$ and proposed solving the quadratic programming problem by optimizing two variables u_i and u_j at a time. According to the constraint given by (11.25), this results in optimizing one variable at a time. In each training cycle, all the $M(M-1)/2$ pairs of variables are selected and optimized, and when all the changes in the variables are within the specified limit, training is terminated. This is similar to the sequential minimal optimization (SMO) algorithm for pattern classification [190]. According to the simulations for the Lorenz chaotic process, the method was faster for a small value of C ($C = 0.1$) than the QP solver. But it slowed down for large C ($C = 10$, 100).

Vogt [262] extended SMO for function approximation when the bias term is not present. Veropoulos [259, pp. 104–6] extended the kernel Adatron to function approximation. Kecman, Vogt, and Huang [130] proved that these algorithms are equivalent.

De Freitas, Milo, Clarkson, Niranjan, and Gee [72] used the Kalman filtering technique for sequential training of support vector regressors.

Anguita, Boni, and Pace [17] proposed speeding up training and reduce memory storage when the training inputs are positioned on a grid using the fact that the matrix associated with the kernel is expressed by the Toeplitz block matrix [96]. They showed the effectiveness of the method for an image interpolation problem.

In the following we extend SMO for support vector regressors, increasing the working set size from two and optimizing the variables in the working set by the steepest ascent method [111, 112]. Calculations of corrections of variables in the working set include inversion of the associated Hessian matrix. But because the Hessian matrix is not guaranteed to be positive definite, we calculate the corrections only for the linearly independent variables in the working set.

11.5 Steepest Ascent Methods

We change the notations: $\alpha_i^* = \alpha_{M+i}$ and $\xi_i^* = \xi_{M+i}$. Then the optimization problem for the L1 support vector regressor given by (11.24) to (11.26) becomes as follows. Maximize

$$Q(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{i,j=1}^{M}(\alpha_i - \alpha_{M+i})(\alpha_j - \alpha_{M+j})\,H(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{M} y_i(\alpha_i - \alpha_{M+i}) - \varepsilon \sum_{i=1}^{M}(\alpha_i + \alpha_{M+i}) \tag{11.50}$$

subject to

$$\sum_{i=1}^{M}(\alpha_i - \alpha_{M+i}) = 0, \tag{11.51}$$

$$0 \le \alpha_i \le C \quad \text{for} \quad i = 1, \ldots, M. \tag{11.52}$$

The optimization problem for the L2 support vector regressor given by (11.43) and (11.45) becomes as follows. Maximize

$$Q(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{i,j=1}^{M}(\alpha_i - \alpha_{M+i})(\alpha_j - \alpha_{M+j})\left(H(\mathbf{x}_i, \mathbf{x}_j) + \frac{\delta_{ij}}{C}\right) + \sum_{i=1}^{M} y_i(\alpha_i - \alpha_{M+i}) - \varepsilon \sum_{i=1}^{M}(\alpha_i + \alpha_{M+i}) \tag{11.53}$$

subject to

$$\sum_{i=1}^{M}(\alpha_i - \alpha_{M+i}) = 0, \tag{11.54}$$

$$\alpha_i \ge 0 \quad \text{for} \quad i = 1, \ldots, M. \tag{11.55}$$

SMO solves two-variable subproblems without using a QP solver. In this section we solve subproblems with more than two variables by the steepest ascent method.

Now we define a candidate set V as the index set of variables that are candidates of support vectors and a working set W as the index set of variables in V.

The rough flow of the training procedure of the support vector regressor is as follows:

1. Add all the indexes that are associated with the training data to V.
2. Select indices from V randomly and set them to W. Selected indices are removed from V.

3. Calculate corrections of the variables in the working set by the steepest ascent method so that the objective function is maximized.
4. If a convergence condition is satisfied, finish training. Otherwise, if V is empty, add new candidates that violate the KKT complementarity conditions. Return to Step 2.

11.5.1 Subproblem Optimization

In this subsection we explain Step 3 in more detail.

Let $\boldsymbol{\alpha}_W$ be vectors whose elements are α_i $(i \in W)$. From (11.51), $\alpha_s \in \boldsymbol{\alpha}_W$ is expressed as follows:

$$\alpha_s = -\sum_{i \neq s, i=1}^{M} \alpha_i + \sum_{i=M+1}^{2M} \alpha_i \quad \text{if} \quad s \leq M, \tag{11.56}$$

$$\alpha_s = \sum_{i=1}^{M} \alpha_i - \sum_{i \neq s, i=M+1}^{2M} \alpha_i \quad \text{if} \quad s > M. \tag{11.57}$$

Substituting (11.56) or (11.57) into the objective function, we eliminate constraint (11.51) from the dual problem. Here W' is defined as the set in which s is removed from W, namely $W' = W - \{s\}$.

Because the objective function is quadratic, the change of the objective function, $\Delta Q(\boldsymbol{\alpha}_{W'})$, for the change of variables, $\Delta\boldsymbol{\alpha}_{W'}$, is given by

$$\Delta Q(\boldsymbol{\alpha}_{W'}) = \frac{\partial Q(\boldsymbol{\alpha}_{W'})}{\partial \boldsymbol{\alpha}_{W'}} \Delta\boldsymbol{\alpha}_{W'} + \frac{1}{2} \Delta\boldsymbol{\alpha}_{W'}^T \frac{\partial^2 Q(\boldsymbol{\alpha}_{W'})}{\partial \boldsymbol{\alpha}_{W'}^2} \Delta\boldsymbol{\alpha}_{W'}. \tag{11.58}$$

If $\partial^2 Q(\boldsymbol{\alpha}_{W'})/\partial \boldsymbol{\alpha}_{W'}^2$ is positive definite, we can calculate corrections by the following formula so that $\Delta Q(\boldsymbol{\alpha}_{W'})$ is maximized:

$$\Delta\boldsymbol{\alpha}_{W'} = -\left(\frac{\partial^2 Q(\boldsymbol{\alpha}_{W'})}{\partial \boldsymbol{\alpha}_{W'}^2}\right)^{-1} \frac{\partial Q(\boldsymbol{\alpha}_{W'})}{\partial \boldsymbol{\alpha}_{W'}}. \tag{11.59}$$

For the L1 support vector regressor, from (11.50), (11.56), and (11.57), the element of $\partial Q(\boldsymbol{\alpha}_{W'})/\partial \boldsymbol{\alpha}_{W'}$ is

$$\begin{aligned}\frac{\partial Q(\boldsymbol{\alpha}_{W'})}{\partial \alpha_i} &= p_i \left\{ y_{i^*} - y_{s^*} - \varepsilon\,(p_i + q) \right. \\ &\quad \left. - \sum_{j=1}^{M} (\alpha_j - \alpha_{M+j})\,(H_{ij} - H_{sj}) \right\} \quad \text{for} \quad i, s \in \{1, \ldots, 2M\},\end{aligned} \tag{11.60}$$

where $H_{ij} = H(\mathbf{x}_{i^*}, \mathbf{x}_{j^*})$ and i^* (s^*), p_i, and q are defined as

$$i^* = \begin{cases} i & \text{for} \quad i \leq M, \\ i - M & \text{for} \quad i > M, \end{cases} \tag{11.61}$$

$$p_i = \begin{cases} +1 & \text{for} \quad i \le M, \\ -1 & \text{for} \quad i > M, \end{cases} \tag{11.62}$$

$$q = \begin{cases} -1 & \text{for} \quad s \le M, \\ +1 & \text{for} \quad s > M. \end{cases} \tag{11.63}$$

The element of $\partial^2 Q(\boldsymbol{\alpha}_{W'})/\partial \boldsymbol{\alpha}_{W'}^2$ is

$$\frac{\partial^2 Q(\boldsymbol{\alpha}_{W'})}{\partial \alpha_i \partial \alpha_j} = -p_i\, p_j\, (H_{ij} + H_{ss} - H_{is} - H_{js}) \quad \text{for} \quad i,\, j, s \in \{1, \ldots, 2M\}. \tag{11.64}$$

For the L2 support vector regressor, the element of $\partial Q(\boldsymbol{\alpha}_{W'})/\partial \boldsymbol{\alpha}_{W'}$ is

$$\frac{\partial Q(\boldsymbol{\alpha}_{W'})}{\partial \alpha_i} = p_i \left\{ y_{i^*} - y_{s^*} - \varepsilon\, (p_i + q) - \sum_{j=1}^{M} (\alpha_j - \alpha_{M+j})\, (H_{ij} - H_{sj}) - \frac{1}{C}\, (\alpha_{i^*} - \alpha_{M+i^*} - \alpha_{s^*} + \alpha_{M+s^*}) \right\} \quad \text{for} \quad i, s \in \{1, \ldots, 2M\} \tag{11.65}$$

and the element of $\partial^2 Q(\boldsymbol{\alpha}_{W'})/\partial \boldsymbol{\alpha}_{W'}^2$ is

$$\frac{\partial^2 Q(\boldsymbol{\alpha}_{W'})}{\partial \alpha_i \partial \alpha_j} = -p_i\, p_j\, (H_{ij} + H_{ss} - H_{is} - H_{js}) - \frac{1}{C}\, (\delta_{ij} + p_i\, p_j) \quad \text{for} \quad i,\, j, s \in \{1, \ldots, 2M\}. \tag{11.66}$$

The solution given by (11.59) is obtained by solving a set of simultaneous equations.

To speed up solving (11.59), we decompose $\partial^2 Q(\boldsymbol{\alpha}_{W'})/\partial \boldsymbol{\alpha}_{W'}^2$ into the upper and lower triangular matrices by the Cholesky factorization. If the Hessian matrix is not positive definite, the Cholesky factorization stops because the input of the square root becomes nonpositive. When this happens, we solve (11.59) only for the variables that are decomposed so far. And we delete the associated variable and the variables that are not decomposed from the working set. For the L2 support vector regressor, because the Hessian matrix is positive definite, the procedure is not necessary.

Now we can calculate the correction of α_s. In the case of $s \le M$,

$$\Delta\alpha_s = -\sum_{i \in W',\, i \le M} \Delta\alpha_i + \sum_{i \in W',\, i > M} \Delta\alpha_i, \tag{11.67}$$

or in the case of $s > M$,

$$\Delta\alpha_s = \sum_{i \in W',\, i \le M} \Delta\alpha_i - \sum_{i \in W',\, i > M} \Delta\alpha_i. \tag{11.68}$$

Now confine our consideration to the L1 support vector regressor. Although the solution calculated by the described procedure satisfies (11.51), it may not satisfy (11.52). Namely, when there are variables that cannot be corrected,

$$\begin{aligned} \Delta\alpha_i < 0 \quad &\text{when} \quad \alpha_i = 0, \\ \Delta\alpha_i > 0 \quad &\text{when} \quad \alpha_i = C, \end{aligned}$$

we remove these variables from the working set and again solve (11.59) for the reduced working set. This does not require much time because recalculation of the kernels for the reduced working set are not necessary by caching them.

Suppose the solution that can make some corrections for all the variables in the working set are obtained. Then the corrections are adjusted so that all the updated variables go into the range $[0, C]$. Let $\Delta\alpha'_i$ be the allowable corrections if each variable is corrected separately. Then

$$\Delta\alpha'_i = \begin{cases} C - \alpha_i^{\text{old}} & \text{if} \quad \alpha_i^{\text{old}} + \Delta\alpha_i > C, \\ -\alpha_i^{\text{old}} & \text{if} \quad \alpha_i^{\text{old}} + \Delta\alpha_i < 0, \\ \Delta\alpha_i & \text{otherwise.} \end{cases} \tag{11.69}$$

Using $\Delta\alpha'_i$ we calculate the minimum ratio of corrections:

$$r = \min_{i \in W} \frac{\Delta\alpha'_i}{\Delta\alpha_i}. \tag{11.70}$$

Then the variables are updated by

$$\alpha_i^{\text{new}} = \alpha_i^{\text{old}} + r\Delta\,\alpha_i. \tag{11.71}$$

Clearly the updated variables satisfy (11.52).

For the L2 support vector regressor, we can calculate corrections without considering the upper bound C for α_i.

11.5.2 Convergence Check

After the update of the variables in the working set, we check whether training should be finished. Namely, in Step 4, when V becomes empty, we check if there are variables that violate the KKT complementarity conditions and terminate training if there are no violating variables. But by this method the training may be slow in some cases.

To accelerate training, if an increase of the objective function becomes very small, we consider the solution sufficiently near the optimal solution. Thus we finish training when the following inequality is satisfied for N consecutive iterations:

$$\frac{Q_n - Q_{n-1}}{Q_{n-1}} < \eta, \tag{11.72}$$

where η is a small positive parameter.

11.6 Candidate Set Selection

If all the variables satisfy KKT complementarity conditions the optimal solution is obtained. Thus to speed up training we need to select violating variables as members of the candidate set.

11.6.1 Inexact KKT Conditions

The KKT conditions for the L1 support vector regressor are as follows:

$$\begin{cases} \alpha_i = C, \alpha_{M+i} = 0 & \text{if} \quad y_i - f(\mathbf{x}_i) > \varepsilon, \\ 0 < \alpha_i < C, \alpha_{M+i} = 0 & \text{if} \quad y_i - f(\mathbf{x}_i) = \varepsilon, \\ \alpha_i = 0, \alpha_{M+i} = 0 & \text{if} \quad |y_i - f(\mathbf{x}_i)| \le \varepsilon, \\ \alpha_i = 0, 0 < \alpha_{M+i} < C & \text{if} \quad y_i - f(\mathbf{x}_i) = -\varepsilon, \\ \alpha_i = 0, \alpha_{M+i} = C & \text{if} \quad y_i - f(\mathbf{x}_i) < -\varepsilon. \end{cases} \tag{11.73}$$

And the KKT conditions for the L2 support vector regressor are as follows:

$$\begin{cases} \alpha_i > 0, \alpha_{M+i} = 0 & \text{if} \quad y_i - f(\mathbf{x}_i) = \varepsilon + \dfrac{\alpha_i}{C}, \\ \alpha_i = 0, \alpha_{M+i} = 0 & \text{if} \quad |y_i - f(\mathbf{x}_i)| \le \varepsilon, \\ \alpha_i = 0, \alpha_{M+i} > 0 & \text{if} \quad y_i - f(\mathbf{x}_i) = -\varepsilon - \dfrac{\alpha_{M+i}}{C}. \end{cases} \tag{11.74}$$

In checking the KKT conditions, we need to calculate the value of $f(\mathbf{x}_i)$. But because the bias term b is not included in the dual problem, during training the value is inexact.

For the L1 support vector regressor, variables α_i have the upper and lower bounds and for the L2 support vector regressor variables have the lower bound. If α_i is bounded, the possibility that the variable is modified in the next iteration is small. Thus, we choose the unbounded variables with high priority.

11.6.2 Exact KKT Conditions

The KKT conditions discussed in the previous section are inexact in that b is estimated during training. Keerthi et al. [131, 133] proposed exact KKT conditions and showed by computer simulations that selecting the violating variables led to training speedup. In the following, we first discuss their method for the L1 support vector regressor.

We define F_i by

$$F_i = y_i - \sum_{j=1}^{M} (\alpha_j - \alpha_{M+j}) H_{ij}, \tag{11.75}$$

where $H_{ij} = H(\mathbf{x}_i, \mathbf{x}_j)$.

We can classify KKT conditions into the following five cases:

$$\begin{aligned} &\text{Case 1.} \quad 0 < \alpha_i < C \\ &\qquad F_i - b = \varepsilon, \end{aligned} \tag{11.76}$$

$$\begin{aligned} &\text{Case 2.} \quad 0 < \alpha_{M+i} < C \\ &\qquad F_i - b = -\varepsilon, \end{aligned} \tag{11.77}$$

$$\begin{aligned} &\text{Case 3.} \quad \alpha_i = \alpha_{M+i} = 0 \\ &\qquad -\varepsilon \le F_i - b \le \varepsilon, \end{aligned} \tag{11.78}$$

$$\begin{aligned} &\text{Case 4.} \quad \alpha_{M+i} = C \\ &\qquad F_i - b \le -\varepsilon, \end{aligned} \tag{11.79}$$

$$\begin{aligned} &\text{Case 5.} \quad \alpha_i = C \\ &\qquad F_i - b \ge \varepsilon. \end{aligned} \tag{11.80}$$

Then we define $\tilde{F}_i, \bar{F}_i$ as follows:

$$\tilde{F}_i = \begin{cases} F_i - \varepsilon & \text{if} \quad 0 < \alpha_i < C \quad \text{or} \quad \alpha_i = \alpha_{M+i} = 0, \\ F_i + \varepsilon & \text{if} \quad 0 < \alpha_{M+i} < C \quad \text{or} \quad \alpha_{M+i} = C, \end{cases} \tag{11.81}$$

$$\bar{F}_i = \begin{cases} F_i - \varepsilon & \text{if} \quad 0 < \alpha_i < C \quad \text{or} \quad \alpha_i = C, \\ F_i + \varepsilon & \text{if} \quad 0 < \alpha_{M+i} < C \quad \text{or} \quad \alpha_i = \alpha_{M+i} = 0. \end{cases} \tag{11.82}$$

Then the KKT conditions are simplified as follows:

$$\bar{F}_i \ge b \ge \tilde{F}_i \quad \text{for} \quad i = 1, \ldots, M. \tag{11.83}$$

We must notice that according to the values of α_i and α_{M+i}, $\bar{F}_i$ or $\tilde{F}_i$ is not defined. For instance, if $\alpha_i = C$, $\tilde{F}_i$ is not defined.

To detect the violating variables, we define $b_{\text{low}}, b_{\text{up}}$ as follows:

$$\begin{aligned} b_{\text{low}} &= \max_i \tilde{F}_i, \\ b_{\text{up}} &= \min_i \bar{F}_i. \end{aligned} \tag{11.84}$$

Then if the KKT conditions are not satisfied, $b_{\text{up}} < b_{\text{low}}$ and the datum i that satisfies

$$b_{\text{up}} < \tilde{F}_i - \tau \quad \text{or} \quad b_{\text{low}} > \bar{F}_i + \tau \quad \text{for} \quad i \in \{1, \ldots, M\} \tag{11.85}$$

violates the KKT conditions, where τ is a positive parameter to loosen the KKT conditions. By this, without calculating b, we can detect the variables that violate the KKT conditions. As training proceeds, b_{up} and b_{low} approach each other and at the optimal solution, b_{up} and b_{low} have the same value if the solution is unique. If not, $b_{\text{up}} > b_{\text{low}}$.

For the L2 support vector regressor, we define $\tilde{F}_i$ and $\bar{F}_i$ as follows:

$$\tilde{F}_i = \begin{cases} F_i - \varepsilon & \text{if} \quad \alpha_i = \alpha_{M+i} = 0, \\ F_i - \varepsilon - \dfrac{\alpha_i}{C} & \text{if} \quad \alpha_i > 0, \alpha_{M+i} = 0, \\ F_i + \varepsilon + \dfrac{\alpha_i}{C} & \text{if} \quad \alpha_i = 0, \alpha_{M+i} > 0, \end{cases} \tag{11.86}$$

$$\bar{F}_i = \begin{cases} F_i + \varepsilon & \text{if} \quad \alpha_i = \alpha_{M+i} = 0, \\ F_i - \varepsilon - \dfrac{\alpha_i}{C} & \text{if} \quad \alpha_i > 0, \alpha_{M+i} = 0, \\ F_i + \varepsilon + \dfrac{\alpha_{M+i}}{C} & \text{if} \quad \alpha_i = 0, \alpha_{M+i} > 0. \end{cases} \tag{11.87}$$

The remaining procedure is the same as that of the L1 support vector regressor.

For selection of variables for the working set, it may be desirable to select the most violating variables. In the following, we discuss one such method.

11.6.3 Selection of Violating Variables

The degree of violation is larger as $\tilde{F}_i$ becomes larger and $\bar{F}_i$ smaller. Thus a procedure of candidate set selection for the steepest ascent training is as follows:

1. Sort $\bar{F}_i$ in increasing order and $\tilde{F}_k$ in decreasing order and set $i = 1, k = 1$.
2. Compare the value of $\tilde{F}_i$ with b_{up} and if the KKT conditions are violated, add i to the candidate set V and increment i by 1.
3. Compare the value of $\bar{F}_k$ with b_{low} and if the KKT conditions are violated, add i to the candidate set V and increment k by 1.
4. Iterate Steps 2 and 3, so that the violating data for $\tilde{F}_i$ and $\bar{F}_k$ are selected alternately until there are no violating data for $\tilde{F}_i$ and $\bar{F}_k$.
5. Move indices in V to the set of working set W in decreasing order of violation.

For the primal-dual interior-point method combined with the decomposition technique, if in Step 4 there are no violating data or the working set is full, we stop selecting the variables, and Step 5 is not necessary.

In function approximation, for $\mathbf{x}_i$, there are two variables α_i and α_{M+i}. In steepest ascent training, we select either variable according to the following conditions. If one is zero and the other is nonzero, we select the nonzero variable. If both are zero, we evaluate the error and if $y_i - f(\mathbf{x}_i) > 0$, we select α_i, and otherwise, α_{M+i}. To estimate $f(\mathbf{x}_i)$, we use $b = (b_{\text{up}} + b_{\text{low}})/2$.

11.7 Variants of Support Vector Regressors

Various variants of support vector regressors have been developed. For example, to enhance approximation ability, mixtures of different kernels are used [150, 221, 228]. To enhance interpolation and extrapolation abilities, Smits and Jordaan [228] proposed mixing global kernels such as polynomials and local kernels such as RBF kernels by $\rho H_{poly} + (1-\rho) H_{RBF}$, where $0 \geq \rho \geq 1$.

Inspired by multiresolution signal analysis, such as wavelet analysis, Shao and Cherkassky [221] proposed multiresolution support vector machines. Instead of using one kernel, two kernels with different resolutions are combined.

For multiresolution signal analysis with orthogonal basis functions, components of different resolutions can be computed separately. But because the kernels in support vector regressors are not orthogonal, the support vector training is reformulated to determine the Lagrange multipliers for two types of kernels simultaneously.

Jordaan and Smits [124] detected, as an outlier, the datum that frequently becomes a bounded support vector whose average slack variable value is the largest for several trained support vector regressors with different values of ε.

In the following, we discuss linear programming support vector regressors, ν-support vector regressors, and least squares support vector regressors.

11.7.1 Linear Programming Support Vector Regressors

Similar to LP support vector machines, LP support vector regressors can be defined. First, we define the approximation function in the dual form as follows:

$$D(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i \, H(\mathbf{x}, \mathbf{x}_i) + b, \tag{11.88}$$

where α_i take on real values. Then we consider minimizing

$$Q(\boldsymbol{\alpha}, b, \boldsymbol{\xi}) = \sum_{i=1}^{M} \left(|\alpha_i| + C\,(\xi_i + \xi_i^*)\right) \tag{11.89}$$

subject to

$$-\varepsilon - \xi_j^* \le \sum_{i=1}^{M} \alpha_i \, H(\mathbf{x}_j, \mathbf{x}_i) + b - y_j \le \varepsilon + \xi_j \quad \text{for} \quad j = 1, \ldots, M. \tag{11.90}$$

Letting $\alpha_i = \alpha_i^+ - \alpha_i^-$ and $b = b^+ - b^-$, where α_i^+, α_i^-, b^+, and b^- are nonnegative, we can solve (11.89) and (11.90) for $\boldsymbol{\alpha}$, b, and $\boldsymbol{\xi}$ by linear programming. For large problems, we use decomposition techniques [38, 249].

Kecman, Arthanari, and Hadzic [128, 129] formulated the linear programming support vector regressors differently. Namely, minimize $\sum_{i=1}^{m} |w_i|$ subject to $|\mathbf{g}^T \mathbf{w} - y| \le \varepsilon$. When RBF kernels are used, elements of $\mathbf{g}$ are given by $\exp(-\gamma \, \|\mathbf{x} - \mathbf{x}'\|^2)$, where $\mathbf{x}'$ is a training datum. The advantage of this formulation is that, unlike support vector regressors, we can place multiple basis functions (i.e., basis functions with different values of γ) for each training datum.

11.7.2 ν-Support Vector Regressors

Usually it is difficult to set the optimal value of ε. One approach to overcome this problem is to estimate the value assuming that it is in proportion to the standard deviation of the noise [121, 231]. Another approach is to modify the model so that it can be optimized during training [211].

Schölkopf et al. [211] proposed controlling the accuracy of the support vector regressor introducing parameter ν as follows. Minimize

$$Q(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \varepsilon) = \frac{1}{2}\|\mathbf{w}\|^2 + C\left(\nu\,\varepsilon + \frac{1}{M}\sum_{i=1}^{M}(\xi_i + \xi_i^*)\right) \tag{11.91}$$

subject to the constraints

$$\begin{aligned} y_i - \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) - b &\le \varepsilon + \xi_i \quad && \text{for} \quad i = 1\ldots, M, \\ \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) + b - y_i &\le \varepsilon + \xi_i^* \quad && \text{for} \quad i = 1\ldots, M, \\ \xi_i \ge 0, \quad \xi_i^* &\ge 0 \quad && \text{for} \quad i = 1\ldots, M, \end{aligned} \tag{11.92}$$

where ν is introduced to control the value of ε.

Introducing the Lagrange multipliers $\alpha_i, \alpha_i^*, \eta_i, \eta_i^*$, and $\beta\,(\ge 0)$, we convert the original constrained problem into an unconstrained one:

$$\begin{aligned} &Q(\mathbf{w}, b, \beta, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \varepsilon, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*, \boldsymbol{\eta}, \boldsymbol{\eta}^*) \\ &= \frac{1}{2}\|\mathbf{w}\|^2 + C\,\nu\,\varepsilon + \frac{1}{M}\,C\sum_{i=1}^{M}(\xi_i + \xi_i^*) - \sum_{i=1}^{M}\alpha_i\,(\varepsilon + \xi_i - y_i + \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) + b) \\ &-\sum_{i=1}^{M}\alpha_i^*\,(\varepsilon + \xi_i^* + y_i - \mathbf{w}^T\mathbf{g}(\mathbf{x}_i) - b) - \beta\,\varepsilon - \sum_{i=1}^{M}(\eta_i\,\xi_i + \eta_i^*\,\xi_i^*). \end{aligned} \tag{11.93}$$

Setting the derivatives of (11.93) with respect to the primal variables to zero, we obtain

$$\mathbf{w} = \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)\,\mathbf{g}(\mathbf{x}_i), \tag{11.94}$$

$$C\,\nu - \sum_{i=1}^{M}(\alpha_i + \alpha_i^*) - \beta = 0, \tag{11.95}$$

$$\sum_{i=1}^{M}(\alpha_i - \alpha_i^*) = 0, \tag{11.96}$$

$$\alpha_i + \eta_i = \frac{C}{M} \quad \text{for} \quad i = 1\ldots, M, \tag{11.97}$$

$$\alpha_i^* + \eta_i^* = \frac{C}{M} \quad \text{for} \quad i = 1\ldots, M. \tag{11.98}$$

Then the dual problem is obtained as follows. Maximize

$$Q(\alpha, \alpha^*) = -\frac{1}{2}\sum_{i,j=1}^{M}(\alpha_i - \alpha_i^*)\,(\alpha_j - \alpha_j^*)\,H(\mathbf{x}_i, \mathbf{x}_j)$$

$$+\sum_{i=1}^{M} y_i \, (\alpha_i - \alpha_i^*) \tag{11.99}$$

subject to the constraints:

$$\sum_{i=1}^{M} (\alpha_i - \alpha_i^*) = 0, \tag{11.100}$$

$$0 \le \alpha_i \le \frac{C}{M}, \quad 0 \le \alpha_i^* \le \frac{C}{M} \quad \text{for} \quad i = 1 \ldots, M, \tag{11.101}$$

$$\sum_{i=1}^{M} (\alpha_i + \alpha_i^*) \le C \, \nu, \tag{11.102}$$

where $H(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{g}^T(\mathbf{x}_i) \, \mathbf{g}(\mathbf{x}_j)$.

From (11.101) and $\alpha_i \, \alpha_i^* = 0$, the left-hand side of (11.102) is bounded by C. Thus, the solution with $\nu > 1$ is the same as that with $\nu = 1$. Assume that the obtained ε is not zero ($\varepsilon > 0$). Then the following relations hold:

$$\frac{\text{Number of errors}}{M} \le \nu \le \frac{\text{Number of support vectors}}{M}, \tag{11.103}$$

where the number of errors is the number of data that are outside of the ε-tube. This can be proved as follows. For the training data outside of the tube, either $\alpha_i = C/M$ or $\alpha_i^* = C/M$. Thus from (11.101) and (11.102), the training data outside of the tube is at most νM. Therefore, the first inequality holds. From the KKT conditions, $\varepsilon > 0$ implies $\beta = 0$. Thus from (11.95), the equality holds in (11.102). Therefore, from (11.101) and (11.102), the number of support vectors is at least νM. Thus the second inequality holds.

11.7.3 Least Squares Support Vector Regressors

Similar to the discussions for least squares (LS) support vector machines for pattern classification, Suykens [234] proposed least squares support vector machines for function approximation, in which the inequality constraints in the original support vector regressors are converted into equality constraints.

Using the M training data pairs $(\mathbf{x}_i, y_i)$ $(i = 1, \ldots, M)$, we consider determining the following function:

$$y(\mathbf{x}) = \mathbf{w}^T \, \mathbf{g}(\mathbf{x}) + b, \tag{11.104}$$

where $\mathbf{w}$ is the l-dimensional vector, b is the bias term, and $\mathbf{g}(\mathbf{x})$ is the mapping function that maps the m-dimensional vector $\mathbf{x}$ into the l-dimensional feature space.

The LS support vector regressor is trained by minimizing

$$\frac{1}{2}\mathbf{w}^T \mathbf{w} + \frac{C}{2} \sum_{i=1}^{M} \xi_i^2 \tag{11.105}$$

with respect to $\mathbf{w}$ and b subject to the equality constraints:

$$y_i = \mathbf{w}^T \mathbf{g}(\mathbf{x}_i) + b + \xi_i \quad \text{for} \quad i = 1, \ldots, M, \tag{11.106}$$

where ξ_i is the slack variable for $\mathbf{x}_i$ and C is the margin parameter.

Introducing the Lagrange multipliers α_i into (11.105) and (11.106), we obtain the unconstrained objective function:

$$\begin{aligned} Q(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}) = {} & \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{2}\sum_{i=1}^{M}\xi_i^2 \\ & - \sum_{i=1}^{M} \alpha_i \left(\mathbf{w}^T \mathbf{g}(\mathbf{x}_i) + b + \xi_i - y_i\right), \end{aligned} \tag{11.107}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)^T$ and $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_M)^T$.

Taking the partial derivatives of (11.107) with respect to $\mathbf{w}, b, \boldsymbol{\alpha}$, and $\boldsymbol{\xi}$, respectively, and equating them to zero, we obtain the optimal conditions as follows:

$$\mathbf{w} = \sum_{i=1}^{M} \alpha_i \, \mathbf{g}(\mathbf{x}_i), \tag{11.108}$$

$$\sum_{i=1}^{M} \alpha_i = 0, \tag{11.109}$$

$$\mathbf{w}^T \, \mathbf{g}(\mathbf{x}_i) + b + \xi_i - y_i = 0, \tag{11.110}$$

$$\alpha_i = C \, \xi_i \quad \text{for} \quad i = 1, \ldots, M. \tag{11.111}$$

Substituting (11.108) and (11.111) into (11.110) and expressing it and (11.109) in matrix form, we obtain

$$\begin{pmatrix} \Omega & \mathbf{1} \\ \mathbf{1}^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ b \end{pmatrix} = \begin{pmatrix} \mathbf{y} \\ 0 \end{pmatrix}, \tag{11.112}$$

where $\mathbf{1}$ is the M-dimensional vector and

$$\{\Omega_{ij}\} = \mathbf{g}^T(\mathbf{x}_i) \, \mathbf{g}(\mathbf{x}_j) + \frac{\delta_{ij}}{C}, \tag{11.113}$$

$$\delta_{ij} = \begin{cases} 1 & i = j, \\ 0 & i \neq j, \end{cases} \tag{11.114}$$

$$\mathbf{y} = (y_1, \ldots, y_M), \tag{11.115}$$

$$\mathbf{1} = (1, \ldots, 1)^T. \tag{11.116}$$

Like the support vector machine, selecting $H(\mathbf{x}, \mathbf{x}') = \mathbf{g}^T(\mathbf{x}) \, \mathbf{g}(\mathbf{x}')$ that satisfies Mercer's condition, we can avoid the explicit treatment of the feature space. The resulting approximation function becomes

$$y(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i \, H(\mathbf{x}, \mathbf{x}_i) + b. \tag{11.117}$$

The original minimization problem is solved by solving the set of simultaneous linear equations (11.112) for $\boldsymbol{\alpha}$ and b. Because the last diagonal element of the coefficient matrix in (11.112) is zero, the matrix is positive semidefinite.

By changing the inequality constraints into equality constraints, training of support vector machines reduces to solving a set of linear equations instead of a quadratic programming problem. But by this formulation, sparsity of $\boldsymbol{\alpha}$ is not guaranteed. To avoid this, Suykens [234, 235] proposed pruning the data whose associated α_i have small absolute values. Namely, first we solve (11.112) using all the training data. Then we sort α_i according to their absolute values and delete a portion of the training data set (say 5 percent of the set) starting from the data with the minimum absolute value in order. Then we solve (11.112) using the reduced training data set and iterate this procedure while the user-defined performance index is not degraded.

11.8 Performance Evaluation

11.8.1 Evaluation Conditions

We evaluated the performance of L1 and L2 support vector regressors (SVRs) using the noisy data for a water purification plant and the noiseless Mackey-Glass data listed in Table 1.1. For the Mackey-Glass data, we measured the performance by NRMSE. If stated otherwise, we used $\varepsilon = 0.01$ and $C = 10{,}000$. In addition to the training data, we artificially generated the outliers and evaluated the robustness of the estimation.

For the water purification plant data, because the number of nonstationary data is too small, we used only the stationary data for most of the evaluation. We evaluated the performance by the average and the maximum estimation errors for the training and test data. We used $\varepsilon = 1$ and $C = 1000$ if stated otherwise.

We used linear kernels, polynomial kernels with $d = 3$, and RBF kernels with $\gamma = 10$.

We set $\eta = 10^{-10}$ in (11.72) and if (11.72) is satisfied for consecutive 10 times, we stopped training. We used an AthlonMP2000+ personal computer under Linux.

We evaluated:

- the effect of the working set size on training time using the steepest ascent method (SAM),
- performance difference of L1 SVRs and L2 SVRs,
- convergence difference by inexact and exact KKT conditions,
- performance difference among several estimation methods, and
- robustness of SVRs to outliers.

11.8.2 Effect of Working Set Size on Speedup

We evaluated how an increase of the working set size accelerates training by the steepest ascent method. In the following we show the results using the L1 SVR when the inexact KKT conditions were used for selecting violating variables. The tendency was the same when the L2 SVR or the exact KKT conditions were used.

Table 11.1 shows the results of the water purification data with polynomial kernels. In the table, “Size,” “SVs, ” and “Time” denote the working set size, the number of support vectors, and the training time. From the table, the training time was shortened by increasing the working set size, and the 50 times speedup against the working set size of 2 was obtained for the working set size of 50.

Table 11.1. Effect of working set size for the water purification data with polynomial kernels

Size	Time	Training data		Test data		SVs
		Ave.err	Max.err	Ave.err	Max.err	
	(s)	(mg/l)	(mg/l)	(mg/l)	(mg/l)	
2	151	0.73	4.30	1.03	11.55	98
3	112	0.73	4.31	1.03	11.56	97
5	59	0.73	4.27	1.03	11.51	97
10	22	0.73	4.34	1.03	11.57	98
20	7.3	0.73	4.37	1.03	11.59	98
30	3.8	0.73	4.33	1.03	11.55	98
40	3.4	0.73	4.44	1.03	11.65	95
50	3.0	0.73	4.31	1.03	11.53	96

Table 11.2 shows the training speedup for the Mackey-Glass data with RBF kernels when the working set size was increased from 2 to 50. By increasing the working set size, training was sped up and 8.3 times speedup against the working set size of 2 was obtained for the working set size of 30. When the size was larger than 30, training slowed down. This is because the calculation of the inverse matrix becomes longer as the size increases. Thus there is an optimal size for training speedup.

11.8.3 Comparison of L1 and L2 Support Vector Regressors

Here, we compare performance of L1 SVRs and L2 SVRs using the Mackey-Glass data. Because the fitting capabilities of L1 SVRs and L2 SVRs are

Table 11.2. Effect of working set size for the Mackey-Glass data with RBF kernels

Size	Time (s)	Training data (NRMSE)	Test data (NRMSE)	SVs
2	58	0.028	0.027	55
3	48	0.028	0.027	57
5	29	0.028	0.027	52
10	16	0.028	0.027	52
20	8.8	0.028	0.027	54
30	7.0	0.028	0.027	52
40	7.7	0.028	0.027	50
50	8.7	0.029	0.028	49

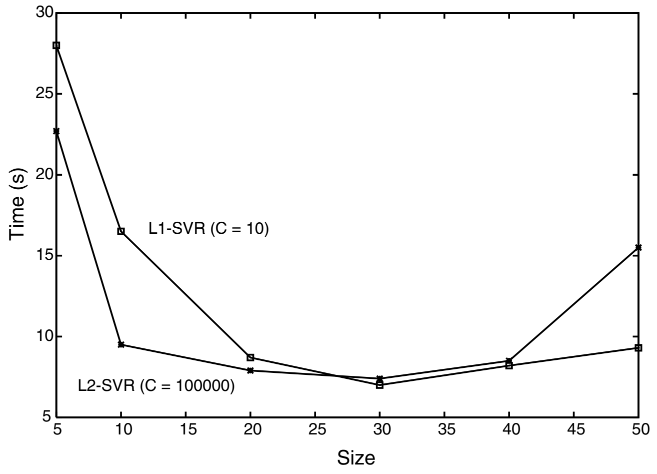


Fig. 11.4. Training time comparison of L1 SAM and L2 SAM for the Mackey-Glass data with RBF kernels. From [112, p. 2069]

different for the same value of C, we set the values of C so that the L1 SVR and the L2 SVR showed the similar NRMSEs for the training data.

Figure 11.4 shows the training time of the L1 SAM and L2 SAM when the working set size was changed. We set $C = 10$ for the L1 SAM and $C = 100{,}000$ for the L2 SAM. When the size is small, the L2 SAM is faster. But for sizes between 20 and 40, where training was the fastest, the difference is small. The reasons may be as follows: Because of RBF kernels the variables did not reach the upper bounds for the L1 SAM and the Hessian matrix for the L1 SVR with RBF kernels was positive definite. Thus, the advantages of the L2 SVR did not appear.

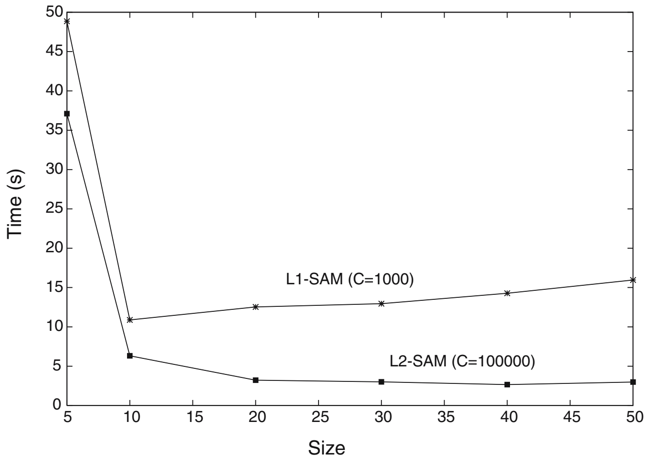


Fig. 11.5. Training time comparison of L1 and L2 SVRs for the Mackey-Glass data with polynomial kernels. From [112, p. 2070]

Figure 11.5 shows the training time comparison of the L1 SAM ($C = 1000$) and the L2 SAM ($C = 100{,}000$) when the working set size was changed. For all the sizes, training by the L2 SAM is faster. For sizes larger than 10, training by the L1 SAM was slowed but that by the L2 SAM was sped up.

Table 11.3 shows the training time of the L1 SAM ($C = 1000$) and the L2 SAM ($C = 10$) with linear kernels. The numeral in parentheses shows the working set size. For sizes 5 and 30, training by the L2 SAM is faster. Training by L1 SAM with size 30 is slower than with size 5. This is because for linear kernels the number of independent variables is the number of input variables plus 1. For the Mackey-Glass data, there are four input variables. Thus, if we set the working set size of more than five, unnecessary calculations increase. But for the L2 SAM, because the Hessian matrix is always positive definite, this sort of thing did not happen.

11.8.4 Comparison of Exact and Inexact KKT Conditions

In selecting violating variables we use either the inexact or exact KKT conditions. Here, we compare training time of the SAM by these conditions. We set $\tau = 0.001$ for the exact KKT conditions and used RBF kernels.

Figure 11.6 shows the training time comparison by the L1 SAM for the water purification data. In general, training by the exact KKT conditions is faster.

Table 11.3. Training time comparison of L1 SAM and L2 SAM with linear kernels

Method	Time (s)	Training data (NRMSE)	Test data (NRMSE)	SVs
L1 SAM (5)	7.4	0.46	0.46	446
L1 SAM (30)	16.9	0.46	0.46	446
L2 SAM (5)	1.0	0.43	0.43	475
L2 SAM (30)	0.8	0.43	0.43	475

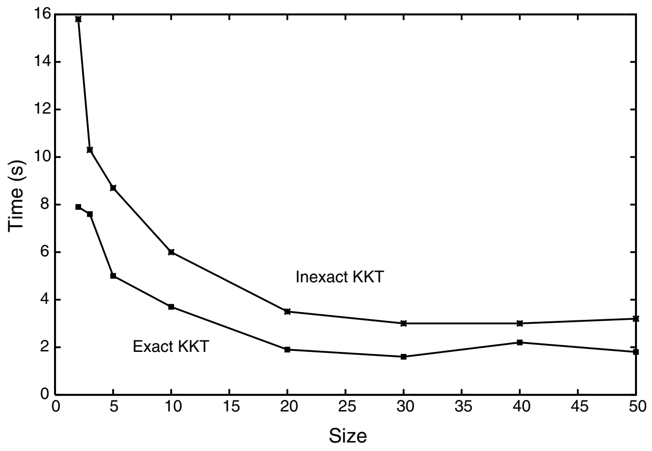


Fig. 11.6. Training time comparison by the L1 SAM for the water purification data

Figure 11.7 shows the training time comparison by the L2 SAM for the water purification data. In general, training by the inexact KKT conditions is faster, but the difference is small. But for sizes 2 and 4, training by the exact KKT conditions is faster.

Figure 11.8 shows training time comparison by the L1 SAM for the Mackey-Glass data. With sizes 2 to 10, training by the exact KKT conditions is faster, but with sizes larger than or equal to 30, training by the inexact KKT conditions is faster.

Figure 11.9 shows training time comparison of the L2 SAM for the Mackey-Glass data. Except for the sizes 2 and 20, training by the inexact KKT conditions is faster.

Using the inexact KKT conditions for the Mackey-Glass data, as the working set size became larger, longer training was needed. For the exact KKT conditions, at the initial stage of training, most of the data are detected as violating data. Thus for the Mackey-Glass data, in which the ratio of support

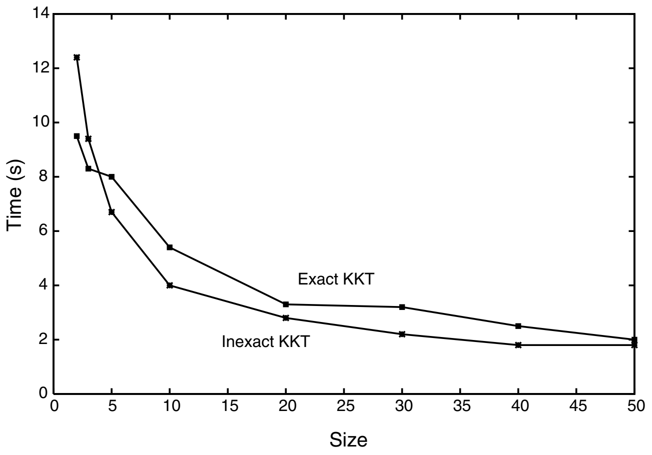


Fig. 11.7. Training time comparison by the L2 SAM for water purification data

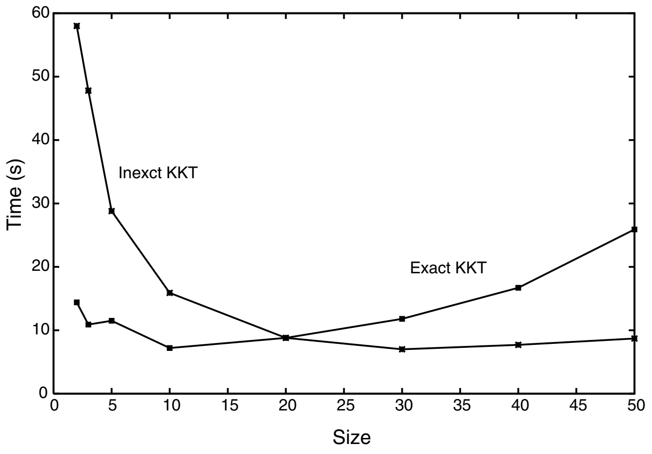


Fig. 11.8. Training time comparison of the L1 SAM for the Mackey-Glass data. From [112, p. 2070]

vectors for the training data is low, redundant calculation was increased. Thus training was considered slower than by the inexact KKT conditions.

11.8.5 Comparison with Other Training Methods

Using polynomial and RBF kernels, we compared training times of the SMA and the primal-dual interior-point method (PDIP) [254] with and without chunking.

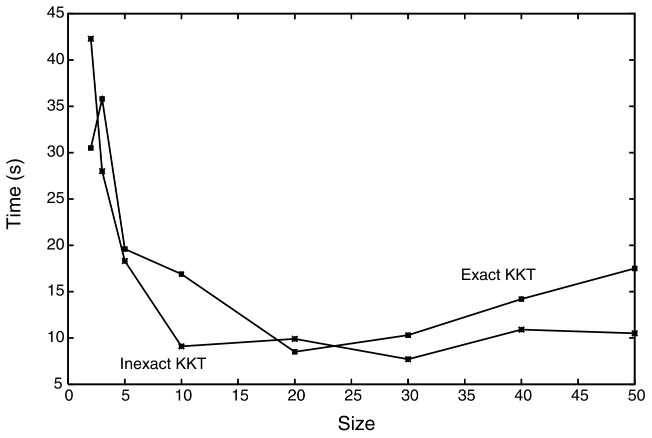


Fig. 11.9. Training time comparison of the L2 SAM for the Mackey-Glass data. From [112, p. 2070]

In the following tables a numeral in parentheses show the working set size. In chunking, we changed the number of added data from 10 to 100 with the increment of 10 and listed the fastest result.

The value of C for the L2 SVR was set so that the estimation performance was comparable to that of the L1 SVR. For the water purification data, because there was not much difference, we set $C = 1000$.

Table 11.4 lists the training time comparison for the Mackey Glass data. For polynomial kernels, training by SMA is faster than by PDIP with and without chunking. But for RBF kernels, PDIP with chunking is the fastest. This is because the number of support vectors is small and the size of the problem of each iteration was very small.

Table 11.5 shows training time comparison for the water purification data with linear kernels. Training by L2 SAM (30) was the fastest, and the PDIP with chunking was the second-fastest.

According to the results, when the number of support vectors is small, the primal-dual interior-point method with chunking was the fastest, but for the data with many support vectors, L2 SAM was the fastest.

11.8.6 Performance Comparison with Other Approximation Methods

Stationary Data

Table 11.6 shows the best estimation performance for different methods. Namely, the three-layer neural network (NN), the fuzzy function approximator with inhibition (FAMI), the fuzzy function approximator with center-of-gravity defuzzification (FACG), and the fuzzy function approximator with

Table 11.4. Training time comparison for the Mackey-Glass data

Kernel	Size	Time (s)	Training data (NRMSE)	Test data (NRMSE)	SVs
Poly	L1 SAM (10)	11.9	0.067	0.066	277
	L2 SAM (30)	2.7	0.067	0.066	282
	PDIP (—)	84.0	0.067	0.066	283
	PDIP (70)	50.6	0.067	0.066	269
RBF	L1 SAM (30)	7.0	0.028	0.027	48
	L2 SAM (20)	6.6	0.028	0.027	55
	PDIP (—)	79.9	0.028	0.027	54
	PDIP (50)	0.9	0.028	0.027	54

Table 11.5. Training comparison for the water purification data with linear kernels

Size	Time	Training data		Test data		SVs
		Ave.err	Max.err	Ave.err	Max.err	
	(s)	(mg/l)	(mg/l)	(mg/l)	(mg/l)	
L1 SAM (10)	2.5	1.14	18.36	1.18	5.77	113
L2 SAM (30)	0.7	1.26	14.43	1.26	7.64	124
PDIP (—)	10.8	1.13	18.60	1.16	5.96	115
PDIP (30)	2.6	1.13	18.60	1.16	5.96	115

a linear combination output (FALC) [3]. The support vector regressor showed the minimum estimation errors for the training data and the second minimum average estimation error for the test data following the neural network. But the maximum estimation error for the test data was the maximum.

With noisy data, usually overfitting to the training data causes low generalization ability. But for the support vector regressor, although the estimation error for the training data is very small, the estimation error for the test data is comparable to those of the other methods.

Nonstationary Data

Because the nonstationary training data included only 45 data and they were noisy, overfitting occurred easily for conventional methods. We evaluated whether this is the case for the support vector regressor.

Table 11.6. Performance comparison for the stationary data

Method	Training data		Test data	
	Ave.err (mg/l)	Max.err (mg/l)	Ave.err (mg/l)	Max.err (mg/l)
L1 SVR	0.69	1.04	1.03	6.99
NN	0.84	4.75	0.99	6.95
FAMI	1.07	4.75	1.18	5.57
FACG	0.91	5.06	1.05	5.33
FALC	1.09	4.34	1.16	5.22

The best estimation performance was achieved for the polynomial kernel with $C = 10$, $d = 4$, and $\varepsilon = 0.5$. Table 11.7 lists the best estimation performance for different methods. The average estimation error for the training data by the support vector regressor was the smallest, but the maximum estimation error was the largest. The average estimation error of the test data by the support vector regressor was the second smallest following FAMI but the maximum estimation error was the largest.

Table 11.7. Performance comparison for the nonstationary data

Method	Training data		Test data	
	Ave.err (mg/l)	Max.err (mg/l)	Ave.err (mg/l)	Max.err (mg/l)
L1 SVR	0.95	9.95	1.62	7.39
NN	1.59	6.83	1.74	6.78
FAMI	1.56	7.20	1.46	4.97
FACG	1.91	6.30	1.95	7.18
FALC	1.63	5.79	1.92	6.30

Mackey-Glass Data

The best estimation performance was obtained when RBF kernels with $\gamma = 10$ and $\varepsilon = 0.001$ were used.

Table 11.8 lists the best estimation results for different methods. In the table, the support vector regressor showed the smallest NRMSE.

Table 11.8. Performance comparison for the Mackey-Glass data

Approximator	Test data (NRMSE)
L1 SVR	0.003
NN [70]	0.02
ANFIS [119]	0.007
Cluster estimation-based [61]	0.014
FAMI [3]	0.092
FACG [3]	0.005
FALC [3]	0.006

11.8.7 Robustness for Outliers

The support vector regressor has a mechanism to suppress outliers. To evaluate this, we randomly selected five data in the Mackey-Glass time series data and multiplied them by 2. Therefore, the five input-output pairs that are generated from the time series data included outliers. Namely, 25 outliers were included in the training data. We used the RBF kernels with $\gamma = 10$ and $\varepsilon = 0.005$ that showed best performance without outliers. We evaluated the performance changing C. The larger the magnitude of C, the larger the effect of outliers to the estimation.

Table 11.9 shows the estimation performance when outliers were included. The numerals in the parentheses are the NRMSEs excluding the outliers. Table 11.10 shows the estimation performance when outliers were not included. When the outliers were included, the best estimation was achieved for $C = 10$. The NRMSE for the test data decreased as the value of C changed from 100,000 to 10. This means that the effect of outliers was effectively suppressed. But when the value of C was further decreased to 0.1, the NRMSE increased. In this case the number of support vectors increased. Thus too small a value of C excluded not only outliers but also normal data. Therefore, to effectively eliminate outliers, we need to set a proper value to C.

The NRMSE for the training data is large when the outliers were included to calculate the NRMSE, but when they were excluded, the NRMSEs were almost the same as those without outliers. For example, for $C = 10$, the difference was only 0.002. Thus the support vector regressor can be trained irrespective of inclusion of outliers.

Table 11.9. Performance of the Mackey-Glass data with outliers

C	Training data (NRMSE)	Test data (NRMSE)	SVs	Time (s)
100,000	0.335 (0.019)	0.033	244	34822
10,000	0.342 (0.019)	0.023	182	6348
1000	0.345 (0.018)	0.019	146	2174
100	0.346 (0.017)	0.018	117	700
10	0.348 (0.017)	0.017	95	414
1	0.349 (0.018)	0.018	114	251
0.1	0.363 (0.031)	0.030	181	129

Table 11.10. Performance of the Mackey-Glass data without outliers

C	Training data (NRMSE)	Test data (NRMSE)	SVs	Time (s)
100,000	0.014	0.014	80	414
10,000	0.014	0.014	80	393
1000	0.014	0.014	80	375
100	0.014	0.014	80	381
10	0.015	0.014	82	381
1	0.017	0.016	101	270
0.1	0.026	0.026	188	155

11.8.8 Summary

In this section, we evaluated support vector regressors (SVRs) for Mackey-Glass and water purification data. The results are summarized as follows:

1. Training by the steepest ascent method (SMA), which optimizes plural data at a time, was sped up by increasing the working set size from 2. There was an optimal working set size. Namely, if a working set size is too large, the training is slowed down.
2. The difference between the L1 SAM and the L2 SAM was small for kernels with a high-dimensional feature space such as RBF kernels. But for kernels with a low-dimensional feature space such as linear kernels, training by L2 SAM was much faster. In such a situation, the advantages of positive

definiteness of the Hessian matrix and the nonupper-bounded variables became evident.

3. The exact KKT conditions did not always lead to faster convergence than the inexact KKT conditions. For the water purification data and for the Mackey-Glass data with a small working set size, the exact KKT conditions performed better for the L1 SAM. But for the Mackey-Glass data, as we increased the working set size, the inexact KKT conditions performed better. This is because, in the initial stage of training, exact KKT conditions detect almost all variables as violating variables, and this leads to useless calculations.
4. For problems with a small number of support vectors, training by the primal-dual interior-point method with chunking was faster than that by the SAM. But for problems with a relatively large number of support vectors, the SAM was faster.
5. There was not much difference in performance among different approximation methods.
6. By proper selection of the value of C, SVRs could suppress the effect of outliers.

A

Conventional Classifiers

A.1 Bayesian Classifiers

Bayesian classifiers are based on probability theory and give the theoretical basis for pattern classification.

Let ω be a random variable and take one of n states: $\omega_1, \ldots, \omega_n$, where ω_i indicates class i, and an m-dimensional feature vector $\mathbf{x}$ be a random variable vector. We assume that we know the a priori probabilities $P(\omega_i)$ and conditional densities $p(\mathbf{x} \,|\, \omega_i)$. Then when $\mathbf{x}$ is observed, the a posteriori probability of ω_i, $P(\omega_i \,|\, \mathbf{x})$, is calculated by the Bayes' rule:

$$P(\omega_i \,|\, \mathbf{x}) = \frac{p(\mathbf{x} \,|\, \omega_i)\, P(\omega_i)}{p(\mathbf{x})}, \tag{A.1}$$

where

$$p(\mathbf{x}) = \sum_{i=1}^{n} p(\mathbf{x} \,|\, \omega_i)\, P(\omega_i). \tag{A.2}$$

Assume that the cost c_{ij} is given when $\mathbf{x}$ is classified into class i although it is class j. Then the expected conditional cost in classifying $\mathbf{x}$ into class i, $C(\omega_i \,|\, \mathbf{x})$, is given by

$$C(\omega_i \,|\, \mathbf{x}) = \sum_{j=1}^{n} c_{ij}\, P(\omega_j \,|\, \mathbf{x}). \tag{A.3}$$

The conditional cost is minimized when $\mathbf{x}$ is classified into the class

$$\arg \min_{i=1,\ldots,n} C(\omega_i \,|\, \mathbf{x}). \tag{A.4}$$

This is called *Bayes' decision rule.*

In diagnosis problems, usually there are normal and abnormal classes. Misclassification of normal data into the abnormal class is less favorable than misclassification of abnormal data into the normal class. In such a situation, we set a smaller cost to the former than the latter.

If we want to minimize the average probability of misclassification, we set the cost as follows:

$$c_{ij} = \begin{cases} 0 & \text{for} \quad i = j, \\ 1 & \text{for} \quad i \neq j, \end{cases} \qquad i, j = 1, \ldots, n. \tag{A.5}$$

Then, from (A.1) and (A.2) the conditional cost given by (A.3) becomes

$$\begin{aligned} C(\omega_i \,|\, \mathbf{x}) &= \sum_{\substack{j \neq i, \\ j = 1}}^{n} P(\omega_j \,|\, \mathbf{x}) \\ &= 1 - P(\omega_i \,|\, \mathbf{x}). \end{aligned} \tag{A.6}$$

Therefore, the Bayes decision rule given by (A.4) becomes

$$\begin{aligned} &\arg \max_{i = 1,\ldots,n} P(\omega_i \,|\, \mathbf{x}) \\ &= \arg \max_{i = 1,\ldots,n} p(\mathbf{x} \,|\, \omega_i)\, P(\omega_i). \end{aligned} \tag{A.7}$$

Now, we assume that the conditional densities $p(\mathbf{x} \,|\, \omega_i)$ are normal:

$$p(\mathbf{x} \,|\, \omega_i) = \frac{1}{\sqrt{(2\pi)^n \det(Q_i)}} \exp\left(- \frac{(\mathbf{x} - \mathbf{c}_i)^T \, Q_i^{-1} \, (\mathbf{x} - \mathbf{c}_i)}{2} \right), \tag{A.8}$$

where $\mathbf{c}_i$ is the mean vector and Q_i is the covariance matrix of the normal distribution for class i. If the a priori probabilities $P(\omega_i)$ are the same for $i = 1, \ldots, n$, $\mathbf{x}$ is classified into class i with the maximum $p(\mathbf{x} \,|\, \omega_i)$.

A.2 Nearest Neighbor Classifiers

Nearest neighbor classifiers use all the training data as templates for classification. In the simplest form, for a given input vector, the nearest neighbor classifier searches the nearest template and classifies the input vector into the class to which the template belongs. In the complex form the classifier treats k nearest neighbors. For a given input vector, the k nearest templates are searched and the input vector is classified into the class with the maximum number of templates. The classifier architecture is simple, but as the number of training data becomes larger, the classification time becomes longer. Therefore many methods for speeding up classification are studied [33, pp. 181–91], [202, pp. 191–201]. One uses the branch-and-bound method [90, pp. 360–2] and another edits the training data, i.e., selects or replaces the data with the suitable templates. It is proved theoretically that as the number of templates becomes larger, the expected error rate of the nearest neighbor classifier is bounded by twice that of the Bayesian classifier [97, pp. 159–75].

Usually the Euclidean distance is used to measure the distance between two data $\mathbf{x}$ and $\mathbf{y}$:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}, \tag{A.9}$$

but other distances, such as the Manhattan distance

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{m} |x_i - y_i| \tag{A.10}$$

are used. It is clear from the architecture that the recognition rate of the training data for the one-nearest neighbor classifier is 100 percent. But for the k-nearest neighbor classifier with $k > 1$, the recognition rate of the training data is not always 100 percent.

Because the distances such as the Euclidean and Manhattan distances are not invariant in scaling, classification performance varies according to the scaling of input ranges.

B

Matrices

B.1 Matrix Properties

In this section, we summarize the matrix properties used in this book. For more detailed explanation, see, e.g., [96].

Vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are *linearly independent* if

$$a_1 \mathbf{x}_1 + \cdots + a_n \mathbf{x}_n = 0 \tag{B.1}$$

holds only when $a_1 = \cdots = a_m = 0$. Otherwise, namely, at least one a_i is nonzero, $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are *linearly dependent.*

Let A be an $m \times m$ matrix:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \multicolumn{3}{c}{\dotfill} \\ a_{m1} & \cdots & a_{mm} \end{pmatrix}. \tag{B.2}$$

Then the *transpose* of A, denoted by A^T, is

$$A^T = \begin{pmatrix} a_{11} & \cdots & a_{m1} \\ \multicolumn{3}{c}{\dotfill} \\ a_{1m} & \cdots & a_{mm} \end{pmatrix}. \tag{B.3}$$

If A satisfies $A = A^T$, A is a *symmetric matrix.* If A satisfies $A^T A = A\,A^T = I$, A is an *orthogonal matrix*, where I is the $m \times m$ *unit matrix*:

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \multicolumn{4}{c}{\dotfill} \\ 0 & 0 & \cdots & 1 \end{pmatrix}. \tag{B.4}$$

If $m \times m$ matrices A and B satisfy $A\,B = I$, B is called the *inverse* of A and is denoted by A^{-1}. If A has the inverse, A is *regular* (or *nonsingular*). Otherwise, A is singular.

Lemma B.1. *Let matrices A, B, C, and D be $n \times n$, $n \times m$, $m \times n$, and $m \times m$ matrices, respectively and A, D, and $D + CA^{-1}B$ be nonsingular. Then the following relation holds:*

$$\left(A + BD^{-1}C\right)^{-1} = A^{-1} - A^{-1}B\left(D + CA^{-1}B\right)^{-1}CA^{-1}. \tag{B.5}$$

This is called the *matrix inversion lemma.*

Let $m = 1$, $B = \mathbf{b}$, $C = \mathbf{c}^T$, and $D = 1$. Then (B.5) reduces to

$$\left(A + \mathbf{b}\mathbf{c}^T\right)^{-1} = A^{-1} - \frac{A^{-1}\mathbf{b}\mathbf{c}^T A^{-1}}{1 + \mathbf{c}^T A^{-1}\mathbf{b}}. \tag{B.6}$$

Using (B.6), LOO error rate estimation of linear equation based–machines such as LS support vector machines can be sped up [42, 55, 209, 278].

The *determinant* of an $m \times m$ matrix $A = \{a_{ij}\}$, $\det(A)$, is defined recursively by

$$\det(A) = \sum_{i=1}^{m} (-1)^{i+1}\, a_{1i} \det(A_{1i}), \tag{B.7}$$

where A_{1i} is the $(m-1) \times (m-1)$ matrix obtained by deleting the first row and the ith column from A. When $m = 1$, $\det(A) = a_{11}$.

If, for the $m \times m$ matrix A, a nonzero m-dimensional vector $\mathbf{x}$ exists for a constant λ:

$$A\,\mathbf{x} = \lambda\,\mathbf{x}, \tag{B.8}$$

λ is called an *eigenvalue* and $\mathbf{x}$ an *eigenvector*. Rearranging (B.8) gives

$$(A - \lambda\, I)\,\mathbf{x} = 0. \tag{B.9}$$

Thus, (B.9) has nonzero $\mathbf{x}$, when

$$\det(A - \lambda\, I) = 0, \tag{B.10}$$

which is called the *characteristic equation.*

Theorem B.2. *All the eigenvalues of a real symmetric matrix are real.*

Theorem B.3. *Eigenvectors associated with different eigenvalues for a real symmetric matrix are orthogonal.*

For an m-dimensional vector $\mathbf{x}$ and an $m \times m$ symmetric matrix A, $Q = \mathbf{x}^T A\,\mathbf{x}$ is called the *quadratic form.* If for any nonzero $\mathbf{x}$, $Q = \mathbf{x}^T A\,\mathbf{x} \geq 0$, Q is *positive semidefinite.* Matrix Q is *positive definite* if the strict inequality holds. Let L be an $m \times m$ orthogonal matrix. By $\mathbf{y} = L\,\mathbf{x}$, $\mathbf{x}$ is transformed into $\mathbf{y}$. This is the transformation from one orthonormal base into another orthonormal basis. The quadratic form Q is

$$\begin{aligned} Q &= \mathbf{x}^T A\,\mathbf{x} \\ &= \mathbf{y}^T L\, A\, L^T\,\mathbf{y}. \end{aligned} \tag{B.11}$$

Theorem B.4. *The characteristic equations for A and $L\,A\,L^T$ are the same.*

Theorem B.5. *If an $m \times m$ real symmetric matrix A is diagonalized by L:*

$$L\,A\,L^T = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \multicolumn{4}{c}{\dotfill} \\ 0 & 0 & \cdots & \lambda_m \end{pmatrix}, \tag{B.12}$$

$\lambda_1, \ldots, \lambda_m$ are the eigenvalues of A and the ith row of L is the eigenvector associated with λ_i.

If all the eigenvalues of A are positive, A is *positive definite.* If all the eigenvalues are nonnegative, A is *positive semidefinite.*

B.2 Least Squares Methods and Singular Value Decomposition

Assume that we have M input-output pairs $\{(\mathbf{a}'_1, b_1), \ldots, (\mathbf{a}'_M, b_M)\}$ in the $(n-1)$-dimensional input space $\mathbf{x}'$ and one-dimensional output space y. Now using the least squares method, we determine the linear relation of the input-output pairs:

$$y = \mathbf{p}^T\,\mathbf{x}' + q, \tag{B.13}$$

where $\mathbf{p}$ is the $(n-1)$-dimensional vector, q is a scalar constant, and $M \geq n$.

Rewriting (B.13), we get

$$(\mathbf{x}'^T, 1) \begin{pmatrix} \mathbf{p} \\ q \end{pmatrix} = y. \tag{B.14}$$

Substituting $\mathbf{a}'_i$ and b_i into $\mathbf{x}'$ and y in (B.14), respectively, and replacing $(\mathbf{p}^T, q)^T$ with the n-dimensional parameter vector $\mathbf{x}$, we obtain

$$\mathbf{a}_i^T\,\mathbf{x} = b_i \qquad \text{for} \quad i = 1, \ldots, M, \tag{B.15}$$

where $\mathbf{a}_i = (\mathbf{a}'^T_i, 1)^T$.

We determine the parameter vector $\mathbf{x}$ so that the sum-of-squares error:

$$E = (A\,\mathbf{x} - \mathbf{b})^T(A\,\mathbf{x} - \mathbf{b}) \tag{B.16}$$

is minimized, where A is an $M \times n$ matrix and $\mathbf{b}$ is an M-dimensional vector:

$$A = \begin{pmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_M^T \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{pmatrix}. \tag{B.17}$$

Here, if the rank of A is smaller than n, there is no unique solution. In that situation, we determine $\mathbf{x}$ so that the Euclidean norm of $\mathbf{x}$ is minimized.

Matrix A is decomposed into singular values [96]:

$$A = U S V^T, \tag{B.18}$$

where U and V are $M \times M$ and $n \times n$ orthogonal matrices, respectively, and S is an $M \times n$ diagonal matrix given by

$$S = \left(\begin{array}{ccc} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_n \\ \hline & 0_{M-n,n} & \end{array} \right). \tag{B.19}$$

Here, σ_i are singular values and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$, and $0_{M-n,n}$ is the $(M-n) \times n$ zero matrix.

It is known that the columns of U and V are the eigenvectors of $A\,A^T$ and $A^T A$, respectively, and that the singular values correspond to the square roots of the eigenvalues of $A\,A^T$, which are the same as those of $A^T A$ [60, pp. 434–5]. Thus when A is a symmetric square matrix, $U = V$ and $A = U\,S\,U^T$. This is similar to the diagonalization of the square matrix given by Theorem B.5. The difference is that the singular values A are the absolute values of the eigenvalues of A. Thus, if A is a positive (semi)definite matrix, both decompositions are the same.

Rewriting (B.16), we get [96, p. 256]

$$\begin{aligned} E &= (A\,\mathbf{x} - \mathbf{b})^T (A\,\mathbf{x} - \mathbf{b}) \\ &= (U\,S\,V^T \mathbf{x} - U\,U^T\,\mathbf{b})^T (A\,\mathbf{x} - \mathbf{b}) \\ &= (S\,V^T \mathbf{x} - U^T\,\mathbf{b})^T (S\,V^T \mathbf{x} - U^T\,\mathbf{b}) \\ &= \sum_{i=1}^{n} (\sigma_i\,\mathbf{v}_i^T\,\mathbf{x} - \mathbf{u}_i^T\,\mathbf{b})^2 + \sum_{i=n+1}^{M} (\mathbf{u}_i^T\,\mathbf{b})^2, \end{aligned} \tag{B.20}$$

where $U = (\mathbf{u}_1, \ldots, \mathbf{u}_M)$ and $V = (\mathbf{v}_1, \ldots, \mathbf{v}_n)$. Assuming that the rank of A is $r\,(\leq n)$, (B.20) is minimized when

$$\sigma_i\,\mathbf{v}_i^T\,\mathbf{x} = \mathbf{u}_i^T\,\mathbf{b} \qquad \text{for} \quad i = 1, \ldots, r, \tag{B.21}$$

$$\mathbf{v}_i^T\,\mathbf{x} = 0 \qquad \text{for} \quad i = r+1, \ldots, n. \tag{B.22}$$

Equation (B.22) is imposed to obtain the minimum Euclidean norm solution. From (B.21) and (B.22), we obtain

$$\mathbf{x} = V\,S^+ U^T\,\mathbf{b} = A^+\,\mathbf{b}, \tag{B.23}$$

where S^+ is the $n \times M$ diagonal matrix given by

$$S^+ = \left(\begin{array}{ccc|c} \dfrac{1}{\sigma_1} & & 0 & \\ & \ddots & & 0 \\ & & \dfrac{1}{\sigma_r} & \\ 0 & & 0 & \end{array}\right). \tag{B.24}$$

We call A^+ the pseudo-inverse of A. We must bear in mind that in calculating the pseudo-inverse, we replace the reciprocal of 0 with 0, not with infinity. This ensures the minimum norm solution.

From (B.18) and (B.23),

$$\begin{aligned} A^+ A &= V\, S^+ U^T\, U\, S\, V^T \\ &= V\, S^+\, S\, V^T \\ &= V \begin{pmatrix} I_r & 0_{r,n-r} \\ 0_{n-r,r} & 0_{n-r} \end{pmatrix} V^T \\ &= \begin{pmatrix} I_r & 0_{r,n-r} \\ 0_{n-r,r} & 0_{n-r} \end{pmatrix}, \qquad (B.25) \\ A\, A^+ &= U\, S\, S^+\, U^T \\ &= \begin{pmatrix} I_r & 0_{r,M-r} \\ 0_{M-r,r} & 0_{M-r} \end{pmatrix}, \qquad (B.26) \end{aligned}$$

where I_r is the $r \times r$ unit matrix, 0_i is the $i \times i$ zero matrix, and $0_{i,j}$ is the $i \times j$ zero matrix. Therefore, if A is a square matrix with rank n, $A^+\, A = A\, A^+ = I$. Namely, the pseudo-inverse of A coincides with the inverse of A, A^{-1}. If $M > n$ and the rank of A is n, $A^+\, A = I$ but $A\, A^+ \neq I$. In this case A^+ is given by

$$A^+ = (A^T\, A)^{-1}\, A^T. \tag{B.27}$$

This is obtained by taking the derivative of (B.16) with respect to $\mathbf{x}$ and equating the result to zero.

When $M > n$ and the rank of A is smaller than n, $A^+\, A \neq I$ and $A\, A^+ \neq I$.

Even when $A^T\, A$ is nonsingular, it is recommended to calculate the pseudo-inverse by singular value decomposition, not using (B.27). Because if $A^T\, A$ is near singular, $(A^T\, A)^{-1}\, A^T$ is vulnerable to the small singular values [195, pp. 59–70].

B.3 Covariance Matrices

Let $\mathbf{x}_1, \ldots, \mathbf{x}_M$ be M samples of the m-dimensional random variable X. Then the sample covariance matrix of X is given by

$$Q = \frac{1}{M} \sum_{i=1}^{M} (\mathbf{x}_i - \mathbf{c})\, (\mathbf{x}_i - \mathbf{c})^T, \tag{B.28}$$

where $\mathbf{c}$ is the mean vector:

$$\mathbf{c} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{x}_i. \tag{B.29}$$

To get the unbiased estimate of the covariance matrix, we replace M with $M - 1$ in (B.28), but in this book we use (B.28) as the sample covariance matrix.

Let

$$\mathbf{y}_i = \mathbf{x}_i - \mathbf{c}. \tag{B.30}$$

Then (B.28) becomes

$$Q = \frac{1}{M} \sum_{i=1}^{M} \mathbf{y}_i \, \mathbf{y}_i^T. \tag{B.31}$$

From (B.29) and (B.30), $\mathbf{y}_1, \ldots, \mathbf{y}_M$ are linearly dependent.

According to the definition, the covariance matrix Q is symmetric. Matrix Q is positive (semi)definite, as the following theorem shows.

Theorem B.6. *The covariance matrix Q given by (B.31) is positive definite if $\mathbf{y}_1, \ldots, \mathbf{y}_M$ have at least m linearly independent data. Matrix Q is positive semidefinite, if any m data from $\mathbf{y}_1, \ldots, \mathbf{y}_M$ are linearly dependent.*

Proof. Let $\mathbf{z}$ be an m-dimensional nonzero vector. From (B.31),

$$\begin{aligned}\mathbf{z}^T Q \, \mathbf{z} &= \mathbf{z}^T \left(\frac{1}{M} \sum_{i=1}^{M} \mathbf{y}_i \, \mathbf{y}_i^T \right) \mathbf{z} \\ &= \frac{1}{M} \sum_{i=1}^{M} \left(\mathbf{z}^T \mathbf{y}_i\right) \left(\mathbf{z}^T \mathbf{y}_i\right)^T \\ &= \frac{1}{M} \sum_{i=1}^{M} \left(\mathbf{z}^T \mathbf{y}_i\right)^2 \geq 0. \end{aligned} \tag{B.32}$$

Thus Q is positive semidefinite. If there are m linearly independent data in $\{\mathbf{y}_1, \ldots, \mathbf{y}_M\}$, they span the m-dimensional space. Because any $\mathbf{z}$ is expressed by a linear combination of these data, the strict inequality holds for (B.32).

Because $\mathbf{y}_1, \ldots, \mathbf{y}_M$ are linearly dependent, at least $m + 1$ samples are necessary so that Q becomes positive definite. ■

Assuming that Q is positive definite, the following theorem holds.

Theorem B.7. *If Q is positive definite, the mean square weighted distance for $\{\mathbf{y}_1, \ldots, \mathbf{y}_M\}$ is m:*

$$\frac{1}{M} \sum_{i=1}^{M} \mathbf{y}_i^T \, Q^{-1} \, \mathbf{y}_i = m. \tag{B.33}$$

Proof. Let P be the orthogonal matrix that diagonalizes Q. Namely,

$$P\,Q\,P^T = \mathrm{diag}(\lambda_1, \ldots, \lambda_m), \tag{B.34}$$

where diag denotes the diagonal matrix, and $\lambda_1, \ldots, \lambda_m$ are the eigenvalues of Q. From (B.34),

$$Q = P^T \mathrm{diag}(\lambda_1, \ldots, \lambda_m) P, \tag{B.35}$$

$$Q^{-1} = P^T \mathrm{diag}(\lambda_1^{-1}, \ldots, \lambda_m^{-1}) P. \tag{B.36}$$

Let

$$\tilde{\mathbf{y}}_i = P\,\mathbf{y}_i. \tag{B.37}$$

Then from (B.31) and (B.37), (B.34) becomes

$$\frac{1}{M} \sum_{i=1}^{M} \tilde{\mathbf{y}}_i\, \tilde{\mathbf{y}}_i^T = \mathrm{diag}(\lambda_1, \ldots, \lambda_m). \tag{B.38}$$

Thus for the diagonal elements of (B.38),

$$\frac{1}{M} \sum_{i=1}^{M} \tilde{y}_{ik}^2 = \lambda_k \qquad \text{for} \quad k = 1, \ldots, m, \tag{B.39}$$

where $\tilde{y}_{ik}$ is the kth element of $\tilde{\mathbf{y}}_i$. From (B.36) and (B.37), the left-hand side of (B.33) becomes

$$\begin{aligned} \frac{1}{M} \sum_{i=1}^{M} \mathbf{y}_i^T\, Q^{-1}\, \mathbf{y}_i &= \frac{1}{M} \sum_{i=1}^{M} \tilde{\mathbf{y}}_i^T\, \mathrm{diag}(\lambda_1^{-1}, \ldots, \lambda_m^{-1})\, \tilde{\mathbf{y}}_i \\ &= \frac{1}{M} \sum_{i=1}^{M} \sum_{k=1}^{m} \lambda_k^{-1}\, \tilde{y}_{ik}^2. \end{aligned} \tag{B.40}$$

Thus from (B.39) and (B.40), the theorem holds. ■

C

Quadratic Programming

Quadratic programming is the basis of support vector machines. Here we summarize some of the basic properties of quadratic programming.

C.1 Optimality Conditions

Consider the following optimization problem. Minimize

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q\,\mathbf{x} + \mathbf{c}^T \mathbf{x} \tag{C.1}$$

subject to

$$g_i(\mathbf{x}) = \mathbf{a}_i^T \mathbf{x} + b_i \geq 0 \quad \text{for} \quad i = 1, \ldots, k, \tag{C.2}$$
$$h_i(\mathbf{x}) = \mathbf{d}_i^T \mathbf{x} + e_i = 0 \quad \text{for} \quad i = 1, \ldots, o, \tag{C.3}$$

where $\mathbf{x}$, $\mathbf{a}_i$, and $\mathbf{d}_i$ are m-dimensional vectors; Q is an $m \times m$ positive semidefinite matrix; and b_i and e_i are scalar constants. This problem is called the *quadratic programming problem.* Because of the linear equality and inequality constraints, $\mathbf{x}$ is in a closed convex domain.

We introduce the Lagrange multipliers:

$$L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) - \sum_{i=1}^{k} \alpha_i\, g_i(\mathbf{x}) + \sum_{i=1}^{o} \beta_i\, h_i(\mathbf{x}), \tag{C.4}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)^T$, $\alpha_i \geq 0$ for $i = 1, \ldots, k$, and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_o)^T$. Then the following theorem holds.

Theorem C.1. *The optimal solution* $(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ *exists if and only if the following conditions are satisfied:*

$$\frac{\partial L(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{x}} = \mathbf{0}, \tag{C.5}$$

$$\alpha_i^* g_i(\mathbf{x}^*) = 0 \qquad \textit{for} \quad i = 1, \ldots, k, \tag{C.6}$$

$$\alpha_i^* \geq 0 \qquad \textit{for} \quad i = 1, \ldots, k, \tag{C.7}$$

$$h_i(\mathbf{x}^*) = 0 \qquad \textit{for} \quad i = 1, \ldots, o. \tag{C.8}$$

These conditions are called the *Karush-Kuhn-Tucker (KKT) conditions* and the conditions given by (C.6) are called the *Karush-Kuhn-Tucker complementarity conditions.* If there is no confusion, the KKT complementarity conditions are abbreviated the *KKT conditions.*

The KKT complementarity condition means that if $\alpha_i^* > 0$, $g_i(\mathbf{x}^*) = 0$ (it is called *active*); and if $\alpha_i^* = 0$, $g_i(\mathbf{x}^*) \geq 0$ (it is called *inactive*).

C.2 Properties of Solutions

The optimal solution can be interpreted geometrically. From (C.4),

$$\frac{\partial L(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{x}} = \frac{\partial f(\mathbf{x}^*)}{\partial \mathbf{x}} - \sum_{i=1}^{k} \alpha_i \frac{\partial g_i(\mathbf{x}^*)}{\partial \mathbf{x}} + \sum_{i=1}^{o} \beta_i \frac{\partial h_i(\mathbf{x}^*)}{\partial \mathbf{x}} = \mathbf{0}. \tag{C.9}$$

If some inequality constraints are inactive (i.e., the associated Lagrange multipliers are zero) for the optimal solution, we can discard the associated terms from (C.9). If they are active, they can be treated as the equality constraints. Thus, without loss of generality, we can assume that the inequality constraints are all inactive. Then the optimal solution must satisfy

$$-\frac{\partial f(\mathbf{x}^*)}{\partial \mathbf{x}} = \sum_{i=1}^{o} \beta_i^* \frac{\partial h_i(\mathbf{x}^*)}{\partial \mathbf{x}}. \tag{C.10}$$

The negative gradient of $f(\mathbf{x})$, $-\partial f(\mathbf{x})/\partial \mathbf{x}$, points the direction in which $f(\mathbf{x})$ decreases the most. And at the optimal solution the negative gradient must be perpendicular to the equality constraint $h_i(\mathbf{x}) = 0$ or parallel to $\partial h_i(\mathbf{x}^*)/\partial \mathbf{x}$. Therefore, the negative gradient must be in the subspace spanned by $\partial g_i(\mathbf{x}^*)/\partial \mathbf{x}$ $(i = 1, \ldots, o)$, which is equivalent to (C.10).

If Q is positive definite, the solution is unique. And if Q is positive semidefinite, the solution may not be unique. But if $\mathbf{x}_\mathrm{o}$ and $\mathbf{x}'_\mathrm{o}$ are solutions, $\lambda \mathbf{x}_\mathrm{o} + (1 - \lambda)\mathbf{x}'_\mathrm{o}$, where $1 \geq \lambda \geq 0$, is also a solution.

For the optimal solution $(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$, the following relation holds:

$$L(\mathbf{x}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \geq L(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \geq L(\mathbf{x}^*, \boldsymbol{\alpha}, \boldsymbol{\beta}). \tag{C.11}$$

Namely, $L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ is minimized with respect to $\mathbf{x}$ and maximized with respect to $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. Thus the optimal point is a saddle point.

Example C.2. Consider the following problem. Minimize

$$f(\mathbf{x}) = \frac{1}{2}\,\mathbf{x}^T\,Q\,\mathbf{x} \tag{C.12}$$

subject to

$$2 \geq x_1 + x_2 \geq 1, \tag{C.13}$$

where $\mathbf{x} = (x_1\, x_2)^T$ and Q is positive definite:

$$Q = \begin{pmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix}. \tag{C.14}$$

Because

$$L(\mathbf{x}, \boldsymbol{\alpha}) = \frac{1}{2}\,(x_1^2 + x_1 x_2 + x_2^2) - \alpha_1\,(x_1 + x_2 - 1) - \alpha_2\,(2 - x_1 - x_2), \tag{C.15}$$

the KKT conditions are given by

$$\frac{\partial L(\mathbf{x}, \boldsymbol{\alpha})}{\partial x_1} = x_1 + \frac{1}{2}\,x_2 - \alpha_1 + \alpha_2 = 0, \tag{C.16}$$

$$\frac{\partial L(\mathbf{x}, \boldsymbol{\alpha})}{\partial x_2} = \frac{1}{2}\,x_1 + x_2 - \alpha_1 + \alpha_2 = 0, \tag{C.17}$$

$$\alpha_1\,(x_1 + x_2 - 1) = 0, \quad \alpha_2\,(2 - x_1 - x_2) = 0, \tag{C.18}$$

$$\alpha_1 \geq 0, \quad \alpha_2 \geq 0. \tag{C.19}$$

Subtracting (C.17) from (C.16), we obtain $x_1 = x_2$. Therefore, from $f(\mathbf{x}) = 3\,x_1^2/2$ and the KKT conditions, the optimal solution satisfies

$$x_1 = x_2 = \frac{1}{2}, \quad \alpha_1 = \frac{3}{2}, \quad \alpha_2 = 0. \tag{C.20}$$

Thus the solution is unique (see Fig. C.1).

Let Q be positive semidefinite:

$$Q = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}. \tag{C.21}$$

Because

$$L(\mathbf{x}, \boldsymbol{\alpha}) = \frac{1}{2}\,(x_1 + x_2)^2 - \alpha_1\,(x_1 + x_2 - 1) - \alpha_2\,(2 - x_1 - x_2), \tag{C.22}$$

the KKT conditions are given by

$$\frac{\partial L(\mathbf{x}, \boldsymbol{\alpha})}{\partial x_1} = \frac{\partial L(\mathbf{x}, \boldsymbol{\alpha})}{\partial x_2} = x_1 + x_2 - \alpha_1 + \alpha_2 = 0, \tag{C.23}$$

$$\alpha_1\,(x_1 + x_2 - 1) = 0, \quad \alpha_2\,(2 - x_1 - x_2) = 0, \tag{C.24}$$

$$\alpha_1 \geq 0, \quad \alpha_2 \geq 0. \tag{C.25}$$

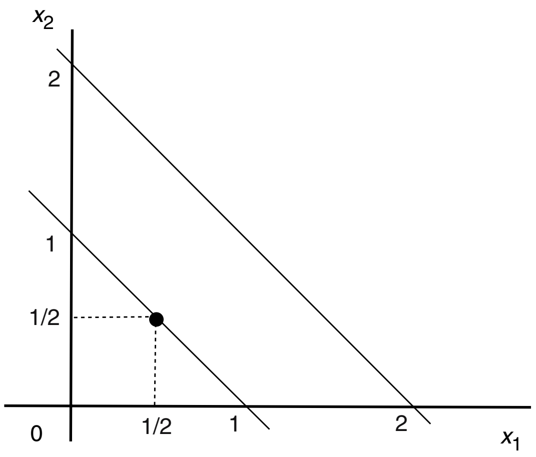


Fig. C.1. Unique solution with a positive definite matrix

So long as $x_1 + x_2$ is constant, the value of the objective function does not change. Thus the optimal solution satisfies $x_1 + x_2 = 1$. Therefore, from (C.23) and (C.24), the optimal solution satisfies

$$x_1 + x_2 = 1, \quad \alpha_1 = 1, \quad \alpha_2 = 0. \tag{C.26}$$

Thus the solution is nonunique (see Fig. C.2).

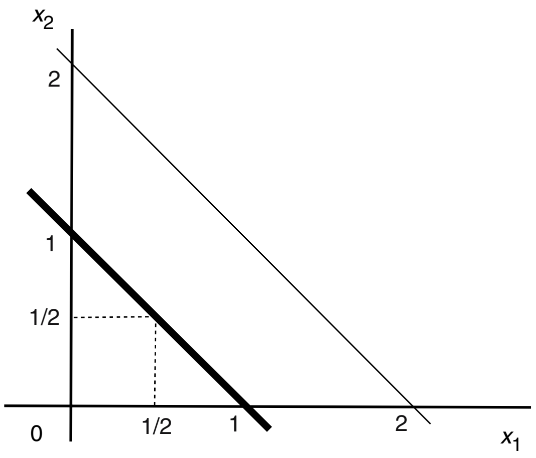


Fig. C.2. Nonunique solution with a positive semidefinite matrix

D

Positive Semidefinite Kernels and Reproducing Kernel Hilbert Space

Support vector machines are based on the theory of reproducing kernel Hilbert space. Here, we summarize some of the properties of positive semidefinite kernels and reproducing kernel Hilbert space based on [30].

D.1 Positive Semidefinite Kernels

Definition D.1. Let $H(\mathbf{x}, \mathbf{x}')$ be a real-valued symmetric function with $\mathbf{x}$ and $\mathbf{x}'$ being m-dimensional vectors. For any set of data $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ and $\mathbf{h}_M = (h_1, \ldots, h_M)^T$ with M being any natural number, if

$$\mathbf{h}_M^T H_M \mathbf{h}_M \geq 0 \tag{D.1}$$

is satisfied (i.e., H_M is a positive semidefinite matrix), we call $H(\mathbf{x}, \mathbf{x}')$ a *positive semidefinite kernel*, where

$$H_M = \begin{pmatrix} H(\mathbf{x}_1, \mathbf{x}_1) & \cdots & H(\mathbf{x}_1, \mathbf{x}_M) \\ \vdots & \ddots & \vdots \\ H(\mathbf{x}_M, \mathbf{x}_1) & \cdots & H(\mathbf{x}_M, \mathbf{x}_M) \end{pmatrix}. \tag{D.2}$$

If (D.1) is satisfied under the constraint

$$\sum_{i=1}^{M} h_i = 0, \tag{D.3}$$

$H(\mathbf{x}, \mathbf{x}')$ is called a *conditionally positive semidefinite kernel.*

From the definition it is obvious that if $H(\mathbf{x}, \mathbf{x}')$ is positive semidefinite, it is also conditionally positive semidefinite. In the following we discuss several properties of (conditionally) positive semidefinite kernels that are useful for constructing positive semidefinite kernels.

Theorem D.2. *If*

$$H(\mathbf{x}, \mathbf{x}') = a, \tag{D.4}$$

where $a > 0$, $H(\mathbf{x}, \mathbf{x}')$ is positive semidefinite.

Proof. Because for any natural number M,

$$H_M = (\sqrt{a}, \ldots, \sqrt{a})^T (\sqrt{a}, \ldots, \sqrt{a}), \tag{D.5}$$

$H(\mathbf{x}, \mathbf{x}')$ is positive semidefinite. ■

Theorem D.3. *If $H_1(\mathbf{x}, \mathbf{x}')$ and $H_2(\mathbf{x}, \mathbf{x}')$ are positive semidefinite kernels,*

$$H(\mathbf{x}, \mathbf{x}') = a_1 H_1(\mathbf{x}, \mathbf{x}') + a_2 H_2(\mathbf{x}, \mathbf{x}') \tag{D.6}$$

is also positive semidefinite, where a_1 and a_2 are positive.

Proof. Because for any M, h_i, and $\mathbf{x}_i$

$$\mathbf{h}_M^T \left(a_1 H_{1M} + a_2 H_{2M}\right) \mathbf{h}_M = a_1 \mathbf{h}_M^T H_{1M} \mathbf{h}_M + a_2 \mathbf{h}_M^T H_{2M} \mathbf{h}_M \geq 0, \tag{D.7}$$

$H(\mathbf{x}, \mathbf{x}')$ is positive semidefinite. ■

Theorem D.4. *If $H(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})\, f(\mathbf{x}')$, where $f(\mathbf{x})$ is an arbitrary scalar function, $H(\mathbf{x}, \mathbf{x}')$ is positive semidefinite.*

Proof. Because for any M, h_i, and $\mathbf{x}_i$

$$\sum_{i,j=1}^{M} h_i\, h_j\, f(\mathbf{x}_i)\, f(\mathbf{x}_j) = \left(\sum_{i=1}^{M} h_i\, f(\mathbf{x}_i)\right)^2 \geq 0, \tag{D.8}$$

$H(\mathbf{x}, \mathbf{x}')$ is positive semidefinite. ■

Theorem D.5. *If $H_1(\mathbf{x}, \mathbf{x}')$ and $H_2(\mathbf{x}, \mathbf{x}')$ are positive semidefinite,*

$$H(\mathbf{x}, \mathbf{x}') = H_1(\mathbf{x}, \mathbf{x}')\, H_2(\mathbf{x}, \mathbf{x}') \tag{D.9}$$

is also positive semidefinite.

Proof. It is sufficient to show that if $M \times M$ matrices $A = \{a_{ij}\}$ and $B = \{b_{ij}\}$ are positive semidefinite, $\{a_{ij}\, b_{ij}\}$ is also positive semidefinite.

Because A is positive semidefinite, A is expressed by $A = F^T F$, where F is an $M \times M$ matrix. Then $a_{ij} = \mathbf{f}_i^T \mathbf{f}_j$, where $\mathbf{f}_j$ is the jth column vector of F. Thus for arbitrary $h_1, \ldots, h_M$,

$$\sum_{i,j=1}^{M} h_i\, h_j\, \mathbf{f}_i^T\, \mathbf{f}_j\, b_{ij} = \sum_{i,j=1}^{M} (h_i\, \mathbf{f}_i)^T (h_j\, \mathbf{f}_j)\, b_{ij} \geq 0. \blacksquare \tag{D.10}$$

Example D.6. The linear kernel $H(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}$ is positive semidefinite because $H_M = (\mathbf{x}_1, \ldots, \mathbf{x}_M)^T (\mathbf{x}_1, \ldots, \mathbf{x}_M)$. Thus, from Theorems D.2 to D.5 the polynomial kernel given by $H(\mathbf{x}, \mathbf{x}') = (1+\mathbf{x}^T \mathbf{x}')^d$ is positive semidefinite.

Corollary D.7. *If $H(\mathbf{x}, \mathbf{x}')$ and $H'(\mathbf{y}, \mathbf{y}')$ are positive semidefinite kernels, where $\mathbf{x}$ and $\mathbf{y}$ may be of different dimensions, $H(\mathbf{x}, \mathbf{x}')\, H'(\mathbf{y}, \mathbf{y}')$ is also a positive semidefinite kernel.*

Corollary D.8. *Let $H(\mathbf{x}, \mathbf{x}')$ be positive semidefinite and satisfy*

$$|H(\mathbf{x}, \mathbf{x}')| \leq \rho, \tag{D.11}$$

where $\rho > 0$. Then if

$$f(y) = \sum_{i=1}^{\infty} a_i\, y^i \tag{D.12}$$

converges for $|y| \leq \rho$, where $a_i \geq 0$ for all integers i, the composed kernel $f(H(\mathbf{x}, \mathbf{x}'))$ is also positive semidefinite. ■

Proof. From Theorem D.5, $H^i(\mathbf{x}, \mathbf{x}')$ is positive semidefinite. Then from Theorem D.5,

$$\sum_{i=0}^{N} a_i\, H^i(\mathbf{x}, \mathbf{x}') \tag{D.13}$$

is positive semidefinite for all integers N. Therefore, so is $f(H(\mathbf{x}, \mathbf{x}'))$. ■

From Corollary D.8, especially for positive semidefinite kernel $H(\mathbf{x}, \mathbf{x}')$, $\exp(H(\mathbf{x}, \mathbf{x}'))$ is also positive semidefinite.

In the following we clarify the relations between positive and conditionally positive semidefinite kernels.

Lemma D.9. *Let*

$$H(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}, \mathbf{x}') + K(\mathbf{x}_0, \mathbf{x}_0) - K(\mathbf{x}, \mathbf{x}_0) - K(\mathbf{x}', \mathbf{x}_0). \tag{D.14}$$

Then $H(\mathbf{x}, \mathbf{x}')$ is positive semidefinite, if and only if $K(\mathbf{x}, \mathbf{x}')$ is conditionally positive semidefinite.

Proof. For $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ and $\mathbf{h}_M = (h_1, \ldots, h_M)^T$ with

$$\sum_{i=1}^{M} h_i = 0, \tag{D.15}$$

we have

$$\mathbf{h}_M^T H_M \mathbf{h}_M = \mathbf{h}_M^T K_M \mathbf{h}_M. \tag{D.16}$$

Thus, if $H(\mathbf{x}, \mathbf{x}')$ is positive semidefinite, $K(\mathbf{x}, \mathbf{x}')$ is conditionally positive semidefinite.

On the other hand, suppose that $K(\mathbf{x}, \mathbf{x}')$ is conditionally positive semidefinite. Then for $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ and $\mathbf{h}_M = (h_1, \ldots, h_M)^T$ with

$$h_0 = -\sum_{i=1}^{M} h_i, \tag{D.17}$$

we have

$$\begin{aligned} 0 &\leq \sum_{i,j=0}^{M} h_i \, h_j \, K(\mathbf{x}_i, \mathbf{x}_j) \\ &= \sum_{i,j=1}^{M} h_i \, h_j \, K(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{M} h_i \, h_0 \, K(\mathbf{x}_i, \mathbf{x}_0) + \sum_{j=1}^{M} h_0 \, h_j \, K(\mathbf{x}_0, \mathbf{x}_j) \\ &\quad + h_0^2 \, K(\mathbf{x}_0, \mathbf{x}_0) \\ &= \sum_{i,j=1}^{M} h_i \, h_j \, H(\mathbf{x}_i, \mathbf{x}_j). \end{aligned} \tag{D.18}$$

Therefore, $H(\mathbf{x}, \mathbf{x}')$ is positive semidefinite. ■

Theorem D.10. *Kernel $K(\mathbf{x}, \mathbf{x}')$ is conditionally positive semidefinite if and only if $\exp(\gamma \, K(\mathbf{x}, \mathbf{x}'))$ is positive semidefinite for any positive γ.*

Proof. If $\exp(\gamma \, K(\mathbf{x}, \mathbf{x}'))$ is positive semidefinite, $\exp(\gamma \, K(\mathbf{x}, \mathbf{x}')) - 1$ is conditionally positive semidefinite. So is the limit

$$K(\mathbf{x}, \mathbf{x}') = \lim_{\gamma \to +0} \frac{\exp(\gamma \, K(\mathbf{x}, \mathbf{x}')) - 1}{\gamma}. \tag{D.19}$$

Now let $K(\mathbf{x}, \mathbf{x}')$ be conditionally positive semidefinite and choose some $\mathbf{x}_0$ and $H(\mathbf{x}, \mathbf{x}')$ as in Lemma D.9. Then for positive γ

$$\gamma K(\mathbf{x}, \mathbf{x}') = \gamma H(\mathbf{x}, \mathbf{x}') - \gamma K(\mathbf{x}_0, \mathbf{x}_0) + \gamma K(\mathbf{x}, \mathbf{x}_0) + \gamma K(\mathbf{x}', \mathbf{x}_0). \tag{D.20}$$

Thus,

$$\begin{aligned} \exp(\gamma K(\mathbf{x}, \mathbf{x}')) &= \exp(\gamma H(\mathbf{x}, \mathbf{x}')) \exp(-\gamma K(\mathbf{x}_0, \mathbf{x}_0)) \\ &\quad \times \exp(\gamma K(\mathbf{x}, \mathbf{x}_0)) \exp(\gamma K(\mathbf{x}', \mathbf{x}_0)). \end{aligned} \tag{D.21}$$

From Theorems D.4 and D.5 and Corollary D.8, $\exp(\gamma K(\mathbf{x}, \mathbf{x}'))$ is positive semidefinite. ■

Example D.11. Kernel $H(\mathbf{x}, \mathbf{x}') = -\|\mathbf{x} - \mathbf{x}'\|^2$ is conditionally positive semidefinite because for $\sum_i^M h_i = 0$,

$$\begin{aligned}
\mathbf{h}_M^T\, H_M\, \mathbf{h}_M &= -\sum_{i=1}^{M} h_i\, h_j\, \|\mathbf{x}_i - \mathbf{x}_j\|^2 \\
&= -\sum_{i,j=1}^{M} h_i\, h_j\, (\mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}_i^T \mathbf{x}_j + \mathbf{x}_j^T \mathbf{x}_j) \\
&= 2 \left(\sum_{i=1}^{M} h_i\, \mathbf{x}_i \right)^T \left(\sum_{i=1}^{M} h_i\, \mathbf{x}_i \right) \geq 0. \qquad \text{(D.22)}
\end{aligned}$$

Thus, $\exp(-\gamma\, \|\mathbf{x} - \mathbf{x}'\|^2)$ is positive semidefinite.

D.2 Reproducing Kernel Hilbert Space

Because a positive semidefinite kernel has the associated feature space called the *reproducing kernel Hilbert space (RKHS)*, support vector machines can determine the optimal hyperplane in that space using the kernel trick. In this section, we discuss reproducing kernel Hilbert spaces for positive and conditionally positive semidefinite kernels.

For the positive semidefinite kernels, the following theorem holds.

Theorem D.12. *Let X be the input space and $H(\mathbf{x}, \mathbf{x}')\,(\mathbf{x}, \mathbf{x}' \in X)$ be a positive semidefinite kernel. Let H_0 be the space spanned by the functions $\{H_{\mathbf{x}} \,|\, \mathbf{x} \in X\}$ where*

$$H_{\mathbf{x}}(\mathbf{x}') = H(\mathbf{x}, \mathbf{x}'). \qquad \text{(D.23)}$$

Then there exist a Hilbert space H, which is a complete space of H_0, and the mapping from X to H such that

$$H(\mathbf{x}, \mathbf{x}') = \langle H_{\mathbf{x}}, H_{\mathbf{x}'} \rangle. \qquad \text{(D.24)}$$

Here, instead of $\mathbf{x}^T \mathbf{x}'$, we use $\langle \mathbf{x}, \mathbf{x}' \rangle$ to denote the dot-product.

Proof. Let $H_{\mathbf{x}}(\mathbf{x}') = H(\mathbf{x}, \mathbf{x}')$ and H_0 be a linear subspace generated by the functions $\{H_{\mathbf{x}} \,|\, \mathbf{x} \in X\}$. Then for $f, g \in H_0$ expressed by

$$f = \sum_{\mathbf{x}_i \in X} c_i\, H_{\mathbf{x}_i}, \qquad \text{(D.25)}$$

$$g = \sum_{\mathbf{x}'_j \in X} d_j\, H_{\mathbf{x}'_j}, \qquad \text{(D.26)}$$

we define the dot-product as follows:

$$\begin{aligned}
\langle f, g \rangle &= \sum_{\mathbf{x}'_j \in X} d_j\, f(\mathbf{x}'_j) \\
&= \sum_{\mathbf{x}_i, \mathbf{x}'_j \in X} c_i\, d_j\, H(\mathbf{x}_i, \mathbf{x}'_j) \\
&= \sum_{\mathbf{x}_i \in X} c_i\, g(\mathbf{x}_i). \qquad \text{(D.27)}
\end{aligned}$$

Now we show that (D.27) satisfies the properties of the dot-product. Clearly, (D.27) is symmetric and linear. Also, according to the assumption of $H(\mathbf{x}, \mathbf{x}')$ being positive semidefinite,

$$\langle f, f \rangle = \sum_{\mathbf{x}_i, \mathbf{x}_j \in X} c_i \, c_j \, H(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \tag{D.28}$$

is satisfied. Here, the strict equality holds if and only if f is identically zero. Thus, (D.27) is the dot-product. Hence, H_0 is a pre-Hilbert space and its completion H is a Hilbert space, which is called *RKHS associated with* $H_\mathbf{x}$.

From (D.27) the following reproducing property is readily obtained:

$$\langle f, H_\mathbf{x} \rangle = f(\mathbf{x}). \tag{D.29}$$

In particular,

$$\langle H_\mathbf{x}, H_{\mathbf{x}'} \rangle = H(\mathbf{x}, \mathbf{x}'). \blacksquare \tag{D.30}$$

For a conditionally positive semidefinite kernel, for $f \in H_0$ the following theorem holds.

Theorem D.13. *Let* $H(\mathbf{x}, \mathbf{x}') \, (\mathbf{x}, \mathbf{x}' \in X)$ *be a conditionally positive semidefinite kernel. Then there exist a Hilbert space* H *and a mapping* $K_\mathbf{x}$ *from* X *to* H *such that*

$$H(\mathbf{x}, \mathbf{x}') - \frac{1}{2} H(\mathbf{x}, \mathbf{x}) - \frac{1}{2} H(\mathbf{x}', \mathbf{x}') = -\|K_\mathbf{x} - K_{\mathbf{x}'}\|^2. \tag{D.31}$$

Proof. For $\mathbf{x}_0$ we define

$$K(\mathbf{x}, \mathbf{x}') = \frac{1}{2} \left(H(\mathbf{x}, \mathbf{x}') + H(\mathbf{x}_0, \mathbf{x}_0) - H(\mathbf{x}, \mathbf{x}_0) - H(\mathbf{x}', \mathbf{x}_0) \right), \tag{D.32}$$

which is a positive semidefinite kernel from Lemma D.9. Let H be the associated RKHS for $K(\mathbf{x}, \mathbf{x}')$ and $K_\mathbf{x}(\mathbf{x}') = K(\mathbf{x}, \mathbf{x}')$. Then

$$\begin{aligned} \|K_\mathbf{x} - K_{\mathbf{x}'}\|^2 &= K(\mathbf{x}, \mathbf{x}) + K(\mathbf{x}', \mathbf{x}') - 2K(\mathbf{x}, \mathbf{x}') \\ &= -H(\mathbf{x}, \mathbf{x}') + \frac{1}{2} H(\mathbf{x}, \mathbf{x}) + \frac{1}{2} H(\mathbf{x}', \mathbf{x}'). \end{aligned} \tag{D.33}$$

Thus the theorem holds. $\blacksquare$

References

1. S. Abe. *Neural Networks and Fuzzy Systems: Theory and Applications.* Kluwer Academic Publishers, Norwell, MA, 1997.
2. S. Abe. Dynamic cluster generation for a fuzzy classifier with ellipsoidal regions. *IEEE Transactions on Systems, Man, and Cybernetics—Part B*, 28(6):869–76, 1998.
3. S. Abe. *Pattern Classification: Neuro-Fuzzy Methods and Their Comparison.* Springer-Verlag, London, 2001.
4. S. Abe. Analysis of support vector machines. In H. Bourlard, T. Adali, S. Bengio, J. Larsen, and S. Douglas, editors, *Neural Networks for Signal Processing XII—Proceedings of the 2002 IEEE Signal Processing Society Workshop*, pages 89–98, 2002.
5. S. Abe. Analysis of multiclass support vector machines. In *Proceedings of International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA 2003)*, pages 385–96, Vienna, Austria, 2003.
6. S. Abe. On invariance of support vector machines. Presented at the Fourth International Symposium on Intelligent Data Engineering and Learning (IDEAL 2003) but not included in the proceedings (http://www2.kobe-u.ac.jp/˜abe/pdf/ideal2003.pdf), 2003.
7. S. Abe. Fuzzy LP-SVMs for multiclass problems. In *Proceedings of the Twelfth European Symposium on Artificial Neural Networks (ESANN 2004)*, pages 429–34, Bruges, Belgium, 2004.
8. S. Abe, Y. Hirokawa, and S. Ozawa. Steepest ascent training of support vector machines. In E. Damiani, L. C. Jain, R. J. Howlett, and N. Ichalkaranje, editors, *Knowledge-Based Intelligent Engineering Systems and Allied Technologies (KES 2002)*, volume 82 Frontiers in Artificial Intelligence and Applications, Part 2, pages 1301–5, IOS Press, Amsterdam, the Netherlands, 2002.
9. S. Abe and T. Inoue. Fast training of support vector machines by extracting boundary data. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Artificial Neural Networks (ICANN 2001)—Proceedings of International Conference, Vienna, Austria*, pages 308–13. Springer-Verlag, Berlin, Germany, 2001.
10. S. Abe and T. Inoue. Fuzzy support vector machines for multiclass problems. In *Proceedings of the Tenth European Symposium on Artificial Neural Networks (ESANN 2002)*, pages 113–8, Bruges, Belgium, 2002.

11. S. Abe and M.-S. Lan. A method for fuzzy rules extraction directly from numerical data and its application to pattern classification. *IEEE Transactions on Fuzzy Systems*, 3(1):18–28, 1995.
12. S. Abe and K. Sakaguchi. Generalization improvement of a fuzzy classifier with ellipsoidal regions. In *Proceedings of the Tenth IEEE International Conference on Fuzzy Systems*, volume 1, pages 207–10, Melbourne, Australia, 2001.
13. S. Abe and R. Thawonmas. A fuzzy classifier with ellipsoidal regions. *IEEE Transactions on Fuzzy Systems*, 5(3):358–68, 1997.
14. E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–41, 2000.
15. S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–9, 1999.
16. S. Amari and S. Wu. An information-geometrical method for improving the performance of support vector machine classifiers. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN '99)*, volume 1, pages 85–90, Edinburgh, UK, 1999.
17. D. Anguita, A. Boni, and S. Pace. Fast training of support vector machines for regression. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN 2000)*, volume 5, pages 210–4, Como, Italy, 2000.
18. D. Anguita, A. Boni, and S. Ridella. Evaluating the generalization ability of support vector machines through the bootstrap. *Neural Processing Letters*, 11(1):51–8, 2000.
19. D. Anguita, S. Ridella, and D. Sterpi. A new method for multiclass support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2004)*, volume 1, pages 407–12, Budapest, Hungary, 2004.
20. C. Angulo, X. Parra, and A. Català. An [*sic*] unified framework for "all data at once" multi-class support vector machines. In *Proceedings of the Tenth European Symposium on Artificial Neural Networks (ESANN 2002)*, pages 161–6, Bruges, Belgium, 2002.
21. J. K. Anlauf and M. Biehl. The Adatron—An adaptive perceptron algorithm. *Europhysics Letters*, 10:687–92, 1989.
22. K. Baba, I. Enbutu, and M. Yoda. Explicit representation of knowledge acquired from plant historical data using neural network. In *Proceedings of 1990 IJCNN International Joint Conference on Neural Networks*, volume 3, pages 155–60, San Diego, 1990.
23. B. Baesens, S. Viaene, T. Van Gestel, J. A. K. Suykens, G. Dedene, B. De Moor, and J. Vanthienen. An empirical assessment of kernel type performance for least squares support vector machine classifiers. In *Proceedings of the Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies (KES 2000)*, volume 1, pages 313–6, Brighton, UK, 2000.
24. T. Ban and S. Abe. Spatially chunking support vector clustering algorithm. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2004)*, volume 1, pages 413–8, Budapest, Hungary, 2004.
25. A. Barla, E. Franceschi, F. Odone, and A. Verri. Image kernels. In S.-W. Lee and A. Verri, editors, *Pattern Recognition with Support Vector Machines: First International Workshop, SVM 2002, Niagara Falls*, pages 83–96. Springer-Verlag, Berlin, Germany, 2002.

26. G. Baudat and F. Anouar. Generalized discriminant analysis using a kernel approach. *Neural Computation*, 12(10):2385–404, 2000.
27. G. Baudat and F. Anouar. Kernel-based methods and function approximation. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '01)*, volume 2, pages 1244–9, Washington, DC, 2001.
28. A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–37, 2001.
29. K. P. Bennett. Combining support vector and mathematical programming methods for classification. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 307–26. MIT Press, Cambridge, MA, 1999.
30. C. Berg, J. P. R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups: Theory of Positive Definite and Related Functions*. Springer-Verlag, New York, 1984.
31. D. P. Bertsekas. *Nonlinear Programming, second edition*. Athena Scientific, Belmont, MA, 1999.
32. J. C. Bezdek, J. M. Keller, R. Krishnapuram, L. I. Kuncheva, and N. R. Pal. Will the real iris data please stand up? *IEEE Transactions on Fuzzy Systems*, 7(3):368–9, 1999.
33. J. C. Bezdek, J. Keller R. Krisnapuram, and N. R. Pal. *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Kluwer Academic Publishers, Norwell, MA, 1999.
34. J. Bi, K. P. Bennett, M. Embrechts, C. Breneman, and M. Song. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, 3:1229–43, 2003.
35. C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK, 1995.
36. S. Borer and W. Gerstner. Support vector representation of multi-categorical data. In J. R. Dorronsoro, editor, *Artificial Neural Networks (ICANN 2002)—Proceedings of International Conference, Madrid, Spain*, pages 733–8. Springer-Verlag, Berlin, Germany, 2002.
37. P. S. Bradley and O. L. Mangasarian. Feature selection via concave minimization and support vector machines. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)*, pages 82–90, Madison, 1998.
38. P. S. Bradley and O. L. Mangasarian. Massive data discrimination via linear support vector machines. *Optimization Methods and Software*, 13(1):1–10, 2000.
39. V. L. Brailovsky, O. Barzilay, and R. Shahave. On global, local, mixed and neighborhood kernels for support vector machines. *Pattern Recognition Letters*, 20(11–13):1183–90, 1999.
40. E. J. Bredensteiner and K. P. Bennett. Multicategory classification by support vector machines. *Computational Optimization and Applications*, 12(1–3):53–79, 1999.
41. M. Brown. Exploring the set of sparse, optimal classifiers. In *Proceedings of Artificial Neural Networks in Pattern Recognition (ANNPR 2003)*, pages 178–84, Florence, Italy, 2003.
42. M. Brown, N. P. Costen, and S. Akamatsu. Efficient calculation of the complete optimal classification set. In *Proceedings of the Seventeenth International Conference on Pattern Recognition (ICPR 2004)*, volume 2, pages 307–10, Cambridge, UK, 2004.

43. C. J. C. Burges. Simplified support vector decision rules. In L. Saitta, editor, *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96)*, pages 71–7. Morgan Kaufmann, San Francisco, 1996.
44. C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–67, 1998.
45. C. J. C. Burges. Geometry and invariance in kernel based methods. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 89–116. MIT Press, Cambridge, MA, 1999.
46. C. J. C. Burges and D. J. Crisp. Uniqueness of the SVM solution. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 223–9. MIT Press, Cambridge, MA, 2000.
47. C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–81, 1997.
48. C. Campbell, T.-T. Frieß, and N. Cristianini. Maximal margin classification using the KA algorithm. In *Proceedings of the First International Symposium on Intelligent Data Engineering and Learning (IDEAL '98)*, pages 355–62, Hong Kong, China, 1998.
49. D. Caragea, D. Cook, and V. Honavar. Towards simple, easy-to-understand, yet accurate classifiers. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM 2003)*, pages 497–500, Melbourne, FL, 2003.
50. G. L. Cash and M. Hatamian. Optical character recognition by the method of moments. *Computer Vision, Graphics, and Image Processing*, 39(3):291–310, 1987.
51. G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 409–15. MIT Press, Cambridge, MA, 2000.
52. G. C. Cawley and N. L. C. Talbot. Manipulation of prior probabilities in support vector classification. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '01)*, volume 4, pages 2433–8, Washington, DC, 2001.
53. G. C. Cawley and N. L. C. Talbot. Efficient formation of a basis in a kernel feature space. In *Proceedings of the Tenth European Symposium on Artificial Neural Networks (ESANN 2002)*, pages 1–6, Bruges, Belgium, 2002.
54. G. C. Cawley and N. L. C. Talbot. A greedy training algorithm for sparse least-squares support vector machines. In J. R. Dorronsoro, editor, *Artificial Neural Networks (ICANN 2002)—Proceedings of International Conference, Madrid, Spain*, pages 681–6. Springer-Verlag, Berlin, Germany, 2002.
55. G. C. Cawley and N. L. C. Talbot. Efficient model selection for kernel logistic regression. In *Proceedings of the Seventeenth International Conference on Pattern Recognition (ICPR 2004)*, volume 2, pages 439–42, Cambridge, UK, 2004.
56. O. Chapelle and V. Vapnik. Model selection for support vector machines. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 230–6. MIT Press, Cambridge, MA, 2000.

57. J.-H. Chen. M-estimator based robust kernels for support vector machines. In *Proceedings of the Seventeenth International Conference on Pattern Recognition (ICPR 2004)*, volume 1, pages 168–71, Cambridge, UK, 2004.
58. S. Chen, S. R. Gunn, and C. J. Harris. The relevance vector machine technique for channel equalization application. *IEEE Transactions on Neural Networks*, 12(6):1529–32, 2001.
59. S. Chen, S. R. Gunn, and C. J. Harris. Errata to "The relevance vector machine technique for channel equalization application." *IEEE Transactions on Neural Networks*, 13(4):1024, 2002.
60. V. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory, and Methods.* John Wiley & Sons, New York, 1998.
61. S. L. Chiu. Fuzzy model identification based on cluster estimation. *Journal of Intelligent and Fuzzy Systems*, 2:267–78, 1994.
62. C. S. Chu, I. W. Tsang, and J. T. Kwok. Scaling up support vector data description by using core-sets. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2004)*, volume 1, pages 425–31, Budapest, Hungary, 2004.
63. V. Chvátal. *Linear Programming.* W. H. Freeman and Company, New York, 1983.
64. C. Cortes, P. Haffner, and M. Mohri. Rational kernels. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 601–8. MIT Press, Cambridge, MA, 2003.
65. K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory (COLT 2000)*, pages 35–46, Palo Alto, CA, 2000.
66. K. Crammer and Y. Singer. Improved output coding for classification using continuous relaxation. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 437–43. MIT Press, Cambridge, MA, 2001.
67. K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–91, 2003.
68. N. Cristianini and C. Campbell. Dynamically adapting kernels in support vector machines. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 204–10. MIT Press, Cambridge, MA, 1999.
69. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* Cambridge University Press, Cambridge, UK, 2000.
70. R. S. Crowder. Predicting the Mackey-Glass time series with cascade-correlation learning. In *Proceedings of 1990 Connectionist Models Summer School*, pages 117–23, Carnegie Mellon University, 1990.
71. M. B. de Almeida, A. de Pádua Braga, and J. P. Braga. SVM-KM: Speeding SVMs learning with a priori cluster selection and k-means. In *Proceedings of the Sixth Brazilian Symposium on Neural Networks (SBRN 2000)*, volume 1, pages 162–7, Rio de Janeiro, Brazil, 2000.
72. N. de Freitas, M. Milo, P. Clarkson, M. Niranjan, and A. Gee. Sequential support vector machines. In *Neural Networks for Signal Processing IX—Proceedings of the 1999 IEEE Signal Processing Society Workshop*, pages 31–40, 1999.

73. T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–86, 1995.
74. T. Downs, K. E. Gates, and A. Masters. Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–7, 2001.
75. P. M. L. Drezet and R. F. Harrison. A new method for sparsity control in support vector classification and regression. *Pattern Recognition*, 34(1):111–25, 2001.
76. K. Duan, S. S. Keerthi, and A. N. Poo. An empirical evaluation of simple performance measures for tuning SVM hyperparameters. In *Proceedings of the Eighth International Conference on Neural Information Processing (ICONIP-2001)*, Paper ID# 159, Shanghai, China, 2001.
77. R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York, 1973.
78. B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap.* Chapman & Hall/CRC Press, Boca Raton, FL, 1993.
79. T. Evgeniou, M. Pontil, C. Papageorgiou, and T. Poggio. Image representations for object detection using kernel classifiers. In *Proceedings of Asian Conference on Computer Vision (ACCV 2000)*, pages 687–92, Taipei, Taiwan, 2000.
80. T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.
81. J. Feng and P. Williams. The generalization error of the symmetric and scaled support vector machines. *IEEE Transactions on Neural Networks*, 12(5):1255–60, 2001.
82. R. Fernández. Behavior of the weights of a support vector machine as a function of the regularization parameter C. In *Proceedings of the Eighth International Conference on Artificial Neural Networks (ICANN '98)*, volume 2, pages 917–22, Skövde, Sweden, 1998.
83. S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–64, 2001.
84. R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–88, 1936.
85. P. Frasconi, A. Passerini, and A. Vullo. A two-stage SVM architecture for predicting the disulfide bonding state of cysteines. In H. Bourlard, T. Adali, S. Bengio, J. Larsen, and S. Douglas, editors, *Neural Networks for Signal Processing XII—Proceedings of the 2002 IEEE Signal Processing Society Workshop*, pages 25–34, 2002.
86. Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–96, 1999.
87. F. Friedrichs and C. Igel. Evolutionary tuning of multiple SVM parameters. In *Proceedings of the Twelfth European Symposium on Artificial Neural Networks (ESANN 2004)*, pages 519–24, Bruges, Belgium, 2004.
88. T.-T. Frieß, N. Cristianini, and C. Campbell. The kernel-Adatron algorithm: A fast and simple learning procedure for support vector machines. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)*, pages 188–96, Madison, 1998.
89. X. Fu, C.-J. Ong, S. Keerthi, G. G. Hung, and L. Goh. Extracting the knowledge embedded in support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2004)*, volume 1, pages 291–6, Budapest, Hungary, 2004.

90. K. Fukunaga. *Introduction to Statistical Pattern Recognition, second edition.* Academic Press, San Diego, 1990.
91. G. Fumera and F. Roli. Support vector machines with embedded reject option. In S.-W. Lee and A. Verri, editors, *Pattern Recognition with Support Vector Machines: First International Workshop, SVM 2002, Niagara Falls*, pages 68–82. Springer-Verlag, Berlin, Germany, 2002.
92. A. Gammerman, V. Vovk, and V. Vapnik. Learning by transduction. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI '98)*, pages 148–55, Madison, 1998.
93. C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–42, 2001.
94. T. Van Gestel, J. A. K. Suykens, J. De Brabanter, B. De Moor, and J. Vandewalle. Least squares support vector machine regression for discriminant analysis. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '01)*, volume 4, pages 2445–50, Washington, DC, 2001.
95. M. Girolami. Mercer kernel-based clustering in feature space. *IEEE Transactions on Neural Networks*, 13(3):780–4, 2002.
96. G. H. Golub and C. F. Van Loan. *Matrix Computations, third edition.* The Johns Hopkins University Press, Baltimore, 1996.
97. E. Gose, R. Johnsonbaugh, and S. Jost. *Pattern Recognition and Image Analysis.* Prentice Hall, Upper Saddle River, NJ, 1996.
98. T. Graepel, R. Herbrich, B. Schölkopf, A. Smola, P. Bartlett, K.-R. Müller, K. Obermayer, and R. Williamson. Classification on proximity data with LP-machines. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN '99)*, volume 1, pages 304–9, Edinburgh, UK, 1999.
99. Y. Grandvalet and S. Canu. Adaptive scaling for feature selection in SVMs. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 569–76. MIT Press, Cambridge, MA, 2003.
100. Y. Guermeur, A. Elisseeff, and H. Paugam-Moisy. A new multi-class SVM based on a uniform convergence result. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN 2000)*, volume 4, pages 183–8, Como, Italy, 2000.
101. S. R. Gunn. Support vector machines for classification and regression. Technical Report ISIS-1-98, Department of Electronics and Computer Science, University of Southampton, 1998.
102. S. R. Gunn and M. Brown. SUPANOVA: A sparse, transparent modelling approach. In *Neural Networks for Signal Processing IX—Proceedings of the 1999 IEEE Signal Processing Society Workshop*, pages 21–30, 1999.
103. G. Guo, S. Z. Li, and K. L. Chan. Support vector machines for face recognition. *Image and Vision Computing*, 19(9–10):631–8, 2001.
104. I. Guyon and D. G. Stork. Linear discriminant and support vector classifiers. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 147–69. MIT Press, Cambridge, MA, 2000.
105. I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1–3):389–422, 2002.

106. A. Hashizume, J. Motoike, and R. Yabe. Fully automated blood cell differential system and its application. In *Proceedings of the IUPAC Third International Congress on Automation and New Technology in the Clinical Laboratory*, pages 297–302, Kobe, Japan, 1988.
107. T. Hastie and R. Tibshirani. Classification by pairwise coupling. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 507–13. MIT Press, Cambridge, MA, 1998.
108. S. Haykin. *Neural Networks: A Comprehensive Foundation, second edition.* Prentice Hall, Upper Saddle River, NJ, 1999.
109. R. Herbrich. *Learning Kernel Classifiers: Theory and Algorithms.* MIT Press, Cambridge, MA, 2002.
110. R. Herbrich and J. Weston. Adaptive margin support vector machines for classification. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN '99)*, volume 2, pages 880–5, Edinburgh, UK, 1999.
111. Y. Hirokawa and S. Abe. Training of support vector regressors based on the steepest ascent method. In *Proceedings of the Ninth International Conference on Neural Information Processing (ICONIP '02)*, volume 2, pages 552–5, Singapore, 2002.
112. Y. Hirokawa and S. Abe. Steepest ascent training of support vector regressors. *IEEJ Transactions of Electronics, Information and Systems*, 124(10):2066–73, 2004 (in Japanese).
113. K. Hotta. Support vector machine with local summation kernel for robust face recognition. In *Proceedings of the Seventeenth International Conference on Pattern Recognition (ICPR 2004)*, volume 3, pages 482–5, Cambridge, UK, 2004.
114. C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–25, 2002.
115. T. M. Huang and V. Kecman. Bias term b in SVMs again. In *Proceedings of the Twelfth European Symposium on Artificial Neural Networks (ESANN 2004)*, pages 441–8, Bruges, Belgium, 2004.
116. T. Inoue and S. Abe. Fuzzy support vector machines for pattern classification. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '01)*, volume 2, pages 1449–54, Washington, DC, 2001.
117. T. Inoue and S. Abe. Improvement of generalization ability of multiclass support vector machines by introducing fuzzy logic and Bayes theory. *Transactions of the Institute of Systems, Control and Information Engineers*, 15(12):643–51, 2002 (in Japanese).
118. K. Ito and R. Nakano. Optimizing support vector regression hyperparameters based on cross-validation. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2003)*, volume 3, pages 2077–82, Portland, OR, 2003.
119. J.-S. R. Jang. ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):665–85, 1993.
120. Jayadeva, A. K. Deb, and S. Chandra. Binary classification by SVM based tree type neural networks. In *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, volume 3, pages 2773–8, Honolulu, 2002.
121. J.-T. Jeng and C.-C. Chuang. A novel approach for the hyperparameters of support vector regression. In *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, volume 1, pages 642–7, Honolulu, 2002.

122. T. Joachims. Estimating the generalization performance of an SVM efficiently. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pages 431–8, Stanford, CA, 2000.
123. T. Joachims. *Learning to Classify Text Using Support Vector Machines: Methods, Theory and Algorithms*. Kluwer Academic Publishers, Norwell, MA, 2002.
124. E. M. Jordaan and G. F. Smits. Robust outlier detection using SVM for regression. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2004)*, volume 3, pages 2017–22, Budapest, Hungary, 2004.
125. A. Juneja and C. Espy-Wilson. Speech segmentation using probabilistic phonetic feature hierarchy and support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2003)*, volume 1, pages 675–9, Portland, OR, 2003.
126. K. Kaieda and S. Abe. A kernel fuzzy classifier with ellipsoidal regions. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2003)*, volume 3, pages 2043–8, Portland, OR, 2003.
127. K. Kaieda and S. Abe. KPCA-based training of a kernel fuzzy classifier with ellipsoidal regions. *International Journal of Approximate Reasoning*, 37(3):145–253, 2004.
128. V. Kecman, T. Arthanari, and I. Hadzic. LP and QP based learning from empirical data. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '01)*, volume 4, pages 2451–5, Washington, DC, 2001.
129. V. Kecman and I. Hadzic. Support vectors selection by linear programming. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN 2000)*, volume 5, pages 193–8, Como, Italy, 2000.
130. V. Kecman, M. Vogt, and T. M. Huang. On the equality of kernel AdaTron and sequential minimal optimization in classification and regression tasks and alike algorithms for kernel machines. In *Proceedings of the Eleventh European Symposium on Artificial Neural Networks (ESANN 2003)*, pages 215–22, Bruges, Belgium, 2003.
131. S. S. Keerthi and E. G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46:351–60, 2002.
132. S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE Transactions on Neural Networks*, 11(1):124–36, 2000.
133. S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13:637–49, 2001.
134. B. Kijsirikul and N. Ussivakul. Multiclass support vector machines using adaptive directed acyclic graph. In *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, volume 1, pages 980–5, Honolulu, 2002.
135. T. Kikuchi and S. Abe. Error correcting output codes vs. fuzzy support vector machines. In *Proceedings of Artificial Neural Networks in Pattern Recognition (ANNPR 2003)*, pages 192–6, Florence, Italy, 2003.
136. T. Kikuchi and S. Abe. Error correcting output codes vs. fuzzy support vector machines. *Pattern Recognition Letters* (to appear).
137. Y. Koshiba. Acceleration of training of support vector machines. Master's thesis, Graduate School of Science and Technology, Kobe University, Japan, 2004 (in Japanese).

138. Y. Koshiba and S. Abe. Comparison of L1 and L2 support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2003)*, volume 3, pages 2054–9, Portland, OR, 2003.
139. Z. Kou, J. Xu, X. Zhang, and L. Ji. An improved support vector machine using class-median vectors. In *Proceedings of the Eighth International Conference on Neural Information Processing (ICONIP-2001)*, Paper ID# 60, Shanghai, China, 2001.
140. U. H.-G. Kreßel. Pairwise classification and support vector machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 255–68. MIT Press, Cambridge, MA, 1999.
141. M.-S. Lan, H. Takenaga, and S. Abe. Character recognition using fuzzy rules extracted from data. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, volume 1, pages 415–20, Orlando, 1994.
142. G. Lebrun, C. Charrier, and H. Cardot. SVM training time reduction using vector quantization. In *Proceedings of the Seventeenth International Conference on Pattern Recognition (ICPR 2004)*, volume 1, pages 160–3, Cambridge, UK, 2004.
143. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–324, 1998.
144. K. K. Lee, S. R. Gunn, C. J. Harris, and P. A. S. Reed. Classification of imbalanced data with transparent kernels. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '01)*, volume 4, pages 2410–5, Washington, DC, 2001.
145. C. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for SVM protein classification. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1441–8. MIT Press, Cambridge, MA, 2003.
146. H. Li, T. Jiang, and K. Zhang. Efficient and robust feature extraction by maximum margin criterion. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 97–104. MIT Press, Cambridge, MA, 2004.
147. Y. Li and P. M. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46(1–3):361–87, 2002.
148. Z. Li and S. Tang. Face recognition using improved pairwise coupling support vector machines. In *Proceedings of the Ninth International Conference on Neural Information Processing (ICONIP '02)*, #1288, Singapore, 2002.
149. S.-P. Liao, H.-T. Lin, and C.-J. Lin. A note on the decomposition methods for support vector regression. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '01)*, volume 2, pages 1474–9, Washington, DC, 2001.
150. C. Ap. M. Lima, A. L. V. Coelho, and F. J. Von Zuben. Ensembles of support vector machines for regression problems. In *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, volume 3, pages 2381–6, Honolulu, 2002.
151. C.-F. Lin and S.-D. Wang. Fuzzy support vector machines. *IEEE Transactions on Neural Networks*, 13(2):464–71, 2002.
152. C.-J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks*, 12(6):1288–98, 2001.

153. C.-J. Lin. Asymptotic convergence of an SMO algorithm without any assumptions. *IEEE Transactions on Neural Networks*, 13(1):248–50, 2002.
154. C.-J. Lin. Errata to"A Comparison of Methods for Multiclass Support Vector Machines." *IEEE Transactions on Neural Networks*, 13(4):1026–7, 2002.
155. H. Lodhi, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 563–9, 2001.
156. B.-L. Lu, K.-A. Wang, M. Utiyama, and H. Isahara. A part-versus-part method for massively parallel training of support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2004)*, volume 1, pages 735–40, Budapest, Hungary, 2004.
157. L. Lukas, A. Devos, J. A. K. Suykens, L. Vanhamme, S. Van Huffel, A. R. Tate, C. Majós, and C. Arús. The use of LS-SVM in the classification of brain tumors based on magnetic resonance spectroscopy signals. In *Proceedings of the Tenth European Symposium on Artificial Neural Networks (ESANN 2002)*, pages 131–6, Bruges, Belgium, 2002.
158. J. Ma and S. Perkins. Time-series novelty detection using one-class support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2003)*, volume 3, pages 1741–5, Portland, OR, 2003.
159. E. Maeda and H. Murase. Kernel based nonlinear subspace method for pattern recognition. *Transactions of the Institute of Electronics, Information and Communication Engineers D-II*, J82D-II(4):600–12, 1999 (in Japanese).
160. O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, 10(5):1032–7, 1999.
161. D. Martinez and G. Millerioux. Support vector committee machines. In *Proceedings of the Eleventh European Symposium on Artificial Neural Networks (ESANN 2000)*, pages 43–8, Bruges, Belgium, 2000.
162. F. Masulli and G. Valentini. Comparing decomposition methods for classification. In *Proceedings of the Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies (KES 2000)*, volume 2, pages 788–91, Brighton, UK, 2000.
163. D. Mattera, F. Palmieri, and S. Haykin. An explicit algorithm for training support vector machines. *IEEE Signal Processing Letters*, 6(9):243–5, 1999.
164. D. Mattera, F. Palmieri, and S. Haykin. Simple and robust methods for support vector expansions. *IEEE Transactions on Neural Networks*, 10(5):1038–47, 1999.
165. E. Mayoraz and E. Alpaydin. Support vector machines for multi-class classification. In J. Mira and J. V. Sanchez-Andres, editors, *Engineering Applications of Bio-Inspired Artificial Neural Networks (IWANN'99)—Proceedings of International Work—Conference on Artificial and Natural Neural Networks, Alicante, Spain*, volume 2, pages 833–42, 1999.
166. S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller. Fisher discriminant analysis with kernels. In Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, *Neural Networks for Signal Processing IX—Proceedings of the 1999 IEEE Signal Processing Society Workshop*, pages 41–8, 1999.
167. S. Mika, B. Schölkopf, A. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. Kernel PCA and de-noising in feature spaces. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 536–42. MIT Press, Cambridge, MA, 1999.

168. S. Miyamoto and D. Suizu. Fuzzy c-means clustering using transformations into high dimensional spaces. In *Proceedings of the First International Conference on Fuzzy Systems and Knowledge Discovery (FSKD '02)*, volume 2, pages 656–60, Singapore, 2002.
169. M. Moreira and E. Mayoraz. Improved pairwise coupling classification with correcting classifiers. In *Proceedings of the Tenth European Conference on Machine Learning (ECML-98)*, pages 160–71, Chemnitz, Germany, 1998.
170. K. Morikawa. Pattern classification and function approximation by kernel least squares. Bachelor's thesis, Electrical and Electronics Engineering, Kobe University, Japan, 2004 (in Japanese).
171. S. Mukherjee, E. Osuna, and F. Girosi. Nonlinear prediction of chaotic time series using support vector machines. In *Neural Networks for Signal Processing VII—Proceedings of the 1997 IEEE Signal Processing Society Workshop*, pages 511–20, 1997.
172. S. Mukherjee, P. Tamayo, D. Slonim, A. Verri, T. Golub, J. P. Mesirov, and T. Poggio. Support vector machine classification of microarray data. Technical Report AI Memo 1677, Massachusetts Institute of Technology, 1999.
173. K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001.
174. K.-R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Artificial Neural Networks (ICANN '97)—Proceedings of the Seventh International Conference, Lausanne, Switzerland*, pages 999–1004. Springer-Verlag, Berlin, Germany, 1997.
175. C. Nakajima, M. Pontil, and T. Poggio. People recognition and pose estimation in image sequences. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2000)*, volume IV, pages 189–94, Como, Italy, 2000.
176. H. Nakayama and T. Asada. Support vector machines using multi objective programming and goal programming. In *Proceedings of the Ninth International Conference on Neural Information Processing (ICONIP '02)*, volume 2, pages 1053–7, Singapore, 2002.
177. A. Navia-Vázquez, F. Pérez-Cruz, A. Artés-Rodríguez, and A. R. Figueiras-Vidal. Weighted least squares training of support vector classifiers leading to compact and adaptive schemes. *IEEE Transactions on Neural Networks*, 12(5):1047–59, 2001.
178. T. Nishikawa and S. Abe. Maximizing margins of multilayer neural networks. In *Proceedings of the Ninth International Conference on Neural Information Processing (ICONIP '02)*, volume 1, pages 322–6, Singapore, 2002.
179. H. Núñez, C. Angulo, and Català. Rule extraction from support vector machines. In *Proceedings of the Tenth European Symposium on Artificial Neural Networks (ESANN 2002)*, pages 107–12, Bruges, Belgium, 2002.
180. C. S. Ong and A. J. Smola. Machine learning using hyperkernels. In T. Fawcett and N. Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), Washington, DC*, pages 568–75. AAAI Press, Menlo Park, CA, 2003.
181. C. S. Ong, A. J. Smola, and R. C. Williamson. Hyperkernels. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 495–502. MIT Press, Cambridge, MA, 2003.

182. 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 Signal Processing Society Workshop*, pages 276–85, 1997.
183. S. K. Pal and S. Mitra. *Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing*. John Wiley & Sons, New York, 1999.
184. C. H. Park and H. Park. Efficient nonlinear dimension reduction for clustered data using kernel functions. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM 2003)*, pages 243–50, Melbourne, FL, 2003.
185. J. P. Pedroso and N. Murata. Optimisation on support vector machines. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN 2000)*, volume 6, pages 399–404, Como, Italy, 2000.
186. F. Pérez-Cruz and A. Artés-Rodríguez. Puncturing multi-class support vector machines. In J. R. Dorronsoro, editor, *Artificial Neural Networks (ICANN 2002)—Proceedings of International Conference, Madrid, Spain*, pages 751–6. Springer-Verlag, Berlin, Germany, 2002.
187. F. Pérez-Cruz, G. Camps-Valls, E. Soria-Olivas, J. J. Pérez-Ruixo, A. R. Figueiras-Vidal, and A. Artés-Rodríguez. Multi-dimensional function approximation and regression estimation. In J. R. Dorronsoro, editor, *Artificial Neural Networks (ICANN 2002)—Proceedings of International Conference, Madrid, Spain*, pages 757–62. Springer-Verlag, Berlin, Germany, 2002.
188. S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3:1333–56, 2003.
189. T. Phetkaew, B. Kijsirikul, and W. Rivepiboon. Reordering adaptive directed acyclic graphs: An improved algorithm for multiclass support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2003)*, volume 2, pages 1605–10, Portland, OR, 2003.
190. J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 185–208. MIT Press, Cambridge, MA, 1999.
191. J. C. Platt. Probabilities for SV machines. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–73. MIT Press, Cambridge, MA, 2000.
192. J. C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAGs for multiclass classification. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 547–53. MIT Press, Cambridge, MA, 2000.
193. M. Pontil and A. Verri. Properties of support vector machines. *Neural Computation*, 10(4):955–74, 1998.
194. M. Pontil and A. Verri. Support vector machines for 3-D object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(6):637–46, 1998.
195. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing, second edition*. Cambridge University Press, Cambridge, UK, 1992.
196. T. Raicharoen and C. Lursinsap. Critical support vector machine without kernel function. In *Proceedings of the Ninth International Conference on Neural*

Information Processing (ICONIP '02), volume 5, pages 2532–6, Singapore, 2002.

197. A. Rakotomamonjy. Variable selection using SVM-based criteria. *Journal of Machine Learning Research*, 3:1357–70, 2003.
198. L. Ralaivola and F. d'Alché-Buc. Incremental support vector machine learning: A local approach. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Artificial Neural Networks (ICANN 2001)—Proceedings of International Conference, Vienna, Austria*, pages 322–30. Springer-Verlag, Berlin, Germany, 2001.
199. G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3):287–320, 2001.
200. G. Rätsch, A. J. Smola, and S. Mika. Adapting codes and embeddings for polychotomies. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 529–36. MIT Press, Cambridge, MA, 2003.
201. R. M. Rifkin, M. Pontil, and A. Verri. A note on support vector machine degeneracy. In O. Watanabe and T. Yokomori, editors, *Proceedings of the Tenth International Conference on Algorithmic Learning Theory (ALT '99), Tokyo, Japan*, pages 252–63. Springer-Verlag, Berlin, Germany, 1999.
202. B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, UK, 1996.
203. D. Roobaert. DirectSVM: A fast and simple support vector machine perceptron. In *Neural Networks for Signal Processing X—Proceedings of the 2000 IEEE Signal Processing Society Workshop*, volume 1, pages 356–65, 2000.
204. R. Rosipal, M. Girolami, and L. J. Trejo. Kernel PCA feature extraction of event-related potentials for human signal detection performance. In H. Malmgren, M. Borga, and L. Niklasson, editors, *Artificial Neural Networks in Medicine and Biology—Proceedings of the ANNIMAB-1 Conference, Göteborg, Sweden*, pages 321–6. Springer-Verlag, Berlin, Germany, 2000.
205. R. Rosipal, M. Girolami, L. J. Trejo, and A. Cichocki. Kernel PCA for feature extraction and de-noising in nonlinear regression. *Neural Computing & Applications*, 10(3):231–43, 2001.
206. A. Ruiz and P. E. López de Teruel. Nonlinear kernel-based statistical pattern analysis. *IEEE Transactions on Neural Networks*, 12(1):16–32, 2001.
207. M. Rychetsky, S. Ortmann, M. Ullmann, and M. Glesner. Accelerated training of support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '99)*, volume 2, pages 998–1003, Washington, DC, 1999.
208. K. Saadi, G. C. Cawley, and L. C. Talbot. Fast exact leave-one-out cross-validation of least-squares support vector machines. In *Proceedings of the Tenth European Symposium on Artificial Neural Networks (ESANN 2002)*, pages 149–54, Bruges, Belgium, 2002.
209. K. Saadi, N. L. C. Talbot, and G. C. Cawley. Optimally regularised kernel fisher discriminant analysis. In *Proceedings of the Seventeenth International Conference on Pattern Recognition (ICPR 2004)*, volume 2, pages 427–30, Cambridge, UK, 2004.
210. C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A. Smola. Support vector machine reference manual. Technical Report CSD-TR-98-03, Royal Holloway, University of London, London, 1998.

211. B. Schölkopf, P. Bartlett, A. Smola, and R. Williamson. Support vector regression with automatic accuracy control. In *Proceedings of the Eighth International Conference on Artificial Neural Networks (ICANN '98)*, volume 2, pages 111–6, Skövde, Sweden, 1998.
212. B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors. *Advances in Kernel Methods: Support Vector Learning*. MIT Press, Cambridge, MA, 1999.
213. B. Schölkopf, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Kernel-dependent support vector error bounds. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN '99)*, volume 1, pages 103–8, Edinburgh, UK, 1999.
214. B. Schölkopf, P. Simard, A. Smola, and V. Vapnik. Prior knowledge in support vector kernels. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 640–6. MIT Press, Cambridge, MA, 1998.
215. B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, 2002.
216. B. Schölkopf, A. J. Smola, and K.-R. Müller. Kernel principal component analysis. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 327–52. MIT Press, Cambridge, MA, 1999.
217. A. Schwaighofer and V. Tresp. The Bayesian committee support vector machine. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Artificial Neural Networks (ICANN 2001)—Proceedings of International Conference, Vienna, Austria*, pages 411–20. Springer-Verlag, Berlin, Germany, 2001.
218. F. Schwenker. Hierarchical support vector machines for multi-class pattern recognition. In *Proceedings of the Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies (KES 2000)*, volume 2, pages 561–5, Brighton, UK 2000.
219. F. Schwenker. Solving multi-class pattern recognition problems with tree structured support vector machines. In B. Radig and S. Florczyk, editors, *Pattern Recognition 2001*, pages 283–90. Springer-Verlag, Berlin, Germany, 2001.
220. M. Seeger. Bayesian model selection for support vector machines, Gaussian processes and other kernel classifiers. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 603–9. MIT Press, Cambridge, MA, 2000.
221. X. Shao and V. Cherkassky. Multi-resolution support vector machine. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '99)*, volume 2, pages 1065–70, Washington, DC, 1999.
222. A. Shilton, M. Palaniswami, D. Ralph, and A. C. Tsoi. Incremental training of support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '01)*, Washington, DC, 2001.
223. H. Shimodaira, K. Noma, M. Nakai, and S. Sagayama. Dynamic time-alignment kernel in support vector machine. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, volume 2, pages 921–8, MIT Press, Cambridge, MA, 2002.
224. T. Shimozaki, T. Takigawa, and S. Abe. A fuzzy classifier with polyhedral regions. *Transactions of the Institute of Systems, Control and Information Engineers*, 14(7):365–72, 2001 (in Japanese).
225. H. Shin and S. Cho. How many neighbors to consider in pattern pre-selection for support vector classifiers? In *Proceedings of International Joint Conference*

on Neural Networks (IJCNN 2003), volume 1, pages 565–70, Portland, OR, 2003.

226. P. K. Simpson. Fuzzy min-max neural networks—Part 1: Classification. *IEEE Transactions on Neural Networks*, 3(5):776–86, 1992.
227. N. Smith and M. Gales. Speech recognition using SVMs. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, volume 2, pages 1197–204, MIT Press, Cambridge, MA, 2002.
228. G. F. Smits and E. M. Jordaan. Improved SVM regression using mixtures of kernels. In *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, volume 3, pages 2785–90, Honolulu, 2002.
229. A. Smola, B. Schölkopf, and G. Rätsch. Linear programs for automatic accuracy control in regression. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN '99)*, volume 2, pages 575–80, Edinburgh, UK, 1999.
230. A. J. Smola, O. L. Mangasarian, and B. Schölkopf. Sparse kernel feature analysis. Technical Report 99-04, University of Wisconsin, Data Mining Institute, Madison, 1999.
231. A. J. Smola, N. Murata, B. Schölkopf, and K.-R. Müller. Asymptotically optimal choice of ε-loss for support vector machines. In *Proceedings of the Eighth International Conference on Artificial Neural Networks (ICANN '98)*, volume 1, pages 105–10, Skövde, Sweden, 1998.
232. S. Sohn and C. H. Dagli. Advantages of using fuzzy class memberships in self-organizing map and support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '01)*, volume 3, pages 1886–90, Washington, DC, 2001.
233. S.-Y. Sun, C. L. Tseng, Y. H. Chen, S. C. Chuang, and H. C. Fu. Cluster-based support vector machines in text-independent speaker identification. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2004)*, volume 1, pages 729–34, Budapest, Hungary, 2004.
234. J. A. K. Suykens. Least squares support vector machines for classification and nonlinear modelling. *Neural Network World*, 10(1–2):29–47, 2000.
235. J. A. K. Suykens, L. Lukas, and J. Vandewalle. Sparse least squares support vector machine classifiers. In *Proceedings of the Eighth European Symposium on Artificial Neural Networks (ESANN 2000)*, pages 37–42, Bruges, Belgium, 2000.
236. J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines.* World Scientific Publishing, Singapore, 2002.
237. J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.
238. J. A. K. Suykens and J. Vandewalle. Multiclass least squares support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '99)*, volume 2, pages 900–3, Washington, DC, 1999.
239. J. A. K. Suykens and J. Vandewalle. Training multilayer perceptron classifiers based on a modified support vector method. *IEEE Transactions on Neural Networks*, 10(4):907–11, 1999.
240. F. Takahashi and S. Abe. Decision-tree-based multiclass support vector machines. In *Proceedings of the Ninth International Conference on Neural Information Processing (ICONIP '02)*, volume 3, pages 1418–22, Singapore, 2002.

241. F. Takahashi and S. Abe. Optimizing directed acyclic graph support vector machines. In *Proceedings of Artificial Neural Networks in Pattern Recognition (ANNPR 2003)*, pages 166–70, Florence, Italy, 2003.
242. F. Takahashi and S. Abe. Optimal structure of decision-tree-based pairwise support vector machines. *Transactions of the Institute of Systems, Control and Information Engineers*, 17(3):122–30, 2004 (in Japanese).
243. T. Takahashi and T. Kurita. Robust de-noising by kernel PCA. In J. R. Dorronsoro, editor, *Artificial Neural Networks (ICANN 2002)—Proceedings of International Conference, Madrid, Spain*, pages 739–44. Springer-Verlag, Berlin, Germany, 2002.
244. H. Takenaga, S. Abe, M. Takatoo, M. Kayama, T. Kitamura, and Y. Okuyama. Input layer optimization of neural networks by sensitivity analysis and its application to recognition of numerals. *Electrical Engineering in Japan*, 111(4):130–8, 1991.
245. T. Takigawa and S. Abe. High speed training of a fuzzy classifier with polyhedral regions. *Transactions of the Institute of Systems, Control and Information Engineers*, 15(12):673–80, 2002 (in Japanese).
246. D. M. J. Tax and R. P. W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20(11–13):1191–9, 1999.
247. D. M. J. Tax and R. P. W. Duin. Outliers and data descriptions. In *Proceedings of the Seventh Annual Conference of the Advanced School for Computing and Imaging*, pages 234–41, Heijen, the Netherlands, 2001.
248. D. M. J. Tax and P. Juszczak. Kernel whitening for one-class classification. In S.-W. Lee and A. Verri, editors, *Pattern Recognition with Support Vector Machines: First International Workshop, SVM 2002, Niagara Falls*, pages 40–52. Springer-Verlag, Berlin, Germany, 2002.
249. T. B. Trafalis and H. Ince. Benders decomposition technique for support vector regression. In *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, volume 3, pages 2767–72, Honolulu, 2002.
250. D. Tsujinishi and S. Abe. Fuzzy least squares support vector machines. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2003)*, volume 2, pages 1599–604, Portland, OR, 2003.
251. D. Tsujinishi and S. Abe. Fuzzy least squares support vector machines for multiclass problems. *Neural Networks*, 16(5–6):785–92, 2003.
252. D. Tsujinishi, Y. Koshiba, and S. Abe. Why pairwise is better than one-against-all or all-at-once. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2004)*, volume 1, pages 693–8, Budapest, Hungary, 2004.
253. V. Uebele, S. Abe, and M.-S. Lan. A neural-network-based fuzzy classifier. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(2):353–61, 1995.
254. R. J. Vanderbei. LOQO: An interior point code for quadratic programming. Technical Report SOR-94-15, Princeton University, 1998.
255. R. J. Vanderbei. *Linear Programming: Foundations and Extensions, second edition.* Kluwer Academic Publishers, Norwell, MA, 2001.
256. V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, 1995.
257. V. N. Vapnik. *Statistical Learning Theory.* John Wiley & Sons, New York, 1998.

258. V. Vapnik and O. Chapelle. Bounds on error expectation for SVM. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 261–80. MIT Press, Cambridge, MA, 2000.
259. K. Veropoulos. Machine learning approaches to medical decision making. Ph.D thesis, Department of Computer Science, University of Bristol, UK, 2001.
260. K. Veropoulos, C. Campbell, and N. Cristianini. Controlling the sensitivity of support vector machines. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99), Workshop ML3*, pages 55–60, 1999.
261. S. V. N. Vishwanathan and M. N. Murty. SSVM: A simple SVM algorithm. In *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, volume 2, pages 2393–8, Honolulu, 2002.
262. M. Vogt. SMO algorithms for support vector machines without bias term. Technical report, Institute of Automatic Control, TU Darmstadt, Germany, 2002.
263. V. Vovk, A. Gammerman, and C. Saunders. Machine-learning applications of algorithmic randomness. In I. Bratko and S. Dzeroski, editors, *Machine Learning, Proceedings of the Sixteenth International Conference (ICML '99)*, pages 444–53. Morgan Kaufmann, San Francisco, 1999.
264. S. M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 781–7, Detroit, 1989.
265. J. Weston. Leave-one-out support vector machines. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, volume 2, pages 727–33, Stockholm, Sweden, 1999.
266. J. Weston, A. Elisseeff, B. Schölkopf, and M. Tipping. Use of the zero-norm with linear models and kernel methods. *Journal of Machine Learning Research*, 3:1439–61, 2003.
267. J. Weston and R. Herbrich. Adaptive margin support vector machines. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 281–95. MIT Press, Cambridge, MA, 2000.
268. J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for SVMs. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 668–74. MIT Press, Cambridge, MA, 2001.
269. J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, Royal Holloway, University of London, London, 1998.
270. J. Weston and C. Watkins. Support vector machines for multi-class pattern recognition. In *Proceedings of the Seventh European Symposium on Artificial Neural Networks (ESANN 1999)*, pages 219–24, Bruges, Belgium, 1999.
271. T. Windeatt and F. Roli, editors. *Multiple Classifier Systems—Proceedings of the fourth International Workshop, MCS 2003, Guildford, UK.* Springer-Verlag, Berlin, Germany, 2003.
272. S. J. Wright. *Primal-Dual Interior-Point Methods.* Society for Industrial and Applied Mathematics, Philadelphia, 1997.
273. R. Xiao, J. Wang, and F. Zhang. An approach to incremental SVM learning algorithm. In *Proceedings of the Twelfth IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2000)*, pages 268–73, Vancouver, BC, Canada, 2000.

274. J. Xu, X. Zhang, and Y. Li. Large margin kernel pocket algorithm. In *Proceedings of International Joint Conference on Neural Networks (IJCNN '01)*, volume 2, pages 1480–5, Washington, DC, 2001.
275. P. Xu and A. K. Chan. Support vector machines for multi-class signal classification with unbalanced samples. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2003)*, volume 2, pages 1116–9, Portland, OR, 2003.
276. J. Yang, V. Estivill-Castro, and S. K. Chalup. Support vector clustering through proximity graph modelling. In *Proceedings of the Ninth International Conference on Neural Information Processing (ICONIP '02)*, volume 2, pages 898–903, Singapore, 2002.
277. M.-H. Yang and N. Ahuja. A geometric approach to train support vector machines. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 430–7, 2000.
278. Z. Ying and K. C. Keong. Fast leave-one-out evaluation and improvement on inference for LS-SVMs. In *Proceedings of the Seventeenth International Conference on Pattern Recognition (ICPR 2004)*, volume 3, pages 494–7, Cambridge, UK, 2004.
279. S. Young and T. Downs. CARVE—A constructive algorithm for real-valued examples. *IEEE Transactions on Neural Networks*, 9(6):1180–90, 1998.
280. C. Yuan and D. Casasent. Support vector machines for class representation and discrimination. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 2003)*, volume 2, pages 1611–6, Portland, OR, 2003.
281. P. Zhang and J. Peng. SVM vs regularized least squares classification. In *Proceedings of the Seventeenth International Conference on Pattern Recognition (ICPR 2004)*, volume 1, pages 176–9, Cambridge, UK, 2004.
282. P. Zhang, J. Peng, and C. Domeniconi. Dimensionality reduction using kernel pooled local discriminant information. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM 2003)*, pages 701–4, Melbourne, FL, 2003.
283. W. Zhang and I. King. Locating support vectors via β-skeleton technique. In *Proceedings of the Ninth International Conference on Neural Information Processing (ICONIP '02)*, volume 3, pages 1423–7, Singapore, 2002.
284. X. Zhang. Using class-center vectors to build support vector machines. In *Neural Networks for Signal Processing IX—Proceedings of the 1999 IEEE Signal Processing Society Workshop*, pages 3–11, 1999.
285. W. Zhou, L. Zhang, and L. Jiao. Linear programming support vector machines. *Pattern Recognition*, 35(12):2927–36, 2002.

Index