Enhanced kernel methods for pattern classification

Tang, Ke

2007

Tang, K. (2007). Enhanced kernel methods for pattern classification. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/3346>

<https://doi.org/10.32657/10356/3346>

Nanyang Technological University

Downloaded on 26 Apr 2023 16:03:03 SGT

Enhanced Kernel Methods for Pattern Classification

Tang Ke

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2007

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Associate Professor Ponnuthurai Nagaratnam Suganthan for his invaluable support and encouragement throughout my PhD study. His continuous and valuable guidance is the most important factor for the completion of this work.

I would also like to thank my external co-supervisor, Professor Xin Yao at the University of Birmingham, UK for his great help and insightful suggestions during the course of my study.

I wish to convey my gratitude to Mr. Shi Yemin, Mr. Qin Kai, Ms. Liang Jing, Ms. Huang Ling and Mr. Lin Jun, who have given me great encouragement and help in study, life and many other aspects. Thanks to all my lab colleagues for having created a warm research environment, and to all friends who help me and make my life in Singapore colorful.

I am grateful that Nanyang Technological University (NTU) has offered me the research scholarship and provided me with research excellent facilities.

Last but not the least, I would like to thank my dearest parents. They always give me boundless and unconditional love. My gratitude is beyond words.

Summary

Kernel methods are a new generation of techniques that are founded on sound theoretical grounds. It has been shown that they could serve as a powerful methodology for dealing with pattern classification effectively. However, many of the newly created kernel methods are far from perfect, so that extensions and improvements are always required to make them competitive.

In this thesis, we study one important class of the kernel methods, namely the least square support vector machines (LS-SVM), and aim to enhance its performance extensively. The LS-SVM is closely related to many other kernel methods and can be regarded as a representative of them. Hence, many of our results can be easily extended to enhance performance of those related kernel methods. In particular, the LS-SVM is enhanced on several sub-problems related to solving the pattern classification problem, namely the model selection, feature selection, building sparse kernel classifier and classifier ensemble construction.

Model selection concerns the problem of choosing optimal values for the hyper-parameters of the LS-SVM. An efficient and exact computation of the leave-one-out error of the LS-SVM is firstly derived for this purpose. Then the computation is combined with two advanced optimization techniques to obtain two new model selection approaches.

When the problem in hand is characterized by a large number of features while only a limited number of training patterns are available, feature selection is usually an

essential pre-processing procedure for classification. It aims to find from the whole feature set a subset of features, which can yield the best classification performance. On the basis of the LS-SVM formulation, we propose two novel feature selection algorithms. The proposed algorithms are then applied to microarray data, and show competitive performance over the existing microarray gene selection methods.

Research on sparse kernel classifiers has drawn much attention in the recent years. Almost all kernel classifiers suffer from the non-sparse solutions. And a non-sparse kernel classifier requires high storage and computational complexity for testing a previously unknown pattern. Generally, building a sparse LS-SVM means to construct the classifier with a small number of support vectors. These support vectors can be either selected from the training patterns, or be generated based on the training set. For support vector selection, we introduce the idea of floating forward selection, which is a very successful technique for feature selection. The resultant support vector selection algorithm is shown to be able to yield more sparseness in the LS-SVM. Generating support vector on basis of the training patterns is a relatively new idea, which has not been well investigated for sparse kernel classifier construction. We propose a novel algorithm along this direction, and show that this algorithm is generally better than almost all the compared support vector selection algorithms. Our algorithm is also demonstrated to be very competitive when comparing to other related sparse kernel classifiers.

Finally, to build an LS-SVM ensemble, an analysis of some existing diversity measures is presented. Diversity is well acknowledged as one of the key issue for the success of a classifier ensemble. But the usefulness of diversity measures in the

construction of classifier ensemble has not been well validated. By relating some diversity measures to the concept of margin, which accounts for the good generalization of a classifier ensemble, we demonstrate that it is NOT necessary to explicitly maximize diversity in a classifier ensemble. Our conclusion applies to any type of base classifier. Further, other potential usages of the diversity measures are also discussed.

Table of Contents

Acknowledgements	i
Summary.....	ii
Table of Contents	v
List of Figures.....	viii
List of Tables	x
1. Introduction.....	1
1.1 Motivation.....	1
1.2 Overview of the Thesis	4
1.3 Contributions of the Thesis	8
2. Preliminary Background.....	11
2.1 Some Basic Issues of Pattern Classification	11
2.2 Least Squares Support Vector Machines (LS-SVM).....	13
2.2.1 Linear Least Squares Support Vector Machines.....	14
2.2.2 Dealing with Non-linearity Using Kernels	16
2.3 The Related Methods	18
2.3.1 Support Vector Machines	19
2.3.2 Proximal Support Vector Machines.....	20
2.3.3 Other Related Algorithms	20
3. Model Selection for LS-SVM	24
3.1 Introduction.....	24
3.2 Grid Search for LS-SVM Model Selection.....	26
3.2.1 Efficient Computation of the Leave-one-out Error.....	26
3.2.2 Shrinking Grid Search for Model Selection.....	28
3.3 Gradient-based Model Selection.....	29
3.4 Evolutionary-based Model Selection.....	33
3.5 A Comparative Study.....	35

3.5.1	Qualitative Comparisons	35
3.5.2	Experiments.....	37
3.6	Conclusions	41
4.	Feature Selection Algorithms Based on LS-SVM	43
4.1	Introduction	43
4.1.1	A Brief Overview of Feature Selection Methods.....	44
4.1.2	Feature Selection Methods Applied to Microarray Data	48
4.2	Leave-one-out Calculation Sequential Forward Selection Algorithm	50
4.2.1	The Algorithm.....	50
4.2.2	Efficient Computation of the LOOC Criterion	53
4.3	Gradient-based Leave-one-out Gene Selection Algorithm	54
4.4	Experiments.....	59
4.4.1	Algorithms Chosen for Comparison	60
4.4.2	Datasets	61
4.4.3	Experimental Setup	62
4.4.4	Results	63
4.5	Further Discussions	74
4.6	Conclusions	78
5.	On Building Sparse LS-SVM Classifier	80
5.1	Introduction	80
5.2	Formulation of the Sparse LS-SVM.....	82
5.3	Existing Support Vector Selection Methods	86
5.3.1	Greedy Support Vector Selection Algorithms	87
5.3.2	Sub-greedy Support Vector Selection Algorithms.....	91
5.3.3	Non-greedy Support Vector Selection Algorithm.....	93
5.4	Floating Support Vector Selection Algorithm	94
5.5	Sparse LS-SVM Using Pseudo Support Vectors	97
5.6	Efficient Computations of the Evaluation Criterion	101
5.7	An Overview of Other Related Algorithms	103
5.8	Experiments.....	106
5.8.1	Algorithms Chosen for Comparison	106

5.8.2	Datasets	107
5.8.3	Model Selection	108
5.8.3	Numerical Results	109
5.7	Conclusions.....	111
6.	An Analysis of Diversity Measures for Constructing LS-SVM Ensemble	113
6.1	Introduction.....	113
6.2	Related Background.....	117
6.2.1	The Diversity Measures	119
6.2.2	The Margins of Classifier Ensembles	123
6.3	Analysis of Diversity Measures	124
6.3.1	Relationship between Diversity and Margins: Majority Vote Case ...	124
6.3.2	Ineffectiveness of Diversity Measures.....	126
6.3.3	Relationship between Diversity and Margins: General Case.....	137
6.4	Exploiting Diversity Measures in LS-SVM Ensemble Constructing	139
6.5	Conclusions.....	142
7.	Conclusions and Future Works	144
7.1	Conclusions.....	144
7.2	Recommendations for Further Research.....	146
	Author's Publications	149
	Appendices	150
A	Proof of Lemma 3.2	150
B.1	Derivations of Equation (4.10).....	158
B.2	The .632+ Bootstrap Method for Model Assessment	159
C	Proof of Lemma 5.1	161
D.1	Derivations of Equations (6.15)-(6.20) and Proof of Lemma 6.1	163
D.2	Proof of Lemma 6.3	168
E	List of Abbreviations	171
	Bibliography	173

List of Figures

Figure 1.1: Structural overview of the thesis	5
Figure 2.1: Demonstration of the kernel trick	17
Figure 2.2: Structural demonstration of LS-SVM and the related algorithms	23
Figure 3.1: The shrinking grid search approach for model selection	29
Figure 3.2: Depiction of the functions $f_1 = (1 - \text{sgn}(t))/2$ and $f_2 = 1/[1 + \exp(\lambda t)]$	31
Figure 3.3: Variation of the leave-one-out error and modified leave-one-out error with respect to $\log_2(\sigma)$ on the Heart dataset	31
Figure 3.4: The gradient-based model selection approach	32
Figure 3.5: The evolutionary-based model selection approach	35
Figure 4.1: Pseudo-code of the LOOC sequential forward selection algorithm	52
Figure 4.2: Pseudo-code of the gradient-based leave-one-out gene selection algorithm	59
Figure 4.3: Comparison of seven gene selection algorithms on Colon dataset	65
Figure 4.4: Comparison of seven gene selection algorithms on Leukemia dataset	66
Figure 4.5: Comparison of seven gene selection algorithms on Carcinoma dataset ...	66
Figure 4.6: Comparison of seven gene selection algorithms on Glioma dataset	67
Figure 4.7a: Comparison of LOOCSFS, LSBSFS and LSBSFFS algorithms on Colon dataset	67
Figure 4.7b: Comparison of LOOCSFS, LSBSFS and LSBSFFS algorithms on Leukemia dataset	68
Figure 4.7c: Comparison of LOOCSFS, LSBSFS and LSBSFFS algorithms on Carcinoma dataset	68
Figure 4.7d: Comparison of LOOCSFS, LSBSFS and LSBSFFS algorithms on Glioma dataset	69
Figure 4.8a: Verification of heuristics components of GLGS on Colon dataset	69
Figure 4.8b: Verification of heuristics components of GLGS on Leukemia dataset	70

Figure 4.8c: Verification of heuristics components of GLGS on Carcinoma dataset	70
Figure 4.8d: Verification of heuristics components of GLGS on Glioma dataset	71
Figure 4.9: The computational time of seven gene selection algorithms on the Carcinoma dataset, shown vs. the number of selected genes.....	71
Figure 4.10: The computational time of seven gene selection algorithms on the Carcinoma dataset, shown vs. the size of the gene set.....	72
Figure 5.1: Pseudo-code of the greedy support vector selection algorithm.....	90
Figure 5.2: Pseudo-code of the sub-greedy support vector selection algorithm.....	92
Figure 5.3: Pseudo-code of the non-greedy support vector selection algorithm	94
Figure 5.4: Pseudo-code of the floating forward support vector selection algorithm.....	96
Figure 5.5: Pseudo-code of the gradient-based support vector generation algorithm	101
Figure 5.6: Depiction of the model selection scheme on the Heart dataset.....	109
Figures 6.1 (a-f): Illustration of the relationship between diversity measures and minimum margin (experiment 1)	129
Figures 6.2 (a-f): Illustration of the relationship between diversity measures and minimum margin (experiment 2)	131
Figure 6.3: Relationship between P and $\sum_{i=1}^N l_i^2$	134
Figure 6.4: Pseudo-code of a greedy approach for building LS-SVM ensemble	135

List of Tables

Table 3.1: Summary of the eight datasets	37
Table 3.2: Comparisons among five combinations of model selection approach and kernel function.....	42
Table 4.1: 20 most frequently selected genes of Hepatocellular Carcinoma dataset selected by LOOCSFS	76
Table 4.2: 20 most frequently selected genes of Hepatocellular Carcinoma dataset selected by GLGS	77
Table 5.1: Results on the Banana, Breast-cancer, Diabetis and Heart datasets.....	110
Table 5.2: Results on the Image, Splice, Thyroid and Waveform datasets.....	111
Table 5.3: Further comparison with related algorithms	112
Table 6.1: Summary of six diversity measures.....	122
Table 6.2: Classification results of single LS-SVM and LS-SVM ensembles achieved by seeking diversity explicitly.....	136

Chapter 1

Introduction

To a large extent, this thesis is devoted to the pattern classification (also known as pattern recognition) problem. Being an interdisciplinary subject, pattern classification can be defined as “the act of taking in raw data and taking an action based on category of the pattern” [1]. Although such tasks appear to be easy for human being as we have evolved highly sophisticated neural and cognitive systems over the past millions of years, it is difficult for a machine to automatically classify data to the correct categories. In fact, it was not until 1960s that research was broadly conducted in the field of pattern classification, based on the developments in the areas of statistics, artificial intelligence and computer science. Since then, pattern classification methods have been continuously renewed and successfully applied to solve real-world problems arising from various areas such as text categorization, information retrieval, image processing, bioinformatics and so on.

1.1 Motivation and Objectives

Although the problem of pattern classification has developed significantly in the past decades and a multitude of sub-fields could be chosen as the focus of attention in order to investigate, it is of course the latest developed topics and (still) unresolved issues that motivate comprehensive research. The works presented in this thesis are

centered around the least squares support vector machines (LS-SVM) [2, 3], which is one important class of the so-called kernel methods.

Kernel methods are a new generation of techniques that are founded on sound theoretical grounds. Due to the successful performance and elegance of the support vector machines (SVM) [4], which was the first method that implemented the theory, kernel methods have been drawing much interest of researchers and heavily influenced the landscape of pattern classification field since the late 1990s. The general idea behind kernel methods is that the data are nonlinearly mapped into a higher dimensional space (the kernel space), in which the non-separable data may become linearly separable. Hence, kernel methods can be intuitively regarded as generalized versions of linear methods in the kernel space. Different from other nonlinear methods, the kernel methods do not require explicit computations of the high-dimensional mapped data, which may be very costly. Instead, only inner products between the mapped data are computed using a so-called kernel function. By this means, any linear method that only involves inner products can be kernelized in a simple and elegant way. In fact, a large number of kernel methods, which originated from various existing linear methods, have been proposed since the invention of SVM [5-13]. However, most of these newly created kernel methods are far from perfect or even not able to provide satisfactory performance. Therefore, extensions and improvements are always required to make them competitive. This thesis aims to make contributions in that matter. Among all the existing kernel methods, the LS-SVM particularly motivated our interest of investigation for two reasons:

- From a pattern classification perspective, the LS-SVM is a competitive tool for solving difficult classification tasks. Compared with other paradigms such as

Chapter 1: Introduction

Bayesian classifiers, neural networks and decision trees, LS-SVM are conceptually simple and very transparent, it provides a unique and global solution and is less sensitive to high input dimensionality. Although the standard SVM also share these theoretical advantages, it involves much more expensive computation than the LS-SVM, while the two methods generally provide comparable classification performance. Besides, as a kernel method without long history, quite a few important problems associated with the LS-SVM have not been well addressed yet. Therefore, the performance of LS-SVM can be further improved.

- From the viewpoint of the development of kernel methods, the standard SVM was proposed with a specific formulation and is self-contained. In comparison, LS-SVM is closely related to quite a few other kernel methods such as the proximal support vector machines [14], the kernel Fisher discriminant analysis [8] and the kernel principal component analysis [6]. In consequence, achievements on LS-SVM may be further extended to those related methods, and thereby stimulate the development of the whole family of kernel methods.

Given the general objective of this thesis as to extensively enhance the LS-SVM for pattern classification tasks, we are particularly motivated by the following limitations and questions associated with the LS-SVM and try to make contributions to their solutions.

1. The performance of an LS-SVM on a pattern classification task is largely determined by the kernel function that is used. So optimization of the kernel functions, which is usually called model selection, is the very starting point for

building a good LS-SVM. Can we enhance the LS-SVM in this stage, with some more powerful optimization techniques?

2. In many situations, the difficulty of a general classification problem does not lie in making classification correctly, but in finding the most useful features for achieving the best classification performance. Originally being proposed as a classification tool, can the formulation of LS-SVM been modified to address the feature selection task? In particular, how can it be done if the problem involved is characterized by a very large numbers of features, while very limited training patterns are given?
3. It is a limitation that the LS-SVM depends on all the training patterns to make classification on a new pattern (i.e. the solution of LS-SVM is not sparse). Therefore, the evaluation of LS-SVM on new data can be very time consuming if it is trained on a large sized training set. Can we improve upon the existing methods to build a sparse LS-SVM?
4. Classifier ensembles have been long known and are appreciated for their competitive classification performance. Can we further improve the LS-SVM by constructing an LS-SVM ensemble?

1.2 Overview of the Thesis

This thesis is divided into five chapters each of which addresses a sub-issue. The Chapters 3-6 are devoted to the four particular questions described above and share the common background exposed in Chapter 2. Chapters 3-6 have been written in

Chapter 1: Introduction

---

such a way that they are as self-contained as possible. In general, each chapter can be related to the other ones either for the underlying problems to be solved, or for the techniques that are utilized to solve them, as demonstrated in Figure 1.1. The contents of the chapters are summarized as follows.

Figure 1.1: Structural overview of the thesis. The arrows indicate relations between the chapters. The titles denote the common theme that links the chapters together, by which we form a logical chain.

- Chapter 1 gives an introduction and describes the background of this thesis in general. In condensed terms, the main motivations and objectives are sketched and an overview is given of the content and the contributions.
- In Chapter 2, we firstly introduce some basic concepts of pattern classification and clarify the terminologies that will be used throughout the subsequent chapters. Then we zoom in on the LS-SVM formulation for pattern classification

and the general paradigm in which kernel methods are rooted. After that, the relations between LS-SVM and other kernel methods are discussed.

- In Chapter 3, we focus on the model selection problem for LS-SVM. Model selection can be understood as the procedure of tuning hyper-parameters, which must be determined at the beginning of classification and directly influence the final performance of the LS-SVM. Traditionally, the hyper-parameters are tuned by grid search based on cross-validation error. In 3.2, we proved that the leave-one-out cross-validation error of an LS-SVM can be calculated efficiently due to its specific formulation. With this efficient calculation, we present two alternative model selection methods based on the gradient descent algorithms and evolutionary algorithms, respectively. Experiments are then conducted to verify the usefulness of our methods. Compared with the grid search, our methods allow usage of more complex kernel functions and thereby result in better classification performance on difficult classification problems.
- In Chapter 4, we elaborate how to adopt the LS-SVM formulation to address feature selection problems that are characterized by large number of features and small number of patterns. In particular, the gene selection problem for microarray data is considered as an example. Based on some results of Chapter 3, we propose two gene selection algorithms in Section 4.2 and 4.3. Experiments are then conducted to compare our algorithms with other commonly used gene selection methods. Our methods are illustrated to be capable of selecting more important genes yielding higher classification accuracy. Finally, since gene selection is a real-world application that usually has multiple objectives, the

Chapter 1: Introduction

---

usages of gene selection algorithms are discussed in the context of different purposes of gene selection.

- Chapter 5 addressed the problem of building a sparse LS-SVM. By approximating the optimal solution of LS-SVM using less support vectors, both computational and storage costs for classifying a pattern can be significantly alleviated. First, we mathematically formulate the problem to be addressed and show the trade-off that needs to be controlled for building a desirable sparse LS-SVM. Then we give an overview of existing forward support vector selection methods for LS-SVM. Based on the overview, we present in Section 5.3 an improved support vector selection method. After that, we explore possible pathways of using a gradient descent algorithm to seek the support vectors more effectively. Related works for the same purpose in the context of other kernel methods are also reviewed. Finally, we experimentally demonstrate the effectiveness of our methods.
- Chapter 6 is devoted to the construction of an LS-SVM ensemble. An LS-SVM ensemble is a combination of multiple LS-SVMs. In general, it is expected that an LS-SVM ensemble contains a diverse set of individual LS-SVM to provide better classification performance than a single LS-SVM. In this chapter, we present an in-depth analysis on some existing diversity measures, which are proposed to quantify how diverse a set of individual LS-SVMs are. Through our analysis, we demonstrate that those diversity measures are closely linked to one another while one CANNOT achieve a desirable LS-SVM ensemble by explicitly

optimizing the diversity measures. Alternative ways of using the diversity measures to build LS-SVM ensemble are also discussed.

- In Chapter 7 we summarize the results obtained in this thesis and give an outlook on several promising directions for future research.

1.3 Contributions of the Thesis

The main contributions obtained in this thesis can be summarized in two folds:

From the LS-SVM point of view, we address four important problems associated with the LS-SVM. We propose solutions for three of them and present an in-depth analysis for the other one:

- In chapter 3, the model selection problem is addressed and two methods are proposed based on advanced optimization techniques, namely the gradient descent algorithm and the evolution strategies. Compared to traditional model selection method, our approaches allow optimizing a large number of hyperparameters simultaneously. Consequently, they make possible the use of highly complex and tunable kernels, with which the LS-SVM may perform significantly better on tough tasks.
- In chapter 4, we modify the LS-SVM formulation to solve feature selection problem. Based on an efficient and exact calculation of the leave-one-out cross-validation error of LS-SVM, we propose two feature selection algorithms, called the leave-one-out calculation sequential forward selection algorithm and the gradient-based gene/feature selection algorithm. Although the advantages of our

Chapter 1: Introduction

---

algorithms are specifically demonstrated on microarray datasets, they can be readily applied to other real-world applications.

- In chapter 5, the problem of building a sparse LS-SVM is addressed. We first survey the existing methods for support vector selection. Then, we suggest including a back-tracking procedure when selecting the support vector in a forward manner, so that more sparseness can be achieved. Further, we also propose a general framework for generating pseudo support vectors in the input feature space and a detailed algorithm based on this framework to perform the pseudo support vector generation.
- In chapter 6, an in-depth analysis is conducted to verify the usefulness of the long-term concept of diversity for construction of LS-SVM ensemble. By relating several commonly-used diversity measures to the margin concept, which accounts for the success of quite a few pattern classification algorithms, we demonstrate that the diversity measures are somewhat inappropriate and should not be used to construct an LS-SVM ensemble.

From a broader perspective, at least two works presented in this thesis can be generalized to many related areas.

- Sparse kernel classifier has been a very hot topic for a few years. Not only in the LS-SVM literature, but also a lot of work has been done on related algorithms such as kernel Fisher discriminant analysis, Gaussian process and even the standard support vector machine. But according to our survey, the general ideas behind our methods for sparse LS-SVM construction have not yet been well

developed on the related algorithms. Hence, by introducing our ideas into those related fields, not only the LS-SVM model, but also a large family of kernel methods, can be improved.

- Although we are particularly interested in the LS-SVM ensemble, the analysis presented in Chapter 6 is actually independent of the specific formulation of the individual classifier involved. Therefore, our analysis and conclusions readily apply to a large variety of classification models and thereby contribute to the very broad areas of pattern classification and machine learning.

Chapter 2

Preliminary Background

In this chapter, some preliminary knowledge is described. We first define the terminologies adopted in this thesis, and then introduce the least squares support vector machines (LS-SVM) for pattern classification. The linear LS-SVM basically assumes the classifier to be a linear function of the input patterns, and is generalized to nonlinear formulation by using the so-called kernel trick. Further, we also briefly introduce some methods that are closely related to the LS-SVM formulation.

2.1 Some Basic Issues of Pattern Classification

Many of the topics we shall discuss have been studied over a range of diverse disciplines, and there is naturally a variety of sometimes contradictory terminologies and descriptions. Hence, we would like to firstly define some essential concepts.

We use the term **pattern** to denote the d -dimensional data vector $\mathbf{x} = [x_1, x_2, \dots, x_d]^T$ (T denotes transpose), whose components x_i are referred to as **features** of the pattern. Associated with each pattern is a **class label** y ($y \in \{+1, -1\}$), with which all the patterns involved in this thesis are naturally categorized into either the “+1” class or the “-1” class.

Given a pattern vector, a **classifier** is a system which is able to classify the pattern to one of the two classes. Specifically, taking a pattern vector as input, a classifier

---

Chapter 2: Preliminary Background

outputs either “-1” or “+1”, which indicates the class membership of the pattern. A classifier can be described in several different but equivalent ways. First, a classifier is a parameterized mapping function $f(\mathbf{x}, \boldsymbol{\zeta})$, which maps from an input space $\mathbf{x} \subseteq R^d$ to an output space $y \subseteq R$, where $\mathbf{x}$ is the pattern submitted to the classifier as input and $\boldsymbol{\zeta}$ is a parameter vector. On the other hand, when the patterns are regarded as data points lying in a d -dimensional space, the classifier can be viewed as a decision boundary in this space which separates one class of patterns from the other.

When applying a classifier to a classification problem, values of the parameters ($\boldsymbol{\zeta}$) need to be determined first. Although they can be in principle specified arbitrarily, arbitrarily determined parameters may yield poor classification performance. Hence, we usually determine the values for parameters by **training** the classifier with a **training set**. The training set is a finite set of **training patterns**, whose class labels are known beforehand.

Once a classifier is trained, it is associated with three types of misclassification rates, namely the **training error**, the **generalization error** and the **testing error**. The training error is the misclassification rate that the trained classifier makes on the training set. The generalization error can be described as the expectation of the probability that the trained classifier will misclassify a new pattern (i.e. this pattern does not belong to the training set). The generalization error is in principle the ultimate criterion for the assessment of a trained classifier. But, directly calculating the generalization error is usually impossible since it theoretically requires applying the trained classifier to an infinite set of new patterns. Hence, in practice the classifier is usually only assessed on a finite set of **testing patterns**, which are kept out of the

Chapter 2: Preliminary Background

---

training phase. And in this way the generalization error is estimated by the so-called **testing error**.

For a pattern classification task, it is natural to expect that a trained classifier with small training error will also achieve small testing error on the testing patterns. However, this is not always true. When applying a trained classifier to the testing patterns, we implicitly assume that the training set is a good representative of all the patterns of interests and is noise-free. If such an assumption is severely violated, a trained classifier which performs extremely well on the training set will perform poorly on testing patterns. Such a classifier is said to be over-fitting. Therefore, achieving optimal performance (in terms of minimizing some error criterion such as training error) on the training set may not be necessary. In addition to the data structure, the over-fitting scenario is also closely related to the complexity of the trained classifier. Generally speaking, a “complex” trained classifier, which corresponds to a complex decision boundary in the Euclidean space, is more likely to be over-fitting than a “simple” trained classifier. The reason is that a complex classifier is able to capture more information from the data and thereby likely to capture more noise. On the other hand, if a trained classifier is too simple, it will under-fit the data and result in both high training error and high testing error. Therefore, complexity of the classifier needs to be carefully controlled to achieve good generalization performance.

2.2 Least Squares Support Vector Machines

In this section we give a brief overview of the least squares support vector machine (LS-SVM). As a variant of the standard support vector machines (SVM) [4], LS-SVM

was proposed by Suykens and Vandewalle [2,3] and have been successfully applied to solve pattern classification problems [15]. We start with the simplest case and then provide the generalized version.

2.2.1 Linear Least Squares Support Vector Machines

Given a training set $S_{tr} = \{(\mathbf{x}_i, y_i), i = 1, 2, \dots, n\}$, $y_i \in \{-1, +1\}$, the linear least squares support vector machines (LS-SVM) assumes a classifier of the form:

$$\hat{y}(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \mathbf{x} + b) \quad (2.1)$$

where the weighting vector $\mathbf{w}$ and bias b are determined by training. With the above form, the linear LS-SVM can be understood as a hyper-plane that separates different classes of patterns in the input feature space R^d . The training phase of LS-SVM is formulated as solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w}, b, \mathbf{e}} J(\mathbf{w}, b, \mathbf{e}) &= \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{\gamma}{2} \sum_{i=1}^n e_i^2 \\ \text{subject to } y_i (\mathbf{w}^T \mathbf{x}_i + b) &= 1 - e_i \quad \text{for } i = 1, \dots, n \end{aligned} \quad (2.2)$$

where $\mathbf{e} = [e_1, e_2, \dots, e_n]^T$. The objective function $J(\mathbf{w}, b, \mathbf{e})$ consists of a sum of squared error term $\sum_{i=1}^n e_i^2$ to fit the classifier to the training set, and a regularization term $\mathbf{w}^T \mathbf{w}$ to prevent over-fitting. The relative importance of these two terms is determined by the positive real constant γ . By taking a small value of γ , one avoids over-fitting in the case of noisy training set. After introducing a set of Lagrange multipliers α_i 's, the optimization problem in Eq. (2.2) becomes:

$$\min_{\mathbf{w}, b, \mathbf{e}, \boldsymbol{\alpha}} L(\mathbf{w}, b, \mathbf{e}, \boldsymbol{\alpha}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{\gamma}{2} \sum_{i=1}^n e_i^2 - \sum_{i=1}^n \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + e_i] \quad (2.3)$$

Chapter 2: Preliminary Background

where $\mathbf{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_n]^T$. Taking the derivatives of the $L(\mathbf{w}, b, \mathbf{e}, \mathbf{\alpha})$ with respect to $\mathbf{w}$, b , e_i and α_i we obtain optimality conditions of Eq. (2.3) as:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = 0 \rightarrow \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \quad (2.4)$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^n \alpha_i y_i = 0 \quad (2.5)$$

$$\frac{\partial L}{\partial e_i} = \gamma e_i - \alpha_i = 0 \rightarrow \alpha_i = \gamma e_i \quad i = 1, \dots, n \quad (2.6)$$

$$\frac{\partial L}{\partial \alpha_i} = y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + e_i = 0 \quad i = 1, \dots, n \quad (2.7)$$

After eliminating the variables $\mathbf{w}$ and $\mathbf{e}$ through substitution, a unique solution of Eq. (2.2) can be computed by solving the following linear system:

$$\begin{bmatrix} \mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n & \mathbf{Y} \\ \mathbf{Y}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{1}_n \\ 0 \end{bmatrix} \quad (2.8)$$

where entries of the n -by- n matrix $\mathbf{\Omega}$ are $\Omega_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$, $\mathbf{Y} = [y_1, y_2, \dots, y_n]^T$, $\mathbf{I}_n$ is an identity matrix and $\mathbf{1}_n$ is a n -by-1 vector with all the entries equal to 1. Let $\tilde{\mathbf{\Omega}} = \mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n$, $\mathbf{\alpha}$ and b can alternatively be computed as:

$$\mathbf{\alpha} = \tilde{\mathbf{\Omega}}^{-1} \mathbf{1}_n - \frac{\tilde{\mathbf{\Omega}}^{-1} \mathbf{Y} \mathbf{Y}^T \tilde{\mathbf{\Omega}}^{-1} \mathbf{1}_n}{\mathbf{Y}^T \tilde{\mathbf{\Omega}}^{-1} \mathbf{Y}} \quad (2.9)$$

$$b = \frac{\mathbf{Y}^T \tilde{\mathbf{\Omega}}^{-1} \mathbf{1}_n}{\mathbf{Y}^T \tilde{\mathbf{\Omega}}^{-1} \mathbf{Y}} \quad (2.10)$$

Once the Lagrange multipliers α_i 's are known, the weighting vector $\mathbf{w}$ can be calculated with Eq. (2.4) and thereby a pattern $\mathbf{x}$ is classified by the trained LS-SVM as:

$$\hat{f}(\mathbf{x}) = \text{sgn} \left(\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b \right) \quad (2.11)$$

The training patterns corresponding to those non-zero α_i 's are referred to as the **support vectors**.

2.2.2 Dealing with Non-linearity Using Kernels

In the previous sub-section, we presented the linear version of LS-SVM, which assumes that the patterns from different classes can be separated by a linear hyper-plane. However, the two classes of patterns may not be linearly separable in many cases. Hence, an LS-SVM is required to be capable of handling non-linear separable cases. A basic strategy is to first non-linearly map the patterns into a new feature space, denoted as $\phi: \mathbf{x} \in R^d \mapsto \phi(\mathbf{x}) \in R^D$. By choosing appropriate ϕ , patterns from different classes may become linearly separable in the new space and thereby the linear LS-SVM can be applied to them. By this means, an LS-SVM classifier is still a linear hyper-plane in the new feature space, but becomes a non-linear hyper-plane in the input feature space. The only difference between this non-linear LS-SVM and the linear LS-SVM is that the entries of matrix $\mathbf{\Omega}$ in Eq. (2.8) now become $\Omega_{ij} = y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$, and Eq. (2.11) is re-written as:

$$\hat{f}(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^n \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b\right) \quad (2.12)$$

When applying the above strategy, a natural question is what non-linear mapping we should use. Since the optimal non-linear mapping varies from case to case, it may be difficult to figure it out. Besides, if such a mapping is too costly to compute (for example, the new feature space is with very high dimensionality), it may not be suitable for a real-application. Fortunately, we can observe from Eq. (2.10) and Eq. (2.11) that only inner products between two pattern vectors are required for both

Chapter 2: Preliminary Background

training and testing. Hence, if there exists a “kernel function” k such that $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$ for two patterns $\mathbf{x}_i$ and $\mathbf{x}_j$, we would never need to know the mapping ϕ explicitly. First proposed in [16] and re-introduced by Boser, Guyon and Vapnik for building the non-linear SVM [17], this idea of using kernel function for ordinary inner products is called the kernel trick. Clearly, the kernel trick is applicable to generalizing any traditional linear method, which only requires calculating the inner products between patterns but not the patterns themselves, to a non-linear version. Hence, it is not surprising that various non-linear methods have been proposed based on kernel trick after the historical work of SVM [4]. All these methods, although aim to solve different problems (such as classification, regression, clustering, etc.) and originated from diverse formulations, are then referred to as the kernel methods in the literature [5, 6, 8, 10-13].

Figure 2.1: Demonstration of the kernel trick. The figure demonstrates the idea of solving nonlinear separable data by non-linearly mapping them into a kernel space, in which they become linear separable.

Although we can easily generalize LS-SVM to non-linear case by utilizing the kernel trick, we still have to choose the kernel function. Then which kind of functions can be

used as the kernel functions? In the last few years, a lot of research works on the design of kernel functions have been conducted, dedicating to both general theoretical foundations and specific applications. As investigating different kernel functions is not the aim of this thesis, we briefly introduce below some commonly-used kernels. Interested readers are referred to the literature [18,19] for theoretical background and more application-oriented kernel functions.

Linear kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$$

Gaussian kernel (RBF kernel):

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\sigma \|\mathbf{x}_i - \mathbf{x}_j\|^2) \quad \sigma \in R_0^+$$

Polynomial kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + b)^p \quad p \in \mathbb{N}, b \in R_0^+$$

Sigmoid kernel (MLP kernel):

$$k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa \mathbf{x}_i^T \mathbf{x}_j + b) \quad \kappa, b \in R$$

With either linear or non-linear kernel functions, we finally get the complete formulation of LS-SVM. The training phase is still done by solving the linear system in Eq. (2.10), but with entries of $\mathbf{\Omega}$ equal to $\Omega_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$ (Note that with the linear kernel we get the linear LS-SVM.). Accordingly, the trained LS-SVM is expressed as

$$\hat{f}(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^n \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b\right) \quad (2.13)$$

2.3 The Related Methods

As we have mentioned, LS-SVM is a specific type of classifier which belongs to the family of kernel classifiers. So it is naturally related to other types of kernel classifiers [20-22]. Besides, several other types of classifier that are not covered by the kernel framework can also be expressed in similar formulations as LS-SVM [21,22]. To

Chapter 2: Preliminary Background

---

provide more insight into the LS-SVM formulation, we next briefly introduce these related methods.

2.3.1 Support Vector Machines

Proposed by Vapnik *et al.* [4,23,24], SVM is probably the most popular kernel classifier. Same as the LS-SVM, a linear SVM adopts the linear classifier in Eq. (2.1) as its basic form and is generalized to non-linear case using the kernel trick. The factor that distinguishes the SVM from the LS-SVM is the following optimization problem involved in its training phase:

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} J(\mathbf{w}, b, \xi) &= \frac{1}{2} \mathbf{w}^T \mathbf{w} + \gamma \sum_{i=1}^n \xi_i \\ \text{subject to} \quad &y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for } i = 1, \dots, n \\ &\xi_i \geq 0 \quad \text{for } i = 1, \dots, n \end{aligned} \quad (2.14)$$

Again, by deriving the optimality conditions, one obtains a quadratic programming (QP) problem that leads to a solution in the same form of LS-SVM:

$$\hat{f}(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^n \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b\right) \quad (2.15)$$

Formulated as different optimization problems, the differences between SVM and LS-SVM are two-folds: First, the training phase of LS-SVM requires solving a linear system of size $(n+1)$ -by- $(n+1)$, while SVM requires solving a QP problem with the same size. Since solving a QP problem is more costly, training an SVM is usually more time consuming than training an LS-SVM. Second, due to the nature of a QP problem, SVM has been proven to contain many zero α_i 's in its solution and thereby is said to provide a sparse solution. On the other hand, most of α_i 's in the solution of LS-SVM are non-zero. Therefore, in comparison to SVM, LS-SVM trades the so-

called sparseness with the training time. Such a scenario motivated a part of the work presented in this thesis. We will discuss this issue in more details in Chapter 5.

2.3.2 Proximal Support Vector Machines

The proximal support vector machine (PSVM), which has almost the same formulation as the LS-SVM, was proposed by Fung and Mangasarian [14,25]. The PSVM can be formulated as the following optimization problem

$$\begin{aligned} \min_{\mathbf{w}, b, \mathbf{e}} J(\mathbf{w}, b, \mathbf{e}) &= \frac{1}{2}(\mathbf{w}^T \mathbf{w} + b^2) + \frac{\gamma}{2} \sum_{i=1}^n e_i^2 \\ \text{subject to } y_i(\mathbf{w}^T \mathbf{x}_i + b) &= 1 - e_i \quad \text{for } i = 1, \dots, n \end{aligned} \quad (2.16)$$

Comparing Eq. (2.16) to Eq. (2.2), we observe only difference between PSVM and LS-SVM is that PSVM includes the bias term b in the regularization term. Accordingly, the solution of PSVM can be computed by solving a linear system

$$\begin{bmatrix} \mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n & \mathbf{Y} \\ \mathbf{Y}^T & -1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{1}_n \\ 0 \end{bmatrix} \quad (2.17)$$

Once $\mathbf{a}$ and b are computed, the trained PSVM is of the exactly same form as the SVM and LS-SVM. Comparing Eq. (2.17) (the training phase of PSVM) with Eq. (2.8) (the training phase of LS-SVM), one can find that the usage of different regularization terms does not change the nature of the optimization problem involved, i.e. the solutions are calculated by solving linear systems of the same size.

2.3.3 Other Related Algorithms

In addition to SVM and PSVM, LS-SVM is also closely related to the kernel Fisher discriminant analysis (KFDA) [8,18,26] and the regularized least squares classifier

Chapter 2: Preliminary Background

(RLSC) [20,27]. The KFDA was proposed as the kernelized version of the very classical Fisher discriminant analysis (FDA). The major goal of Fisher discriminant analysis is to project the data from the original space to a one dimensional space (i.e. a straight line), in which the patterns from different classes are kept *as distant as possible* so that they can be easily separated [28,29]. Under some assumptions, Fisher formulated the problem as simultaneously maximizing the between-class variance and minimizing the within-class variance of the two classes [28]. Solution of this formulation can then be calculated by performing an eigenvalue decomposition of a square matrix. When generalizing FDA to KFDA, the involved square matrix is of size n -by- n . It is notable that there exist several KFDA algorithms [8,18,26], which are slightly different from one another. And one of them directly leads to an LS-SVM formulation with only minor difference from Eq. (2.8) [3]. The RLSC is defined by Rifkin to solve binary classification problems [20]. Rifkin theoretically showed that the RLSC is actually within the same general framework as LS-SVM and PSVM.

In this thesis, we mainly investigate the LS-SVM for classification. But it is noteworthy that the LS-SVM framework has also been adapted to regression tasks [3]. To be specific, Eq. (2.2) is modified as Eq. (2.18):

$$\begin{aligned} \min_{\mathbf{w}, b, \mathbf{e}} J(\mathbf{w}, b, \mathbf{e}) &= \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{\gamma}{2} \sum_{i=1}^n e_i^2 \\ \text{subject to} \quad &\mathbf{w}^T \mathbf{x}_i + b = y_i - e_i \quad \text{for } i = 1, \dots, n \end{aligned} \quad (2.18)$$

Resultantly, the solution can be computed by solving the following linear system

$$\begin{bmatrix} \mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n & \mathbf{1}_n \\ \mathbf{1}_n^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{Y} \\ 0 \end{bmatrix} \quad (2.19)$$

Although Eq. (2.2) and Eq. (2.18) were proposed for different problems, and Eq. (8) and Eq. (19) definitely lead to different solutions, we can observe that the

---

Chapter 2: Preliminary Background

modifications merely result in limited difference of the involved optimization problems. This is reasonable since the areas of classification and regression are closely related to each other. If we further consider other methods originally proposed for regression task, the regularization networks (RN) [30], Gaussian processes (GP) [31,32], kriging [33,34] and kernel ridge regression (KRR) [7] are closely related to the regression formulation of LS-SVM, and hence can be easily related to the classification formulation of LS-SVM. Similar to LS-SVM, all the above regression methods can be formulated as optimization problems that require solving linear systems of approximately the same size. The difference is that no bias term b is included in the formulation of GP, kriging and KRR, while RN is oriented from different theoretical background. Finally, it is noteworthy that the radial basis function neural network (RBFNN) is also related to the LS-SVM, though it originated from a distinct background and usually neglects the regularization term.

We close this chapter by remarking again the relations between LS-SVM and all the above-mentioned methods. Except for the standard SVM, the training phase of all the other methods are conducted by solving a linear system of size n -by- n (or $(n+1)$ -by- $(n+1)$). The SVM is trained by solving a QP problem. Besides, all the trained classifiers can be expressed in the same basic form in Eq. (2.1).

Chapter 2: Preliminary Background

Figure 2.2: Structural demonstration of LS-SVM and the related algorithms

Chapter 3

Model Selection for LS-SVM

3.1 Introduction

Suppose that we are given a training set $S_{tr} = \{(\mathbf{x}_i, y_i), i = 1, 2, \dots, n\}$, $y_i \in \{-1, +1\}$, where $\mathbf{x}_i \in R^d$ is the training pattern, and y_i denotes the corresponding class label. For any pattern $\mathbf{x}$, a trained LS-SVM can be expressed as,

$$\hat{f}(\mathbf{x}) = \text{sgn}(f(\mathbf{x})) = \text{sgn}\left(\sum_{i=1}^n \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b\right) \quad (3.1)$$

where $k(\mathbf{x}_i, \mathbf{x})$ is calculated using a pre-defined kernel function. α_i 's and b are computed as:

$$\begin{bmatrix} \mathbf{a} \\ b \end{bmatrix} = \left(\begin{bmatrix} \mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n & \mathbf{Y} \\ \mathbf{Y}^T & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} \mathbf{1}_n \\ 0 \end{bmatrix} \quad (3.2)$$

where $\mathbf{\Omega}$ is a matrix whose entry $\Omega_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$ and γ is the regularization parameter. It can be seen that once the matrix $\mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n$ is fixed, $\mathbf{a}$ and b can be readily computed. Hence, LS-SVM is parameterized by the kernel function k and the regularization parameter γ . To achieve good classification performance, we must choose an appropriate kernel function and the optimal value of γ . Such a task falls in the category of model selection problems. Since selecting a suitable type of the kernel function (such as Gaussian, polynomial or sigmoid) is relatively easy, the model selection problem reduces to tuning the regularization parameter and the kernel

Chapter 3: Model Selection for LS-SVM

---

parameters that are associated with the kernel function (e.g. the value of σ in the Gaussian kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\sigma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$). As the kernel and regularization parameters are not on the same level as parameters α and b , they are called the hyper-parameters of LS-SVM. In practice, the task of tuning hyper-parameters can be understood as searching in a hyper-parameter space for the optimal hyper-parameter vector (consisting of the regularization parameter and all the kernel parameters), and is usually done by grid search [3,15]. That is, the hyper-parameters are varied through a wide range of values and the performance of every candidate is evaluated using some performance measures. Due to its high computational complexity, grid search is only suitable for the adjustment of a very few hyper-parameters.

In this chapter, we propose two advanced approaches for tuning the hyper-parameters of the LS-SVM. The first approach employs a gradient descent algorithm and iterates the following procedure: The LS-SVM is trained using the current hyper-parameters, the gradient of the performance measure with respect to the hyper-parameters is calculated, and a gradient step is performed in the hyper-parameter space. In the second approach, the hyper-parameters are optimized by an evolutionary algorithm. In pattern classification literature, evolutionary algorithms have been successfully applied to model selection for neural networks [35-38]. They have also been successfully applied to standard SVM for both feature selection and model selection [39-43]. Specifically, we use evolution strategies (ES) [44-47] to search for the appropriate hyper-parameters that optimize the generalization performance. Compared to the traditional approach based on grid search, both approaches can handle much more hyper-parameters and lead to better generalization performance.

In Section 3.2, we introduce the leave-one-out error, which is commonly used for estimating generalization error of a classifier, and show that it can be computed efficiently for the LS-SVM. Based on this efficient computation, we also briefly describe the procedure of using grid search to tune hyper-parameters. After that, the gradient-based and evolutionary-based model selection approaches are described in Sections 3.3 and 3.4 respectively. Finally, we present a comparative study of the three approaches in Section 3.5 and draw our conclusions in Section 3.6.

3.2 Grid Search for LS-SVM Model Selection

3.2.1 Efficient Computation of the Leave-one-out Error

Although model selection is usually kept as a separate step from the training phase, it is actually the starting point of training. That is, the hyper-parameters must be chosen in advance. Hence, we can use only the training set to assess hyper-parameters in the model selection procedure. For this purpose, one usually employs the cross-validation technique [48-50]. In a K -fold cross-validation procedure, the training set is divided into K roughly equal-sized parts. The LS-SVM is trained using a candidate hyper-parameter vector based on $K-1$ parts and is then tested (validated) on the remaining one. This process is repeated K times and the hyper-parameters are evaluated by averaging the validation error across the K validation parts. When $K=n$, the cross-validation is referred to as the leave-one-out cross-validation procedure and the corresponding cross-validation error is called the leave-one-out error for short. Let us denote the number of errors in the leave-one-out cross-validation procedure by $L(\mathbf{x}_1, y_1, \dots, \mathbf{x}_n, y_n)$. It is well known that the

Chapter 3: Model Selection for LS-SVM

---

leave-one-out error provides an almost unbiased estimate of the expected generalization error [51].

Lemma 3.1:
$$E(p_{err}^{n-1}) = \frac{1}{n} E(L(\mathbf{x}_1, y_1, \dots, \mathbf{x}_n, y_n))$$

where p_{err}^{n-1} is the probability of test error for the classifier trained on a training set of size $n-1$ and the expectations are taken over the random choice of the pattern.

According to the above Lemma, the leave-one-out error is a good choice for assessing hyper-parameters in the model selection procedure. However, it may be very costly if we compute it by training the classifier n times for each candidate hyper-parameter vector. In particular, since training the LS-SVM requires a computational cost of $O(n^3)$, the computational cost of leave-one-out cross-validation procedure is $O(n^4)$, which quickly becomes intractable as the number of training patterns increases. Fortunately, the leave-one-out error of LS-SVM can be calculated using an efficient implementation.

Let us denote by $\hat{f}^p$ the LS-SVM trained with training pattern $\mathbf{x}_p$ being kept out, and f^p corresponds to the f computed without $\mathbf{x}_p$. The leave-one-out error of LS-SVM can be written as:

$$looe = \frac{1}{2n} \sum_{i=1}^n (1 - y_i \hat{f}^i(\mathbf{x}_i)) = \frac{1}{2n} \sum_{i=1}^n (1 - y_i \cdot \text{sgn}(f^i(\mathbf{x}_i))) \quad (3.3)$$

It can be proved that

Lemma 3.2: For any training pattern $\mathbf{x}_p$, there is

$$1 - y_p f^p(\mathbf{x}_p) = \frac{\alpha_p}{(\mathbf{H}^{-1})_{pp}} \quad (3.4)$$

where α_p is the p th Lagrange multiplier that is computed by training the LS-SVM

using the entire training set. $\mathbf{H} = \begin{bmatrix} \mathbf{K} + \gamma^{-1} \mathbf{I}_n & \mathbf{1}_n \\ \mathbf{1}_n^T & 0 \end{bmatrix}$ (where $\mathbf{K}$ is the kernel matrix

with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$) and $(\mathbf{H}^{-1})_{pp}$ denotes the p th diagonal entry of $\mathbf{H}^{-1}$ (see Appendix A.1 for proof).

A straightforward corollary of Lemma 3.2 is that the leave-one-out error of LS-SVM can be computed by¹:

$$looe = \frac{1}{2} - \frac{1}{2n} \sum_{i=1}^n (y_i \cdot \text{sgn}(f^i(\mathbf{x}_i))) = \frac{1}{2} - \frac{1}{2n} \sum_{i=1}^n \text{sgn} \left(1 - \frac{\alpha_i}{(\mathbf{H}^{-1})_{ii}} \right) \quad (3.5)$$

One can find that Eq. (3.5) only requires training the LS-SVM once with the whole training set, which can be computed as a cost $O(n^3)$. Hence, the hyper-parameters can be tuned more efficiently by using Eq. (3.5).

3.2.2 Shrinking Grid Search for Model Selection

Given leave-one-out error as the assessment criterion, the hyper-parameters of LS-SVM are traditionally tuned through a shrinking grid search [3,15]. Using Gaussian kernel as an example (for which we need to tune the kernel parameter σ and the regularization parameter γ), the shrinking grid search is summarized in Figure 3.1.

It can be easily verified that the traditional approach requires a computational cost of $O(\text{size-of-grid} \times \text{iteration}_{\max} \times n^3)$, where size-of-grid denotes the cardinality of the set $\Sigma_{\text{iteration}} \times \Gamma_{\text{iteration}}$.

¹It should be noted that Cawley and Talbot [52] and Gestel *et al.* [53] proposed different efficient computation for the leave-one-out error of LS-SVM. Although these computations were proved in a quite different way and take different forms, they generally provide equivalent performance and property as our computation.

Chapter 3: Model Selection for LS-SVM

Model selection using shrinking grid search

---

1. Set $iteration = 1$, initialize tuning sets Σ_1 and Γ_1 for σ and γ respectively, e.g. $\Sigma_1 = \{0.02, 0.2, 2, 20, 50, 100\}$, $\Gamma_1 = \{0.01, 0.05, 0.2, 2, 10, 20, 50\}$. The first candidate set of hyper-parameter vectors are formed by combining Σ_1 with Γ_1 (denoted by $\Sigma_1 \times \Gamma_1$).
  2. For each hyper-parameter vector from $\Sigma_{iteration} \times \Gamma_{iteration}$, employ Eq. (3.5) to calculate the leave-one-out error of LS-SVM.
  3. Choose the optimal (σ, γ) corresponding to the lowest leave-one-out error.
  4. **If** $iteration = iteration_{max}$ (the $iteration_{max}$ is a pre-defined natural number), outputs the currently optimal (σ, γ) .
Otherwise, set $iteration = iteration + 1$, construct a locally refined grid $\Sigma_{iteration} \times \Gamma_{iteration}$ around the currently optimal (σ, γ) and go back to step 2.
-

Figure 3.1: The shrinking grid search approach for model selection

---

3.3 Gradient-based Model Selection

Although grid search has shown some satisfactory performance for tuning the hyper-parameters of LS-SVM, it quickly becomes intractable as the number of hyper-parameters increases. Further, the performance of grid search is largely influenced by the initial set $\Sigma_1 \times \Gamma_1$ of hyper-parameter vectors, which is manually determined. If the optimal hyper-parameter vector does not lie in the neighborhood of the best hyper-parameter vector in the initial set, the grid search will never have a chance to find the optimal hyper-parameters. Hence, one may prefer alternative methods that can not only handle more hyper-parameters simultaneously, but also are less sensitive to the initializations. In the following, we present an approach based on a modified version of the leave-one-out error and the gradient descent algorithms.

Chapter 3: Model Selection for LS-SVM

In [54], Bengio firstly sketched the general methodology of using a gradient descent algorithm to tune hyper-parameters of a learning algorithm. This methodology was then extensively studied and employed to tune hyper-parameters for the standard SVM [55-58] and the kernel Fisher discriminant analysis [59]. The key point of Bengio's work is to design a differentiable estimator of the generalization error, and then employ a gradient descent algorithm to minimize it with respect to the hyper-parameters. Although Eq. (3.5) provides an efficient computation of the leave-one-out error of LS-SVM, it is not differentiable since an indicator function $\text{sgn}(\cdot)$ is included. Hence, Eq. (3.5) cannot be minimized directly using a gradient descent algorithm and needs to be modified. Taking a closer look at Eq. (3.3), we can find that

$$looe = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} (1 - \text{sgn}(y_i f^i(\mathbf{x}_i))) \quad (3.6)$$

where $\frac{1}{2} (1 - \text{sgn}(y_i f^i(\mathbf{x}_i)))$ is zero if $f^i(\mathbf{x}_i) > 0$, otherwise $\frac{1}{2} (1 - \text{sgn}(y_i f^i(\mathbf{x}_i)))$ is one. So we can replace the term $\frac{1}{2} (1 - \text{sgn}(y_i f^i(\mathbf{x}_i)))$ with a logistic function of the form $\frac{1}{1 + \exp(\lambda y_i f^i(\mathbf{x}_i))}$ and modify Eq. (3.6) as

$$mlooe = \frac{1}{n} \sum_{i=1}^n \frac{1}{1 + \exp(\lambda y_i f^i(\mathbf{x}_i))} \quad (3.7)$$

Figure 3.2 depicts the functions $\frac{1}{2} (1 - \text{sgn}(t))$ and $\frac{1}{1 + \exp(\lambda t)}$, it can be seen that the latter formula can well approximate the former one if λ is sufficiently large.

In addition, Figure 3.3 depicts variations of the leave-one-out error and modified leave-one-out error with respect to the kernel parameter σ of a Gaussian kernel on

Chapter 3: Model Selection for LS-SVM

the Heart dataset. We can observe that the modified leave-one-out error successfully follows the trend of the leave-one-out error. Thus, we should be able to achieve good generalization performance by minimizing the modified leave-one-out error.

Figure 3.2: Depiction of the functions $f1 = \frac{1}{2}(1 - \text{sgn}(t))$ and $f2 = \frac{1}{1 + \exp(\lambda t)}$

Figure 3.3: Variation of the leave-one-out error and modified leave-one-out error with respect to $\log_2(\sigma)$ on the Heart dataset.

Since the modified leave-one-out error is differentiable, we can minimize it using the gradient descent algorithms. According to the chain rule, derivative of the modified

Chapter 3: Model Selection for LS-SVM

leave-one-out error with respect to any hyper-parameter t can be computed as:

$$\frac{\partial m_{looe}}{\partial t} = - \sum_{i=1}^n \left(\frac{\lambda y_i \exp(\lambda y_i f^i(\mathbf{x}_i))}{[1 + \exp(\lambda y_i f^i(\mathbf{x}_i))]^2} \cdot \frac{\partial f^i(\mathbf{x}_i)}{\partial t} \right) \quad (3.8)$$

As the term $\frac{\lambda y_i \exp(\lambda y_i f^i(\mathbf{x}_i))}{[1 + \exp(\lambda y_i f^i(\mathbf{x}_i))]^2}$ can be readily calculated in the leave-one-out

cross-validation procedure, we only need to focus on computing the term $\frac{\partial f^i(\mathbf{x}_i)}{\partial t}$,

which is

$$\frac{\partial f^i(\mathbf{x}_i)}{\partial t} = - \frac{1}{(\mathbf{H}^{-1})_{ii}} \cdot \frac{\partial \alpha_i}{\partial t} + \frac{\alpha_i}{(\mathbf{H}^{-1})_{ii}^2} \cdot \frac{\partial (\mathbf{H}^{-1})_{ii}}{\partial t} \quad (3.9)$$

Let $\bar{\mathbf{\Omega}} = \begin{bmatrix} \mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n & \mathbf{Y} \\ \mathbf{Y}^T & 0 \end{bmatrix}$ and $\bar{\mathbf{1}}_n = \begin{bmatrix} \mathbf{1}_n \\ 0 \end{bmatrix}$. According to Eq. (3.2), α_i can be

computed by $\alpha_i = (\bar{\mathbf{\Omega}}^{-1} \bar{\mathbf{1}}_n)_i$. Since $\frac{\partial \mathbf{M}^{-1}}{\partial t} = -\mathbf{M}^{-1} \frac{\partial \mathbf{M}}{\partial t} \mathbf{M}^{-1}$ for any invertible matrix

$\mathbf{M}$, Eq. (3.9) can be further written as:

$$\frac{\partial f^i(\mathbf{x}_i)}{\partial t} = \frac{1}{(\mathbf{H}^{-1})_{ii}} \cdot \left(\bar{\mathbf{\Omega}}^{-1} \frac{\partial \bar{\mathbf{\Omega}}}{\partial t} \bar{\mathbf{\Omega}}^{-1} \bar{\mathbf{1}}_n \right)_i - \frac{\alpha_i}{(\mathbf{H}^{-1})_{ii}^2} \cdot \left(\mathbf{H}^{-1} \frac{\partial \mathbf{H}}{\partial t} \mathbf{H}^{-1} \right)_{ii} \quad (3.10)$$

Gradient-based model selection approach

1. Initialize the hyper-parameters to some value
2. Apply the hyper-parameters to the LS-SVM and calculate the modified leave-one-out error.
3. Update the hyper-parameters such that the modified leave-one-out error is minimized, this is typically achieved by a gradient step using Eqs. (3.8)-(3.10).
4. Go to step 2 until minimum of the modified leave-one-out error is reached.

Figure 3.4: The gradient-based model selection approach

Chapter 3: Model Selection for LS-SVM

With Eqs. (3.8)-(3.10), the gradient-based model selection approach is described in Figure 3.4. Since calculating derivative of the modified leave-one-out error requires inverting the matrices $\mathbf{H}$ and $\bar{\mathbf{\Omega}}$, computational complexity of the gradient-based approach is $O(\text{No. of hyper-parameters} \times \text{No. of iterations} \times 2n^3)$. Further, it is noteworthy that the gradient-based approach is only applicable to the cases where a differentiable kernel function is used.

3.4 Evolutionary-based Model Selection

Although the gradient-based model selection approach is more advantageous than the grid search method, it still has some drawbacks. First, the kernel function has to be differentiable. Secondly, the gradient descent algorithm actually does not minimize the leave-one-out error directly, but an approximation of it. One must set a new parameter λ to an appropriate value so that the optima of the modified leave-one-out error and the leave-one-out error lie in the approximately same position in the hyper-parameter space. Finally, it is well known that a gradient descent algorithm is likely to find a local optimum. If the modified leave-one-out error is discontinuous or contains a lot of local minima, the gradient-based approach may not work well. Here, we suggest another model selection approach that does not suffer from the above-described limitations. More specifically, we propose using the evolution strategies (ES) [44-47,60] to minimize the leave-one-out error with respect to the hyper-parameters.

As a class of evolution-based optimization algorithms, evolution strategies can be viewed as population-based variants of generate-and-evaluate algorithms. Here, we

employ the standard ES algorithm. The algorithm is initialized by generating a set of μ_p individuals that form the parent population. Each individual represents a candidate solution, in our case a real-valued hyper-parameter vector, for the optimization problem at hand. At each iteration of the ES algorithm, $\mu_o > \mu_p$ offspring individuals are generated by partially stochastic variations of parent individuals. The fitness of both the parent and offspring population is evaluated and the μ_p best individuals among them form the next parent population. This loop of variation and selection is repeated until a termination criterion is met. Based on the ES algorithm, the major steps of our evolutionary-based model selection approach are summarized as in Figure 3.5.

Fitness evaluation

One main advantage of the evolutionary algorithms over the gradient descent algorithms is that the evolutionary algorithms can directly optimize a non-differentiable objective function. Hence, instead of using the modified leave-one-out error, the evolutionary-based approach assesses the fitness of a hyper-parameter vector using the leave-one-out error of LS-SVM. That is, the fittest hyper-parameter vector corresponds to the smallest leave-one-out error.

Gaussian mutation

In the standard ES algorithm, offspring are generated using a Gaussian mutation. Given a parent individual $\mathbf{v}_p$, an offspring individual $\mathbf{v}_o$ is generated by the following formula

$$\mathbf{v}_o^h = \mathbf{v}_p^h + N(0,1) \quad (3.11)$$

Chapter 3: Model Selection for LS-SVM

---

where $N(0,1)$ denotes a Gaussian random variable with zero mean and unit standard deviation. v_o^h and v_p^h denote the h th entries of $\mathbf{v}_o$ and $\mathbf{v}_p$ respectively.

Evolutionary-based model selection approach

---

1. Randomly generate an initial population of μ_p hyper-parameter vectors and calculate their fitness using Eq. (3.5).
  2. Use all initial vectors as parents to create μ_o offspring vectors by Gaussian mutation.
  3. Calculate the fitness of the both $\mu_p + \mu_o$ vectors using Eq. (3.5), and prune the whole population to μ_p fittest vectors.
  4. Go to the next step if the maximum number of generations (iterations) has been reached. Otherwise, go to Step 2.
  5. Choose the fittest one in the population as the optimal hyper-parameter vector.
-

Figure 3.5: The evolutionary-based model selection approach

The approach requires a computational cost of $O((\mu_p + \mu_o \times \text{No. of generations}) \times n^3)$.

3.5 A Comparative Study

3.5.1 Qualitative Comparisons

By now, three types of model selection approaches have been presented for the LS-SVM. By comparing them, we observe the following main differences:

1. Grid search scales exponentially in the number of hyper-parameters. So it becomes highly time consuming or even intractable as the number of hyper-parameters increases. Practically, grid search is claimed to be only suited

for problems with no more than three hyper-parameters [55]. In contrast, a simple evolution strategy scales linearly on functions that are monotone with respect to the distance to the optimum [46] [61]. So the evolutionary-based model selection approach can generally handle more hyper-parameters. But the performance of an evolutionary algorithm may not be satisfactory when optimizing more than 50 variables simultaneously. Finally, gradient descent algorithms scale very well to high-dimensional problems. Thus, the gradient-based model selection approach is able to handle even more hyper-parameters than the evolutionary-based approach.

2. As discussed before, computational complexities of the three approaches are given as:

Grid search: $O(\text{Size of the grid} \times \text{No. of iterations} \times n^3)$

Gradient-based: $O(\text{No. of hyper-parameters} \times \text{No. of iterations} \times 2n^3)$

Evolutionary-based: $O((\mu_p + \mu_o \times \text{No. of generations}) \times n^3)$

It can be observed that the time requirement of the three approaches is largely influenced by the size of the grid, the number of iterations (for the gradient-based approach) and the number of generations (for the evolutionary-based approach).

For the cases where all the three approaches are applicable (i.e. the kernel functions only involves one or two kernel hyper-parameters), the evolutionary-based approach is generally the most time consuming. Comparing the remaining two approaches, the gradient-based approach will be less costly if we have little knowledge about the optimization problem at hand and thereby have to build a grid of very large size. Otherwise, the grid search approach can be implemented more efficiently.

3. The gradient-based model selection approach is only applicable to differentiable

Chapter 3: Model Selection for LS-SVM

---

kernel functions, while the other two methods can deal with any type of kernels.

3.5.2 Experiments

Datasets

For the evaluation of our model selection approaches, we conducted a series of experiments on eight benchmark datasets, namely the Banana, Breast-cancer, Diabetes, Heart, Image segment, Splice, Thyroid and Waveform datasets. These datasets were first processed by Rätsch *et al.* [8] and then widely used in subsequent studies [12,62-64]. Every input feature of the datasets is normalized to zero mean and unit standard deviation. Each dataset is independently partitioned into disjoint training and testing sets for 100 times. Thus, the performance of an algorithm can be evaluated by averaging the testing error over the 100 partitions. These datasets, as well as the 100 training and testing splits are available at:

<http://ida.first.gmd.de/~raetsch/data/benchmarks.htm>.

Dataset	No. of training patterns	No. of testing patterns	No. of input features
Banana	400	4900	2
B-Cancer	200	77	9
Diabetis	468	300	8
Heart	170	100	13
Image	1300	1010	18
Splice	1000	2175	60
Thyroid	140	75	5
Waveform	400	4600	21

Table 3.1: Summary of the eight datasets. The No. of training and testing patterns indicate the patterns in one training/testing partition.

Kernels

In the experiments, model selection approaches were employed to optimize the standard Gaussian kernel function and the feature-scaling Gaussian kernel function. The standard Gaussian function contains only one kernel parameter σ and takes the form $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\sigma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$. For this kernel, all the three approaches can be exploited to tune the two hyper-parameters. In a feature-scaling kernel, each input feature of the pattern vector is associated with a scaling factor. That is, the feature-scaling Gaussian kernel takes the form $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\sum_{k=1}^d \sigma_k \|x_i^k - x_j^k\|^2)$, where x_i^k denotes the k th entry (feature) of pattern vector $\mathbf{x}_i$. As the feature-scaling Gaussian kernel can be regarded as a generalized version of the standard Gaussian kernel, it was also referred to as the general Gaussian kernel in some related works [43,65]. Since the feature-scaling Gaussian kernel is parameterized by more variables (i.e. σ_k 's), it is more complex in comparison to the standard Gaussian kernel. When using the feature-scaling Gaussian kernel, one is required to tune $d+1$ hyper-parameters for the LS-SVM. Thus, only the gradient-based and evolutionary-based approaches are applicable to this kernel.

Setups

For every dataset, each combination of the kernel function and model selection approach was independently applied to all the 100 partitions. For each partition, the hyper-parameters were first tuned based on the training set, and then the LS-SVM was trained and applied to the corresponding testing set. Performance of the model selection approach was evaluated by averaging the testing error over the 100

Chapter 3: Model Selection for LS-SVM

---

partitions.

For grid search, we took the settings used by Suykens *et al.* in their benchmarking work [3,15]. Specifically, the grid search started with the initial tuning set $\Sigma_1 = \{4 \times 10^{-6}, 1.6 \times 10^{-5}, 10^{-4}, 4 \times 10^{-4}, 1.6 \times 10^{-3}, 4.5 \times 10^{-3}, 10^{-2}, 4 \times 10^{-2}, 4\}/d$ for σ and $\Gamma_1 = \{0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000\}$ for γ . The number d of input features was considered because the value of $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ in the standard Gaussian kernel grows with the value of d . Further, the value of $iteration_{\max}$ was chosen to be 3.

For the gradient-based approach, λ was set to be 10 for all the datasets. We have found that such a setting works well on all the eight datasets. In practice, one can also try several different values for λ and choose the one yielding the smallest leave-one-out error. The gradient descent algorithm provided in the optimization toolbox of Matlab was used in our experiments. It includes second order updates to improve the convergence speed. We used a loose stopping criterion.

For the evolutionary-based approach, μ_p and μ_o were set to be 60 and 100 respectively. The number of generations was set at 200.

Results

Table 3.2 presents the average testing errors (with standard deviation) achieved the datasets with respect to the corresponding combinations of kernel function and model selection approach. We also used a two-tailed t -test (significance level 0.05) to compare the results. It showed that:

1. In the experiments, both proposed algorithms were randomly initialized. From the Table 3.2, we can observe that both the evolutionary-based approach and the gradient-based approach can achieve quite stable results (i.e. the standard deviation of the classification error is sufficiently small). Hence, we can generally say that both approaches are not very sensitive with regard to the initialization. The difference is, evolutionary algorithms are more likely to find the global optimal solution even with a totally random initialization, while gradient descent algorithms often “stably” find the local optimal solution.
2. Usage of the feature-scaling Gaussian kernel generally leads to better performance in comparison to the standard Gaussian kernel. In other words, the usage of a more complicated kernel function may generally lead to better classification performance. But, feature-scaling Gaussian kernel is only significantly better on the image datasets. The reason is that the standard Gaussian kernel may not perform ideally if the problem in hand is quite complex. In such case, we obtain more flexibility by using the feature-scaling Gaussian kernel and thereby can achieve a classifier that fits the problem better. On the other hand, the standard Gaussian kernel is a simplified version of the feature-scaling Gaussian kernel. Thus, the two kernels will yield similar generalization performance if the problem in hand is relatively easy, as we can observe from the remaining seven datasets.
3. In case of using the feature-scaling Gaussian kernel, the evolutionary-based model selection approach performs better than the gradient-based approach on 6 out of 8 datasets. As we mentioned, gradient descent algorithms are usually

Chapter 3: Model Selection for LS-SVM

---

regarded as local search methods, while evolutionary algorithms are believed to be global search method that are more likely to find the global optimum. Hence, we can generally conclude that conducting a global search instead of a local search might benefit in many classification problems.

4. When the number of hyper-parameters is small (i.e. the standard Gaussian kernel is used), the gradient-based and evolutionary-based model selection approaches do not give significantly better generalization performance than the grid search method. But since the grid search approach is more sensitive to the initialization. Our approaches still provide some practical advantages.

In summary, usefulness of the gradient-based and evolutionary-based model selection approaches are finally verified as: The two approaches can effectively optimize the feature-scaling kernels, which cannot be tuned by traditional model selection method. And the optimized feature-scaling kernels may significantly outperform the standard kernels for tough classification problems.

3.6 Conclusions

In this chapter, we mainly address the model selection problem for the LS-SVM. First, an efficient computation of the leave-one-out error of LS-SVM is proposed. Based on this efficient computation, two approaches are proposed to tune hyper-parameters of LS-SVM. They employ gradient descent algorithms and evolution strategies, respectively. Compared to the traditional grid search method that is only able to handle very small number of hyper-parameters, our approaches allow optimizing a large number of hyper-parameters simultaneously. Consequently, they make possible

Chapter 3: Model Selection for LS-SVM

the use of highly complex and tunable kernels, with which the LS-SVM may perform significantly better on tough tasks.

Dataset	Grid-G	Grad-G	Evo-G	Grad-FSG	Evo-FSG
Banana	10.79 ± 0.61	10.61 ± 0.67	10.72 ± 0.65	10.59 ± 0.55	10.59 ± 0.61
B-Cancer	26.77 ± 4.52	27.17 ± 4.63	26.31 ± 4.36	26.99 ± 4.52	26.28 ± 4.47
Diabetis	23.49 ± 1.81	23.47 ± 1.81	23.74 ± 1.86	23.38 ± 1.77	23.60 ± 1.82
Heart	16.41 ± 3.55	16.70 ± 3.51	16.75 ± 3.60	16.61 ± 3.87	16.33 ± 3.68
Image	2.84 ± 0.67	2.87 ± 0.67	2.84 ± 0.55	2.79 ± 0.62	$2.05 \pm 0.53^*$
Splice	10.89 ± 0.72	10.80 ± 0.63	10.88 ± 0.71	10.84 ± 0.58	10.93 ± 0.70
Thyroid	5.11 ± 2.27	5.03 ± 1.89	5.09 ± 2.19	4.83 ± 2.07	4.55 ± 2.19
Waveform	10.01 ± 0.56	10.08 ± 0.56	9.90 ± 0.53	9.85 ± 0.50	9.70 ± 0.52

Table 3.2: Comparisons among five combinations of model selection approach and kernel function: Grid search with the standard Gaussian kernel (Grid-G), Gradient-based method with the standard Gaussian kernel (Grad-G) and the feature-scaling Gaussian kernel (Grad-FSG), Evolutionary-based method with the standard Gaussian kernel (Evo-G) and the feature-scaling kernel (Evo-FSG). The testing error (with standard deviation) on each dataset is calculated using the LS-SVM and averaging over all the partitions. The best results for each dataset are presented in bold, and “*” indicate that the result is significantly better than the remaining results in the same row.

Chapter 4

Feature Selection Algorithms Based on LS-SVM

4.1 Introduction

In the previous chapter, we have demonstrated that LS-SVM may achieve better performance with more complicated nonlinear kernel functions. But, it has also been experimentally shown that not all real-world classification problems are so difficult that it can only be well-solved by complex nonlinear classifiers. If a problem is characterized by both very large number of features (say, more than 1000) and small number of training patterns (typically less than 100), a linear classifier will be sufficient and a nonlinear classifier may even over-fit the data. In such a scenario, classification is no longer the most challenging task. Instead, one may be more interested in the problem of feature selection. That is, to find the most important features from the original feature set. By using only those important features for classification, not only the over-fit problem can be restrained, but also the computational cost of classification can be reduced.

Among real-world applications, microarray data classification is exactly the problem that we described above. Recently, pattern classification techniques have been widely applied to microarray data to assist diagnosis [66-69]. Given some microarray data characterized by a large number of genes' expressions, a classifier is constructed based

---

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

on the given data to distinguish between different disease types. In practice, a gene selection procedure is usually implemented prior to the construction of classifiers. There are several reasons for performing gene selection. First, the cost of clinical diagnosis can be reduced with gene selection since it is much cheaper to focus on only the expressions of a few genes for diagnosis instead of the whole gene set. Second, many of the genes are redundant. Although the training error of a classifier on the given data will decrease as more and more genes are included, the generalization error when classifying new data eventually will increase [70]. Hence, a preceding gene selection procedure can remove the redundant genes, reduce storage requirement and computational complexity of classification, and possibly reduce the generalization error. Finally, and maybe the most important to biologists, gene selection provides a more compact gene set, which can help understand the functions of particular genes and plan the diagnosis process.

In this chapter, we demonstrate that the LS-SVM is not only a classifier, but also can be modified to resolve feature selection tasks for high-dimensional small-size dataset. In particular, we present our work in the context of gene selection since it is one of the representative real-world applications.

4.1.1 A Brief Overview of Feature Selection Methods

Given n training patterns represented by d features, a feature selection method aims to select a feature subset that leads to a low generalization error. It searches for an optimal or near optimal subset of features with respect to a given criterion, and consists of two basic components: an evaluation criterion and a search strategy.

Chapter 4: Feature Selection Algorithms Based on LS-SVM

In the literature, feature selection methods are usually categorized according to the evaluation criteria they use. Generally feature selection methods can be categorized into three major groups: filters, wrappers and embedded methods [71,72]. Filter methods use general characteristics of the training set as the evaluation criteria to select the features. They are independent of the classifier that will use the selected features. Thus, filter methods can be combined with any type of classifier. In a wrapper method, feature selection and classification are simultaneously considered. That is, a classifier is repeatedly built in the feature selection procedure and its classification accuracy is used for evaluation of feature subsets. After that, the same type of classifier is employed for the final classification. In this way, the feature selection procedure is closely related to the decision mechanism of the specific classifier and therefore the wrapper methods are expected to lead to higher classification accuracy. If a feature selection method derives the evaluation criterion from intrinsic properties of a classifier (i.e. a metric which is different from classification accuracy) and the same type of classifier is employed for the subsequent classification, this feature selection method will be categorized as an embedded method [71]. The above-described categorization can be demonstrated by the following example. Assume that we are going to select a feature subset and feed it to a support vector machine (SVM) classifier. If we select features according to their standard deviations over the training patterns, then the resultant feature selection method is a filter method, because the standard deviation of a feature has nothing to do with the classifier. Alternatively, we can build a SVM classifier for each candidate feature subset, and select the subset corresponding to the lowest training or cross-validation error of the SVM. Such a method should be categorized as a wrapper

---

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

method. Finally, we can also firstly train a SVM with the whole feature set, and then sequentially eliminate the features corresponding to the smallest weights in the vector normal to the optimal hyper-plane [73]. In this way, we arrive at the SVM Recursive Feature Elimination (SVM-RFE) algorithm, which is an embedded method. From the above example, it can be observed that wrapper and embedded methods are often closely related to each other. Both the methods require training SVM classifiers in the feature selection procedure. The only difference is whether the classification accuracy or an intrinsic metric is employed as the evaluation criterion. Since the term “intrinsic” is quite vague and actually covers a very broad range, numerous existing feature selection algorithms can be categorized as embedded methods, recent related examples are the SVM-based feature selection methods [74].

Besides the evaluation criteria, feature selection methods are also characterized by the search strategies they employ. In general, feature selection can be understood as searching for the optimal feature subset in a feature subset space, which typically contains 2^d elements. From the perspective of search strategies, perhaps the simplest feature selection method is to score each feature individually based on the evaluation criterion, and then to select the l features with the highest scores. This type of methods is referred to as marginal filter or individual feature ranking [73]. By taking different values of l , a marginal filter method merely covers a small region (of size d) of the feature subset space and thereby results in sub-optimal feature subsets in most cases. On the other hand, if a method explicitly (such as using an exhaustive search) or implicitly (such as using the branch-and-bound strategy [75]) searches over the whole feature subset space, it is guaranteed to discover the optimal feature subset with respect to the evaluation criterion. However, an exhaustive search or the

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

branch-and-bound strategy is computationally prohibitive except for small-sized problems. Therefore, one usually searches a part of the whole feature subset space, which is more practicable but provide no optimality guarantees [76]. The sequential forward selection, sequential floating forward selection, sequential backward elimination and sequential floating backward elimination strategies [77] fall into this category. The sequential forward selection strategy starts from an empty set and sequentially includes a new feature into the feature subset so that the largest improvement on the evaluation criterion can be achieved. Once a feature is selected, it will not be removed from the subset. Differently, the sequential floating forward selection strategy contains two steps. First, the feature leading to the largest improvement is included, and then the strategy back-tracks the search path and removes one or more previously selected features if improvement can be achieved by doing so. Sequential backward elimination strategy sequentially removes features from the whole feature set until an optimal feature subset is remained. And the sequential floating backward elimination strategy allows inclusion of previously removed features to the current feature subset. Hence, the floating strategies cover a larger portion of all the possible feature subsets, while it is more time consuming [77]. Recently, genetic algorithms (GAs) have also been employed as search strategies [78-81]. Compared with the traditional search strategies, GAs provide a more flexible search procedure, the feature subset space is searched in parallel and multiple feature subsets instead of a single subset are evaluated simultaneously to avoid being trapped in a local optimum. GAs are generally even more time consuming than the floating strategies, although it can cover more feature subsets.

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Based on the above, two issues should be considered when assessing feature selection methods: the generalization error that can be achieved on the selected feature subset and the time requirement of the selection procedure. A good feature selection method should contain following characteristics: The evaluation criterion can guarantee low generalization error, computational cost for a single evaluation is low, the search strategy requires a small number of evaluations while can still cover a sufficiently large portion of the whole feature subset space in order to include the optimal solutions. Among the methods discussed above, the marginal filter methods are the most efficient, but the selected feature subsets are usually sub-optimal. The wrapper/embedded methods using an exhaustive search are the most time consuming, but optimality can be guaranteed. All the other methods lie between these two cases, providing a trade-off between optimality and computational cost.

4.1.2 Feature Selection Methods Applied to Microarray Data

In the context of microarray data analysis, many of the methodologies discussed above have been employed for gene selection. Typical examples are the marginal filter methods using t -statistics, Fisher's ratio and information gain, the wrapper methods based on k -nearest neighbor classifier [81] and the embedded methods originated from the SVM formulation [73,74]. Not only traditional search strategies, but also GA-based algorithms are now available in the literature of gene selection [79-81]. In comparison to a general feature selection task, gene selection is *special* for two reasons: First, a gene selection problem usually involves both small number of training patterns and large number of features. Hence, many existing feature selection methods may encounter difficulties when applied to gene selection problems. And it is

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

necessary to modify them accordingly. Besides, in addition to finding an optimal gene subset for classification, gene selection also aims to identify important genes. Identifying important genes is essentially different from finding a single optimal gene subset. For the microarray data, a classifier may be able to achieve the lowest generalization error on many different gene subsets, and all of them consist of important genes. Therefore, knowing these different gene subsets can help gain more insight into the biological functions of genes.

In the following, we propose two methods to address the gene selection problem. First, an evaluation criterion called the leave-one-out calculation (LOOC) criterion is presented. This criterion is derived from the exact and efficient calculation of the leave-one-out error of LS-SVM, which has been presented in Chapter 3. By combining the LOOC criterion with the sequential forward selection strategy, we then propose the leave-one-out calculation sequential forward selection (LOOCSFS) algorithm. After that, we present a novel gene selection algorithm, named gradient-based leave-one-out gene selection (GLGS) algorithm. Employing none of the traditional search strategies, the GLGS algorithm combines a variant of the LOOC criterion with the gradient descent algorithm and the principal component analysis (PCA). Performance of the proposed methods is evaluated experimentally on four microarray datasets. It is noteworthy that the terms “gene” and “feature” generally refer to the same thing (the input feature of microarray data) throughout this chapter. The only exception is given when describing the GLGS algorithm, where we define the term “feature” in a slightly different way.

We present the LOOCSFS algorithm in Section 4.2, and then describe the GLGS

algorithm in Section 4.3. After that, the two algorithms are experimentally compared with some state-of-art gene selection methods in Section 4.4. Finally, discussion and conclusions are presented in Section 4.5.

4.2 Leave-one-out Calculation Sequential Forward Selection (LOOCSFS) Algorithm

4.2.1 The Algorithm

Generally speaking, the LOOCSFS algorithm is a direct by-product of the works that have been presented in Chapter 3. Thus, let us firstly recall some important equations. Assume that the training set is given as $S_{tr} = \{(\mathbf{x}_i, y_i), i = 1, 2, \dots, n\}$, $y_i \in \{-1, +1\}$, a trained LS-SVM is expressed as

$$\hat{f}(\mathbf{x}) = \text{sgn}(f(\mathbf{x})) = \text{sgn}\left(\sum_{i=1}^n \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b\right) \quad (4.1)$$

In microarray data classification, we deal with a small number of high-dimensional pattern vectors. According to previous literature [82-84], the linear kernel is usually more preferable to a non-linear kernel in this scenario. Therefore, throughout this chapter, the linear kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$ is employed, while the presented algorithms can be easily extended by replacing it with nonlinear kernels.

In the leave-one-out cross-validation procedure, let us denote by $\hat{f}^p$ the LS-SVM trained with training pattern $\mathbf{x}_p$ being kept out, and define f^p in a similar manner. For any pattern $\mathbf{x}_p$, there is:

$$y_p f^p(\mathbf{x}_p) = 1 - \frac{\alpha_p}{(\mathbf{H}^{-1})_{pp}} \quad (4.2)$$

Chapter 4: Feature Selection Algorithms Based on LS-SVM

where $\mathbf{H} = \begin{bmatrix} \mathbf{K} + \gamma^{-1} \mathbf{I}_n & \mathbf{1}_n \\ \mathbf{1}_n^T & 0 \end{bmatrix}$ ($\mathbf{K}$ is the kernel matrix with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$) and

$(\mathbf{H}^{-1})_{pp}$ denotes the p th diagonal entry of $\mathbf{H}^{-1}$ and α_p is computed by training the LS-SVM using the entire training set. Consequently, the leave-one-out error of the LS-SVM can be efficiently calculated by

$$looe = \frac{1}{2} - \frac{1}{2n} \sum_{i=1}^n \text{sgn} \left(1 - \frac{\alpha_i}{(\mathbf{H}^{-1})_{ii}} \right) \quad (4.3)$$

Since the leave-one-out error has been proven to be an almost unbiased estimator of the generalization error [51] and can be calculated efficiently for the LS-SVM using Eq. (4.3), it appears to be a good choice for evaluating feature subsets. But since a microarray dataset usually contains only limited number of training patterns and large number of genes (and thereby large number of possible gene subsets), it is very likely that many candidate gene subsets may provide the same leave-one-out error. To further figure the optimal gene subset from those gene subsets that yield the lowest leave-one-out error, we propose the C bound as a supplement to Eq. (4.3):

$$C = \sum_{i=1}^n (1 - \alpha_i / (\mathbf{H}^{-1})_{ii})_- \quad (4.4)$$

where $(t)_- = \min(0, t)$ for t . The C bound is motivated by the following consideration: A training pattern $\mathbf{x}_i$ is misclassified in the leave-one-out cross-validation procedure if $y_i f^i(\mathbf{x}_i) < 0$. Since f^i defines the decision hyper-plane in the kernel space, the absolute value of $y_i f^i(\mathbf{x}_i)$ indicates how close a pattern $\mathbf{x}_i$ is to the decision hyper-plane. Therefore, for those patterns misclassified in the leave-one-out procedure, a small absolute value of $y_i f^i(\mathbf{x}_i)$ is more preferable. Because a small value means that the pattern is close to the decision hyper-plane and

---

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

might be classified correctly with a few more training patterns, while a large absolute value indicates that the pattern may be difficult to classify even if more training patterns are available.

The LOOC sequential forward selection (LOOCSFS) algorithm

Input: The gene set $G = \{g_1, g_2, \dots, g_d\}$

Initialization: Set S as empty, the candidate gene set $P = G$

Do for $i = 1$ to l (l is the number of genes to be selected)

1. for $j = 1 : |P|$ ($|P|$ denotes the number of genes in P)

Temporarily take gene g_j from P , put it into S , calculate the leave-one-out error and C bound using all genes in S .
end

2. If only one gene yield the minimal leave-one-out error, set it as $g_{selected}$.

Otherwise, the gene with the maximal C bound is set as $g_{selected}$.

3. Set $S = S \cup g_{selected}$, remove $g_{selected}$ from P .

End

Output: The selected gene subset S .

Figure 4.1: Pseudo-code of the LOOC sequential forward selection algorithm

---

By combining Eq. (4.3) with Eq. (4.4), we obtain the two-step LOOC criterion for gene selection. First, the optimal gene subset must yield the smallest leave-one-out error. If the same leave-one-out error can be achieved on multiple gene subsets, the one with the largest value of the C bound is preferred (note that calculating the C bound only requires negligible additional computation since the term $1 - \alpha_i / (\mathbf{H}^{-1})_{ii}$

Chapter 4: Feature Selection Algorithms Based on LS-SVM

has been computed by Eq. (4.3)). The LOOC criterion can be calculated by training the LS-SVM with the whole training set only once. Hence, computation of the LOOC criterion requires solving a linear system and thereby is more efficient than the SVM-based criteria that involve solving quadratic programming (QP) problems [73,74]. By combining the LOOC criterion with the sequential forward selection strategy, we propose the LOOCSFS algorithm, which is described in Figure 4.1.

4.2.2 Efficient Computation of the LOOC Criterion

In the above-described LOOCSFS algorithm, each candidate gene subset needs to be evaluated by the LOOC criterion. For a high dimensional dataset such as a microarray dataset, the repeated computation of the Lagrange multipliers α_i 's is computationally intensive since it involves the inverse of an $n \times n$ matrix $\mathbf{M} = \mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n$ (where $\Omega_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$). Fortunately, we can compute the inverse efficiently with the Sherman-Morrison-Woodbury formula [85]:

$$(\mathbf{A} + \mathbf{V}\mathbf{B}\mathbf{V}^T)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{V}(\mathbf{B}^{-1} + \mathbf{V}^T\mathbf{A}^{-1}\mathbf{V})^{-1}\mathbf{V}^T\mathbf{A}^{-1} \quad (4.5)$$

Suppose that we have selected a particular gene subset S and the matrix $\mathbf{M}$ for S is denoted by $\mathbf{M}_S$. During the sequential forward selection procedure, we have to temporarily put each of the candidate genes into S to assess the performance of each candidate gene. When gene g_j is added into S to form a temporary set $S_t = S \cup g_j$, the matrix $\mathbf{M}$ for S_t , denoted by $\mathbf{M}_{S_t}$, can be formulated as:

$$\mathbf{M}_{S_t} = \mathbf{M}_S + \mathbf{z}\mathbf{z}^T \quad (4.6)$$

where $\mathbf{z} = [y_1 x_{1j}, y_2 x_{2j}, \dots, y_n x_{nj}]^T$ and x_{ij} is the j th entry of pattern $\mathbf{x}_i$. According to the Sherman-Morrison-Woodbury formula, we have:

$$\mathbf{M}_{S_t}^{-1} = \mathbf{M}_S^{-1} - \frac{\mathbf{M}_S^{-1} \mathbf{z} \mathbf{z}^T \mathbf{M}_S^{-1}}{1 + \mathbf{z}^T \mathbf{M}_S^{-1} \mathbf{z}} \quad (4.7)$$

The above equation reveals that the inverse of $\mathbf{M}_{S_t}$ can be recursively computed from the inverse of $\mathbf{M}_S$ using simple matrix operations. Thus, in the sequential forward selection procedure, the inverse operation can be implemented efficiently by using the trick of Eq. (4.7).

4.3 Gradient-based Leave-one-out Gene Selection (GLGS) Algorithm

In addition to the sequential forward selection, sequential floating forward selection, sequential backward elimination and sequential floating backward elimination search strategies, a possible alternative search strategy for gene selection is the gradient descent algorithms. Utilizing gradient descent algorithm is not a totally new idea in the literature. On basis of SVM, Chapelle *et al.* [55] suggested using the gradient descent algorithms to address feature selection problems. But the resultant algorithm requires repeatedly solving an optimization problem, whose dimensionality is the same as the total number of features. As the number of genes is usually huge in microarray data, this framework is almost intractable for gene selection problems. Considering the specific properties of microarray data, we propose a novel gene selection algorithm based on LS-SVM, namely the gradient-based leave-one-out gene selection (GLGS) algorithm.

The basic formula

Generally speaking, the basic idea of the GLGS algorithm directly comes from the

Chapter 4: Feature Selection Algorithms Based on LS-SVM

gradient-based model selection approach that has been presented in Chapter 3. That is, we introduce scaling factors $\mathbf{v} = [v_1, v_2, \dots, v_d]^T$ to the linear kernel (i.e. using the feature-scaling linear kernel), and using a gradient descent algorithm to minimize the modified leave-one-out error of LS-SVM (as given in Eq. (4.8)) with respect to the scaling factors. Those genes corresponding to large scaling factors are selected.

$$mlooe = \frac{1}{n} \sum_{i=1}^n \frac{1}{1 + \exp(y_i f^i(\mathbf{x}_i))} \quad (4.8)$$

By introducing the scaling factors, we get the feature-scaling linear kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^d v_k x_{ik} x_{jk} \quad (4.9)$$

where x_{ik} is the k th entry of $\mathbf{x}_i$ and v_k is the corresponding scaling factor. Same as in Chapter 3, we have the denotations: $\mathbf{K} = \{k(\mathbf{x}_i, \mathbf{x}_j)\}$, $\mathbf{\Omega} = \{y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)\}$,

$$\mathbf{H} = \begin{bmatrix} \mathbf{K} + \gamma^{-1} \mathbf{I}_n & \mathbf{1}_n \\ \mathbf{1}_n^T & 0 \end{bmatrix}, \quad \bar{\mathbf{\Omega}} = \begin{bmatrix} \mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n & \mathbf{Y} \\ \mathbf{Y}^T & 0 \end{bmatrix} \quad \text{and} \quad \bar{\mathbf{1}}_n = \begin{bmatrix} \mathbf{1}_n \\ 0 \end{bmatrix}.$$

It can be derived that (see Appendix B.1):

$$\begin{aligned} \frac{\partial mlooe}{\partial v_k} = & \frac{1}{n} \sum_{i=1}^n \frac{\exp(1 - \alpha_i / (\mathbf{H}^{-1})_{ii})}{[1 + \exp(1 - \alpha_i / (\mathbf{H}^{-1})_{ii})]^2 (\mathbf{H}^{-1})_{ii}^2} \cdot \left[\alpha_i \left(\mathbf{H}^{-1} \begin{bmatrix} \frac{\partial \mathbf{K}}{\partial v_k} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} \mathbf{H}^{-1} \right)_{ii} \right. \\ & \left. - \mathbf{H}_{ii}^{-1} \left(\bar{\mathbf{\Omega}}^{-1} \begin{bmatrix} \frac{\partial \mathbf{\Omega}}{\partial v_k} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} \bar{\mathbf{\Omega}}^{-1} \bar{\mathbf{1}}_n \right)_i \right] \end{aligned} \quad (4.10)$$

Substituting $\frac{\partial \mathbf{\Omega}}{\partial v_k} = \{y_i y_j x_{ik} x_{jk}\}$ and $\frac{\partial \mathbf{K}}{\partial v_k} = \{x_{ik} x_{jk}\}$ into Eq. (4.10), the derivatives of

the modified leave-one-out error with respect to the scaling factors can be computed.

Acceleration using Principal Component Analysis

As we know, minimizing Eq. (4.8) with respect to the scaling factors

---

Chapter 4: Feature Selection Algorithms Based on LS-SVM

$\mathbf{v} = [v_1, v_2, \dots, v_d]^T$ results in a d -dimensional optimization problem. Since d is huge for microarray data, direct usage of Eq. (4.10) and a gradient descent algorithm may still be computationally prohibitive. Hence, we propose the following method to reduce the computational cost:

1. The microarray data is linearly transformed into a lower dimensional (say, d_{low}) space by $\mathbf{X}_{low} = \mathbf{TX}$, where the $d \times n$ matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$ consists of the pattern vectors in the original d -dimensional space, $\mathbf{T}$ is a $d_{low} \times d$ transformation matrix and thereby $\mathbf{X}_{low}$ is a $d_{low} \times n$ matrix that represents the pattern vectors in the lower dimensional space. In the literature, this procedure is referred to as dimensionality reduction. For clarity, we say that a pattern vector is represented by d_{low} features in the lower dimensional space and is represented by d genes in the original space.
2. Since $d_{low} < d$, we can optimize the scaling factors (denoted as $\mathbf{v}_{low}$) corresponding to the features of $\mathbf{X}_{low}$ with much less cost.
3. Clearly, absolute values of the entries of $\mathbf{v}_{low}$ indicate the importance of the transformed features for achieving the minimal value of the modified leave-one-out error. Further, it can be seen that each feature in the lower dimensional space is actually a linear combination of the original genes, and absolute values of the matrix $\mathbf{T}$'s entries reveal how important the corresponding genes are for constructing the features of $\mathbf{X}_{low}$. Therefore, we propose to estimate the pseudo scaling factors of the original genes, which are called pseudo because

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

they are not truly optimized, using Eq. (4.11). And then the gene selection is conducted on basis of these pseudo scaling factors.

$$\mathbf{v}_{pseudo} = \text{abs}(\mathbf{T})^T \text{abs}(\mathbf{v}_{low}) \quad (4.11)$$

In the step 1 above, the transformation matrix $\mathbf{T}$ needs to be pre-specified. We recommend to using a principal component analysis (PCA) procedure to find it. (i.e. the rows of $\mathbf{T}$ are the principal components calculated by a PCA procedure.) The reason is that most variation presented in the original data can be preserved in the lower dimensional space when dimensionality reduction is carried out using PCA. More specifically, we use the PCA to transform the original microarray data into an n -dimensional space (i.e. $d_{low} = n$). By this means, we only need to solve an optimization problem whose dimensionality is the number of training patterns and thereby the computational cost is considerably reduced.

Final selection with heuristics

Although we can employ the above-described steps to estimate the pseudo scaling factors for genes, correlations between genes are not considered during the estimation procedure. In the literature, it is well known that correlation between features plays an important role in feature selection problems. On one hand, if a gene set contains highly correlated genes, some of them can be said to be redundant for classification purpose. Since gene selection always aims to find the smallest gene subset that leads to the best generalization performance, redundant genes that marginally contribute to the classification accuracy should be avoided. Hence, highly correlated genes are not desirable in the selected gene subset. On the other hand, the more a gene correlates with other genes, the better it can represent other genes and the more informative it is.

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Therefore, although correlation between selected genes is not desired, correlation between the selected genes and the unselected genes can be helpful. Taking into account the information provided by the estimated pseudo scaling factors, we propose the following heuristic procedure in GLGS algorithm to select the gene subset in a sequential manner. At each iteration, one gene is selected as:

$$g_{selected} = \arg \max_{g_i \in P} \{(1 - \beta_i) \mathbf{R}(i,:) \mathbf{v}_{pseudo}\} \quad (4.12)$$

where P is the set of candidate genes, $\mathbf{R}$ is the $n \times n$ correlation coefficient matrix of the original gene set, $\mathbf{R}(i,:)$ is the i th row of $\mathbf{R}$ and β_i is the largest correlation coefficient between gene g_i and the previously selected genes (i.e. $\beta_i = \max_{g_j \in S} \{R_{ij}\}$). In Eq. (4.12), $\mathbf{R}(i,:)$ assesses the correlation between each candidate gene g_i and all the other genes. The term $\mathbf{R}(i,:) \mathbf{v}_{pseudo}$ can be understood as sharing the pseudo scaling factors among correlated genes. If two genes have the same pseudo scaling factor, the one that is highly correlated to other genes will get a higher score on $\mathbf{R}(i,:) \mathbf{v}_{pseudo}$. In this way, we attempt to bias toward the genes that are more representative. Further, the term $(1 - \beta_i)$ is introduced to restrain correlations between $g_{selected}$ and the previously selected genes. To summarize, the detailed steps of the GLGS algorithm are presented in Figure 4.2.

According to the definitions in [72], the GLGS algorithm can be categorized as an embedded method. Hence, it is more time consuming than a marginal filter method. But the GLGS algorithm differs from previous embedded approaches. It optimizes the evaluation criterion in a transformed space with significantly reduced dimensions, while it selects genes from the original gene set based on results of optimization.

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Therefore, the GLGS algorithm can scale well to high dimensional data and is particularly suitable for microarray data analysis.

The Gradient-based leave-one-out gene selection (GLGS) algorithm

---

Input: The gene set $G = \{g_1, g_2, \dots, g_d\}$, given in the $d \times n$ matrix $\mathbf{X}$

Initialization: Set S as empty, the candidate gene set $P = G$

Do

1. Calculate the correlation coefficient matrix $\mathbf{R}$
2. Perform dimensionality reduction using PCA, compute $\mathbf{X}_{low} = \mathbf{TX}$
3. Introduce a scaling factor vector $\mathbf{v}_{low}$ to $\mathbf{X}_{low}$, optimize it using Eqs. (4.8)-(4.10) and a gradient descent algorithm.
4. Estimate the pseudo scaling factors of the genes using Eq. (4.11)
5. For $i = 1$ to l (l is the number of genes to be selected)
  - a. Using Eq. (4.12) to select a gene $g_{selected}$ from the candidate gene set
  - b. Set $S = S \cup g_{selected}$ and remove $g_{selected}$ from P

End

Output: the selected gene subset S .

Figure 4.2: Pseudo-code of the gradient-based leave-one-out gene selection algorithm

4.4 Experiments

We conducted experiments on four public domain microarray datasets to evaluate the LOOCSFS and GLGS algorithms. They were also compared to other related gene selection algorithms.

4.4.1 Algorithms Chosen for Comparison

As one of the major research topics in the field of pattern recognition, feature selection has been extensively studied for quite a few years. Therefore, numerous feature selection algorithms can be found in the literature. Since the main concern of this chapter is gene selection for microarray data, we decided to compare our LOOCSFS and GLGS algorithms with five other algorithms that had been used to solve the same problem. First, although it usually selects a sub-optimal gene subset for classification, a marginal filter method using Fisher's ratio was employed to provide a baseline for the comparison. Fisher's ratio is a criterion that evaluates how well a single gene is correlated with the separation between classes. For every gene, the

Fisher's ratio is defined as $f = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$, where $\mu_1, \mu_2, \sigma_1, \sigma_2$ denote the means and

standard deviations of the two classes. Besides, two evaluation criteria, namely the Mahalanobis class separability measure [75] and the LS bound criterion [84] were combined with the sequential forward selection strategy and resulted in the Mahalanobis class separability sequential forward selection (MAHSFS) and the LS bound sequential forward selection (LSBSFS) algorithms [84]. Furthermore, we also compared our algorithms with the SVM-RFE [73] and the LS bound sequential floating forward selection algorithms (LSBSFFS) [84].

Among the above algorithms, the MAHSFS algorithm is a traditional approach and has been commonly used in literature. The SVM-RFE algorithm was proposed based on the standard SVM and has been shown to be one of the state-of-the-art approaches for gene selection. The LSBSFS and LSBSFFS algorithms were chosen because they are closely related to the LOOCSFS algorithm. Similar to the LOOC criterion, the LS

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

bound criterion also originated from the LS-SVM formulation. For a training pattern $\mathbf{x}_p$, Zhou and Mao [84] proved that

$$y_p f^p(\mathbf{x}_p) \geq 1 - \alpha_p [(D_{\min}^p)^2 + 2/\gamma] \quad (4.13)$$

where $D_{\min}^p$ is defined as the distance between $\mathbf{x}_p$ and its nearest neighbor in the kernel space and all the other denotations are same as defined for the LOOCSFS algorithm. Since Eq. (4.13) provides a lower bound for the term $y_p f^p(\mathbf{x}_p)$, Mao and Zhou suggested maximizing it. And thereby the LS bound criterion is proposed as:

$$\text{LSB} = \sum_{i=1}^n (\alpha_i [(D_{\min}^i)^2 + 2/\gamma] - 1)_+ \quad (4.14)$$

where $(t)_+ = \max(0, t)$. From the theoretical point of view, LS bound criterion may not be that appealing, since $y_p f^p(\mathbf{x}_p)$ can be exactly calculated by Eq. (4.2) with almost the same computational cost. However, LS bound criterion has been shown to be very successful for practical use, and whether the LOOC criterion is superior to the LS bound criterion must be verified by experimental results. Hence, we chose the LSBSFS and LSBSFFS algorithms for the comparative study.

4.4.2 Datasets

Four public domain microarray datasets were used in our experiments.

Colon Cancer Dataset

The colon cancer dataset [66] contains gene expression levels of 40 tumor and 22 normal colon tissues for 2000 genes. The task is to distinguish colon cancer from normal tissues.

Leukemia Dataset

The leukemia dataset was first described in [67]. The dataset contains gene expression levels of 72 patients with either acute lymphoblastic leukemia (ALL, 47 cases) and acute myeloid leukemia (AML, 25 cases) for 7129 human genes. The raw data are available at <http://www-genome.wi.mit.edu/cancer/>.

Hepatocellular Carcinoma dataset

This dataset comprises information of 60 patients with hepatocellular, with oligonucleotide microarrays representing 7129 gene expression levels [68].

Glioma dataset

The 50 patterns of the Glioma dataset [69] are expressed by 12625 genes. Twenty eight of the patterns are glioblastomas and the other 22 are anaplastic oligodendrogliomas.

4.4.3 Experimental Setup

The comparisons were conducted based on the generalization error achieved on the selected gene subset and the time requirement of the selection procedure. Unlike the datasets used in the previous chapter, all the above-described microarray datasets do not provide separate training and testing sets. We need to manipulate the datasets to estimate the generalization error. In a previous works, Ambroise and McLachlan [70] assessed several generalization error estimation techniques for evaluating gene selection algorithms. They showed that the external 10-fold cross-validation error and the external B.632+ error are the two most unbiased estimators of the generalization error. The “external” means that the cross-validation or bootstrap procedure should be

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

kept external to the gene selection procedure [70,86]. Since cross-validation is claimed to have a relatively higher variance when the training set is small [87], we utilized the external B.632+ technique [88,89] to estimate generalization error. The B.632+ technique employs the bootstrap method [90,91] to generate different training and testing sets (which are called bootstrap samples) from the original dataset (More details about the B.632+ bootstrap is given in Appendix B.2). Then the algorithms are applied to the bootstrap samples as well as the original dataset. Specifically, we employed 200 replicates of balanced bootstrap samples to reduce variance of the B.632+ error, i.e. each pattern in the original dataset was restricted to appear exactly 200 times in total in all the 200 balanced bootstrap samples. A linear LS-SVM was employed as the final classifier for the seven gene selection algorithms. All the compared algorithms terminated if a predefined number of genes were selected. We set this number as 100. Furthermore, we conducted another experiment to study the computational complexity and scalability of the seven gene selection algorithms. The required computational time of the algorithms were studied with respect to the number of genes to be selected (l) and the size of the whole gene set (d). This experiment was conducted on the Hepatocellular Carcinoma dataset, but similar scenario can be easily shown on the other datasets. All our experiments were implemented in the Matlab environment on a computer with 3GHz P4 CPU and 1024 MB RAM.

4.4.4 Results

On the generalization performance

Figures 4.3-4.6 present the external B.632+ errors achieved on the genes selected by the seven gene selection algorithms. Like many other metrics of classification

---

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

accuracy, the external B.632+ error ranges between 0 (the estimated generalization error is 0%) and 1 (the estimated generalization error is 100%). It can be observed that the GLGS algorithm generally performs the best on all four datasets. Its superiority is particularly obvious on the Hepatocellular Carcinoma and Glioma datasets.

The LOOCSFS algorithm does not perform as well as the GLGS algorithm. But according to the figures, it still provides competitive performance on all the four datasets. To compare our LOOC criterion with the LS bound criterion, the performance of LOOCSFS, LSBSFS and LSBSFFS algorithms are also separately plotted in Figures 4.7a-4.7d. As shown in the figures, LOOCSFS is consistently superior to LSBSFS. Although a gene selection algorithm employing the sequential forward selection strategy is expected to be inferior to the methods employing the sequential floating forward selection strategy, LOOCSFS also outperforms the LSBSFFS on all the four datasets. Therefore, we can conclude that our LOOC criterion is superior to the LS bound criterion and thereby results in a better gene selection algorithm.

As we mentioned, the GLGS algorithm is proposed largely based on heuristics. In particular, the usage of Eq. (4.12) is totally heuristic with no theoretical background. Hence, we also expected to verify the heuristic components of GLGS through experiments. Specifically, with the exactly same setups, we conducted gene selection solely based on the pseudo scaling factors calculated by Eq. (4.11). In other words, the term $(1 - \beta_i)\mathbf{R}(i,:)$ is removed from Eq. (4.12) and we have

$$g_{selected} = \arg \max_{g_i \in P} \{\mathbf{v}_{pseudo}\} \quad (4.15)$$

which means the genes corresponding to the l largest pseudo scaling factors were

Chapter 4: Feature Selection Algorithms Based on LS-SVM

selected. The external B.632+ error was then computed for this variant of GLGS and compared to the original GLGS algorithm. The results are plotted in Figures 4.8a-4.8d. We can observe that inclusion of the heuristic component leads to consistent improvements on three (Leukemia, Carcinoma and Glioma) datasets. The results are somewhat mixed on the Colon dataset, but we can generally say that lower generalization error can be achieved on this dataset by including the heuristics.

Figure 4.3: Comparison of seven gene selection methods on Colon dataset

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Figure 4.4: Comparison of seven gene selection methods on Leukemia dataset

Figure 4.5: Comparison of seven gene selection methods on Carcinoma dataset

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Figure 4.6: Comparison of seven gene selection methods on Glioma dataset

Figure 4.7a: Comparison of LOOCSFS, LSBSFS and LSBSFFS on Colon dataset

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Figure 4.7b: Comparison of LOOCSFS, LSBSFS and LSBSFFS on Leukemia dataset

Figure 4.7c: Comparison of LOOCSFS, LSBSFS and LSBSFFS on Carcinoma dataset

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Figure 4.7d: Comparison of LOOCSFS, LSBSFS and LSBSFFS on Glioma dataset

Figure 4.8a: Verification of heuristic components of GLGS on Colon dataset

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Figure 4.8b: Verification of heuristic components of GLGS on Leukemia dataset

Figure 4.8c: Verification of heuristic components in GLGS on Carcinoma dataset

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Figure 4.8d: Verification of heuristic components of GLGS on Glioma dataset

Figure 4.9: The computational time of seven gene selection algorithms on the Carcinoma dataset, shown vs. the number of selected genes

Figure 4.10: The computational time of seven gene selection algorithms on the Carcinoma dataset, shown vs. the size of the gene set.

On the computational complexity

From the perspective of computational complexity, the scalability of a gene selection algorithm should also be considered when assessing it. In a gene selection algorithm, the evaluation criterion is computed repeatedly to assess candidate gene subsets. Hence, the computational cost of a gene selection algorithm is determined not only by computational complexity of the evaluation criterion, but also by the number of required evaluations. Our proposed methods employ evaluation criteria that can be efficiently computed. To analyze the computational complexity issue in more details, the numbers of evaluations required by different search strategies need to be examined. If the microarray data contain d genes and l of them are to be selected, the sequential forward selection and the sequential backward elimination strategies

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

require $(2d - l + 1)/2$ and $(2d - l - 1)(d - l)/2$ evaluations respectively. The sequential floating selection (or elimination) strategy requires more evaluations than the former two strategies and GAs generally requires even more evaluations than the floating strategies. As d and l increase, the number of evaluations will increase significantly. Since microarray data usually contain thousands of genes, all the traditional strategies are relatively time consuming for the gene selection problem even if we employ an evaluation criterion that is easy to compute. In comparison, because the computational complexity of Eq. (4.10) is mainly determined by the number of patterns rather than the size of gene set or the number of genes to be selected, time requirement of the GLGS algorithm will not increase much with d and l . By performing minimization of Eq. (4.10) in the lower dimensional space, it requires much less evaluations for high dimensional data than the traditional search strategies. For example, if 50 genes are to be selected from 5000, then the LOOCSFS algorithm requires 123775 evaluations, and SVM-RFE requires solving the QP problem for 4950 times because of its specific mechanism. For the GLGS algorithm, the computational complexity is dominated by PCA and gradient descent procedure. Generally, the gradient descent procedure can converge within 300 iterations in our cases. As the computational complexity of Eq. (4.10) is approximately two times of Eq. (4.3), the time requirement of the gradient descent procedure is comparable to 600 evaluations of the LOOCSFS algorithm, and less than 600 evaluations of SVM-RFE. The computational cost of the PCA procedure is relatively difficult to estimate, but our experimental results show that it can almost be neglected when d and l are large.

Experimental studies have also been carried out to compare the computational costs of different gene selection algorithms. In Figures 4.9 and 4.10, the required

---

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

computational time are plotted with respect to the number of genes to be selected (l) and the size of the whole gene set (d), respectively. As shown in the two figures, the marginal filter method is always the most efficient one among all the algorithms. The computational costs of LSBSFS, MAHSFS, LOOCSFS and LSBSFFS all increase significantly when d or l increases. In Figure 4.9, LSBSFFS is the most time consuming since the sequential floating forward selection strategy is employed. MAHSFS also requires expensive computation. In comparison, the LSBSFS and the LOOCSFS algorithms can be implemented more efficiently. The LOOCSFS requires slightly more time than the LSBSFS. The computational time of SVM-RFE and GLGS does not change significantly with l , and SVM-RFE is more time consuming than GLGS. In Figure 4.10, computational costs of all methods except GLGS increase significantly with d , with the LSBSFFS algorithm being the most time consuming and the other four are comparable. Hence, the GLGS algorithm can better scale to microarray data with large number of genes as well as the problems that require selecting a large number of genes from the original gene set.

4.5 Further Discussions

In practice, choosing a gene selection algorithm for classification usually depends on the problem involved. According to the presented experimental results, if only one gene subset that leads to the lowest generalization error is needed, the GLGS algorithm is more appealing. However, in some cases we have to trade some accuracy for efficiency. Among the seven compared gene selection methods, the marginal filter method is the most efficient, but it leads to the highest generalization error. If $d < 1000$ and $l < 50$, the GLGS algorithm will be more time consuming than the

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

algorithms employing the sequential forward selection strategy. In this case, if one wants to obtain the solution faster and also achieve higher accuracy than a marginal filter method, LOOCSFS and SVM-RFE may be more suitable than the GLGS.

To identify important genes and study the possible interactions between them, one may need to select a number of different gene subsets that can all yield comparably high accuracy. And the important genes can be identified by counting the frequency of the genes been selected. The rationale is that the important genes should be selected irrespective of small change of the training data. To get different feature subsets, one can manipulate the original training data to generate different training set. This procedure can be carried out using the bootstrap method. By applying the same selection algorithm to these different training set, different feature subsets can be selected. Reminding that we employed 200 bootstrap samples in the experimental study. Correspondingly, we have achieved 200 feature subsets. To demonstrate usage of LOOCSFS and GLGS for identifying important genes, The 20 genes most frequently selected by LOOCSFS and GLGS on the Carcinoma dataset are listed in Tables 4.1 and 4.2 respectively. Some comments about the selected genes are worthy of mention. Genes M59465, X75042, Y10032, L08895, AB000409, L11695, X15341 and L76927 are frequently selected by both algorithms. Among them, M59465, X75042, Y10032 and L08895 are also used to construct an SVM classifier in the original work [68]. The Y10032 and M59465 are claimed as greatly downregulated in hepatocellular carcinoma with venous invasion and the levels of Y10032 transcript are altered in hepatoma cells in response to osmotic changes or cell volume changes [68].

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Gene no.	Frequency of selection	Description
X03100	198	HLA-SB alpha gene (class II antigen) extracted from Human HLA-SB(DP) alpha gene
M33600	196	Human MHC class II HLA-DR-beta-1 (HLA-DRB1) mRNA
X16663	194	Human HS1 gene for hematopoietic lineage cell specific protein
U19713	193	Human allograft-inflammatory factor-1 mRNA
L36033	193	Human pre-B cell stimulating factor homologue (SDF1b) mRNA
X00274	192	Human gene for HLA-DR alpha heavy chain a class II antigen (immune response gene) of the major histocompatibility complex (MHC)
L08895	191	Homo sapiens MADS/MEF2-family transcription factor (MEF2C) mRNA
X15341	190	Human COX VIa-L mRNA for cytochrome c oxidase liver-specific subunit VIa (EC 1.9.3.1)
M59465	190	Human tumor necrosis factor alpha inducible protein A20 mRNA
HG1872-HT1907	190	Major Histocompatibility Complex, Dg
L11695	189	Human activin receptor-like kinase (ALK-5) mRNA
Y10032	185	H.sapiens mRNA for putative serine/threonine protein kinase
M13560	184	Human Ia-associated invariant gamma-chain gene
L76927	182	Human galactokinase (GALK1) gene
X16323	181	Human mRNA for hepatocyte growth factor (HGF)
U69546	181	Human RNA binding protein Etr-3 mRNA
AB000409	180	Human mRNA for MNK1
X75042	177	H.sapiens rel proto-oncogene mRNA
HG3576-HT3779	175	Major Histocompatibility Complex, Class II Beta W52
M87503	175	Human IFN-responsive transcription factor subunit mRNA

Table 4.1: 20 most frequently selected genes of Hepatocellular Carcinoma dataset selected by LOOCFS

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Gene no.	Frequency of selection	Description
AB000409	128	Human mRNA for MNK1
L11695	109	Human activin receptor-like kinase (ALK-5) mRNA
X15341	105	Human COX VIa-L mRNA for cytochrome c oxidase liver-specific subunit VIa (EC 1.9.3.1)
U79294	105	Human clone 23748 mRNA
Y10032	103	H.sapiens mRNA for putative serine/threonine protein kinase
L76927	103	Human galactokinase (GALK1) gene
D28915	80	Human gene for hepatitis C-associated microtubular aggregate protein p44
L08895	79	Homo sapiens MADS/MEF2-family transcription factor (MEF2C) mRNA
M64925	75	Human palmitoylated erythrocyte membrane protein (MPP1) mRNA
X75042	73	H.sapiens rel proto-oncogene mRNA
X58377	72	Human mRNA for adipogenesis inhibitory factor
M59465	70	Human tumor necrosis factor alpha inducible protein A20 mRNA
X15422	68	Human mRNA for mannose-binding protein C
D78335	66	Human mRNA for 5'-terminal region of UMK
U26710	64	Human cbl-b mRNA
L36033	64	Human pre-B cell stimulating factor homologue (SDF1b) mRNA
HG4063-HT4333	61	Transcription Factor Hbf-2
D90086	60	Human pyruvate dehydrogenase (EC 1.2.4.1) beta subunit gene, exons 10-Jan
U03105	59	Human B4-2 protein mRNA
L22343	58	Human nuclear phosphoprotein mRNA

Table 4.2: 20 most frequently selected genes of Hepatocellular Carcinoma dataset selected by GLGS

Since every gene selection method may have its specific mechanism and underlying assumption, it is possible that different algorithms favor different types of genes.

Chapter 4: Feature Selection Algorithms Based on LS-SVM

Therefore, it is reasonable to not only use different training data (as in [92]), but also employ different selection algorithms to render a comprehensive exploration of the useful genes. In the context of machine learning, this approach is referred as an ensemble, which has also been used to solve gene selection problems. Since any algorithm that selects a single gene subset could be used as a component of such an ensemble system, our algorithms can also be viewed as providing new choices to build an ensemble system.

4.6 Conclusions

In this chapter, we considered modifying the LS-SVM formulation to address feature selection problem for high-dimensional small-size datasets. In particular, we present a case study on the gene selection problem for microarray data. Two gene selection algorithms, the LOOCSFS and the GLGS algorithms, are proposed based on the efficient and exact computation of the leave-one-out error of LS-SVM. The LOOCSFS is proposed by combining our new evaluation criterion with the existing sequential forward selection scheme. The GLGS algorithm is different from many traditional gene selection algorithms for that it solves the involved optimization problem in a much lower dimensional space, thereby significantly reduces the computational cost of the selection procedure, while still selects genes from the original gene set. As both LOOCSFS and GLGS algorithms are derived from the exact calculation of leave-one-out error, they are promising to select gene subsets that yield low generalization error. Experimental results show that the GLGS algorithm is more efficient than traditional algorithms when the microarray data are represented by a large number of genes. The GLGS algorithm is also capable of efficiently selecting a

Chapter 4: Feature Selection Algorithms Based on LS-SVM

---

large number of genes from the whole gene set. Finally, although the two algorithms are presented in the context of gene selection, no biological knowledge about the microarray data is incorporated in the design of them. Therefore, both LOOCSFS and GLGS algorithm can be viewed as general feature selection methods that are specifically proposed for high-dimensional small-size datasets, and they can be readily applied to other real-world applications.

Chapter 5

On Building Sparse LS-SVM Classifier

5.1 Introduction

Although LS-SVM had been shown to be a competitive tool for classification problems, it has two main drawbacks when applying to large size datasets. Firstly, LS-SVM requires the computation and storage of the full-order kernel matrix $\mathbf{K}$ of size $n \times n$ and the computational cost of training is $O((n+1)^3)$, where n is the number of training patterns. Secondly, if a nonlinear kernel function is employed (which is usually the case), we will have to evaluate a testing pattern on all the support vectors (SVs) to classify it. Since usually all the training patterns are SVs in the case of LS-SVM, the storage and time requirement for testing will be $O(nd)$ and $O(n)$ respectively, where d is the dimensionality of the pattern vectors. For these reasons, a sparse LS-SVM classifier is required when applying the LS-SVM to large-scale problems. In comparison to the original LS-SVM that employs all the training patterns as SVs, the sparse LS-SVM uses only a SV set of much smaller size. By this means, the storage requirement and computational cost of the testing phase can be significantly reduced.

In the literature, the most commonly-used approach for building a sparse LS-SVM is support vector selection. That is, the sparse LS-SVM is built using a small number of SVs that are chosen from the whole training set. There are generally two categories of

Chapter 5: On Building Sparse LS-SVM Classifier

SV selection methods, namely the forward SV selection and the backward SV selection. A forward selection method starts from an empty set and sequentially selects SVs from the training set based on some criteria [93,94]. In backward selection, one starts with the whole training set and then the SV set is chosen by means of pruning unimportant training patterns [95-99]. Both forward and backward SV selection methods require sequentially solving a set of linear systems, but at different costs. The smallest linear system involved in a backward selection method is of the size $(l+1) \times (l+1)$, where l is the number of SVs. The forward selection methods only incur linear systems whose sizes are no more than $(l+1) \times (l+1)$. Thus, forward SV selection methods can be implemented at lower storage and computational costs in comparison to the backward SV selection methods. Nevertheless, they have attracted much less attention than the backward SV selection methods in the LS-SVM literature. On comparison, forward selection methods have shown competitive performance in the literature of several other algorithms that are closely related to the LS-SVM [63,100-105]. Hence, forward SV selection methods for LS-SVM have not been explored to its full extent and deserve more attention.

The SV selection methods naturally assume that all the SVs belong to the training set. But such an assumption is not necessarily true since what we need is simply a small set of SVs. By relaxing it, we should be able to design a large family of promising approaches for building the sparse LS-SVM. However, few studies have been conducted along this direction.

Motivated by the above observations, we dedicate this chapter to designing forward-type approaches for building sparse LS-SVM classifiers. Firstly, by

---

Chapter 5: On Building Sparse LS-SVM Classifier

---

incorporating a pruning step in the forward selection procedure, we propose a floating forward SV selection (FFSVS) algorithm. In comparison to the existing methods, this algorithm can be employed to build a sparse LS-SVM using smaller number of SVs. Secondly, we propose a novel approach based on the considerations that an SV is not necessarily an existing training pattern. Instead of selecting SVs from the training set, our approach searches in the input feature space to generate “pseudo” SVs. In this way, acquiring the SVs is addressed as a continuous optimization task (generating SVs) instead of a discrete optimization task (selecting SVs). More specifically, we utilize a gradient descent algorithm to implement the general idea and propose the gradient-based pseudo SV generation algorithm (GPSVG). Employing the GPSVG algorithm, we may achieve even better performance in very sparse LS-SVM classifiers.

In Section 5.2, we briefly formulate the problem of building a sparse LS-SVM. Then we review some previous works in Section 5.3. The FFSVS and GPSVG algorithms are described in Sections 5.4 and 5.5 respectively. Two methods that can accelerate the implementation of all the SV selection/generation algorithms are mentioned in Section 5.6. Section 5.7 further reviewed some related approaches. All the involved algorithms are experimentally compared in Section 5.8. Finally, we draw conclusions in Section 5.9.

5.2 Formulation of the Sparse LS-SVM

Suppose the d -dimensional training patterns are given as $S_{tr} = \{(\mathbf{x}_i, y_i), i = 1, 2, \dots, n\}$, where $y_i \in \{-1, +1\}$. We have shown that the LS-SVM is trained by computing the

Chapter 5: On Building Sparse LS-SVM Classifier

---

following equation:

$$\begin{bmatrix} \mathbf{a} \\ b \end{bmatrix} = \left(\begin{bmatrix} \mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n & \mathbf{Y} \\ \mathbf{Y}^T & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} \mathbf{1}_n \\ 0 \end{bmatrix} \quad (5.1)$$

and the trained LS-SVM takes the form:

$$\hat{f}(\mathbf{x}) = \text{sgn}(f(\mathbf{x})) = \text{sgn}\left(\sum_{i=1}^n \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b\right) \quad (5.2)$$

From Eq. (5.1), we can observe that the training phase of LS-SVM requires the cost of $O((n+1)^3)$ time and $O(n^2)$ memory respectively. Since usually all entries of $\mathbf{a} = [\alpha_1, \alpha_2, \dots, \alpha_n]^T$ are non-zero, one needs to evaluate $k(\mathbf{x}_i, \mathbf{x})$ on all training pattern $\mathbf{x}_i$'s for classifying the testing pattern $\mathbf{x}$, which leads to $O(n)$ cost. Therefore, it is impractical to directly apply the LS-SVM to large-scale training or testing datasets. Instead, one can build a sparse LS-SVM.

Intuitively speaking, the “sparse” means that many entries of $\mathbf{a}$ are exactly zero, so that the corresponding training patterns will not be used as SVs and can be neglected immediately after the training phase. Assume that a sparse LS-SVM (denoted as $\hat{f}_{\text{sparse}}$) is built with l ($l < n$) SVs $Z = \{\mathbf{z}_1, \dots, \mathbf{z}_l\}$, it takes the similar form to the original LS-SVM:

$$\hat{f}_{\text{sparse}}(\mathbf{x}) = \text{sgn}(f_{\text{sparse}}(\mathbf{x})) = \text{sgn}(\mathbf{w}^T \mathbf{x} + b) = \text{sgn}\left(\sum_{i=1}^l \beta_i k(\mathbf{z}_i, \mathbf{x}) + b\right) \quad (5.3)$$

where the β_i 's are determined by training. Given Eq. (5.3), the sparse LS-SVM can be trained by solving the same type optimization problem as the original LS-SVM:

$$\begin{aligned} \min_{\mathbf{w}, b, \mathbf{e}} J_{\text{sparse}}(\mathbf{w}, b, \mathbf{e}) &= \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{\gamma}{2} \sum_{i=1}^n e_i^2 \\ \text{subject to} \quad &y_i (\mathbf{w}^T \phi(\mathbf{x}_i) + b) = 1 - e_i \quad \text{for } i = 1, \dots, n \end{aligned} \quad (5.4)$$

Chapter 5: On Building Sparse LS-SVM Classifier

where $\phi(\mathbf{x})$ denotes $\mathbf{x}$'s mapping in the kernel space. According to Eq. (5.3), we

have $\mathbf{w} = \sum_{i=1}^l \beta_i \phi(\mathbf{z}_i)$ in the sparse LS-SVM. Thus, the cost function in Eq. (5.4) can

be written as:

$$\begin{aligned} J_{\text{Sparse}} &= \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{\gamma}{2} \sum_{i=1}^n e_i^2 \\ &= \frac{1}{2} \sum_{j=1}^l \sum_{k=1}^l \beta_j \beta_k k(\mathbf{z}_j, \mathbf{z}_k) + \frac{\gamma}{2} \sum_{i=1}^n [1 - y_i \sum_{j=1}^l \beta_j k(\mathbf{z}_j, \mathbf{x}_i) - y_i b]^2 \end{aligned} \quad (5.5)$$

Setting the partial derivatives with respect to β_j to zero, we get

$$\frac{\partial J_{\text{Sparse}}}{\partial \beta_j} = \sum_{k=1}^l \beta_k k(\mathbf{z}_k, \mathbf{z}_j) + \gamma \sum_{i=1}^n [k(\mathbf{z}_j, \mathbf{x}_i) \sum_{j=1}^l \beta_j k(\mathbf{z}_j, \mathbf{x}_i) + (b - y_i) k(\mathbf{z}_j, \mathbf{x}_i)] = 0 \quad (5.6)$$

which gives

$$\sum_{k=1}^l \beta_k [k(\mathbf{z}_k, \mathbf{z}_j) + \gamma \sum_{i=1}^n k(\mathbf{z}_j, \mathbf{x}_i) k(\mathbf{z}_k, \mathbf{x}_i)] + \gamma b \sum_{i=1}^n k(\mathbf{z}_j, \mathbf{x}_i) = \gamma \sum_{i=1}^n y_i k(\mathbf{z}_j, \mathbf{x}_i) \quad (5.7)$$

Similarly, from

$$\frac{\partial J_{\text{Sparse}}}{\partial b} = -\gamma \sum_{i=1}^n y_i [1 - y_i \sum_{j \in L} \beta_j k(\mathbf{x}_j, \mathbf{x}_i) - y_i b] = 0 \quad (5.8)$$

we get

$$\sum_{i=1}^n \sum_{j=1}^l \beta_j k(\mathbf{z}_j, \mathbf{x}_i) + nb = \sum_{i=1}^n y_i \quad (5.9)$$

According to Eqs. (5.7) and (5.9), solution of the sparse LS-SVM can be calculated by

solving the following $(l+1)$ -by- $(l+1)$ linear system

$$\begin{bmatrix} \mathbf{P} & \Phi \\ \Phi^T & n \end{bmatrix} \begin{bmatrix} \beta \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \sum_{i=1}^n y_i \end{bmatrix} \quad (5.10)$$

where $\mathbf{P}$ is an $l \times l$ matrix and $P_{jk} = \sum_{i=1}^n k(\mathbf{z}_j, \mathbf{x}_i) k(\mathbf{z}_k, \mathbf{x}_i) + k(\mathbf{z}_j, \mathbf{z}_k) / \gamma$. Φ and

$\mathbf{c}$ are column vectors. Their entries can be calculated by $\Phi_j = \sum_{i=1}^n k(\mathbf{z}_j, \mathbf{x}_i)$ and

Chapter 5: On Building Sparse LS-SVM Classifier

$c_j = \sum_{i=1}^n y_i k(\mathbf{z}_j, \mathbf{x}_i)$, respectively. Let $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_l]$ be the $d \times l$ matrix whose j th column is $\mathbf{z}_j$, we define $k(\mathbf{Z}, \mathbf{x}_i)$ as the $l \times 1$ vector whose j th component is computed by $k(\mathbf{z}_j, \mathbf{x}_i)$. Accordingly, $k(\mathbf{Z}, \mathbf{X})$ is defined as the $l \times n$ matrix

whose entries can be computed by $k(\mathbf{z}_j, \mathbf{x}_i)$. Let $\bar{\mathbf{P}} = \begin{bmatrix} \mathbf{P} & \Phi \\ \Phi & n \end{bmatrix}$ and $\bar{\mathbf{c}} = \begin{bmatrix} \mathbf{c} \\ \sum_{i=1}^n y_i \end{bmatrix}$.

From Eq. (5.10), we have $\begin{bmatrix} \beta \\ b \end{bmatrix} = \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}}$ and $e_i = 1 - y_i \left(\bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \begin{bmatrix} k(\mathbf{Z}, \mathbf{x}_i) \\ 1 \end{bmatrix} \right)$. Further,

we can get

$$\begin{aligned} \sum_{i=1}^n e_i^2 &= n - 2 \sum_{i=1}^n \left[y_i \left(\bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \begin{bmatrix} k(\mathbf{Z}, \mathbf{x}_i) \\ 1 \end{bmatrix} \right) \right] + \sum_{i=1}^n \left(\bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \begin{bmatrix} k(\mathbf{Z}, \mathbf{x}_i) \\ 1 \end{bmatrix} \right)^2 \\ &= n - 2 \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \begin{bmatrix} k(\mathbf{Z}, \mathbf{X}) \\ \mathbf{1} \end{bmatrix} \mathbf{Y} + \left(\bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \begin{bmatrix} k(\mathbf{Z}, \mathbf{X}) \\ \mathbf{1} \end{bmatrix} \right) \left(\bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \begin{bmatrix} k(\mathbf{Z}, \mathbf{X}) \\ \mathbf{1} \end{bmatrix} \right)^T \\ &= n - 2 \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} + \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \begin{bmatrix} k(\mathbf{Z}, \mathbf{X}) k(\mathbf{Z}, \mathbf{X})^T & \Phi \\ \Phi & n \end{bmatrix} \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} \end{aligned} \quad (5.11)$$

Then

$$\begin{aligned} J_S &= \min(J_{\text{Sparse}}) \\ &= \frac{\gamma}{2} (\frac{1}{\gamma} \mathbf{w}^T \mathbf{w} + \mathbf{e}^T \mathbf{e}) \\ &= \frac{\gamma}{2} \left\{ n - 2 \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} + \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \begin{bmatrix} k(\mathbf{Z}, \mathbf{X}) k(\mathbf{Z}, \mathbf{X})^T & \Phi \\ \Phi & n \end{bmatrix} \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} + \frac{1}{\gamma} k(\mathbf{Z}, \mathbf{Z}) \quad \mathbf{0} \right. \\ &\quad \left. \mathbf{0} \quad 0 \right\} \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} \\ &= \frac{\gamma}{2} \left\{ n - 2 \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} + \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \begin{bmatrix} k(\mathbf{Z}, \mathbf{X}) k(\mathbf{Z}, \mathbf{X})^T + \frac{1}{\gamma} k(\mathbf{Z}, \mathbf{Z}) & \Phi \\ \Phi & n \end{bmatrix} \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} \right\} \\ &= \frac{\gamma}{2} (n - 2 \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} + \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \bar{\mathbf{P}} \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}}) \\ &= \frac{\gamma}{2} (n - \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}}) \end{aligned} \quad (5.12)$$

Chapter 5: On Building Sparse LS-SVM Classifier

Consequently, an important property of sparse LS-SVM can be summarized as below:

Lemma 5.1: *For two SV sets Z_1 and Z_2 , let $J_s(Z_1)$ and $J_s(Z_2)$ be the corresponding J_s 's. If $Z_1 \subseteq Z_2$, then $J_s(Z_1) \geq J_s(Z_2)$. (See Appendix C for proof.)*

According to Lemma 5.1, we can achieve smaller value of J_s , which is more desirable, by using more SVs to build the sparse LS-SVM. But since the training and testing phase of the sparse LS-SVM lead to $O((l+1)^3)$ and $O(l)$ costs respectively, we prefer small number of SVs to alleviate both the computational and storage costs. Therefore, when building a sparse LS-SVM, our objective is to find a good trade-off between the classification performance and the sparseness of LS-SVM. In this way, the costs can be considerably reduced while the classification accuracy does not deteriorate.

5.3 Existing Support Vector Selection Methods

In LS-SVM literature, the problem of building a sparse LS-SVM is usually addressed by SV selection algorithms. An SV selection algorithm selects patterns from the training set and the chosen patterns are then used as SVs of the sparse LS-SVM. An SV selection algorithm typically consists of two crucial components, namely an evaluation criterion and a search strategy. Employing a search strategy, the algorithm iteratively selects l SVs from a candidate pool based on the evaluation criterion. In the literature, various evaluation criteria have been designed based on specific considerations of the LS-SVM formulation. Due to the fact that choosing l SVs out of n training patterns involves a prohibitive combinatorial search over C_n^l space

Chapter 5: On Building Sparse LS-SVM Classifier

[101,106], various search strategies have also been investigated. When employing selection algorithms to build a sparse LS-SVM, we actually use a subset of the SVs of the original LS-SVM. According to Lemma 5.1, the optimized J_s of the sparse LS-SVM cannot be smaller than the optimized J of the original LS-SVM. Therefore, when an SV selection algorithm is used to acquire the SVs, the resultant sparse LS-SVM can always be viewed as an approximation of the original LS-SVM. As we mentioned, although SV selection algorithms can be categorized as forward or backward, only forward SV selection algorithms are considered in this chapter. For a comprehensive study on backward selection methods, readers are referred to [99].

5.3.1 Greedy Support Vector Selection Algorithms

Greedy Selection Strategy

Among all the SV selection algorithms, greedy selection may be the most commonly used search strategy. A greedy SV selection (GSVS) algorithm iteratively selects the SVs by repeating the following steps: At each iteration, all the previously unselected training patterns are put into the candidate pool. One assesses these candidate patterns using an evaluation criterion, The optimal pattern is chosen as the new SV and is included into the SV set. Then all the related quantities, such as J_s and β_i 's in Eq. (5.5), are updated. In the feature selection literature, this greedy selection strategy is usually referred to as the sequential forward selection strategy [75,77,107].

Evaluation Criteria

Assume that we have collected $\bar{l}$ SVs using a GSVS algorithm and the $(\bar{l} + 1)$ th one is to be chosen from the candidate pool, which contains $n - \bar{l}$ previously

---

Chapter 5: On Building Sparse LS-SVM Classifier

---

unselected training patterns. Cawley and Talbot suggested choosing the candidate pattern whose inclusion yields the largest reduction of J_S [93]. More specifically, given the current SV set and the corresponding value of J_S (denoted as J_{Sold}), each candidate pattern is temporarily included into the current SV set and the corresponding J_S (denoted as J_{Snew}) is computed using equations (5.5) and (5.10).

The pattern that leads to the largest value of $lossJ = J_{Sold} - J_{Snew}$ is selected.

Based on the work presented in [108], a criterion that is similar to the $lossJ$ criterion was suggested in [99]. It assesses each candidate pattern by its contribution to the reduction of the regression error $\sum_{i=1}^n e_i^2$. Since this criterion employs QR-factorization

to simplify the calculation of candidate patterns' contribution, it is denoted as the dQR criterion. Computations of both $lossJ$ and dQR criteria require the class label of training patterns. Hence, they are *supervised* in nature. Although the two criteria have slightly different formulation, no significant difference between them has been reported in practical use.

In addition to the *supervised* criteria, an *unsupervised* criterion, denoted as the $lossSpan$ criterion, has also been proposed [102]. This criterion aims to select the SV set by which all the unselected patterns can be best approximated. The rationale is: According to Eq. (5.2), the weighting vector $\mathbf{w}$ of the original LS-SVM is a linear combination of all the training patterns. In the kernel space, if the SV set can express all the remaining training patterns as its linear combination, we should be able to exactly reconstruct the original LS-SVM classifier with this SV set. Mathematically, if

Chapter 5: On Building Sparse LS-SVM Classifier

---

a pattern vector $\mathbf{x}_i$ can be expressed as linear combination of the SV set

$Z = \{\mathbf{z}_1, \dots, \mathbf{z}_l\}$, we have

$$k(\mathbf{x}_i, \mathbf{x}_i) = k(\mathbf{Z}, \mathbf{x}_i)^T k(\mathbf{Z}, \mathbf{Z})^{-1} k(\mathbf{Z}, \mathbf{x}_i) \quad (5.13)$$

where $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_l]$ is a $d \times l$ matrix whose j th column is $\mathbf{z}_j$. $k(\mathbf{Z}, \mathbf{x}_i)$ is an

$l \times 1$ vector whose j th component is computed by $k(\mathbf{z}_j, \mathbf{x}_i)$ and $k(\mathbf{Z}, \mathbf{Z})$ is an $l \times l$

matrix whose component is $k(\mathbf{z}_j, \mathbf{z}_i)$. Correspondingly, if all the training patterns can

be expressed as a linear combination of $\mathbf{z}_i$'s, we have

$$Span = \frac{1}{n} \sum_{i=1}^n \left(1 - \frac{k(\mathbf{Z}, \mathbf{x}_i)^T k(\mathbf{Z}, \mathbf{Z})^{-1} k(\mathbf{Z}, \mathbf{x}_i)}{k(\mathbf{x}_i, \mathbf{x}_i)} \right) = 0 \quad (5.14)$$

Both of Eqs. (5.13) and (5.14) were presented and verified in [102]. When including a new SV, let us denote the corresponding reduction of $Span$ by $lossSpan$. Based on equations (5.13) and (5.14), it was suggested that the candidate pattern leading to the largest $lossSpan$ should be selected at each iteration of the GSVS algorithm.

Recently, Eq. (5.13) has also been suggested as a “filtering” criterion in the SV selection algorithms. At each iteration, the candidate patterns that can be expressed as linear combinations of the current SVs are omitted from the candidate pool. Once a candidate is omitted, it will not be included in the candidate pool again in later iterations. After that, any above-mentioned criterion, such as the $lossJ$ criterion, the dQR criterion or the $lossSpan$ criterion is applied to the remaining candidates. By this means, fewer candidate patterns are involved in the subsequent iterations, thus the whole selection procedure is accelerated.

The greedy support vector selection (GSVS) algorithm

Input: The whole training set $S_{tr} = \{(\mathbf{x}_i, y_i), i = 1, 2, \dots, n\}$

Initialization: Set Z as empty, the candidate pool $\Psi = S_{tr}$ and $s_c = +\infty$

Do While $s_c > c_1$, where $c_1 \in (0, 1)$ is the pre-defined stop condition

1. $\forall \mathbf{x}_i \in \Psi$, if $k(\mathbf{x}_i, \mathbf{x}_i) = k(\mathbf{Z}, \mathbf{x}_i)^T k(\mathbf{Z}, \mathbf{Z})^{-1} k(\mathbf{Z}, \mathbf{x}_i)$, delete it from Ψ .
2. Evaluate all $\mathbf{x}_i \in \Psi$ using the $lossJ$ criterion, keep the optimal one as $\mathbf{x}_{opt}$ and the corresponding value of $lossJ$ as $lossJ_{opt}$.
3. If $lossJ_{opt} > c_1$, set $Z = \{Z, \mathbf{x}_{opt}\}$ and $s_c = lossJ_{opt}$, remove $\mathbf{x}_{opt}$ from Ψ and update J_s with the new Z .

Otherwise, terminate the algorithm.

End While

Output: The set of selected support vectors $Z = \{\mathbf{z}_1, \dots, \mathbf{z}_l\}$

Note: the omitting stage is negligible when Z is empty.

Figure 5.1: Pseudo-code of the greedy support vector selection algorithm

The Stop Criterion

The last important issue that is worthy of mention is the stop condition of a GSVS algorithm. Generally, one can terminate the selection when a good approximation of the original LS-SVM has been achieved. More specifically, it can be observed that

$0 < J_s \leq \frac{\gamma n}{2}$ and $0 < Span \leq 1$, where γ is the regularization parameter. Hence, we

can stop the GSVS algorithm when $\frac{lossJ}{\gamma n} < t$ (if $lossJ$ criterion is used) or

$lossSpan < t$ (if $lossSpan$ criterion is used), where $t \in (0, 1)$ is a pre-defined

Chapter 5: On Building Sparse LS-SVM Classifier

---

constant and the number of selected SVs increases with the value of t^1 . In Figure 5.1, we present the pseudo code of the GSVS algorithms. The $lossJ$ criterion is used as the example and the code can be easily extended to the cases of dQR and $lossSpan$ criteria.

5.3.2 Sub-greedy Support Vector Selection Algorithms

When using the greedy algorithms for SV selection, one needs to evaluate $n - \bar{l}$ candidate patterns at the $\bar{l}$ th iteration. Thus, $\frac{(2n - l + 1)l}{2}$ evaluations are required to choose l SVs out of n training patterns. If n is large, a greedy SV selection algorithm may not be computationally feasible. Alternatively, a sub-greedy search strategy can be employed [94,101]. In [18,101], Schölkopf and Smola proved that:

Lemma 5.2: *A random subset of 59 candidates, with a probability of 95%, contains a candidate with an error reduction within the lowest 5% of the entire pool of candidates.*

According to Lemma 5.2, we are able to select at least a sub-optimal candidate by only evaluating a small number L (say, 59) of the previously unselected patterns. That is, at each iteration, only L previously unselected training patterns are randomly chosen and put into the candidate pool, and the SV is selected from the candidate pool in the same way as the greedy SV algorithms. With this minor modification, we summarize the sub-greedy SV selection (SGSVS) algorithms in Figure 5.2.

---

¹ One can also stop the selection when a pre-defined number of SVs have been chosen. But since the minimum number of SVs may vary over different problems, the former stop condition is more commonly used.

The sub-greedy support vector selection (SGSVS) algorithm**Input:** The whole training set $S_{tr} = \{(\mathbf{x}_i, y_i), i = 1, 2, \dots, n\}$ **Initialization:** Set Z as empty, the candidate pool $\Psi = S_{tr}$ and $s_c = +\infty$ **Do While** $s_c > c_1$, where $c_1 \in (0, 1)$ is the pre-defined stop condition

1. $\forall \mathbf{x}_i \in \Psi$, if $k(\mathbf{x}_i, \mathbf{x}_i) = k(\mathbf{Z}, \mathbf{x}_i)^T k(\mathbf{Z}, \mathbf{Z})^{-1} k(\mathbf{Z}, \mathbf{x}_i)$, delete it from Ψ
2. Randomly choose L ($L \geq 59$) patterns from Ψ , put them into the candidate pool Ψ_{sub} .
3. Evaluate all $\mathbf{x}_i \in \Psi_{sub}$ using $lossJ$ criterion, keep the optimal one as $\mathbf{x}_{opt}$ and the corresponding $lossJ$ as $lossJ_{opt}$.
4. If $lossJ_{opt} > c_1$, set $Z = \{Z, \mathbf{x}_{opt}\}$ and $s_c = lossJ_{opt}$, remove $\mathbf{x}_{opt}$ from Ψ and update J_s with the new Z .

Otherwise, terminate the algorithm.

End While**Output:** The set of selected support vectors $Z = \{\mathbf{z}_1, \dots, \mathbf{z}_l\}$

Figure 5.2: Pseudo-code of the sub-greedy support vector selection algorithms

In an SGSVS algorithm, at most Ll evaluations are required and thereby the selection procedure can be implemented more efficiently. However, since the SGSVS algorithms may select a sub-optimal candidate pattern at each iteration, it is inferior to the greedy selection algorithms from the classification viewpoint. In other words, to achieve comparable classification performance, SGSVS algorithms usually chooses more SVs than the GSVS methods.

5.3.3 Non-greedy Support Vector Selection Algorithm

In addition to the greedy and sub-greedy selection methods, Sun recently proposed a fast SV selection method that is non-greedy in nature² [109]. This method does not greedily select the candidate which is *optimal* according to some criteria. Instead, a candidate pattern $\mathbf{x}_i$ will be selected once it satisfies two conditions:

1. $1 - \frac{k(\mathbf{Z}, \mathbf{x}_i)^T k(\mathbf{Z}, \mathbf{Z})^{-1} k(\mathbf{Z}, \mathbf{x}_i)}{k(\mathbf{x}_i, \mathbf{x}_i)} > c_1$, where c_1 is a predefined threshold.

Satisfying this condition indicates that $\mathbf{x}_i$ cannot be well approximated by linear combinations of the current SVs.

2. Inclusion of $\mathbf{x}_i$ into the SV set leads to a $lossJ > c_2$, where c_2 is a pre-defined threshold. According to this condition, only the candidate patterns that lead to significant reductions on J_C will be selected.

Otherwise, $\mathbf{x}_i$ is discarded. The whole procedure of this non-greedy support vector selection (NGSVS) algorithm is described in more details in Figure 5.3.

In the NGSVS algorithm, each candidate pattern is only evaluated once throughout the whole selection procedure. Hence, in comparison to the GSVS and SGSVS algorithms, the NGSVS algorithm incurs only n evaluations and thereby considerably accelerates the selection procedure. However, the NGSVS algorithm inevitably has drawbacks. And the most important one is its high sensitivity to the order in which we evaluate the candidates. The NGSVS algorithm will definitely select the first candidate pattern. Hence, if we evaluate the candidate patterns in different orders, the

---

² This method was not originally proposed for the LS-SVM, we made some minor modifications to adapt it to the LS-SVM formulation.

Chapter 5: On Building Sparse LS-SVM Classifier

---

first selected SV will be different. And thereby we may finish with different SV sets. Even if these SV sets lead to comparable classification performance of the sparse LS-SVM, their sizes may be considerably different. As a result, the NGSVS algorithm is quite likely to choose a larger set of SVs than the GSVS and SGSVS algorithms.

The non-greedy support vector selection (NGSVS) algorithm

Input: The whole training set $S_{tr} = \{(\mathbf{x}_i, y_i), i = 1, 2, \dots, n\}$

Initialization: Set Z as empty, the candidate pool $\Psi = S_{tr}$ and $s_C = +\infty$

Do For $i = 1$ to n

If $1 - \frac{k(\mathbf{Z}, \mathbf{x}_i)^T k(\mathbf{Z}, \mathbf{Z})^{-1} k(\mathbf{Z}, \mathbf{x}_i)}{k(\mathbf{x}_i, \mathbf{x}_i)} > c_1$ and $lossJ > c_2$, set $Z = \{Z, \mathbf{x}_i\}$,

remove $\mathbf{x}_i$ from Ψ and update J_S with the new Z .

Otherwise remove $\mathbf{x}_i$ from Ψ .

End For

Output: The set of selected support vectors $Z = \{\mathbf{z}_1, \dots, \mathbf{z}_l\}$

Figure 5.3: Pseudo-code of the non-greedy support vector selection algorithm

5.4 Floating Support Vector Selection Algorithm

As we described above, although the NGSVS and SGSVS algorithms provide computational advantage, the GSVS algorithms may result in an SV set of smaller size. Nevertheless, the greedy forward selection (or the sequential forward selection) is after all a sub-optimal search strategy, and there is no guarantee that it can choose the smallest SV set leading to sufficiently good classification performance. In a GSVS algorithm, an SV will not be further re-evaluated once it is selected. On the other hand,

---

Chapter 5: On Building Sparse LS-SVM Classifier

---

the sparse LS-SVM is re-trained after any change of the SV set and thereby an SV's contribution to the performance of the sparse LS-SVM varies during the selection procedure. Although the GSVS algorithms always select the candidate pattern which contributes the most, an SV may become unimportant after inclusion of some new SVs. If such a scenario happens, the GSVS algorithm will definitely result in a redundant SV set. To tackle this drawback, we propose here a floating forward support vector selection (FFSVS) algorithm.

In the FFSVS algorithm, we employ the $lossJ$ criterion and the sub-greedy strategy to forwardly choose the SVs and perform a pruning step after the inclusion of each new SV. That is, we re-evaluate the previously chosen SVs and those SVs that only provide limited contributions to the sparse LS-SVM are removed. More specifically, given the SV set Z at hand, the SVs whose deletions lead to the smallest increase on J_s are sequentially removed, and the pruning phase is terminated when the smallest increase is larger than a pre-defined threshold. In Figure 5.4, we describe the FFSVS algorithm in more details.

The floating forward support vector selection (FFSVS) algorithm

---

Input: The whole training set $S_{tr} = \{(\mathbf{x}_i, y_i), i = 1, 2, \dots, n\}$

Initialization: Set Z as empty, the candidate pool $\Psi = S_{tr}$, $s_c = +\infty$ and

$$s_{cp} = 0$$

Do While $s_c > c_1$, where $c_1 \in (0, 1)$ is a pre-defined constant (**While1**)

A. Selection phase

Chapter 5: On Building Sparse LS-SVM Classifier

1. $\forall \mathbf{x}_i \in \Psi$, if $k(\mathbf{x}_i, \mathbf{x}_i) = k(\mathbf{Z}, \mathbf{x}_i)^T k(\mathbf{Z}, \mathbf{Z})^{-1} k(\mathbf{Z}, \mathbf{x}_i)$, delete it from Ψ .
2. Randomly choose L ($L \geq 59$) samples from Ψ , put them into the candidate pool Ψ_{sub} .
3. Evaluate all $\mathbf{x}_i \in \Psi_{sub}$ using the $lossJ$ criterion, keep the optimal one as $\mathbf{x}_{opt}$ and the corresponding $lossJ$ as $lossJ_{opt}$.
4. If $lossJ_{opt} > c_1$, set $Z = \{Z, \mathbf{x}_{opt}\}$ and $s_c = lossJ_{opt}$, remove $\mathbf{x}_{opt}$ from Ψ and update J_s with the new Z .

Otherwise, terminate the algorithm.

B. Pruning phase

While $s_{cp} < c_1$ (**While2**)

1. For each $\mathbf{z}_i \in Z$, temporarily remove it, compute Eq. (5.10) with the remaining support vectors and calculate the corresponding $J_s(\mathbf{z}_i)$ using Eq. (5.12). The support vector which leads to the smallest increase on J_s is: $\mathbf{z}_{prune} = \underset{\mathbf{z}_i \in Z}{\operatorname{argmin}} \{J_s(\mathbf{z}_i) - J_s\}$.

2. If $\frac{J_s(\mathbf{z}_{prune}) - J_s}{\gamma n} \geq c_1$, terminate the pruning phase and set

$$s_{cp} = \frac{J_s(\mathbf{z}_{prune}) - J_s}{\gamma n}.$$

Otherwise, remove $\mathbf{z}_{prune}$ from Z , include it into Ψ .

End While2

End While1

Output: The set of selected support vectors $Z = \{\mathbf{z}_1, \dots, \mathbf{z}_l\}$

Figure 5.4: Pseudo-code of the floating forward support vector selection algorithm

In the FFSVS algorithm, the same threshold constant c_1 is used as the stop criterion

Chapter 5: On Building Sparse LS-SVM Classifier

---

for both pruning and selection procedures. In practice, one can also define different thresholds for these two procedures. Since the FFSVS algorithm repeatedly prunes redundant SVs, it is promising to select smaller number of SVs than the SGSVS algorithm. In fact, it may even be expected to yield smaller number of SVs than the GSVS algorithm. But at the same time, the FFSVS algorithm is computationally more expensive than the SGSVS algorithm³. Finally, it is noteworthy that the general idea of floating forward selection is originally proposed for feature selection problems and then studied extensively in the same context [107,110,111]. But to our knowledge, they have not yet been seriously adapted to the SV selection problems to build sparse kernel classifiers (such as LS-SVM, KFDA, PSVM, etc.).

5.5 Sparse LS-SVM Using Pseudo Support Vectors

Generally speaking, building sparse LS-SVM can be regarded as searching for an optimal SV set in the input feature space with respect to Eq. (5.5). To address this problem, we have introduced various SV selection algorithms, which assume that all the SVs belong to the training set. Although this assumption makes the problem easier to deal with (as we only need to search in a discrete space instead of a continuous space), it is not necessary. By extending our search to the whole continuous input feature space, we may probably find some vectors that are more suitable than the training patterns for building the sparse LS-SVM. Therefore, we propose to search the input feature space for a set of pseudo SVs, which are not given in the original training set but can still be used to build the sparse LS-SVM. More specifically, we consider utilizing gradient descent algorithms to implement such a general idea, and

---

³ Accordingly, if we replace the sub-greedy selection strategy with the greedy selection strategy, the resultant algorithm will be more costly than the GSVS algorithm.

Chapter 5: On Building Sparse LS-SVM Classifier

the resultant method is named the gradient-based support vector generation (GPSVG) algorithm.

The general framework of GPSVG algorithm is straightforward. Starting from an empty set, we sequentially add new SVs to the SV set. Instead of being selected from a candidate pool, the SVs are *generated* based on the $lossJ$ criterion using a gradient descent algorithm. By generating a new SV instead of selecting one from the training set, we expect to achieve a large reduction of the J_s at each iteration. To exploit the gradient descent algorithms to generate a new SV (denoted as $\mathbf{z}_{new}$), we need to calculate the derivative of J_s with respect to a variable t as:

$$\begin{aligned}
\frac{\partial J_s}{\partial t} &= \frac{\gamma}{2} \left(\bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \frac{\partial \bar{\mathbf{P}}}{\partial t} \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} - 2 \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \frac{\partial \bar{\mathbf{c}}}{\partial t} \right) \\
&= \frac{\gamma}{2} \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \left[\begin{array}{c} \frac{\partial [k(\mathbf{Z}, \mathbf{X})k(\mathbf{Z}, \mathbf{X})^T]}{\partial t} + \frac{1}{\gamma} \frac{\partial k(\mathbf{Z}, \mathbf{Z})}{\partial t} \sum_{k=1}^n \frac{\partial k(\mathbf{Z}, \mathbf{x}_k)}{\partial t} \\ \sum_{k=1}^n \frac{\partial k(\mathbf{Z}, \mathbf{x}_k)^T}{\partial t} \quad 0 \end{array} \right] \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} \\
&\quad - \gamma \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \left[\begin{array}{c} \sum_{k=1}^n y_k \frac{\partial k(\mathbf{Z}, \mathbf{x}_k)}{\partial t} \\ 0 \end{array} \right] \\
&= \frac{\gamma}{2} \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \left[\begin{array}{c} 2 \frac{\partial k(\mathbf{Z}, \mathbf{X})}{\partial t} k(\mathbf{Z}, \mathbf{X})^T + \frac{1}{\gamma} \frac{\partial k(\mathbf{Z}, \mathbf{Z})}{\partial t} \sum_{k=1}^n \frac{\partial k(\mathbf{Z}, \mathbf{x}_k)}{\partial t} \\ \sum_{k=1}^n \frac{\partial k(\mathbf{Z}, \mathbf{x}_k)^T}{\partial t} \quad 0 \end{array} \right] \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} \\
&\quad - \gamma \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \left[\begin{array}{c} \sum_{k=1}^n y_k \frac{\partial k(\mathbf{Z}, \mathbf{x}_k)}{\partial t} \\ 0 \end{array} \right] \tag{5.15}
\end{aligned}$$

When employing Eq. (5.15) to generate the first SV $\mathbf{z}_1 = [z_{11}, z_{12}, \dots, z_{1d}]^T$, for each

entry z_{1q} we have

Chapter 5: On Building Sparse LS-SVM Classifier

$$\frac{\partial J_S}{\partial z_{1q}} = \frac{\gamma}{2} \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \begin{bmatrix} 2 \frac{\partial k(\mathbf{z}_1, \mathbf{X})}{\partial z_{1q}} k(\mathbf{z}_1, \mathbf{X})^T & \sum_{k=1}^n \frac{\partial k(\mathbf{z}_1, \mathbf{x}_k)}{\partial z_{1q}} \\ \sum_{k=1}^n \frac{\partial k(\mathbf{z}_1, \mathbf{x}_k)^T}{\partial z_{1q}} & 0 \end{bmatrix} \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}} - \gamma \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \begin{bmatrix} \sum_{k=1}^n y_k \frac{\partial k(\mathbf{z}_1, \mathbf{x}_k)}{\partial z_{1q}} \\ 0 \end{bmatrix} \quad (5.16)$$

Assume that we have already collected $\bar{l}$ SVs, denoted by $\mathbf{Z}_{old} = [\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{\bar{l}}]$ and is to generate a new SV $\mathbf{z}_{new} = [z_{new1}, z_{new2}, \dots, z_{newd}]^T$. An $(\bar{l} + 1) \times n$ matrix $k(\mathbf{Z}, \mathbf{X})$

can be written as $k(\mathbf{Z}, \mathbf{X}) = \begin{bmatrix} k(\mathbf{Z}_{old}, \mathbf{X}) \\ k(\mathbf{z}_{new}, \mathbf{X}) \end{bmatrix}$ and

$$\frac{\partial k(\mathbf{Z}, \mathbf{X})}{\partial z_{newq}} = \begin{bmatrix} \frac{\partial k(\mathbf{Z}_{old}, \mathbf{X})}{\partial z_{newq}} \\ \frac{\partial k(\mathbf{z}_{new}, \mathbf{X})}{\partial z_{newq}} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \frac{\partial k(\mathbf{z}_{new}, \mathbf{X})}{\partial z_{newq}} \end{bmatrix} \quad (5.17)$$

$$\frac{\partial k(\mathbf{Z}, \mathbf{Z})}{\partial z_{newq}} = \begin{bmatrix} \mathbf{0} & \frac{\partial k(\mathbf{Z}_{old}, \mathbf{z}_{new})}{\partial z_{newq}} \\ \frac{\partial k(\mathbf{z}_{new}, \mathbf{Z}_{old})}{\partial z_{newq}} & 0 \end{bmatrix} \quad (5.18)$$

where z_{newq} is the q th entry of SV $\mathbf{z}_{new}$. Substituting Eqs. (5.17) and (5.18) into Eq. (5.15), $\partial J_S / \partial z_{newq}$ can be computed.

Since the gradient descent algorithms are sub-optimal optimization approaches, one cannot expect them to always find the truly optimal SV. Fortunately, given a carefully determined starting point, a gradient descent algorithm is usually able to achieve a sufficiently good solution for our problems. Here, we heuristically suggest two choices for deciding the starting point. To generate the first SV, we start from the mean vector of all the training patterns, because the mean vector is a statistical

---

Chapter 5: On Building Sparse LS-SVM Classifier

---

representation of the whole training set. When generating the other SVs, we just randomly choose a training pattern from the candidate pool.

We have mentioned in Section 5.4 that contributions of the previously selected SVs vary over time. This scenario is also true when pseudo SVs are generated. Thus, a pruning step will be beneficial to our algorithm. Due to computational consideration, we do not conduct pruning after each new SV's generation. Instead, all the pseudo SVs are generated first, and then those unimportant ones are pruned. The same pruning approach employed in the FFSVS algorithm is used. The detailed GPSVG algorithm is described in Figure 5.5.

The gradient-based pseudo support vector generation (GPSVG) algorithm

Input: The whole training set $S_{tr} = \{(\mathbf{x}_i, y_i), i = 1, 2, \dots, n\}$

Initialization: Set the candidate pool $\Psi = S_{tr}$, $s_c = +\infty$ and $s_{ep} = 0$

Do

A. Generation phase

1. Set $\mathbf{x}_{start} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$ as the starting point, utilize Eq. (5.16) and a

gradient descent algorithm to find the first pseudo support vector $\mathbf{z}_1$

that minimizes J_s , set $Z = \mathbf{z}_1$.

2. **While** $s_c > c_1$, where $c_1 \in (0, 1)$ is the pre-defined stop condition

1). Randomly choose a starting point $\mathbf{x}_{start}$ from Ψ , utilize Eqs.

(5.15), (5.17), (5.18) and a gradient descent algorithm to generate a

new pseudo support vector $\mathbf{z}_{new}$.

Chapter 5: On Building Sparse LS-SVM Classifier

If $\mathbf{z}_{new}$ is linearly dependent on Z in the kernel space, terminate the whole algorithm.

- 2). Calculate the $lossJ$ introduced by inclusion of the $\mathbf{z}_{new}$ into the support vector set. If $lossJ > c_1$, set $Z = \{Z, \mathbf{z}_{new}\}$, $s_c = lossJ$ and update J_S with the new Z . Otherwise, terminate the algorithm.

End While

B. Pruning phase

While $s_{cp} < c_1$

1. For each $\mathbf{z}_i \in Z$, temporarily remove it, compute Eq. (5.10) with the remaining support vectors and calculate the corresponding $J_S(\mathbf{z}_i)$ using Eq. (5.12). The support vector which leads to the smallest increase on J_S is: $\mathbf{z}_{prune} = \underset{\mathbf{z}_i \in Z}{\operatorname{argmin}} \{J_S(\mathbf{z}_i) - J_S\}$.
2. If $\frac{J_S(\mathbf{z}_{prune}) - J_S}{\gamma n} \geq c_1$, terminate the pruning phase.

Otherwise, remove $\mathbf{z}_{prune}$ from Z

End While

Output: The set of generated support vectors $Z = \{\mathbf{z}_1, \dots, \mathbf{z}_l\}$

Figure 5.5: Pseudo code of the gradient-based support vector generation algorithm.

5.6 Efficient Computations of the Evaluation Criterion

By now, we have described a few SV selection and generation algorithms. In general, two types of evaluations are involved. That is, evaluation for selection/generation and evaluation for pruning. Suppose that we have collected a set Z that contains $\bar{l}$ SVs,

Chapter 5: On Building Sparse LS-SVM Classifier

and the corresponding matrix $\bar{\mathbf{P}} = \begin{bmatrix} \mathbf{P} & \Phi \\ \Phi & n \end{bmatrix}$ (see Eq. (5.10) for detailed definition)

for set Z is denoted by $\bar{\mathbf{P}}_Z$. During the selection/generation procedure, each candidate pattern $\mathbf{x}_i$ is temporarily included into Z and evaluated using the *lossJ* criterion. Such an evaluation involves computation of the inverse of a $(\bar{l} + 2) \times (\bar{l} + 2)$ matrix $\bar{\mathbf{P}}_{ZS}$ corresponding to the SV set $Z \cup \{\mathbf{x}_i\}$. During the pruning procedure, each previously selected SV $\mathbf{z}_j$ is temporarily removed from Z and evaluated, which requires computing the inverse of $\bar{l} \times \bar{l}$ matrix $\bar{\mathbf{P}}_{ZP}$. Since the direct calculation of $\bar{\mathbf{P}}_{ZS}^{-1}$ and $\bar{\mathbf{P}}_{ZP}^{-1}$ requires computational costs of $O((\bar{l} + 2)^3)$ and $O(\bar{l}^3)$ respectively, repeated evaluations of all the candidate patterns may be computationally intensive even with a sub-greedy strategy. Fortunately, by employing some tricks, we do not have to do the inversion directly.

It is given in classical linear algebra text book [85] that

$$\begin{aligned} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}^{-1} &= \begin{bmatrix} \mathbf{F}_{11}^{-1} & -\mathbf{A}_{11}^{-1} \mathbf{A}_{12} \mathbf{F}_{22}^{-1} \\ -\mathbf{F}_{22}^{-1} \mathbf{A}_{12} \mathbf{A}_{11}^{-1} & \mathbf{F}_{22}^{-1} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{F}_{11}^{-1} & -\mathbf{F}_{11}^{-1} \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \\ -\mathbf{A}_{22}^{-1} \mathbf{A}_{21} \mathbf{F}_{11}^{-1} & \mathbf{A}_{22}^{-1} + \mathbf{A}_{22}^{-1} \mathbf{A}_{21} \mathbf{F}_{11}^{-1} \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \end{bmatrix} \end{aligned} \quad (5.19)$$

where $\mathbf{F}_{11} = \mathbf{A}_{11} - \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \mathbf{A}_{21}$ and $\mathbf{F}_{22} = \mathbf{A}_{22} - \mathbf{A}_{21} \mathbf{A}_{11}^{-1} \mathbf{A}_{12}$.

With the same definitions used in Eq. (5.10), $\bar{\mathbf{P}}_{ZS}$ can be written as

$$\bar{\mathbf{P}}_{ZS} = \begin{bmatrix} P_{SS} & \mathbf{P}_{ZS}^T \\ \mathbf{P}_{ZS} & \bar{\mathbf{P}}_Z \end{bmatrix} \quad (5.20)$$

where P_{SS} and $\mathbf{P}_{ZS}$ are introduced by the inclusion of $\mathbf{x}_i$ into Z . Since $\bar{\mathbf{P}}_Z^{-1}$ has been computed in the previous iteration, we can directly substitute Eq. (5.20) into Eq.

Chapter 5: On Building Sparse LS-SVM Classifier

(5.19). Therefore, $\bar{\mathbf{P}}_{ZS}^{-1}$ can be computed with a cost of $O((\bar{l} + 2)^2)$.

Similar to Eq. (5.20), during the pruning procedure, $\bar{\mathbf{P}}_{ZP}$ can be written in form of:

$$\bar{\mathbf{P}}_Z = \begin{bmatrix} P_{PP} & \mathbf{P}_{ZP}^T \\ \mathbf{P}_{ZP} & \bar{\mathbf{P}}_{ZP} \end{bmatrix} \quad (5.21)$$

Hence, the problem is to compute the inverse of $\bar{\mathbf{P}}_{ZP}$ based on $\bar{\mathbf{P}}_Z^{-1}$. According to Eq.

(5.19), if $\mathbf{F}_{11}$ is a scalar (which requires $\mathbf{A}_{11}$ being a scalar), $\mathbf{A}_{22}^{-1}$ can be computed by:

$$\mathbf{A}_{22}^{-1} = \mathbf{F}_{22}^{-1} - \frac{(-\mathbf{A}_{22}^{-1}\mathbf{A}_{21}\mathbf{F}_{11}^{-1})(-\mathbf{F}_{11}^{-1}\mathbf{A}_{12}\mathbf{A}_{22}^{-1})}{\mathbf{F}_{11}^{-1}} \quad (5.22)$$

Therefore, $\bar{\mathbf{P}}_{ZP}^{-1}$ can be computed from $\bar{\mathbf{P}}_Z^{-1}$ with a cost of $O(\bar{l}^2)$.

It should be noted that although Eq. (5.19) and Eq. (5.22) reveal that evaluations can be implemented much more efficiently, they do not change any of our comparisons between the computational time of two SV selection/generation algorithms as the two equations are generally applicable to all the algorithms involved in this chapter.

5.7 An Overview of Other Related Algorithms

As discussed in Chapter 2, the LS-SVM is closely related to quite a few other algorithms, such as kernel Fisher discriminant analysis (KFDA), Gaussian process (GP), standard support vector machine (SVM) and RBF neural networks (RBFNN). Specifically, both KFDA and GP suffer from non-sparse solutions. Thus, it is not surprising that the very general idea employed in the above-described SV selection/generation algorithms has been more or less investigated in the context of

---

Chapter 5: On Building Sparse LS-SVM Classifier

---

algorithms other than the LS-SVM.

In [63], Billings and Lee proposed an SV selection algorithm for KFDA, namely the NFD-OLS algorithm. The NFD-OLS algorithm employs the greedy search strategy and the evaluation criterion is designed based on the orthogonal least square algorithm (OLS) [108]. In GP literature, both greedy and sub-greedy strategies have been employed to build a sparse GP model. Compared with the search strategies, more efforts have been put into designing better evaluation criteria for selecting basis vectors of the GP model. For example, Smola and Bartlett proposed a criterion based on the log posterior probability and used the sub-greedy strategy to optimize it [103]. Candela then suggested an alternative criterion by maximizing the approximate model evidence [112]. In [104], Seeger *et al.* presented another greedy selection method for building sparse GP models. Due to the special formulation of the involved evaluation criterion, this method can be implemented very efficiently. Using ideas from the matching pursuit formulation, Keerthi and Chu presented a promising new evaluation criterion in [113]. Further improved version of the criterion proposed in [103] has also been proposed by Sun and Yao and compared to some existing criteria [105].

Very recently, the idea of generating pseudo pattern vectors has also been considered by Snelson and Ghahramani [114] for building a sparse pseudo-input Gaussian process (SPGP) model. Similar to GPSVG, the SPGP employs a gradient descent algorithm to seek a small number of pseudo-input patterns. The difference is that the pseudo SVs are sought in a greedy manner in the GPSVG algorithm while the SPGP optimizes all of them simultaneously. Thus, the number of pseudo-input patterns must be decided in advance for SPGP, and then the gradient descent algorithm is used to

Chapter 5: On Building Sparse LS-SVM Classifier

---

optimize an ld -dimensional optimization problem. Although the standard SVM is claimed to provide sparse solutions, Wu *et al.* have shown that an even more sparse solution can be achieved for the SVM by seeking some pseudo support vectors [115,116]. In their works, Wu *et al.* also proposed to optimize all the pseudo support vectors together and thereby the number of pseudo support vectors must be pre-defined and the resultant algorithm was named as the sparse large margin classifier (SLMC). More important, Wu *et al.* also sketched a general framework for adapting their formulation to different kernel algorithms and a sparse KFDA (SKFDA) algorithm was presented as an example. Hence, one surely can expect more and more subsequent works along the direction of generating pseudo pattern vectors.

Although the general idea of incorporating a pruning step into the forward selection procedure has not been seriously studied in literatures of LS-SVM, KFDA, GP and SVM, it has been exploited repeatedly in the RBFNN literature for determining the RBF hidden nodes. Generally speaking, a patterns vector will be used as a hidden node if it can give significant contribution to the performance of the RBFNN. Thus, the problem of choosing *novel* pattern vectors as hidden nodes is in general similar to the problem of SV selection. The resource-allocating network (RAN) [117] and the resource-allocating network via Extended Kalman Filter (RANEKF) [118] are two important representative algorithms that sequentially include hidden nodes. In both the RAN and RANEKF algorithms, a hidden node will not be removed once it is created. Subsequently, quite a few RBFNN algorithms that allow pruning insignificant nodes have been proposed, such as the minimal RAN (MRAN) [100,119], the growing and pruning RBFNN (GAP-RBF) [120] and the generalized growing and pruning RBFNN (GGAP-RBF) [121]. Besides the natural differences between the formulations of

LS-SVM and RBFNN (such as different regularization techniques, different objective functions and different learning schemes), our FFSVS algorithm differs from its RBFNN counterparts in several aspects. In the MRAN, the nodes are evaluated independently on the training patterns and multiple nodes can be pruned simultaneously. In the FFSVS algorithm, SVs are always pruned one by one. In the GAP-RBF and GGAP-RBF, the information about sampling distribution or sampling range of the pattern vectors is required, while in FFSVS this information is dispensable.

5.8 Experiments

To evaluate the usefulness of our proposed algorithms, experiments were conducted to compare our algorithms with related approaches.

5.8.1 Algorithms Chosen for Comparison

In the context of LS-SVM, we compared our FFSVS and GPSVG algorithms with the following four algorithms: The NGSVS algorithm (described in Figure 5.3), two SGSVS algorithms (described in Figure 5.2, using the $lossJ$ and $lossSpan$ criteria respectively), and a GSVS algorithm (described in Figure 5.1, using the $lossJ$ criterion). A sparse LS-SVM classifier can be readily trained once the SVs are available. Hence, the performance of an SV selection/generation algorithm was evaluated on the sparse LS-SVM that is built using this algorithm. We did not consider the dQR criterion because it is almost equivalent to the $lossJ$ criterion in practice. From the experimental results for the sub-greedy algorithms, we found that

Chapter 5: On Building Sparse LS-SVM Classifier

---

the *lossSpan* criterion is significantly inferior to the *lossJ* criterion. Therefore, the GSVS algorithm using *lossSpan* was not included in the experiments. Furthermore, we also compared the sparse LS-SVMs built using our algorithms with five other kernel classification methods that are claimed to provide sparse solutions. These methods are the standard SVM, the relevance vector machine (RVM) [122], the SLMC [116], the SKFDA [116] and two sparse greedy kernel-based learning algorithms proposed in [123] (denoted by QR+AIC and QR+MDL in the original work). By comparing with them, we can get a better insight into the usefulness of our algorithms in the larger context of the sparse kernel classifiers.

5.8.2 Datasets

Same as in Chapter 3, eight datasets were used in our experiments. They are the Banana, Breast-cancer, Diabetis, Heart, Image, Splice, Thyroid and Waveform datasets. When comparing our algorithm with the NGSVS, SGSVS, GSVS and SVM, our results were achieved by averaging over all the 100 splits (20 splits for image and splice datasets) of every dataset.

Results of the remaining five approaches (i.e. the RVM, SLMC, SKFDA, QR+AIC and QR+MDL) were taken directly from previous literature [116,122,123]. These approaches were assessed on only four (Banana, Breast-cancer, Image and Waveform) of the above-mentioned eight datasets, and the available results were presented as averages over the first ten splits for each dataset. To make a fair comparison, the results for FFSVS and GPSVG in Table 5.3 were also computed by averaging over the first ten splits.

5.8.3 Model Selection

Same as in [116,122,123], a Gaussian kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\sigma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$ was employed in the experiments. Thus, in addition to the hyper-parameters σ and γ , we were required to determine two more parameters for NGSVS (c_1 and c_2 defined in Figure 5.3) and one more parameter (c_1) for SGSVS, GSVS, FFSVS and GPSVG algorithms. All these parameters were estimated using 5-fold cross-validation. Specifically, we employed a shrinking grid search method described in Chapter 3 to search for the optimal hyper-parameters σ and γ . Since c_1 (and c_2 for the NGSVS) plays a very important role in finding a good trade-off between sparseness and classification accuracy, the optimal value of it was chosen cautiously from the pool $\{0.1, 0.05, 0.02, 0.01, 0.005, 0.002, 0.001, 0.0005, 0.0002, 0.0001, 0.00005, 0.00002, 0.00001, 0.000005, 0.000002, 0.000001\}$. With each element of the pool, we calculated the 5-fold cross-validation error 30 times and located the candidate value that resulted in the lowest cross-validation error (denoted by c^*). Then two-tailed t -tests with significance level 0.05 were performed to find other candidate values that resulted in approximately the same (i.e. statistically indifferent) cross-validation errors. In this way, we typically acquired a few values, and the largest one (which will result in the smallest number of support vectors) among them is chosen for our experiments. An example of model selection of c_1 for the GSVS algorithm is shown in Figure 5.6 on the Heart dataset. As observed from the figure, the lowest cross-validation error can be achieved by setting c_1 as 0.002, 0.001 or 0.0005 (the corresponding $-\log(c_1)$ is 6.21, 6.91 and 7.60, respectively). Hence c_1 is set as 0.002 for this case.

Figure 5.6: Depiction of the model selection scheme on the Heart dataset

5.8.4 Numerical Results

Experimental results are shown in Tables 5.1-5.3. We used a two-tailed t -test with significance level 0.05 to compare the results in the first two tables, and it was shown that all the compared approaches resulted in comparable classification accuracy (we did not apply the t -test to Table 5.3 because standard deviations are not available for the compared methods). Among the 8 methods, the GPSVG algorithm provided the sparsest models on five datasets. The sparsest models for the Breast-cancer and Diabetis datasets were achieved using the FFSVS algorithm and the GSVS algorithm performed the best on the Banana dataset. Compared to the standard SVM, our algorithms consistently resulted in less SVs on all the eight datasets.

Another important observation from Tables 5.1 and 5.2 is that the SGSVS algorithm with $lossJ$ criterion consistently performed better (in terms of number of support vectors) than the SGSVS algorithm with $lossSpan$ criterion (although the latter still

Chapter 5: On Building Sparse LS-SVM Classifier

gave a sparse model). The reason is that the underlying rationale of the *lossSpan* criterion actually violates one of Vapnik’s cornerstone viewpoints. In [23], Vapnik stated that when a problem is solved, one has to try to avoid solving a more general problem as an intermediate step. Hence, when building a sparse LS-SVM for classification, we should not aim to re-construct all the training patterns, which is actually an unnecessary intermediate goal.

	Methods	Banana	B-cancer	Diabetis	Heart
Errors	SVM	10.46 ± 0.49	25.99 ± 5.58	23.17 ± 1.69	15.64 ± 3.30
	LS-SVM	10.72 ± 0.60	26.77 ± 4.52	23.59 ± 1.72	16.59 ± 3.47
	NGSVS	10.73 ± 0.62	26.92 ± 5.43	23.62 ± 1.72	16.42 ± 3.38
	SGSVS _{lossJ}	10.75 ± 0.62	26.94 ± 5.68	23.67 ± 1.82	16.54 ± 3.93
	SGSVS _{lossSpan}	10.73 ± 0.64	27.30 ± 5.89	23.59 ± 1.72	16.95 ± 5.43
	GSVS	10.84 ± 0.70	26.94 ± 5.68	23.67 ± 1.88	16.58 ± 4.04
	FFSVS	10.78 ± 0.67	27.25 ± 5.64	23.62 ± 1.75	16.68 ± 5.14
	GPSVG	11.00 ± 0.79	27.17 ± 5.56	23.60 ± 1.81	16.57 ± 3.82
N_{SV}	SVM	117.66	122.53	263.97	85.57
	LS-SVM	400	200	468	170
	NGSVS	55.26	27.32	28.53	8.36
	SGSVS _{lossJ}	42.91	5.11	17.14	6.16
	SGSVS _{lossSpan}	82.98	22.79	135.31	50.4
	GSVS	26.03	5.11	18.11	5.64
	FFSVS	42.26	3.5	5.19	6.12
	GPSVG	31.13	8.2	15.62	3.24

Table 5.1: Results on the Banana, Breast-cancer, Diabetis and Heart datasets. The testing error (with standard deviation) and number of support vectors (N_{SV}) were averaged over all the available training/testing splits. Since all approaches resulted in comparable testing errors, only the smallest N_{SV} for each dataset is shown in boldface.

The results presented in Table 5.3 are relatively mixed. Overall, FFSVS and GPSVG outperformed QR+AIC and QR+MDL. For the other three compared approaches, RVM achieved distinctively better performance on the Banana and Image datasets,

Chapter 5: On Building Sparse LS-SVM Classifier

while FFSVS and GPSVG algorithms are better on waveform datasets. Hence, we conservatively conclude that our algorithms are capable of constructing competitive sparse kernel classifiers.

	Methods	Image	Splice	Thyroid	Waveform
Errors	SVM	3.0 ± 0.52	10.80 ± 0.72	4.15 ± 2.43	9.87 ± 0.43
	LS-SVM	2.92 ± 0.66	10.86 ± 0.65	5.04 ± 2.02	10.01 ± 0.50
	NGSVS	2.84 ± 0.57	11.24 ± 0.78	5.99 ± 2.04	9.95 ± 0.51
	SGSVS _{lossJ}	2.87 ± 0.58	11.86 ± 1.12	5.11 ± 2.04	10.00 ± 0.86
	SGSVS _{lossSpan}	2.98 ± 0.74	11.36 ± 1.06	5.25 ± 2.76	10.04 ± 1.10
	GSVS	2.92 ± 0.58	11.51 ± 1.20	5.05 ± 2.04	10.16 ± 0.96
	FFSVS	3.04 ± 0.63	11.96 ± 1.16	5.96 ± 2.01	10.34 ± 1.38
	GPSVG	3.07 ± 0.56	11.02 ± 1.20	5.39 ± 1.97	10.08 ± 0.69
N_{SV}	SVM	162.2	707.95	15.44	158.42
	LS-SVM	1300	1000	140	400
	NGSVS	527.8	633.75	80.41	64.06
	SGSVS _{lossJ}	425.5	558.1	85.33	40.61
	SGSVS _{lossSpan}	661.5	700.6	67.83	297.69
	GSVS	382.6	526.2	71.97	26.08
	FFSVS	346.4	503.35	61.46	26.27
	GPSVG	153.1	37.65	15.46	13.19

Table 5.2: Results on the Image, Splice, Thyroid and Waveform datasets. The testing error (with standard deviation) and number of support vectors (N_{SV}) were averaged over all the available training/testing splits. Most approaches resulted in comparable testing errors (except for the standard SVM on Thyroid dataset, which is shown in boldface), the smallest N_{SV} for each dataset is shown in boldface.

5.9 Conclusions

In this chapter, we investigated the problem of constructing a sparse LS-SVM classifier. We presented an overview on existing support vector selection methods. Subsequently, we proposed to select support vectors in a floating manner. Further, since the support vectors do not necessarily belong to the “real” training set, we

Chapter 5: On Building Sparse LS-SVM Classifier

propose an algorithm to generate pseudo support vectors. Experimental studies indicate that our algorithms consistently lead to smaller number of support vectors, at comparable classification accuracy, than those existing methods for building sparse LS-SVM. When compared to other related sparse kernel classifiers, the sparse LS-SVMs built by our algorithms still exhibit competitive performance. Finally, our methods provide appealing by-products. That is, the general ideas used in our algorithms can be exploited to sparsify many other current kernel classifiers and thereby contribute to the development of the whole family of kernel methods.

	Methods	Banana	B-cancer	Image	Waveform
Errors	FFSVS	10.9	29.9	3.2	10.6
	GPSVG	10.9	29.5	3.0	10.0
	RVM	10.8	29.9	3.9	10.9
	QR+AIC	10.7	28.0	3.1	12.6
	QR+MDL	10.9	31.3	3.5	11.6
	SLMC	11.0	27.9	3.6	9.9
	SKFDA	10.8	25.5	4.0	9.9
N_{SV}	FFSVS	26.4	3.4	155.4	17.5
	GPSVG	32.5	3.5	128.5	8
	RVM	11.4	6.3	35.6	15.6
	QR+AIC	41	11	199	101
	QR+MDL	29	5	175	27
	SLMC	40	20	130	40
	SKFDA	40	20	130	40

Table 5.3: Further comparison with related algorithms. Results of the FFSVS and GPSVG algorithms are averaged over the first 10 splits of each dataset. Results of RVM are taken from [122], results of QR+AIC and QR+MDL are given in [123] and results of SLMC and SKFDA are presented in [116]. Since these previous works did not present standard deviation of the testing error, it is also omitted for our algorithms.

Chapter 6

An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

6.1 Introduction

Also known as committees of learners, mixtures of experts and multiple classifier systems, classifier ensemble has been well established as a research area in the past decades. In general, a classifier ensemble is a combination of multiple individual classifiers (usually referred to as the base classifiers). Since different base classifiers potentially offer complementary information about the patterns to be classified, a classifier ensemble is usually expected to achieve better classification performance than a single classifier. Actually, various classifier ensembles have been repeatedly shown to be very successful in the literature [124-131]. Intuitively speaking, the key to the success of a classifier ensemble is that the base classifiers perform diversely. Empirical studies have shown that there exists positive correlation between accuracy of the classifier ensemble and diversity among the base classifiers [132-134]. Hence, large diversity among the base classifiers is usually desired in the classifier ensembles.

When constructing the classifier ensemble using a so-called ensemble learning algorithm, if the term “diversity” is defined explicitly and maximized, we say the algorithm seeks diversity explicitly. Otherwise the algorithm seeks diversity implicitly. In the literature,

---

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

most existing ensemble learning algorithms, such as bagging, boosting and negative correlation learning can be interpreted as building diverse base classifiers implicitly. And many classifier ensembles have been successfully proposed by applying these ensemble learning algorithms to different types of base classifiers. In the context of kernel classifier, the support vector machine (SVM) ensemble has been proposed using the bagging method and outperformed the single SVM classifier [135-137]. Recently, the effectiveness of bagging LS-SVM ensemble was also verified in [138].

Since LS-SVM ensemble has been successfully proposed by seeking diversity implicitly (i.e. using the bagging method), it is natural to consider whether we can perform better by seeking the diversity explicitly. Unfortunately, despite the popularity of the term diversity, there is no single definition and measure of it. Several measures have been proposed to represent the diversity and are optimized explicitly in different ensemble learning algorithms, but none of these measures is proven superior to the others. Furthermore, even the validity of explicitly maximizing diversity measures has been questioned. Although generally we can achieve satisfactory accuracy with a set of diverse base classifier, experimental studies on other types of base classifiers have shown that we may not achieve optimal classification accuracy by explicitly maximizing a diversity measures [133] [134]. Why these diversity measures may lead to contradictory results is still unclear. Even though some extensive empirical studies have been conducted and provide more insight into the true meaning of diversity [133,139], theoretical validation of the explicitly diversity-driven algorithms is absent from the literature. In particular, before we

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

utilize the concept of diversity to construct classifier ensemble, three main questions that arose from several previous empirical studies [132,133,140] must be addressed.

1. When one seeks a set of diverse and accurate base classifiers, what cost function is optimized? Does optimizing the cost function guarantee good generalization performance?
2. Is there a trade-off between diversity and accuracy of the base classifiers? In other words, do we have to sacrifice the accuracy of some base classifiers in order to increase the diversity?
3. How to make use of the existing diversity measures for designing good classifier ensembles? Besides the existing diversity measures, is a more precise diversity measure necessary for designing ensemble learning algorithms?

Although all the three questions are important, none of them has been thoroughly answered, while several contradictory conclusions have been drawn from experimental studies. Motivated by this observation, we present an in-depth analysis of some existing diversity measures in this chapter. More specifically, the margin of the classifier ensemble is employed as the main tool in our analysis. The concept of margin was originally proposed for analyzing the behavior of boosting type algorithms [141] and later generalized to other classes of ensemble learning algorithms [131,142]. We analyze the underlying relationships between the diversity measures and the margin of a classifier ensemble. By this means, we put the study of diversity measures into the context of so-called large margin classifiers and attempt to answer the above-mentioned questions by

---

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

both theoretical and experimental analysis. Instead of finishing with a promising ensemble learning algorithm for building LS-SVM ensemble, the most important conclusion that can be drawn from our analysis is: Given the existing diversity measures, the LS-SVM ensemble built by seeking diversity explicitly CANNOT consistently achieve good classification performance. Since similar conclusion has been drawn for several other base classifier models [133], such a conclusion for LS-SVM is not that surprising. However, to the best of our knowledge, this is the first time that the topic is viewed from the angle of margin maximization.

In comparison to the previous chapters, this chapter is somewhat different. To a large extent, the concepts of diversity, margin and even the classifier ensemble are independent of the base classifiers. Hence, the specific formulation of LS-SVM (and even the kernel classifiers) does not play a key role in this chapter. Instead, LS-SVM is only considered as one type of base classifier to which our analysis directly applies. Besides, in chapters 3 to 5, we formulated problems and proposed particular solutions. While we demonstrate in this chapter *what we cannot do* when building the LS-SVM ensemble, and thereby contribute to the development of LS-SVM in a different way.

The remainder of this chapter is organized as follows. In Section 6.2, the diversity measures, the margin and some relevant background of classifier ensembles are introduced. In Section 6.3, we analyze theoretically and experimentally the relationships among diversity measures, average classification accuracy of base classifiers and margin of the classifier ensemble. After that, applications of diversity measures in LS-SVM

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

ensemble construction are discussed in Section 6.4, and conclusions are presented in Section 6.5.

6.2 Related Background

Assume that a labeled training set is given as $S_{tr} = \{(\mathbf{x}_i, y_i), i = 1, 2, \dots, N\}$, where $y_i \in \{-1, +1\}$. The base classifiers $H = \{h_1, h_2, \dots, h_L\}$ of an ensemble are trained on the training set, and the output of a base classifier h_j on training pattern $\mathbf{x}_i$ is $h_j(\mathbf{x}_i)$. Given this set of base classifiers, together with a corresponding set of weights $\mathbf{w} = [w_1, w_2, \dots, w_L]^T$, where $w_j \geq 0$ and $\sum w_j = 1$, the ensemble classifies the patterns by taking a weighted vote among the base classifiers and choosing the class label that receives the largest weighted vote. Given L base classifiers and N training patterns, a classifier ensemble is associated with an oracle output matrix $\mathbf{O}$, which is defined as:

$$O_{ij} = 1 \quad \text{if training pattern } \mathbf{x}_i \text{ is classified correctly by base classifier } h_j$$

$$O_{ij} = -1 \quad \text{otherwise}$$

Hence, the oracle output matrix is an $N \times L$ matrix whose entries take either 1 or -1. Although it appears to be simple, an oracle output matrix provides very detailed information about the behavior of the corresponding classifier ensemble. In the literature, many other authors discussed the diversity in terms of oracle outputs and most of the diversity measures were defined based on oracle outputs. For example, all of the ten diversity measures summarized by Kuncheva and Whitaker [133] are based on oracle

---

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

outputs and the majority vote rule. Hence, the oracle output matrix is an indispensable tool for our analysis.

For a classifier ensemble, let us denote the classification accuracy of the base classifier h_j and the average training accuracy of all the base classifiers by p_j and P respectively.

Defining $l_i = L \sum_{O_{ij}=-1} w_j$ as the product of L and sum of the weights of the base classifiers

that misclassify the training pattern $\mathbf{x}_i$, we get

$$P = \sum_{j=1}^L w_j p_j, \quad (6.1)$$

$$P = 1 - \frac{\sum_{i=1}^N l_i}{NL}, \quad (6.2)$$

$$p_j = \frac{1}{2} + \frac{\sum_{i=1}^N O_{ij}}{2N}. \quad (6.3)$$

When all base classifiers of the ensemble are uniformly weighted (i.e. $w_j = 1/L$, $\forall h_j \in H$), we have the majority vote rule. Since majority vote is simple and is employed in most works concerned with diversity measures, we will first present our analysis on the basis of it and then generalize our conclusions to the non-uniformly weighted case. For the uniformly weighted case, we simply replace the weights $1/L$ with 1 for the definition of l_i . Hence l_i becomes the number of base classifiers that classify the training pattern $\mathbf{x}_i$ incorrectly. This small modification will not influence our analysis. In the subsequent sections, unless we refer explicitly to the non-uniform weights, all definitions and derivations are based on the majority vote rule.

6.2.1 The Diversity Measures

Since a unique definition of diversity does not exist, we consider six diversity measures in this chapter. These measures were proposed by different researchers independently and summarized in [133]. When we mention diversity, we refer to the diversity that is defined by these measures.

The Disagreement Measure

In [143], Skalak proposed the disagreement measure to evaluate the diversity between two base classifiers. Ho also employed the disagreement to measure diversity in a decision forest [129]. This measure is defined based on the intuition that two diverse classifiers perform differently on the same training data. Given two base classifiers h_j and h_k , let $n(a,b)$ be the number of training patterns on which the oracle output of h_j and h_k is a and b respectively. The diversity between the two base classifiers is measured by

$$dis_{j,k} = \frac{n(1,-1) + n(-1,1)}{n(1,1) + n(-1,1) + n(1,-1) + n(-1,-1)} \quad (6.4)$$

Diversity within the whole set of base classifiers is then calculated by averaging over all pairs of base classifiers:

$$dis = \frac{2}{L(L-1)} \sum_{j=1}^L \sum_{k=j+1}^L dis_{j,k} \quad (6.5)$$

Since for any pair of base classifiers, $n(1,1) + n(1,-1) + n(-1,1) + n(-1,-1) = N$, then we can get

$$dis = \frac{2}{NL(L-1)} \sum_{j=1}^L \sum_{k=j+1}^L (n_{j,k}(1,-1) + n_{j,k}(-1,1)) \quad (6.6)$$

The diversity increases with the value of the disagreement measure.

The Double-Fault Measure

Giacinto and Roli proposed the double-fault measure to select classifiers that are least related from a pool of classifiers [144]. The double-fault measure between a pair of base classifiers is calculated by

$$DF_{j,k} = \frac{n(-1, -1)}{n(1, 1) + n(-1, 1) + n(1, -1) + n(-1, -1)} \quad (6.7)$$

This measure also arose from the intuition that two classifiers should perform differently to be diverse. Giacinto and Roli claimed that the more different two classifiers are, the fewer the coincident errors between them. Same as the disagreement measure, the diversity within the whole set of base classifiers is calculated as follows:

$$DF = \frac{2}{NL(L-1)} \sum_{j=1}^L \sum_{k=j+1}^L n_{j,k}(-1, -1) \quad (6.8)$$

The diversity decreases when the value of the double-fault measure increases.

Kohavi-Wolpert Variance

The Kohavi-Wolpert variance was proposed by Kohavi and Wolpert in their decomposition formula of the classification error of a classifier [145]. This measure originated from the bias-variance decomposition of the error of a classifier. The expression of the variability of the predicted class label y for a pattern $\mathbf{x}$ is defined as

$$\text{var}_{\mathbf{x}} = \frac{1}{2} \left(1 - \sum_{i=1}^C P(y = \omega_i | \mathbf{x})^2 \right) \quad (6.9)$$

where C is the number of classes. Since $C=2$ and $P(y=1|\mathbf{x}) + P(y=-1|\mathbf{x}) = 1$ in the case of oracle output, we can get

---

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

$$\begin{aligned}
 \text{var}_{\mathbf{x}} &= \frac{1}{2} \left(1 - P(y=1|\mathbf{x})^2 - P(y=-1|\mathbf{x})^2 \right) \\
 &= P(y=1|\mathbf{x})P(y=-1|\mathbf{x}) \\
 &= P(O=1|\mathbf{x})P(O=-1|\mathbf{x})
 \end{aligned}$$

Let $l_i = L \sum_{O_{ij}=-1} w_j$ denote the product of L and sum of the weights of the base classifiers

that misclassify the training pattern $\mathbf{x}_i$, the term $P(O=-1|\mathbf{x})$ can be estimated by

$P(O=-1|\mathbf{x}) = l_i / L$. Based on this estimation, Kuncheva and Whitaker [133] presented a

modified version of equation (6.9) to measure the diversity of a classifier ensemble:

$$KW = \frac{1}{NL^2} \sum_{i=1}^N l_i (L - l_i) \quad (6.10)$$

The diversity increases with values increasing of the KW variance.

Measurement of Inter-rater Agreement

This measure is developed as a measure of inter-rater (inter-classifier) reliability [146]. It can be used to measure the level of agreement within a set of classifiers, hence it is also based on the assumption that a set of classifiers should disagree with one another to be diverse. The diversity decreases when inter-rater agreement (denoted by Ir) increases.

The Ir is calculated by

$$Ir = 1 - \frac{\sum_{i=1}^N (L - l_i) l_i}{NL(L-1)P(1-P)} \quad (6.11)$$

Generalized Diversity

The generalized diversity measure (GD) is proposed by Partidge and Krzanowski [147].

The heuristic behind this measure is similar to that of the Double-Fault measure. Given

---

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

two classifiers, Partidge and Krzanowski argued that maximum diversity is achieved when failure of one classifier is accompanied by correct classification by the other classifier and minimum diversity occurs when two classifiers fail together. Therefore, for a pattern $\mathbf{x}_i$ that is randomly drawn from the training set, let T_j denote the probability that $l_i = j$, the generalized diversity is defined as

$$GD = 1 - \frac{\sum_{j=1}^L \frac{j(j-1)}{L(L-1)} T_j}{\sum_{j=1}^L \frac{j}{L} T_j} \quad (6.12)$$

The diversity increases with increasing values of the generalized diversity.

The Measure of “Difficulty”

This measure comes from the study of Hansen and Salamon [148]. Defining a discrete random variable V , $V_i = (L - l_i) / L$ for a pattern $\mathbf{x}_i$ that is randomly drawn from the training set, the measure of difficulty was defined as the variance of V over the whole training set.

$$diff = \text{var}(V_i) \quad (6.13)$$

The diversity increases with decreasing values of the measure of difficulty. The intuition of this measure can be explained as: a diverse classifier ensemble has a smaller value for this measure since every training pattern can at least be classified correctly by a portion of all the base classifiers, which is likely to result in lower variance of V .

	<i>dis</i>	<i>DF</i>	<i>KW</i>	<i>Ir</i>	<i>GD</i>	<i>diff</i>
<i>div</i>	+	-	+	-	+	-

Table 6.1: Summary of six diversity measures. *div* denotes the diversity, the “+” means that diversity increases with the measure, and the “-” means that diversity decreases with the measure.

6.2.2 The Margins of Classifier Ensembles

To explain the remarkable success of the boosting algorithm, Schapire *et al.* introduced the concept of margin into the area of ensemble learning [141]. Let $v_{i,C}$ be the total vote that the weighted ensemble casts for label C on pattern $\mathbf{x}_i$, the margin of the classifier ensemble on this pattern is defined as $m_i = v_{i,y_j} - \sum_{C \neq y_j} v_{i,C}$. Given a set of base classifiers and the weights $\mathbf{w} = [w_1, w_2, \dots, w_L]^T$, where $w_j \geq 0$ and $\sum w_j = 1$, margin of the ensemble can be calculated by

$$m_i = \sum_{j=1}^L w_j O_{ij} \quad (6.14)$$

Since boosting type algorithms construct a classifier ensemble with respect to the vote rule, the concept of margin was easily generalized to all the classifier ensembles that employ the vote rule as the combination method [141]. Several extensive studies have shown that the generalization performance of a classifier ensemble is related to the distribution of its margins on the training patterns. Schapire *et al.* proved that achieving a larger margin on the training set results in an improved bound on the generalization error of the ensemble. They also proposed the upper bound explicitly as the sum of a function of distribution of the margins and a complexity penalty term [141]. Rätsch *et al.* analyzed the margins by focusing on the “minimum margin” of the ensemble on training patterns [62]. Employing boosting as an example, they explained the good generalization performance of a classifier ensemble in terms of the minimum margin that can be achieved by it: the ensemble with the largest minimum margin will have the best

---

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

generalization error bound¹ [23,141]. Since the generalization error itself is actually immeasurable, this relationship provides us an approach to analyze the relationship between diversity and generalization performance.

6.3 Analysis of Diversity Measures

6.3.1 Relationship between Diversity and Margins: Majority Vote Case

If we regard the generalization performance of a classifier ensemble as a function F that is parameterized by the average classification accuracy P of the base classifiers and the diversity among the base classifiers (div), our analysis can be formulated as follows: we analyze how F changes with respect to div if P is fixed, and how P and div interact with each other to influence F . Aiming to maximize the diversity between base classifiers, we begin with answering the first question that is mentioned in Section 1, the cost functions for the six diversity measures can be reformulated as:

Disagreement measure:
$$dis = \frac{2L(1-P)}{(L-1)} - \frac{2}{NL(L-1)} \sum_{i=1}^N l_i^2 \quad (6.15)$$

Double-fault measure:
$$DF = \frac{1}{NL(L-1)} \sum_{i=1}^N l_i^2 - \frac{1-P}{L-1} \quad (6.16)$$

Kohavi-Wolpert variance:
$$KW = 1 - P - \frac{1}{NL^2} \sum_{i=1}^N l_i^2 \quad (6.17)$$

---

¹However, the ensemble that is constructed by maximizing the margin may over-fit when the data is noisy. That is, the generalization performance of the ensemble may decrease when the minimum margin is maximized. Some discussion is presented in Section 6.3.

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

Measurement of inter-rater agreement:
$$Ir = \frac{LP - P - L}{LP - P} + \frac{\sum_{i=1}^N l_i^2}{NL(L-1)P(1-P)} \quad (6.18)$$

Generalized diversity:
$$GD = \frac{L}{L-1} - \frac{\sum_{i=1}^N l_i^2}{NL(L-1)(1-P)} \quad (6.19)$$

Measure of “Difficulty”:
$$diff = \frac{1}{NL} \sum_{i=1}^N l_i^2 - L(1-P)^2 \quad (6.20)$$

It can be observed from Eqs. (6.15)-(6.20) that all these cost functions contain the terms P and $\sum_{i=1}^N l_i^2$. Based on this observation, we propose the Lemma below.

Lemma 6.1: *If we regard the average classification accuracy P of the base classifiers as a constant, the diversity (div) is maximized only when all the training patterns are classified correctly by the same number of base classifiers, which means:*

$$l_i = L(1-P), \quad \forall i \quad (6.21)$$

Appendix D.1 provides detailed derivations of Eqs. (6.15)-(6.20) and proof of Lemma 6.1.

We call equation (6.21) the *uniformity condition* for maximizing the diversity. In the uniformly weighted case, the margin of a classifier ensemble on $\mathbf{x}_i$ is calculated by

$$m_i = \frac{1}{L} \sum_{j=1}^L O_{ij} \quad (6.22)$$

and
$$m_i = \frac{L - 2l_i}{L}. \quad (6.23)$$

then
$$\sum_{i=1}^N m_i = \frac{1}{L} \sum_{i=1}^N (L - 2l_i) = N(2P - 1) \quad (6.24)$$

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

Since $\min(m_i) \leq \frac{1}{N} \sum_{i=1}^N m_i$, we have

$$\min(m_i) \leq 2P - 1 \quad (6.25)$$

As mentioned in Section 6.2.2, the best generalization error bound of a classifier ensemble can be achieved by maximizing the minimum margin of the ensemble on the training patterns. Hence, the best generalization error bound of the ensemble is achieved when $\min(m_i) = 2P - 1$. It is obvious that the equality in Eq. (6.25) holds only when

$$m_i = 2P - 1, \quad \forall i \quad (6.26)$$

which means:

$$l_i = L(1 - P), \quad \forall i \quad (6.21)$$

Therefore, for the six diversity measures discussed in this chapter, the answer to the first question posed in Section 6.1 can be summarized by Eqs. (6.15)-(6.20) and the following Lemma.

Lemma 6.2: *If P is regarded as a constant and the maximum diversity is **achievable**, maximizing the diversity among the base classifiers is equivalent to maximizing the minimum margin of the ensemble on the training patterns.*

Lemma 6.2 can be readily proved by combining Lemma 6.1 with Eq. (6.26). Hence, seeking diversity in a classifier ensemble can be viewed as an implicit way to maximize the minimum margin of the classifier ensemble.

6.3.2 Ineffectiveness of Diversity Measures

In addition to theoretical derivations, we conduct several experiments here to illustrate relationships among the average accuracy P , the diversity and the minimum margin of a

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

classifier ensemble. In the first experiment, setting the number of patterns as $N=100$, the number of base classifiers $L=150$ and the average accuracy $P=0.7$, we randomly generate 10000 pseudo oracle output matrices. These matrices are used to represent a set of classifier ensembles. The corresponding diversity measures and the minimum margins are calculated and relationships among them are plotted in figures 6.1a -6.1f. The maximum minimum margin in this case is 0.4. For each diversity measure, the optimized value is also presented in the figures. In the past, experimental study has revealed that large diversity does not always correspond to good generalization performance even when P is fixed [133]. That is why usefulness of diversity measures was questioned. From the previous subsection and figures 6.1a-6.1f, two reasons of this discrepancy can be summarized:

1. *For a given P , the maximum diversity is usually not achievable.*

According to the definition, l_i 's take discontinuous values, while $L(1-P)$ is a continuous variable since P is continuous. Hence, Eq. (6.21) cannot be satisfied in general. When we attempt to seek diversity, we usually achieve only *large* diversity but not the *maximum* diversity.

2. *The minimum margin of an ensemble is not monotonically increasing with the diversity.*

From figures 6.1a-6.1f, we observe that the minimum margin of a classifier ensemble is not monotonically increasing with the diversity, albeit positive correlation is as obvious as in the literature. Therefore, enlarging diversity is different from enlarging the minimum margin. When P is a constant, we can find from Eqs. (6.15)-(6.20) that the smaller the

---

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

term $\sum l_i^2$, the larger the diversity. From Eq. (6.23), we can also find that the minimum margin is determined by the maximum l_i . The smaller the $\max(l_i)$, the larger the minimum margin. Since a smaller $\sum l_i^2$ does not always yield a smaller $\max(l_i)$, a larger value for the diversity does not guarantee a larger value for the minimum margin.

The two above-mentioned reasons result in the fact that large diversity may not consistently correspond to a better generalization performance. This is a main drawback of using diversity measures in the implementation of an ensemble learning algorithm.

The first experiment only demonstrates the diversity measures under the assumption that P is a constant. However, P and div actually interact with each other to influence the performance of a classifier ensemble. By taking a closer look at Eqs. (6.21) and (6.25), one finds that P determines the upper bound of the minimum margin, while distribution of l_i over the training set determines difference between the achieved minimum margin and this bound. To illustrate this in the second experiment, we set L at 150 and tune P to $3/5$, $2/3$, $3/4$, $4/5$ and $9/10$. In Figures 6.2a-6.2f, we plot the relationship between each measure and the minimum margin with different values of P in one figure. The figures show that increasing average accuracy does increase the upper-bound of the minimum margin, but the realized minimum margin may or may not increase (although the positive correlation between diversity and minimum margin is obvious in Figures 6.2). In order to maximize the minimum margin of a classifier ensemble, we need to maximize P and simultaneously satisfy the uniformity condition in Eq. (6.21). However, we have presented that the diversity measures does not represent the uniformity condition well. So,

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

Figure 6.1a: Disagreement measure

Figure 6.1b: Double-fault measure

Figure 6.1c: Kohavi-Wolpert variance

Figure 6.1d: Measurement of inter-rater agreement

Figure 6.1e: Generalized diversity

Figure 6.1f: Measure of "Difficulty"

Figures 6.1(a-f): Illustration of the relationships between diversity measures and the minimum margin (experiment 1). $L=150$, horizontal axis represents the diversity measures and vertical axis represents the minimum margin.

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

we cannot expect a monotonic relationship between the generalization performance and the interactions between accuracy and diversity. This explains why experimental results in the past were inconclusive and sometimes contradictory.

It was usually deemed that there exists a trade-off between the average accuracy and diversity. In other words, smaller P may correspond to larger div . This hypothesis is not reasonable according to our decomposition of the diversity measures. From Eqs. (6.15)-(6.20), all the diversity measures described in Section 6.2.1 can be formulated as

$div = a - \left(bP + c \sum_{i=1}^N l_i^2 \right)$, where a , b and c are constants. Hence, relationship between

div and P is influenced by the term $\sum_{i=1}^N l_i^2$. To study the relationship between P and

$\sum_{i=1}^N l_i^2$, we carry out the third set of experiments. We set L at 15, randomly choose values

for P between $[0.5, 1]$ and generate ensembles with the chosen P . 10000 ensembles are

generated again and relationship between P and the term $\sum_{i=1}^N l_i^2$ is plotted in Fig. 6.3. An

obvious negative correlation between P and the term $\sum_{i=1}^N l_i^2$ can be observed with a

correlation coefficient value of -0.9689 . The strong negative correlation between P and

$\sum_{i=1}^N l_i^2$ implies that a smaller P is not likely to result in a smaller value of $\left(bP + c \sum_{i=1}^N l_i^2 \right)$,

and hence may not corresponds to larger div . Since the term $\sum_{i=1}^N l_i^2$ may take many

different values for a given P , the true relationship between diversity and accuracy is

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

complex. Therefore, the answer to the second question presented in Section 6.1 is that we do not have to sacrifice some accuracy for diversity.

Figure 6.2a: Disagreement measure

Figures 6.2(a-f): Illustration of the relationships between diversity measures and the minimum margin (experiment 2). $L=15$, X-axis represents the diversity measures and Y-axis represents the minimum margin. The upper bound of the minimum margin is also shown in the figures corresponding to $P=3/5$, $2/3$, $3/4$, $4/5$, and $9/10$ with “*”.

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

Figure 6.2b: Double-fault measure

Figure 6.2c: Kohavi-Wolpert variance

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

Figure 6.2d: Measurement of inter-rater agreement

Figure 6.2e: Generalized diversity

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

Figure 6.2f: Measure of “Difficulty”

Figure 6.3: Relationship between P and $\sum_{i=1}^N l_i^2$. Horizontal axis is P and vertical axis is $\sum_{i=1}^N l_i^2$.

Another noteworthy property that we have observed about the existing diversity measures is the lack of a regularization term. When designing a classification system, regularization

---

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

is usually considered to avoid the over-fitting problem. In particular, regularization terms have been incorporated in the boosting type methods and the SVM, two main algorithms that are based on the concept of margin maximization [23,62,149]. In contrast, we cannot find a regularization term in the discussed six diversity measures. This observation implies that even when maximum values for the diversity measures are achieved, we may only obtain undesirably over-fitted solutions.

-
1. Generate 300 LS-SVM using the training set.
  2. Select the LS-SVM h_1 which has the smallest training error, the oracle output of this LS-SVM is denoted by $\mathbf{O}_1$, $\mathbf{O} = \mathbf{O}_1$
  3. Greedily select other LS-SVMs

For $i=1:149$

for all previously unselected h_j , calculate $div([\mathbf{O}, \mathbf{O}_j])$;

 select classifier h_j such that $div([\mathbf{O}, \mathbf{O}_j])$ is maximized;

 Set $\mathbf{O} = [\mathbf{O}, \mathbf{O}_j]$,

End
  4. Apply the ensemble to the testing set using majority vote rule.
-

Figure 6.4: Pseudo code of a greedy approach for building LS-SVM ensemble

All the experiments above do not really construct classifier ensembles and evaluate their performance on real-world problems. Hence, we have carried out a simple empirical experiment to demonstrate our analysis. In this experiment, we employed the LS-SVM as the base classifier, and the LS-SVM ensembles were evaluated on 11 different 2-class

---

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

datasets described in [8,62]. For each of the datasets, the experiment was conducted for 100 times on randomly partitioned training and test sets with 60% for training and 40% for testing. For each pair of training and test sets, 300 different LS-SVMs were generated by using random subsets of the training set, random subsets of the input features or even using randomly determined hyper-parameters. We first selected the LS-SVM with the best performance. Then the diversity measures were employed to sequentially select other base classifiers. Pseudo code of the experiment is presented in Figure 6.4.

	<i>dis</i>	<i>DF</i>	<i>KW</i>	<i>Ir</i>	<i>GD</i>	<i>diff</i>	LS-SVM
Banana	15.3 ± 1.4	12.7 ± 0.9	15.3 ± 1.4	13.4 ± 0.7	12.6 ± 0.7	13.1 ± 1.2	10.8 ± 0.6
B-cancer	25.8 ± 4.6	25.4 ± 4.0	25.8 ± 4.6	25.4 ± 4.8	25.4 ± 4.0	25.3 ± 4.5	26.8 ± 4.5
Diabetis	28.7 ± 2.0	24.7 ± 1.7	28.7 ± 2.0	28.9 ± 2.1	27.2 ± 1.7	26.9 ± 1.6	23.5 ± 1.8
F-solar	35.2 ± 1.6	35.1 ± 1.8	35.2 ± 1.9	35.2 ± 2.7	35.3 ± 1.7	35.2 ± 2.0	34.2 ± 1.9
German	25.1 ± 2.2	25.3 ± 2.2	25.1 ± 2.2	26.1 ± 2.1	25.9 ± 2.0	25.8 ± 2.0	23.6 ± 2.1
Heart	26.2 ± 4.7	16.5 ± 3.6	26.2 ± 4.5	20.2 ± 3.5	17.5 ± 2.9	18.2 ± 3.2	16.4 ± 3.6
Ringnorm	6.5 ± 0.7	4.3 ± 0.8	6.5 ± 0.7	3.4 ± 0.3	3.5 ± 0.3	3.3 ± 0.2	1.6 ± 0.2
Thyroid	7.4 ± 3.7	4.9 ± 2.3	7.4 ± 3.3	4.9 ± 2.3	4.5 ± 2.0	4.9 ± 2.2	5.1 ± 2.3
Titanic	24.8 ± 4.7	22.7 ± 1.2	24.8 ± 0.7	23.2 ± 0.8	22.7 ± 1.2	24.7 ± 1.7	22.6 ± 0.8
Twonorm	8.4 ± 3.5	3.2 ± 0.3	8.4 ± 3.8	3.1 ± 0.2	3.5 ± 0.3	3.1 ± 0.3	2.5 ± 0.2
Waveform	18.5 ± 1.3	11.8 ± 0.7	18.5 ± 1.5	13.0 ± 0.7	12.3 ± 0.8	12.2 ± 0.6	10.0 ± 0.6

Table 6.2: Classification results of single LS-SVM and LS-SVM ensembles achieved by seeking diversity explicitly

Results of the experiment are presented in Table 6.2, in which the single LS-SVM is also included for comparison. Among all the 11 datasets, exploiting diversity to construct classifier ensemble only yielded better performance on the Breast Cancer and the Thyroid datasets. For the other nine datasets, sometimes results of the ensembles are comparable to that of single LS-SVM (e.g. *DF* on the Heart dataset, *DF*, *Ir* and *diff* on the

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

Two norm dataset). But we can observe that the classifier ensembles perform significantly worse in many cases, such as *Dis* for the Heart dataset and *Ir* for the Diabetis dataset. According to previous analysis, these empirical results demonstrate that seeking diversity explicitly is not likely to result in promising classification performance consistently.

Based on all the results presented above, we are able to make the following conclusions: Compared to those algorithms that seek diversity implicitly, exploiting diversity measures to seek diversity explicitly is ineffective in consistently achieving classifier ensembles with good generalization performance. Firstly, the change of measured diversity cannot provide consistent guidance on whether a set of base classifiers possesses good generalization performance. Secondly, the measures are naturally related to the average accuracy of the base classifiers. This property is not desirable since we do not require the diversity measures to become another estimate for the classification accuracy.

6.3.3 Relationship between Diversity and Margins: General Case

So far we have presented our analysis based on the assumption that all the individual base classifiers are assigned the same weights. Now we will show that this assumption is not strictly necessary. Lemmas 6.1 and 6.2 can be generalized to the non-uniform weights of the base classifiers, which also validates our experimental analysis in Section 6.3.2 for the non-uniformly weighted case. For brevity we present the disagreement measure as an example here, detailed derivation for the other five measures can be found in Appendix D.2.

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

Since the weights of base classifiers are no longer same now, some equations we introduced in previous sections need to be modified. Given L base classifiers, the $N \times L$ oracle output matrix $\mathbf{O}$ and a weighting vector $\mathbf{w} = [w_1, w_2, \dots, w_L]^T$ for the base classifiers, Eq. (6.5) is modified as:

$$dis = \frac{2}{L(L-1)} \sum_{j=1}^L \sum_{k=j+1}^L w_i w_j dis_{j,k} \quad (6.27)$$

and the definition of l_i remains as:

$$l_i = L \sum_{o_{ij}=-1} w_j \quad (6.28)$$

With the definitions presented in Eqs. (6.1) and (6.14), Eqs. (6.2) and (6.23) still hold in this case. Using some simple derivations, we have:

$$\sum_{j=1}^L \sum_{k=j+1}^L w_i w_j (n_{j,k}(1, -1) + n_{j,k}(-1, 1)) = \frac{N\mathbf{w}^T \mathbf{w} - \mathbf{w}^T \mathbf{O}^T \mathbf{O} \mathbf{w}}{4} \quad (6.29)$$

Let $\{\mathbf{Ow}\}_i$ be the i th element of vector $\mathbf{Ow}$. For each training pattern $\mathbf{x}_i$, we have

$$\{\mathbf{Ow}\}_i = \sum_{o_{ij}=1} w_j - \sum_{o_{ij}=-1} w_j$$

Then

$$l_i = L \sum_{o_{ij}=-1} w_j = \frac{L(1 - \{\mathbf{Ow}\}_i)}{2}$$

and

$$L - l_i = L - L \sum_{o_{ij}=-1} w_j = \frac{L(1 + \{\mathbf{Ow}\}_i)}{2}$$

which gives

$$\sum_{i=1}^N l_i (L - l_i) = \frac{L^2 (N - \mathbf{w}^T \mathbf{O}^T \mathbf{O} \mathbf{w})}{4} \quad (6.30)$$

So equation (6.27) can be re-written as:

$$dis = \frac{2}{NL^3(L-1)} \sum_{i=1}^N l_i (L - l_i) - \frac{1 - \mathbf{w}^T \mathbf{w}}{2L(L-1)}$$

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

Since equation (6.2) still holds in this case, we have

$$\sum_{i=1}^N l_i = NL(1-P)$$

and hence

$$dis = \frac{2(1-P)}{L(L-1)} - \frac{2}{NL^3(L-1)} \sum_{i=1}^N l_i^2 - \frac{1-\mathbf{w}^T \mathbf{w}}{2L(L-1)} \quad (6.31)$$

As $\mathbf{w}$ is the given weighting vector, the term $\frac{1-\mathbf{w}^T \mathbf{w}}{2L(L-1)}$ can be regarded as a constant

when optimizing dis with respect to l_i . Therefore, except for an additional L^2 term in the denominator, Eq. (6.31) is almost the same as Eq. (6.15). When optimizing Eq. (6.31), since both the cost function and the constraint are in the same form as we have shown for majority vote case in Appendix D.1, the solution of this problem will also be in the same form as in the majority vote case. Therefore, the previous analysis is also applicable to non-uniformly weighted case, and Lemma 6.2 can be generalized as:

Lemma 6.3 (generalized form of Lemma 6.2): *Given L base classifiers weighted by weights $\mathbf{w} = [w_1, w_2, \dots, w_L]^T$, if P is regarded as a constant and the maximum diversity is achievable, maximizing diversity among the base classifiers is equivalent to maximizing the minimum margin of the ensemble on the training patterns.*

6.4 Exploiting Diversity Measures in LS-SVM Ensemble Construction

Up to this point, we have analyzed six diversity measures. All of them require satisfying the uniformity condition. When the uniformity condition is maximized, the minimum

---

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

margin of the classifier ensemble is maximized. In order to design a classifier ensemble, simply analyzing relationship between diversity, training accuracy and margin of the ensemble is not enough. All the conclusions of the theoretical and experimental results must be considered in a much larger context. In the following, we try to propose some answers for the third question posed in Section 6.1 by examining possible applications of diversity measures for building an LS-SVM ensemble.

Typically, a classifier ensemble can be constructed in three stages:

1. Given the training patterns, generate a set of base classifiers
2. Select a subset of the generated base classifiers
3. Construct the ensemble with a combination scheme

These procedures can be illustrated using two examples, a boosting algorithm [127] and the neural network ensemble proposed by Giacinto and Roli in [144]. In a boosting algorithm, we first generate the base classifiers sequentially. After that, the base classifiers are pruned either to avoid over-fitting or to reduce computational complexity for testing. Here pruning functions as a classifier selector and several methods have been proposed in the literature [150,151]. Finally, the selected base classifiers are combined according to the pre-defined weighted vote rule. In the second example, after generating the base classifiers, a clustering method is employed to cluster them, then only one classifier is chosen from each cluster to construct the ensemble, and majority vote is employed as the combination rule.

In these procedures, a classifier ensemble is actually constructed in stages 2 and 3. What is the optimal base classifier is unclear in stage 1 and any base classifier generated in this

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

stage may be useful for the final ensemble. Therefore, the goal in the first stage is only to generate diverse classifiers rather than achieving a good classifier ensemble. Recently, a nice survey of diversity creation work for classifier ensemble has been presented in [152]. According to that work, to generate diverse LS-SVMs, we can either manipulate the training patterns that are fed to the LS-SVMs, or simply use different hyper-parameters for them. Clearly, a combination of both the approaches of course can also yield a set of diverse LS-SVMs. For example, we can use the bagging method or random sampling to acquire different training subsets from the original training set. The same idea can also be implemented along the features (i.e. different feature subsets are fed to different LS-SVMs), which results in the so-called random subspace method [129]. Since the bagging and random subspace methods do not seek diversity explicitly, they may not lead to LS-SVMs that are sufficiently diverse. By making use of the diversity measures, one may generate more diverse LS-SVMs since difference between them is required more explicitly. For this purpose, the definition of diversity is not necessary to be very precise and any above-mentioned diversity measure can be used.

The problem to be solved in the second stage is much more difficult than in the first stage. One can easily find that the base classifier selection problem is naturally similar to the feature selection problem. Hence, we can employ a well-developed feature selection method to select base classifiers. For example, after acquiring a number of different LS-SVMs, we can employ either a simple sequential forward selection method (as we employed in the experiment in Fig. 6.4) or the LOOCSFS proposed in Chapter 4 to select LS-SVMs for building the LS-SVM ensemble. If a really large number (which is much

---

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

large than the number of training patterns) of LS-SVMs are generated, the GLGS algorithm presented in Chapter 4 can also be used. In principle, any diversity measure can be exploited as the evaluation criterion to select the LS-SVMs. But it is required to be precise since the choice of diversity measure will directly influence the final LS-SVM ensemble and subsequently the classification result.

After selecting L base classifiers, a (typically linear) combination scheme should be decided in the third stage. This problem is naturally a classification problem in an L -dimensional space. Hence effective classification methods can be easily exploited. Diversity measures are generally not applicable in this stage. Another application of the diversity measures is to visualize relationships of the base classifiers in a classifier ensemble, or relationship between different ensembles. This application is not directly related to classifier ensemble construction. Hence, the precise definition of diversity is again not quite important.

6.5 Conclusions

Motivated by the attempt of constructing an LS-SVM ensemble, we review in this chapter six existing measures that quantify the diversity among base classifiers in a classifier ensemble. We demonstrate explicitly the relation between these measures and the margin maximization concept, which accounts for the success of several pattern classification algorithms. Since many diversity measures are heuristically motivated from different areas of pattern classification, it is necessary to verify first whether they are useful or not for building our LS-SVM ensemble. In this process, we presented the uniformity

Chapter 6: An Analysis of Diversity Measures for Constructing LS-SVM Ensemble

---

condition for maximizing both the diversity and the minimum margin of a classifier ensemble. We have also demonstrated the ineffectiveness of diversity measures for constructing LS-SVM ensembles. Although this chapter does not readily lead to a novel approach for constructing LS-SVM ensemble, we do expect that such a goal can be achieved on basis of our analysis.

Chapter 7

Conclusions and Future Works

7.1 Conclusions

In this thesis, we have extensively investigated one important kernel method, namely the least squares support vector machines (LS-SVM), for pattern classification, and enhanced the original LS-SVM model from the following aspects.

The classification performance of LS-SVM is influenced significantly by its hyper-parameters. In Chapter 3, we proposed two new model selection approaches for tuning the hyper-parameters of LS-SVM. First, we derived an efficient and exact computation for the leave-one-out error of LS-SVM, and the hyper-parameters can be tuned by minimizing the leave-one-out error using some optimization techniques. Specifically, a gradient descent algorithm and an evolution strategy algorithm were exploited in our model selection approaches. In the context of kernel classifiers, it is commonly expected that a more complex kernel function is likely to yield better classification performance. However, the grid search, which is the most commonly-used model selection approach in LS-SVM literature, can only handle relatively simple kernels. In comparison, our gradient-based and evolutionary-based model selection approaches allow highly complex kernels. Using both of our approaches, the expected advantages of complex kernel functions have been validated experimentally.

Chapter 7: Conclusions and Future Research

In some real-world classification problems, the data of interest may be characterized by a large number of features while the number of training patterns is relatively small. In these cases, an LS-SVM with linear kernel can well solve the classification task, and the more challenging problem is finding those features that are the most important for classification. One typical example of such a scenario is selecting important genes for microarray data classification. Originally proposed as a standard classification tool, the LS-SVM can also be modified to address feature selection problems. Two feature selection methods, namely the leave-one-out calculation sequential forward selection (LOOCSFS) algorithm and the gradient-based gene selection (GLGS) algorithm, have been proposed based on the LS-SVM formulation (in Chapter 4) and applied to the gene selection problems. Although experimental study was restricted to microarray datasets, no biological knowledge is incorporated in the design of the two proposed algorithms. Hence, both LOOCSFS and GLGS can be readily applied to other real-world problems, and provide attractive alternatives to the existing feature selection methods.

In spite of its desirable properties, one main drawback of LS-SVM is the non-sparseness. To classify a testing pattern, a trained LS-SVM requires evaluating it on all the training patterns. When the training set is large-sized, both time and storage complexity of the LS-SVM will be high for testing. In Chapter 5, approaches of building sparse LS-SVM have been investigated. With the introduction of the idea of sequential floating forward selection, we firstly proposed an approach to select support vectors from the training patterns. Then, we suggested that one might be able to build more effectively a sparse LS-SVM by generating pseudo support vectors in the input feature space. A detailed algorithm has also been proposed to implement

such a general idea.

By the tools of the margin concept, the usefulness of some diversity measures for LS-SVM ensemble construction was examined in Chapter 6. Although it is well acknowledged that diversity is one of the most important issues for classifier ensembles, we have found that explicitly optimizing diversity measures does not yield good classification performance. Such an observation was theoretically and experimentally analyzed. We showed that optimizing the diversity measures is equivalent to maximizing the margin of the LS-SVM ensemble, but the margin does not consistently increase with the diversity. Since the maximum diversity usually cannot be achieved in real-world problems, optimizing diversity may not yield a large margin LS-SVM ensemble. Therefore, good classification performance cannot be guaranteed. Further, suggestions have been presented on how to employ the diversity measures to construct LS-SVM ensemble.

In summary, this thesis has investigated four main problems in the context of LS-SVM, namely model selection, feature selection, sparse kernel classifier and classifier ensemble construction. Explicit algorithms have been proposed to address the first three problems. For the last problem, we conducted an in-depth analysis, which will help construct a successful LS-SVM ensemble in the near future.

7.2 Recommendations for Further Research

Finally, we point out several related directions which deserve further investigations.

- The first direction is to build a successful LS-SVM ensemble. The negative correlation (NC) learning [35,130,153] provides a quite appealing framework for

Chapter 7: Conclusions and Future Research

---

building classifier ensemble. In the literature, negatively correlated neural network ensemble has been shown to be very successful classification tool. Hence, it is reasonable to expect an LS-SVM ensemble by modifying the NC framework according to the LS-SVM formulation.

- In Chapter 4, the leave-one-out calculation sequential forward feature selection (LOOCSFS) algorithm and the gradient-based gene selection (GLGS) algorithm are proposed for 2-class classification problems. A natural question is how to extend them to multi-class problems. In the literature, most of the existing kernel classifiers (including the LS-SVM) were originally proposed for 2-class problems. When employing kernel classifiers to solve multi-class problems, one usually divides the problem into several 2-class sub-problems, and kernel classifiers are then applied to each sub-problem. Final classification is conducted by combining the outputs of all the kernel classifiers involved. Along the above-described idea, quite a few structures have been investigated for multi-class kernel classifiers and showed their effectiveness [24,154-158]. However, the feature selection algorithms associated with them is not well developed. If we directly apply the existing feature selection algorithms to each of the sub-problems, different feature subsets will be achieved. Then we will have to store all of them. Assume that the original feature set is F and feature subsets F_1 , F_2 and F_3 are selected for different sub-problems. If $F_1 \cup F_2 \cup F_3 = F$, the advantage of conducting feature selection is totally lost since we may still have to store the whole original feature set. Even if $F_1 \cup F_2 \cup F_3 \subset F$, it is not appealing to store different feature subsets for

---

Chapter 7: Conclusions and Future Research

---

different sub-problems. Instead, we will prefer a unified feature subset that can be used to solve all the sub-problems well. Hence, the LOOCSFS and GLGS algorithms need to be generalized to tackle multi-class problems.

- For the support vector selection/generation algorithms, generalization from 2-class to multi-class problems is also required.
- All the work in LS-SVM literature, including those presented in this thesis, can be further extended to the research field of semi-supervised learning. Semi-supervised learning can be viewed as a special form of classification task [159,160]. Traditional classifiers need all the training patterns to be labeled. Labeled patterns however are often difficult, expensive, or time consuming to obtain, as they require the efforts of experienced human annotators. Meanwhile it may be relatively easy to collect unlabeled patterns, but there has been few ways to use them in the training phase. Semi-supervised learning addresses this problem by using large amount of unlabeled patterns, together with the labeled patterns, to train classifiers. Because semi-supervised learning requires less human effort and gives higher accuracy (as the training phase is conducted using more patterns), it is of great interest both in theory and in practice. To our knowledge, the LS-SVM has never been investigated in the semi-supervised manner. Hence, a lot of work is to be done in the future.

Author's Publications

Journal Publications

1. **E. K. Tang**, P. N. Suganthan and X. Yao, "An analysis on ensemble learning algorithms," *Machine Learning*, **65**: 247-271, 2006.
2. **E. K. Tang**, P. N. Suganthan and X. Yao, "Gene selection for microarray data by using LS-SVM," *BMC-Bioinformatics*, 7:95, 27 February 2006.
3. **E. K. Tang**, P. N. Suganthan, X. Yao and A. K. Qin, "Linear Dimensionality Reduction Using Relevance Weighted LDA," *Pattern Recognition*, 38(4):485-493, 2005.

Conference Publications

1. A. Ashish, G. Fogel, **E. K. Tang** and P. N. Suganthan, "Feature Selection Approach for Quantitative Prediction of Transcriptional Activities," *Proceedings of the 2006 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)* 2006.
2. **E. K. Tang**, P. N. Suganthan, X. Yao, "Feature Selection for Microarray Data Using Least Squares SVM and Particle Swarm Optimization," *Proceedings of the 2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (Best Student Paper Award)*, 2005.
3. **E. K. Tang**, P. N. Suganthan and X. Yao, "Nonlinear Feature Extraction Using Evolutionary Algorithm," *11th Int. Conference on Neural Information Processing*, Calcutta, India, LNCS Vol. 3316, pp. 1014-1019, November 2004.
4. **E. K. Tang**, P. N. Suganthan and X. Yao, "Generalized LDA Using Relevance Weighting and Evolution Strategy", *Proceedings of the 2004 Congress on Evolutionary Computation*, Portland, USA, Vol. 2, pp. 2230-2234, June 2004.

Appendix A: Proof of Lemma 3.2

Summary of denotations

Scalars:

b : the bias term in LS-SVM

γ : the regularization parameter of LS-SVM

μ : a variable defined in the proof of Lemma 3.2

Vectors:

$\mathbf{x}_i$: a d -dimensional column vector representing the i th pattern

$\phi(\mathbf{x}_i)$: $\mathbf{x}_i$ in the kernel space

$\mathbf{Y}$: the class label of n training patterns, $\mathbf{Y} = [y_1, y_2, \dots, y_n]^T$

$\mathbf{1}_n$: an n -by-1 vector with all the entries equal to 1

$\mathbf{e}$: regression errors of the training patterns, $\mathbf{e} = [e_1, e_2, \dots, e_n]^T$

$\boldsymbol{\alpha}$: the vector of Lagrange multipliers that are to be computed in the training phase of LS-SVM, $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_n]^T$

$\boldsymbol{\beta}$, $\boldsymbol{\delta}$, $\boldsymbol{\lambda}$, $\boldsymbol{\lambda}^*$, $\boldsymbol{\lambda}'$ and $\bar{\boldsymbol{\lambda}}$ are n -dimensional vectors that are defined in the proof of Lemma 3.2.

Matrices:

$\mathbf{I}_n$: the identity matrix of size n

$\mathbf{K}$ and $\hat{\mathbf{K}}$: the kernel matrix, $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and $\tilde{\mathbf{K}} = \mathbf{K} + \gamma^{-1}\mathbf{I}_n$

$\boldsymbol{\Omega}$ and $\hat{\boldsymbol{\Omega}}$: two $n \times n$ matrices, $\Omega_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$ and $\tilde{\boldsymbol{\Omega}} = \boldsymbol{\Omega} + \gamma^{-1}\mathbf{I}_n$

$\mathbf{H}$: an $(n+1) \times (n+1)$ matrix, $\mathbf{H} = \begin{bmatrix} \mathbf{K} + \gamma^{-1}\mathbf{I}_n & \mathbf{1}_n \\ \mathbf{1}_n^T & 0 \end{bmatrix}$

$\bar{\boldsymbol{\Omega}}$: an $(n+1) \times (n+1)$ matrix, $\bar{\boldsymbol{\Omega}} = \begin{bmatrix} \boldsymbol{\Omega} + \gamma^{-1}\mathbf{I}_n & \mathbf{Y} \\ \mathbf{Y}^T & 0 \end{bmatrix}$

For any matrix $\mathbf{A}$, $\mathbf{A}_{/ij}$ is the j th column of $\mathbf{A}$ with the i th entry removed. $\mathbf{A}_{/i}$ is the sub-matrix of $\mathbf{A}$ with the row and column i removed.

Appendix A

Preliminaries

As we know, the LS-SVM can be formulated as

$$\begin{aligned} \min_{\mathbf{w}, b, \mathbf{e}} J(\mathbf{w}, b, \mathbf{e}) &= \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{\gamma}{2} \sum_{i=1}^n e_i^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) = 1 - e_i \quad \text{for } i = 1, \dots, n \end{aligned} \quad (2.2)$$

By introducing Lagrange multipliers $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_n]^T$, solution of the optimization problem is given by the saddle point of the Lagrangian:

$$\min_{\mathbf{w}, b, \mathbf{e}, \boldsymbol{\alpha}} L(\mathbf{w}, b, \mathbf{e}, \boldsymbol{\alpha}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{\gamma}{2} \sum_{i=1}^n e_i^2 - \sum_{i=1}^n \alpha_i [y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + e_i] \quad (2.3)$$

$$\text{Subject to} \quad \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \phi(\mathbf{x}_i) \quad (2.4)$$

$$\sum_{i=1}^n \alpha_i y_i = 0 \quad (2.5)$$

$$\alpha_i = \gamma e_i \quad i = 1, \dots, n \quad (2.6)$$

$$y_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) - 1 + e_i = 0 \quad i = 1, \dots, n \quad (2.7)$$

Substitute Eqs. (2.4)-(2.7) into Eq. (2.3), Eq. (2.2) can be further converted to another optimization problem:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} L(\boldsymbol{\alpha}) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) - \frac{1}{2\gamma} \sum_{i=1}^n \alpha_i^2 \\ \text{subject to} \quad & \sum_{i=1}^n \alpha_i y_i = 0 \quad \text{for } i = 1, \dots, n \end{aligned} \quad (A.1)$$

Eq. (2.2) and Eq. (A.1) share the same optimal solution, which can be computed by:

a unique solution of Eq. (2.2) can be computed by solving the following linear system:

$$\begin{bmatrix} \boldsymbol{\Omega} + \gamma^{-1} \mathbf{I}_n & \mathbf{Y} \\ \mathbf{Y}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{1}_n \\ 0 \end{bmatrix} \quad (2.8)$$

where $\boldsymbol{\Omega} = \{y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)\} = \{y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)\}$, $\mathbf{Y} = [y_1, y_2, \dots, y_n]^T$, $\mathbf{I}_n$ is an identity matrix and $\mathbf{1}_n$ is an n -by-1 vector with all the entries equal to 1.

Lemma 3.2: For any training pattern $\mathbf{x}_p$, there is

$$1 - y_p f^p(\mathbf{x}_p) = \frac{\alpha_p}{(\mathbf{H}^{-1})_{pp}} \quad (3.4)$$

where α_p is the p th Lagrange multiplier that is computed by training the LS-SVM using all the training patterns.

Proof:

Note: This proof is applicable to any kernel functions $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$, but we use $\mathbf{K} = \{k(\mathbf{x}_i, \mathbf{x}_j)\} = \{\mathbf{x}_i^T \mathbf{x}_j\}$ for simplicity.

When a pattern $\mathbf{x}_p$ is removed from the training set in the leave-one-out cross-validation procedure, the variables associated with the trained LS-SVM f^p are denoted by $\mathbf{a}^p$, $\mathbf{w}_p$ and b_p respectively. In the leave-one-out cross-validation procedure, $\mathbf{a}^p = [\alpha_1^p, \alpha_2^p, \dots, \alpha_n^p]^T$ can be computed by solving the maximization problem in Eq. (A.1) with an additional constraint:

$$\alpha_p^p = 0 \quad (A.2)$$

Accordingly, we have:

$$\mathbf{w}_p = \sum_{i=1, i \neq p}^n \alpha_i^p y_i \mathbf{x}_i = \sum_{i=1}^n \alpha_i^p y_i \mathbf{x}_i \quad (A.3)$$

$$f^p(\mathbf{x}_p) = \mathbf{w}_p^T \mathbf{x}_p + b_p \quad (A.4)$$

To prove Lemma 3.2, we first introduce another Lemma as below:

Lemma A: For any training pattern $\mathbf{x}_p$ the following equality holds:

$$1 - y_p f^p(\mathbf{x}_p) = \min_{\boldsymbol{\lambda}} \alpha_p \boldsymbol{\lambda}^T (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}$$

where $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \dots, \lambda_n]^T$ and $\lambda_p = -1, \sum_{i=1}^n \lambda_i = 0$.

Proof of Lemma A:

As we know, $\mathbf{a} = [\alpha_1, \alpha_2, \dots, \alpha_n]^T$ is the solution of the maximization problem

Appendix A

---

presented in Eq. (A.1). Thus,

$$L(\mathbf{a}) \geq L(\mathbf{a}^p + \boldsymbol{\beta}) \quad (\text{A.5})$$

where $\boldsymbol{\beta} = [\beta_1, \beta_2, \dots, \beta_n]^T$ satisfies the following conditions:

$$\sum_{i=1}^n \beta_i y_i = 0 \quad (\text{A.6})$$

Since $\mathbf{a}^p$ also optimize Eq. (A.1) with the additional constraint $\alpha_p^p = 0$, we have:

$$L(\mathbf{a}^p) \geq L(\mathbf{a} - \boldsymbol{\delta}) \quad (\text{A.7})$$

where $\boldsymbol{\delta} = [\delta_1, \delta_2, \dots, \delta_n]^T$ satisfies the conditions:

$$\delta_p = 0, \quad \sum_{i=1}^n \delta_i y_i = 0 \quad (\text{A.8})$$

Hence, we obtain:

$$L(\mathbf{a}^p + \boldsymbol{\beta}) - L(\mathbf{a}^p) \leq L(\mathbf{a}) - L(\mathbf{a}^p) \leq L(\mathbf{a}) - L(\mathbf{a} - \boldsymbol{\delta}) \quad (\text{A.9})$$

Let us define $I_1 = L(\mathbf{a}^p + \boldsymbol{\beta}) - L(\mathbf{a}^p)$ and $I_2 = L(\mathbf{a}) - L(\mathbf{a} - \boldsymbol{\delta})$ and calculate them separately:

$$\begin{aligned} I_1 &= L(\mathbf{a}^p + \boldsymbol{\beta}) - L(\mathbf{a}^p) \\ &= \sum_{i=1}^n (\alpha_i^p + \beta_i) - \frac{1}{2} \sum_{i,j=1}^n (\alpha_i^p + \beta_i)(\alpha_j^p + \beta_j) y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \frac{1}{2\gamma} \sum_{i=1}^n (\alpha_i^p + \beta_i)^2 \\ &\quad - \sum_{i=1}^n \alpha_i^p + \frac{1}{2} \sum_{i,j=1}^n \alpha_i^p \alpha_j^p y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \frac{1}{2\gamma} \sum_{i=1}^n (\alpha_i^p)^2 \\ &= \sum_{j=1}^n \beta_j (1 - y_j \sum_{i=1}^n \alpha_i^p y_i \mathbf{x}_i^T \mathbf{x}_j - \frac{\alpha_j^p}{\gamma}) - \frac{1}{2} (\sum_{i,j=1}^n \beta_i \beta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\beta_i^2}{\gamma}) \\ &= \sum_{j=1}^n \beta_j (1 - y_j \mathbf{w}_p^T \mathbf{x}_j - \frac{\alpha_j^p}{\gamma}) - \frac{1}{2} (\sum_{i,j=1}^n \beta_i \beta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\beta_i^2}{\gamma}) \\ &= \sum_{j=1}^n \beta_j (1 - y_j \mathbf{w}_p^T \mathbf{x}_j - \frac{\alpha_j^p}{\gamma}) - \frac{1}{2} (\sum_{i,j=1}^n \beta_i \beta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\beta_i^2}{\gamma}) - b_p \sum_{i=1}^n \beta_i y_i \\ &= \sum_{j=1}^n \beta_j [1 - y_j (\mathbf{w}_p^T \mathbf{x}_j + b_p) - \frac{\alpha_j^p}{\gamma}] - \frac{1}{2} (\sum_{i,j=1}^n \beta_i \beta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\beta_i^2}{\gamma}) \\ &= \beta_p [1 - y_p (\mathbf{w}_p^T \mathbf{x}_p + b_p)] - \frac{1}{2} (\sum_{i,j=1}^n \beta_i \beta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\beta_i^2}{\gamma}) \end{aligned} \quad (\text{A.10})$$

$$\begin{aligned}
I_2 &= L(\mathbf{a}) - L(\mathbf{a} - \delta) \\
&= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \frac{1}{2\gamma} \sum_{i=1}^n \alpha_i^2 \\
&\quad - \sum_{i=1}^n (\alpha_i - \delta_i) + \frac{1}{2} \sum_{i,j=1}^n (\alpha_i - \delta_i)(\alpha_j - \delta_j) y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \frac{1}{2\gamma} \sum_{i=1}^n (\alpha_i - \delta_i)^2 \\
&= \sum_{j=1}^n \delta_j (1 - y_j \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_j - \frac{\alpha_j}{\gamma}) + \frac{1}{2} (\sum_{i,j=1}^n \delta_i \delta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\delta_i^2}{\gamma}) \\
&= \sum_{j=1}^n \delta_j (1 - y_j \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_j - \frac{\alpha_j}{\gamma}) + \frac{1}{2} (\sum_{i,j=1}^n \delta_i \delta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\delta_i^2}{\gamma}) - b \sum_{i=1}^n \delta_i y_i \\
&= \sum_{j=1}^n \delta_j [1 - y_j (\mathbf{w}^T \mathbf{x}_j + b) - \frac{\alpha_j}{\gamma}] + \frac{1}{2} (\sum_{i,j=1}^n \delta_i \delta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\delta_i^2}{\gamma}) \\
&= \frac{1}{2} (\sum_{i,j=1}^n \delta_i \delta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\delta_i^2}{\gamma}) \tag{A.11}
\end{aligned}$$

Let $\beta_i = \delta_i = \alpha_i - \alpha_i^p \quad \forall i$, then:

$$\begin{aligned}
I_1 &= I_2 \\
&= L(\mathbf{a}) - L(\mathbf{a}^p) \\
&= \beta_p [1 - y_p (\mathbf{w}_p^T \mathbf{x}_p + b_p)] - \frac{1}{2} (\sum_{i,j=1}^n \beta_i \beta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\beta_i^2}{\gamma}) \\
&= \frac{1}{2} (\sum_{i,j=1}^n \delta_i \delta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\delta_i^2}{\gamma}) \tag{A.12}
\end{aligned}$$

Hence, we have

$$\begin{aligned}
\beta_p [1 - y_p (\mathbf{w}_p^T \mathbf{x}_p + b_p)] &= (\alpha_p - \alpha_p^p) [1 - y_p (\mathbf{w}_p^T \mathbf{x}_p + b_p)] \\
&= \alpha_p [1 - y_p (\mathbf{w}_p^T \mathbf{x}_p + b_p)] \\
&= \frac{1}{2} (\sum_{i,j=1}^n \beta_i \beta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\beta_i^2}{\gamma} + \sum_{i,j=1}^n \delta_i \delta_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{\delta_i^2}{\gamma}) \\
&= 2[L(\mathbf{a}) - L(\mathbf{a}^p)] \\
&= \sum_{i,j=1}^n (\alpha_i - \alpha_i^p)(\alpha_j - \alpha_j^p) y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \frac{(\alpha_i - \alpha_i^p)^2}{\gamma} \tag{A.13}
\end{aligned}$$

Assume that pattern $\mathbf{x}_p$ is removed from the training set in the leave-one-out cross-validation procedure, we define the T_p as a constrained set of linear

Appendix A

combinations of the remaining $n-1$ training patterns:

$$T_p = \{ \sum_{i=1, i \neq p}^n \lambda_i \mathbf{x}_i, \text{ subject to } \sum_{i=1, i \neq p}^n \lambda_i = 1 \} \quad (\text{A.14})$$

We also define a quantity S_p as the square distance between $\mathbf{x}_p$ and this set:

$$S_p = d^2(\mathbf{x}_p, T_p) = \min_{\mathbf{z} \in T_p} (\|\mathbf{x}_p - \mathbf{z}\|^2) \quad (\text{A.15})$$

By setting $\lambda_p = -1$, we can re-write S_p as:

$$S_p = \min \left\{ \left\| \sum_{i=1}^n \lambda_i \mathbf{x}_i \right\|^2, \text{ subject to } \sum_{i=1}^n \lambda_i = 0 \text{ and } \lambda_p = -1 \right\} \quad (\text{A.16})$$

Finally, we define

$$\boldsymbol{\lambda}'^T (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}' = \min \{ \boldsymbol{\lambda}^T (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}, \text{ subject to } \sum_{i=1}^n \lambda_i = 0 \text{ and } \lambda_p = -1 \} \quad (\text{A.17})$$

Based on all the above definitions and Eq. (A.13), it is easy to show that:

$$(\alpha_p - \alpha_p^p)[1 - y_p(\mathbf{w}_p^T \mathbf{x}_p + b_p)] = \alpha_p^2 \boldsymbol{\lambda}^{*T} (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}^* \quad (\text{A.18})$$

where $\boldsymbol{\lambda}^* = [\lambda_1^*, \lambda_2^*, \dots, \lambda_n^*]^T$ and $\lambda_i^* = \frac{y_p y_i (\alpha_i^p - \alpha_i)}{\alpha_p}$.

Let $\delta_i = -\alpha_p y_p y_i \lambda_i'$, we have

$$2[L(\boldsymbol{\alpha}) - L(\boldsymbol{\alpha} - \boldsymbol{\delta})] = \alpha_p^2 \boldsymbol{\lambda}'^T (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}' \quad (\text{A.19})$$

Since $L(\boldsymbol{\alpha}) - L(\boldsymbol{\alpha}^p) \leq L(\boldsymbol{\alpha}) - L(\boldsymbol{\alpha} - \boldsymbol{\delta})$, then according to Eq. (A.18) and Eq. (A.19):

$$\alpha_p^2 \boldsymbol{\lambda}^{*T} (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}^* \leq \alpha_p^2 \boldsymbol{\lambda}'^T (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}'$$

However, according to the definitions of $\boldsymbol{\lambda}^*$ and $\boldsymbol{\lambda}'$, we know that:

$$\alpha_p^2 \boldsymbol{\lambda}^{*T} (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}^* \geq \alpha_p^2 \boldsymbol{\lambda}'^T (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}'$$

Therefore

$$\alpha_p^2 \boldsymbol{\lambda}^{*T} (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}^* = \alpha_p^2 \boldsymbol{\lambda}'^T (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}' = \alpha_p [1 - y_p(\mathbf{w}_p^T \mathbf{x}_p + b_p)]$$

which gives

$$1 - y_p(\mathbf{w}_p^T \mathbf{x}_p + b_p) = \min_{\boldsymbol{\lambda}} \alpha_p \boldsymbol{\lambda}^T (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda}$$

Hence we complete the proof of Lemma A.

Proof of Lemma 3.2

Since the matrix $\mathbf{K} + \gamma^{-1}\mathbf{I}_n$ is symmetric and positive definite, the minimization problem involved in Lemma A can be modified as:

$$\begin{aligned}
& \min_{\lambda} \lambda^T (\mathbf{K} + \gamma^{-1}\mathbf{I}_n) \lambda \\
& = \min_{\lambda_i} (\lambda_p^2 \tilde{K}_{pp} + 2\lambda_p \sum_{i=1, i \neq p}^n \lambda_i \tilde{K}_{ip} + \sum_{i,j=1, i,j \neq p}^n \lambda_i \lambda_j \tilde{K}_{ij}) \\
& = \min_{\lambda_i} (\tilde{K}_{pp} - 2 \sum_{i=1, i \neq p}^n \lambda_i \tilde{K}_{ip} + \sum_{i,j=1, i,j \neq p}^n \lambda_i \lambda_j \tilde{K}_{ij}) \\
& \text{subject to } \sum_{i=1, i \neq p}^n \lambda_i = 1 \text{ and } \lambda_p = -1
\end{aligned} \tag{A.20}$$

where $\tilde{\mathbf{K}} = \mathbf{K} + \gamma^{-1}\mathbf{I}_n$. By introducing a new Lagrange multiplier μ , we can further modify Eq. (A.20) as:

$$\begin{aligned}
& \min_{\lambda_i} (\tilde{K}_{pp} - 2 \sum_{i=1, i \neq p}^n \lambda_i \tilde{K}_{ip} + \sum_{i,j=1, i,j \neq p}^n \lambda_i \lambda_j \tilde{K}_{ij}) \\
& = \min_{\lambda_i} \{ \max_{\mu} [\tilde{K}_{pp} - 2 \sum_{i=1, i \neq p}^n \lambda_i \tilde{K}_{ip} + \sum_{i,j=1, i,j \neq p}^n \lambda_i \lambda_j \tilde{K}_{ij} + 2\mu (\sum_{i=1, i \neq p}^n \lambda_i - 1)] \} \\
& = \min_{\lambda_i} \{ \max_{\mu} (\tilde{K}_{pp} - 2\mathbf{H}_{/pp} \bar{\lambda} + \bar{\lambda}^T \mathbf{H}_{/p} \bar{\lambda}) \}
\end{aligned} \tag{A.21}$$

where $\bar{\lambda} = [\lambda_1, \dots, \lambda_{p-1}, \lambda_{p+1}, \dots, \lambda_n, \mu]^T$, $\mathbf{H} = \begin{bmatrix} \mathbf{K} + \gamma^{-1}\mathbf{I}_n & \mathbf{1}_n \\ \mathbf{1}_n^T & 0 \end{bmatrix}$, $\mathbf{H}_{/pp}$ is the p th column of $\mathbf{H}$ with the p th entry removed. $\mathbf{H}_{/p}$ is the sub-matrix of $\mathbf{H}$ with the row and column p removed.

From the fact that the optimal $\bar{\lambda}$ can be computed by $(\mathbf{H}_{/p})^{-1} \mathbf{H}_{/pp}$, it follows:

$$\min_{\lambda_i} \{ \max_{\mu} (\tilde{K}_{pp} - 2\mathbf{H}_{/pp} \bar{\lambda} + \bar{\lambda}^T \mathbf{H}_{/p} \bar{\lambda}) \} = \tilde{K}_{pp} - \mathbf{H}_{/pp}^T (\mathbf{H}_{/p})^{-1} \mathbf{H}_{/pp}$$

For a block matrix, there is

$$\begin{aligned}
\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}^{-1} &= \begin{bmatrix} \mathbf{F}_{11}^{-1} & -\mathbf{A}_{11}^{-1} \mathbf{A}_{12} \mathbf{F}_{22}^{-1} \\ -\mathbf{F}_{22}^{-1} \mathbf{A}_{12} \mathbf{A}_{11}^{-1} & \mathbf{F}_{22}^{-1} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{F}_{11}^{-1} & -\mathbf{F}_{11}^{-1} \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \\ -\mathbf{A}_{22}^{-1} \mathbf{A}_{21} \mathbf{F}_{11}^{-1} & \mathbf{A}_{22}^{-1} + \mathbf{A}_{22}^{-1} \mathbf{A}_{21} \mathbf{F}_{11}^{-1} \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \end{bmatrix}
\end{aligned}$$

where $\mathbf{F}_{11} = \mathbf{A}_{11} - \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \mathbf{A}_{21}$ and $\mathbf{F}_{22} = \mathbf{A}_{22} - \mathbf{A}_{21} \mathbf{A}_{11}^{-1} \mathbf{A}_{12}$. By re-arranging the matrix

$\mathbf{H}$ (change the positions of rows and columns in the matrix), we can get:

Appendix A

$$(\tilde{K}_{pp} - \mathbf{H}_{/pp}^T (\mathbf{H}_{/p})^{-1} \mathbf{H}_{/pp})^{-1} = (\mathbf{H}^{-1})_{pp} \quad (\text{A.22})$$

Therefore

$$\begin{aligned} 1 - y_p (\mathbf{w}_p^T \mathbf{x}_p + b_p) &= \min_{\lambda} \alpha_p \boldsymbol{\lambda}^T (\mathbf{K} + \gamma^{-1} \mathbf{I}_n) \boldsymbol{\lambda} \\ &= \alpha_p \min_{\lambda_i} \{ \max_{\mu} (\tilde{K}_{pp} - 2\mathbf{H}_{/pp} \bar{\boldsymbol{\lambda}} + \bar{\boldsymbol{\lambda}}^T \mathbf{H}_{/p} \bar{\boldsymbol{\lambda}}) \} \\ &= \frac{\alpha_p}{(\mathbf{H}^{-1})_{pp}} \end{aligned}$$

and we complete the proof of Lemma 3.2.

Appendix B

B.1: Partial Derivatives of the Modified Leave-one-out Error with respect to Scaling Factors in Linear Kernel (Eq. (4.10))

As defined in Chapter 4, the modified leave-one-out error of LS-SVM is computed by:

$$mlooe = \frac{1}{n} \sum_{i=1}^n \frac{1}{1 + \exp(y_i f^i(\mathbf{x}_i))} \quad (4.8)$$

Further, we have

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^d v_k x_{ik} x_{jk} \quad (4.9)$$

where v_k is the scaling factor x_{ik} is the k th component of $\mathbf{x}_i$. And the following

denotations are used: $\mathbf{K} = \{k(\mathbf{x}_i, \mathbf{x}_j)\}$, $\mathbf{H} = \begin{bmatrix} \mathbf{K} + \gamma^{-1} \mathbf{I}_n & \mathbf{1}_n \\ \mathbf{1}_n^T & 0 \end{bmatrix}$, $\mathbf{\Omega} = \{y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)\}$

and $\bar{\mathbf{\Omega}} = \begin{bmatrix} \mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n & \mathbf{Y} \\ \mathbf{Y}^T & 0 \end{bmatrix}$.

Let $g_i = y_i f^i(\mathbf{x}_i) = 1 - \frac{\alpha_p}{(\mathbf{H}^{-1})_{pp}}$, the partial derivative of the modified leave-one-out

error with respect to any hyper-parameter t can be computed by

$$\begin{aligned} \frac{\partial mlooe}{\partial t} &= -\frac{1}{n} \sum_{i=1}^n \frac{\exp(g_i)}{[1 + \exp(g_i)]^2} \cdot \frac{\partial g_i}{\partial t} \\ &= \frac{1}{n} \sum_{i=1}^n \frac{\exp(g_i)}{[1 + \exp(g_i)]^2} \cdot \left[\frac{1}{(\mathbf{H}^{-1})_{ii}} \cdot \frac{\partial \alpha_i}{\partial t} - \frac{\alpha_i}{(\mathbf{H}^{-1})_{ii}^2} \cdot \frac{\partial (\mathbf{H}^{-1})_{ii}}{\partial t} \right] \\ &= \frac{1}{n} \sum_{i=1}^n \frac{\exp(g_i)}{[1 + \exp(g_i)]^2 (\mathbf{H}^{-1})_{ii}^2} \cdot [(\mathbf{H}^{-1})_{ii} \cdot \frac{\partial \alpha_i}{\partial t} + \alpha_i \cdot (\mathbf{H}^{-1} \frac{\partial \mathbf{H}}{\partial t} \mathbf{H}^{-1})_{ii}] \end{aligned}$$

Let $\bar{\mathbf{1}}_n = \begin{bmatrix} \mathbf{1}_n \\ 0 \end{bmatrix}$, we have $\alpha_i = (\bar{\mathbf{\Omega}}^{-1} \bar{\mathbf{1}}_n)_i$, hence

$$\frac{\partial \alpha_i}{\partial t} = -(\bar{\mathbf{\Omega}}^{-1} \frac{\partial \bar{\mathbf{\Omega}}}{\partial t} \bar{\mathbf{\Omega}}^{-1} \bar{\mathbf{1}}_n)_i$$

Appendix B

Since $\bar{\mathbf{\Omega}} = \begin{bmatrix} \mathbf{\Omega} + \gamma^{-1} \mathbf{I}_n & \mathbf{Y} \\ \mathbf{Y}^T & 0 \end{bmatrix}$ and $\mathbf{H} = \begin{bmatrix} \mathbf{K} + \gamma^{-1} \mathbf{I}_n & \mathbf{1}_n \\ \mathbf{1}_n^T & 0 \end{bmatrix}$, then $\frac{\partial \bar{\mathbf{\Omega}}}{\partial t} = \begin{bmatrix} \frac{\partial \mathbf{\Omega}}{\partial t} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix}$ and

$$\frac{\partial \mathbf{H}}{\partial t} = \begin{bmatrix} \frac{\partial \mathbf{K}}{\partial t} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix}.$$

Therefore, $\frac{\partial m_{looe}}{\partial t}$ can finally be computed as:

$$\begin{aligned} \frac{\partial m_{looe}}{\partial t} = & \frac{1}{n} \sum_{i=1}^n \frac{\exp(g_i)}{[1 + \exp(g_i)]^2 (\mathbf{H}^{-1})_{ii}^2} \cdot [\alpha_i \cdot (\mathbf{H}^{-1})_{ii} \begin{bmatrix} \frac{\partial \mathbf{K}}{\partial t} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} (\mathbf{H}^{-1})_{ii} \\ & - (\mathbf{H}^{-1})_{ii} \cdot (\bar{\mathbf{\Omega}}^{-1})_{ii} \begin{bmatrix} \frac{\partial \mathbf{\Omega}}{\partial t} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} \bar{\mathbf{\Omega}}^{-1} \mathbf{1}_n] \end{aligned}$$

Replace t with the scaling factor v_k , we can get the Eq. (4.10).

B.2: The .632+ Bootstrap Method for Model Assessment

Firstly studied by Efron [90], The bootstrap method is a general tool for estimating the generalization error [88,91]. Suppose we have n training patterns, bootstrap basically re-use the training data to generate a set of bootstrap samples. For each sample, n patterns are randomly drawn *with replacement* from the training set for training, and the patterns that are not chosen are kept for testing. This is done for K times and K bootstrap samples are produced. Then, a given model M (in our case, a feature selection algorithm followed by a classifier) is applied to the bootstrap samples as well as the original training set for evaluation.

The B.632+ of a model is computed by the following steps: First, the leave-one-out bootstrap error $B1$ is estimated. To define $B1$ more precisely, we define a quantity R_{ik} , where i is the index of pattern and k is the index of bootstrap sample. For the k th bootstrap sample, if the pattern $\mathbf{x}_i$ is kept for testing and is misclassified, then $R_{ik} = 1$. Otherwise, R_{ik} is zero. Then the R_{ik} for each pattern is averaged over all

the bootstrap samples as:

$$E_i = \frac{\sum_{k=1}^K I_{ik} R_{ik}}{\sum_{k=1}^K I_{ik}} \quad (\text{B.1})$$

where I_{ik} is one if the pattern $\mathbf{x}_i$ is kept for testing in the k th bootstrap sample and is zero otherwise. Then the (Monte Carlo) estimate of $B1$ on the basis of the K bootstrap samples is given by:

$$B1 = \sum_{i=1}^n \frac{E_i}{n} \quad (\text{B.2})$$

Typically, $B1$ is based on $\approx 0.632n$ of the original patterns, and it was confirmed that $B1$ is upwardly biased. Thus, Efron proposed the .632 estimator:

$$B.632 = 0.368 \times AE + 0.632 \times B1 \quad (\text{B.3})$$

where AE is computed by applying the model to the whole training set. Based on the B.632 error, the B.632+ estimate is proposed as [88]:

$$B.632+ = (1 - w)AE + wB1 \quad (\text{B.4})$$

where the weight w is given by:

$$w = \frac{0.632}{1 - 0.368r} \quad (\text{B.5})$$

and where

$$r = \frac{B1 - AE}{\gamma - AE} \quad (\text{B.6})$$

is the relative over-fitting rate and γ is called the non-information error rate. By applying the model to the whole training set, γ can be estimated by

$$\gamma = \sum_{j=1}^C p_j (1 - q_j) \quad (\text{B.7})$$

where C is the number of classes, p_j is the proportion of the training patterns from class j and q_j is the proportion of training patterns that are classified to class j by the model.

Following the above-described steps, we can finally compute the B.632+ error.

Appendix C: Proof of Lemma 5.1

Given a set of support vectors $Z = \{\mathbf{z}_1, \dots, \mathbf{z}_l\}$, the entries of a $l \times l$ matrix $\mathbf{P}$ is

defined as $P_{jk} = \sum_{i=1}^n k(\mathbf{z}_j, \mathbf{x}_i)k(\mathbf{z}_k, \mathbf{x}_i) + k(\mathbf{z}_j, \mathbf{z}_k) / \gamma$, $\Phi = [\sum_{i=1}^n k(\mathbf{z}_1, \mathbf{x}_i), \dots, \sum_{i=1}^n k(\mathbf{z}_l, \mathbf{x}_i)]^T$

and $\bar{\mathbf{c}} = [\sum_{i=1}^n y_i k(\mathbf{z}_1, \mathbf{x}_i), \dots, \sum_{i=1}^n y_i k(\mathbf{z}_l, \mathbf{x}_i), \sum_{i=1}^n y_i]^T$. Let $\bar{\mathbf{P}} = \begin{bmatrix} \mathbf{P} & \Phi \\ \Phi & n \end{bmatrix}$ and define J_s as

$J_s = \frac{\gamma}{2} (n - \bar{\mathbf{c}}^T \bar{\mathbf{P}}^{-1} \bar{\mathbf{c}})$, the Lemma 5.1 can be described as:

Lemma 5.1: For two support vector sets Z_1 and Z_2 , let $J_s(Z_1)$ and $J_s(Z_2)$ be the corresponding J_s 's. If $Z_1 \subseteq Z_2$, then $J_s(Z_1) \geq J_s(Z_2)$.

PROOF:

Given $Z_1 \subseteq Z_2$, we can assume that $Z_1 = \{\mathbf{z}_1, \dots, \mathbf{z}_l\}$ and $Z_2 = Z_1 \cup \{\mathbf{z}_{l+1}, \dots, \mathbf{z}_L\}$. As defined in Chapter 5, J_s is computed by minimizing the following term:

$$J_{\text{Sparse}} = \frac{1}{2} \sum_{j=1}^l \sum_{k=1}^l \beta_j \beta_k k(\mathbf{z}_j, \mathbf{z}_k) + \frac{\gamma}{2} \sum_{i=1}^n [1 - y_i \sum_{j=1}^l \beta_j k(\mathbf{z}_j, \mathbf{x}_i) - y_i b]^2$$

with respect to the Lagrange multipliers β_i 's.

Let $\beta_2 = [\beta_{21}, \dots, \beta_{2L}]^T$ be the vector of Lagrange multipliers optimized on Z_2 and $\beta_1 = [\beta_{11}, \dots, \beta_{1l}]^T$ be the vector of Lagrange multipliers optimized on Z_1 . We further extend β_1 as

$$\bar{\beta}_1 = [\beta_{11}, \dots, \beta_{1l}, \beta_{l(l+1)}, \dots, \beta_{1L}]^T$$

subject to $\beta_{li} = 0$ for $i > l$

Hence, β_2 and $\bar{\beta}_1$ can both be substitute into the term below:

$$J_{\text{Sparse}} = \frac{1}{2} \sum_{j=1}^L \sum_{k=1}^L \beta_j \beta_k k(\mathbf{z}_j, \mathbf{z}_k) + \frac{\gamma}{2} \sum_{i=1}^n [1 - y_i \sum_{j=1}^L \beta_j k(\mathbf{z}_j, \mathbf{x}_i) - y_i b]^2 \quad (\text{C.1})$$

It can be easily verified that:

$$J_S(Z_1, \beta_1) = J_S(Z_1, \bar{\beta}_1) = \frac{1}{2} \sum_{j=1}^L \sum_{k=1}^L \beta_{1j} \beta_{1k} k(\mathbf{z}_j, \mathbf{z}_k) + \frac{\gamma}{2} \sum_{i=1}^n [1 - y_i \sum_{j=1}^L \beta_{1j} k(\mathbf{z}_j, \mathbf{x}_i) - y_i b]^2$$

and
$$J_S(Z_2, \beta_2) = \frac{1}{2} \sum_{j=1}^L \sum_{k=1}^L \beta_{2j} \beta_{2k} k(\mathbf{z}_j, \mathbf{z}_k) + \frac{\gamma}{2} \sum_{i=1}^n [1 - y_i \sum_{j=1}^L \beta_{2j} k(\mathbf{z}_j, \mathbf{x}_i) - y_i b]^2$$

Since β_2 is the optimal solution that minimizes Eq. (C.1), we can directly get

$$J_S(Z_1) \geq J_S(Z_2)$$

Therefore, the proof of Lemma 5.1 is completed.

Appendix D

D.1: Derivations of Eqs. (6.15)-(6.20) and Proof of Lemma 6.1

The disagreement measure (Eq. (6.15))

As defined in Section 6.2, suppose that a pattern $\mathbf{x}_i$ is misclassified by l_i base classifiers and the other $L-l_i$ classifiers classify it correctly. Then for this pattern, there are $(L-l_i)l_i$ pairs of base classifiers whose oracle outputs are different. Hence,

the term $\sum_{j=1}^L \sum_{k=j+1}^L (n_{j,k}(1,-1) + n_{j,k}(-1,1))$ is equivalent to the term $\sum_{i=1}^N l_i(L-l_i)$, and we

can re-write Eq. (6.6) as Eq. (D.1).

$$dis = \frac{2}{NL(L-1)} \sum_{j=1}^L \sum_{k=j+1}^L (n_{j,k}(1,-1) + n_{j,k}(-1,1)) \quad (6.6)$$

$$\begin{aligned} dis &= \frac{2}{NL(L-1)} \sum_{i=1}^N l_i(L-l_i) \\ &= \frac{2}{N(L-1)} \sum_{i=1}^N l_i - \frac{2}{NL(L-1)} \sum_{i=1}^N l_i^2 \end{aligned} \quad (D.1)$$

Since $P = 1 - \frac{\sum_{i=1}^N l_i}{NL}$, we have

$$\sum_{i=1}^N l_i = NL(1-P) \quad (D.2)$$

To maximize the diversity, dis need to be maximized. Substituting equation (D.2) into equation (D.1), we get

$$dis = \frac{2L(1-P)}{(L-1)} - \frac{2}{NL(L-1)} \sum_{i=1}^N l_i^2 \quad (6.15)$$

If P is regarded as a constant, the diversity maximization problem becomes a Lagrangian formulation. We are given the constrained maximization problem:

$$\begin{aligned} \max(dis) &= \frac{2L(1-P)}{(L-1)} - \frac{2}{NL(L-1)} \sum_{i=1}^N l_i^2 \\ \text{subject to } &\sum_{i=1}^N l_i = NL(1-P) \end{aligned}$$

Introducing the Lagrangian multiplier, we obtain

$$Ldis = \frac{2L(1-P)}{(L-1)} - \frac{2}{NL(L-1)} \sum_{i=1}^N l_i^2 + \lambda \cdot \sum_{i=1}^N l_i - \lambda \cdot NL(1-P) \quad (D.3)$$

Differentiating Eq. (D.3) with respect to l_i , we obtain

$$\frac{d(Ldis)}{d(l_i)} = -\frac{4l_i}{NL(L-1)} + \lambda = 0. \quad (D.4)$$

Hence, we have

$$l_i = \frac{\lambda NL(L-1)}{4}, \quad \forall i \quad (D.5)$$

which means

$$\lambda = \frac{1-P}{N(L-1)} \quad (D.6)$$

and

$$l_i = L(1-P), \quad \forall i \quad (6.21)$$

Therefore, Eq. (6.15) is maximized if and only if Eq. (6.21) is satisfied.

$$\max(dis) = \frac{2LP(1-P)}{L-1} \quad (D.7)$$

The double-fault measure (Eq. (6.16))

According to the definition, for each pattern $\mathbf{x}_i$, there exist $l_i(l_i-1)/2$ pairs of base

classifiers whose oracle outputs on $\mathbf{x}_i$ are -1. Since the term $\sum_{j=1}^L \sum_{k=j+1}^L n_{j,k}(-1,-1)$ is

equivalent to the term $\sum_{i=1}^N l_i(l_i-1)/2$, Eq. (6.8) can be directly re-written as Eq. (6.16).

$$DF = \frac{2}{NL(L-1)} \sum_{j=1}^L \sum_{k=j+1}^L n_{j,k}(-1,-1) \quad (6.8)$$

$$DF = \frac{1}{NL(L-1)} \sum_{i=1}^N l_i^2 - \frac{1-P}{L-1} \quad (6.16)$$

In this case, we need to minimize DF to maximize the diversity. Similar to the derivation for the disagreement measure, after solving the minimization problem by introducing a Lagrangian multiplier, we obtain:

$$\min(DF) = \frac{(1-P)(L-LP-1)}{L-1} \quad (D.8)$$

Appendix D

---

if and only if Eq. (6.21) is satisfied.

The Kohavi-Wolpert variance (Eq. (6.17))

From the derivations for the first two measures, it is easy to get Eq. (6.17).

$$\begin{aligned} KW &= \frac{1}{NL^2} \sum_{i=1}^N l_i(L-l_i) \\ &= 1 - P - \frac{1}{NL^2} \sum_{i=1}^N l_i^2 \end{aligned} \quad (6.17)$$

We need to maximize Eq. (6.17) to maximize the diversity. Then we can use exact the same technique that has been presented for the disagreement measure. Only when Eq. (21) is satisfied, the maximum KW can be achieved as:

$$\max(KW) = P(1 - P) \quad (D.9)$$

The measurement of inter-rater agreement (Eq. (6.18))

It can be derived directly from the definition of this measure that

$$\begin{aligned} Ir &= 1 - \frac{\sum_{i=1}^N (L-l_i)l_i}{NL(L-1)P(1-P)} \\ &= 1 - \frac{NL^2(1-P) - \sum_{i=1}^N l_i^2}{NL(L-1)P(1-P)} \\ &= \frac{LP - P - L}{LP - P} + \frac{\sum_{i=1}^N l_i^2}{NL(L-1)P(1-P)}. \end{aligned} \quad (6.18)$$

The maximum diversity can be achieved when Eq. (6.18) is minimized, which means to minimize the term $\sum_{i=1}^N l_i^2$. Similar to the former three cases, we only need to satisfy Eq. (6.21) and:

$$\min(Ir) = -\frac{1}{L-1}. \quad (D.10)$$

The generalized diversity (Eq. (6.19))

We first prove that the terms $\sum_{j=1}^L \frac{j}{L} T_j$ and $\sum_{j=1}^L \frac{j(j-1)}{L(L-1)} T_j$ are equivalent to the terms

$1 - P$ and $\frac{\sum_{i=1}^N l_i(l_i - 1)}{NL(L-1)}$ respectively.

Let $n(j)$ be the number of patterns misclassified by j base classifiers, then:

$$\sum_{j=0}^L n(j) = N \quad (\text{D.11})$$

$$T_j = \frac{n(j)}{N}. \quad (\text{D.12})$$

From the definition of l_i , we can get

$$\begin{aligned} \sum_{j=1}^L \frac{j}{L} T_j &= \sum_{j=1}^L \frac{j}{L} T_j + \frac{0}{L} T_0 \\ &= \sum_{j=0}^L \frac{n(j)j}{NL} \\ &= \frac{1}{NL} \sum_{j=0}^L \sum_{i=1, l_i=j}^N l_i \end{aligned} \quad (\text{D.13})$$

Since

$$\sum_{j=0}^L \sum_{i=1, l_i=j}^N l_i = \sum_{i=1}^N l_i \quad (\text{D.14})$$

We have

$$\sum_{j=1}^L \frac{j}{L} T_j = \frac{\sum_{i=1}^N l_i}{NL} = 1 - P \quad (\text{D.15})$$

Similarly

$$\begin{aligned} \sum_{j=1}^L \frac{j(j-1)}{L(L-1)} T_j &= \sum_{j=0}^L \frac{j(j-1)}{L(L-1)} T_j \\ &= \frac{1}{NL(L-1)} \sum_{j=0}^L j(j-1)n(j) \\ &= \frac{1}{NL(L-1)} \sum_{j=0}^L \sum_{i=1, l_i=j}^n l_i(l_i - 1) \\ &= \frac{\sum_{i=1}^N l_i(l_i - 1)}{NL(L-1)} \end{aligned} \quad (\text{D.16})$$

From Eqs. (D.15) and (D.16), Eq. (6.12) can finally be re-written as Eq. (6.19).

$$GD = 1 - \frac{\sum_{j=1}^L \frac{j(j-1)}{L(L-1)} T_j}{\sum_{j=1}^L \frac{j}{L} T_j} \quad (\text{6.12})$$

Appendix D

$$\begin{aligned}
 GD &= 1 - \frac{\sum_{j=1}^L \frac{j(j-1)}{L(L-1)} T_j}{\sum_{j=1}^L \frac{j}{L} T_j} \\
 &= 1 - \frac{\sum_{i=1}^N l_i(l_i-1)}{NL(L-1)(1-P)} \\
 &= 1 - \frac{\sum_{i=1}^N l_i(l_i-1)}{NL(L-1)(1-P)} \\
 &= \frac{L}{L-1} - \frac{\sum_{i=1}^N l_i^2}{NL(L-1)(1-P)} \tag{6.19}
 \end{aligned}$$

If and only if Eq. (6.21) is satisfied, the generalized diversity is maximized and

$$\max(GD) = \frac{LP}{L-1} \tag{D.17}$$

The measure of “Difficulty” (Eq. (6.20))

From the definition of this measure, we need to minimize the term:

$$\begin{aligned}
 diff &= \text{var}\left(\frac{L-l_i}{L}\right) \\
 &= \frac{1}{N} \sum_{i=1}^N \left(\frac{1}{NL} \sum_{j=1}^N l_j - l_i \right)^2 \\
 &= \frac{1}{NL} \sum_{i=1}^N l_i^2 - L(1-P)^2 \tag{6.20}
 \end{aligned}$$

It is obvious that only when Eq. (6.21) is satisfied

$$\min(diff) = 0 \tag{D.18}$$

Combining the derivations of all the six diversity measures, the proof of Lemma 6.1 can be readily completed.

D.2: Proof of Lemma 6.3 (The generalized version of Lemma 6.1)

In this section we show that our analysis can also be generalized to non-uniformly weighted case for the other diversity measures besides the disagreement measure.

Proof for Double-Fault measure

Similar to Eq. (6.27), Eq. (6.8) is re-defined for the non-uniformly weighted case as

$$DF = \frac{2}{NL(L-1)} \sum_{j=1}^L \sum_{k=j+1}^L n_{j,k}(-1, -1) \quad (6.8)$$

$$DF = \frac{2}{NL(L-1)} \sum_{j=1}^L \sum_{k=j+1}^L w_k w_j n_{j,k}(-1, -1) \quad (D.19)$$

Let $\mathbf{1}$ be an $N \times L$ matrix with all the entries as one. It is easy to get

$$\sum_{j=1}^L \sum_{k=j+1}^L w_k w_j n_{j,k}(-1, -1) = \frac{1}{2} \left(\frac{\mathbf{w}^T (\mathbf{O} - \mathbf{1})^T (\mathbf{O} - \mathbf{1}) \mathbf{w}}{4} - \sum_{j=1}^L w_j^2 N(1 - p_j) \right)$$

Since there is:

$$\sum_{i=1}^N l_i^2 = \frac{L^2 \mathbf{w}^T (\mathbf{O} - \mathbf{1})^T (\mathbf{O} - \mathbf{1}) \mathbf{w}}{4} \quad (D.20)$$

Eq. (D.19) can be re-written as:

$$DF = \frac{1}{NL^3(L-1)} \left(\sum_{i=1}^N l_i^2 - \sum_{j=1}^L w_j^2 L^2 N(1 - p_j) \right) \quad (D.21)$$

Again, by optimizing DF with respect to l_i , we can validate our analysis in the non-uniformly weighted case. If w_j is one for all base classifiers, Eq. (D.21) is almost the same as Eq. (6.16) except for an additional L^2 term in the denominator. Therefore, Eq. (6.16) is a special case of Eq. (D.21).

Proof for the Kohavi-Wolpert variance

Following the same rationale described in Section 6.2.1, we have

$$P(O = -1 | \mathbf{x}) = \sum_{O_{ij} = -1} w_j = \frac{l_i}{L} \quad \text{and} \quad P(O = 1 | \mathbf{x}) = \sum_{O_{ij} = 1} w_j = 1 - \frac{l_i}{L}$$

Appendix D

Then,

$$\begin{aligned} KW &= \frac{1}{2N} \sum_{i=1}^N \left(1 - \sum_{k=1}^C P(y = \omega_k | \mathbf{x})^2 \right) \\ &= \frac{1}{2N} \sum_{i=1}^N \left(1 - \frac{l_i^2}{L^2} - \left(1 - \frac{l_i}{L} \right)^2 \right) = \frac{1}{NL^2} \sum_{i=1}^N l_i (L - l_i) \end{aligned} \quad (\text{D.22})$$

Eq. (D.22) is exactly same as Eq. (6.10), hence our analysis is equivalent for both uniformly and non-uniformly weighted cases.

Proof for the measurement of inter-rater agreement and “Difficulty”

According to Eqs. (6.11) and (6.20), the definitions of these two measures directly include the term l_i and do not contain any additional constraints on $\mathbf{w}$. Therefore, Eqs. (6.11) and (6.20) are valid for both uniformly and non-uniformly weighted cases. Since Eq. (6.2) also remains unchanged in non-uniformly weighted case, it is obvious that any analysis based on l_i 's in uniformly weighted case can be generalized to non-uniformly weighted case for these two measures, and the Lemma 3 holds.

Proof for the generalized diversity

Because Eqs. (D.15) and (D.16) directly result in Eq. (6.19), to generalize Eq. (6.19) to non-uniformly weighted case, we only need to check whether Eqs. (D.15) and (D.16) still hold in non-uniformly weighted case. If the weights are non-uniform, l_i and j in the original definition will no longer only take integers from 0 to L , but they are still discontinuous. We define Ω as the set that contains all possible values of l_i . Then Eq. (D.11) can be re-written as (D.23) and equation (D.12) still holds.

$$\sum_{j \in \Omega} n(j) = N \quad (\text{D.23})$$

Further, Eqs. (D.13) and (D.14) can be modified as

$$\sum_{j \in \Omega} \frac{j}{L} T_j = \frac{1}{NL} \sum_{j \in \Omega} \sum_{i=1, l_i=j}^N l_i \quad (\text{D.24})$$

and

$$\frac{1}{NL} \sum_{j \in \Omega} \sum_{i=1, l_i=j}^N l_i = \sum_{i=1}^N l_i \quad (\text{D.25})$$

Appendix D

Combining Eqs. (D.24) and (D.25) with Eq. (6.2), we can prove that Eq. (D.15) still holds when the weights are non-uniform. Eq. (D.15) can be verified in a similar manner. Therefore, Eq. (6.19) also holds in non-uniformly weighted case and Lemma 3 is true for the generalized diversity measure.

Appendix E: List of Abbreviations

ES	Evolution Strategies
FFSVS	Floating Forward Support Vector Selection
GA	Genetic Algorithm
GAP-RBF	Growing And Pruning Radial Basis Function Neural Network
GGAP-RBF	Generalized GAP-RBF
GLGS	Gradient-based Leave-one-out Gene Selection
GP	Gaussian Process
GPSVG	Gradient-based Pseudo Support Vector Generation
GSVS	Greedy Support Vector Selection
KFDA	Kernel Fisher Discriminant Analysis
KRR	Kernel Ridge Regression
<i>looe</i>	Leave-One-Out Error
LOOC	Leave-One-Out Calculation
LOOCSFS	Leave-One-Out Calculation Sequential Forward Selection
LSBSFS	LS Bound Sequential Forward Selection
LSBSFFS	LS Bound Sequential Floating Forward Selection
LS-SVM	Least Squares Support Vector Machines
MAHSFS	Mahalanobis class separability Sequential Forward Selection
<i>mlooe</i>	Modified Leave-One-Out Error
MRAN	Minimal Resource-Allocating Network
NC	Negative Correlation
PCA	Principal Component Analysis
PSVM	Proximal Support Vector Machines
QP	Quadratic Programming
RAN	Resource-Allocating Network
RANEKF	Resource-Allocating Network via Extended Kalman Filter
RBFNN	Radial Basis Function Neural Network
RLSC	Regularized Least Squares Classifier

Appendix E

RN	Regularized Network
RVM	Relevance Vector Machine
SLMC	Sparse Large Margin Classifier
SKFDA.....	Sparse Kernel Fisher Discriminant Analysis
SPGP	Sparse Pseudo-input Gaussian Process
SV	Support Vector
SVM	Support Vector Machines
SVM-RFE.....	SVM Recursive Feature Elimination

Bibliography

- [1] R. Duda, P. Hart and D. Stork. *Pattern classification*. John Wiley & Sons, the second edition, 2000.
- [2] J. A. K. Suykens and J. Vandewalle, “Least squares support vector machines,” *Neural Processing Letters*, 9:293-300, 1999.
- [3] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific, Singapore, 2002.
- [4] C. Cortes and V. Vapnik, “Support vector networks,” *Machine Learning*, 20:273-297, 1995.
- [5] K. R. Müller, S. Mika, G. Rätsch, K. Tsuda and B. Schölkopf, “An introduction to kernel-based learning algorithms,” *IEEE Transactions on Neural Networks*, 12:181-202, 2001.
- [6] B. Schölkopf, A. J. Smola, and K. R. Müller, “Nonlinear component analysis as a kernel eigenvalue problem,” *Neural Computation*, 10:1299-1319, 1998.
- [7] C. Saunders, A. Gammerman and V. Vovk, “Ridge regression learning algorithm in dual variables,” In *Proceedings of the 15th International Conference on Machine Learning*, Morgan Kaufmann, 1998.
- [8] S. Mika, G. Rätsch, J. Weston, B. Schölkopf and K. R. Müller, “Fisher discriminant analysis with kernels,” *Neural Networks for Signal Processing IX*, 41-48. IEEE Press, New York, 1999.
- [9] P. -L. L and C. Fyfe, “Kernel and nonlinear canonical correlation analysis,” In *Proceedings of International Joint Conference on Neural Networks*, 2000.

- [10] A. Ruiz and P. E. Lopez-de-Teruel, "Nonlinear kernel-based statistical pattern analysis," *IEEE Transactions on Neural Networks*, 12:16-32, 2001.
- [11] F. Bach and M. Jordan, "Kernel independent component analysis," *Journal of Machine Learning Research*, 3:1-48, 2002.
- [12] C. H. Park and H. Park, "Nonlinear feature extraction based on centroids and kernel functions," *Pattern Recognition*, 37:801-810, 2004.
- [13] A. K. Qin and P. N. Suganthan, "Kernel neural gas algorithms with application to cluster analysis," In *Proceedings of the International Conference on Pattern Recognition*, Cambridge, UK, 2004.
- [14] G. Fung and O. L. Mangasarian, "Proximal support vector classifiers," In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 77-86, 2001.
- [15] T. Van Gestel, J. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor and J. Vandewalle, "Benchmarking least squares support vector machine classifiers," *Machine Learning*, 54:5-32, 2004.
- [16] M. Aizerman, E. Braverman and L. Rozonoer, "Theoretical foundations of the potential function method in pattern recognition learning," *Automation and Remote Control*, 25:821-837, 1964.
- [17] B. E. Boser, I. M. Guyon and V. N. Vapnik, "A training algorithm for optimal margin classifiers," In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 144-152, 1992.
- [18] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and beyond*. MIT Press, Cambridge, MA, 2001.

Bibliography

- [19] J. S. Taylor and N. Crisitanini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
- [20] R. Rifkin. *Everything Old Is New Again: A Fresh Look at Historical Approaches in Machine Learning*. PhD thesis, MIT, Cambridge, MA, 2002.
- [21] J. Xu, X. Zhang and Y. Li, “Kernel MSE algorithm: a unified framework for KFD, LS-SVM and KRR,” In *Proceedings of the International Joint Conference on Neural Networks*, pages 1486-1491, 2001.
- [22] T. Van Gestel, J. Suykens, G. Lanckriet, A. Lambrechts, B. De Moor and J. Vandewalle, “Bayesian framework for least squares support vector machine classifiers, Gaussian processes and kernel Fisher discriminant analysis,” *Neural Computation*, 14:1115-1147, 2002.
- [23] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.
- [24] V. N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.
- [25] G. Fung and O. L. Mangasarian, “Proximal support vector classifiers,” Technical report, Data Mining Institute, 2001.
- [26] G. Baudat and F. Anouar, “Generalized discriminant analysis using a kernel approach,” *Neural Computation*, 12:2385-2404, 2000.
- [27] R. Rifkin, G. Yeo and T. Poggio, “Regularized least-squares classification,” In *Advances in Learning Theory: Methods, Models and Applications*, volume 190 of *NATO Science Series III: Computer and Systems Sciences*, pages 131–153, IOS Press Amsterdam, 2003.
- [28] R.A. Fisher, “The use of multiple measurements in taxonomic problems,” *Annals of Eugenics*, 7:178-188, 1936.
- [29] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press,

- the second Edition, 1990.
- [30] T. Evgeniou, M. Pontil and T. Poggio, “Regularization networks and support vector machines,” *Advances in Computational Mathematics*, 13:1-50, 2000.
  - [31] C. K. L. Williams and C. E. Rasmussen, “Gaussian processes for regression,” *Advances in Neural Processing Systems* 8:598-604, MIT Press, Cambridge, MA, 1996.
  - [32] C. E. Rasmussen and C. K. L. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
  - [33] D.G. Krige, “A statistical approach to some basic mine valuation problems on the Witwatersrand,” *Journal of the Chemical, Metallurgical and Mining Society of South Africa*, 52(6):119–139, 1951.
  - [34] N. Cressie. *Statistics for Spatial Data*. John Wiley & Sons, New York, 1993.
  - [35] Y. Liu, X. Yao and T. Higuchi, “Evolutionary Ensembles with Negative Correlation Learning,” *IEEE Transactions on Evolutionary Computation*, 4:380-387, 2000.
  - [36] S. Nolfi, “Evolution and learning in neural networks,” In *The Handbook of Brain Theory and Neural Networks*, pages 415-418, MIT Press, Cambridge, the second edition, 2002.
  - [37] M. M. Islam, X. Yao and K. Murase, “A constructive algorithm for training cooperative neural network ensembles,” *IEEE Transactions on Neural Networks*, 14:820-834, 2003.
  - [38] C. Igel, S. Wiegand and F. Friedrichs, “Evolutionary optimization of neural systems: the use of self-adaptation,” In *Trends and Applications in Constructive Approximation, International Series of Numerical Mathematics*,

Bibliography

- volume 151, Birkhäuser Verlag, Basel, 2005.
- [39] D. R. Eads, D. Hill, S. Davis, S.J. Perkins, J. Ma, R. B. Porter, J. P. Theiler, “Genetic algorithms and support vector machines for time series classification,” In *Applications and Science of Neural Networks, Fuzzy Systems and Evolutionary Computation V, Proceedings of the SPIE*, vol. 4787, pages 74-85, 2002.
  - [40] H. Fröhlich, O. Chapelle and B. Schölkopf, “Feature selection for support vector machines by means of genetic algorithms,” In *Proceedings of the 15th IEEE International Conference on Tools with AI*, pages 142–148, 2003.
  - [41] M. T. Miller, A. K. Jerebko, J. D. Malley and R. M. Summers, “Feature selection for computer-aided polyp detection using genetic algorithms,” In *Medical Imaging 2003: Physiology and Function: Methods, Systems, and Applications, Proceedings of the SPIE*, vol. 5031, 2003, pp. 102–110.
  - [42] K. Jong, E. Marchiori and A. van der Vaart, “Analysis of proteomic pattern data for cancer detection,” In *Applications of Evolutionary Computing, Lecture Notes in Computer Science*, vol. 3005, pages 41–51, Springer, Berlin, 2004.
  - [43] F. Friedrichs and C. Igel, “Evolutionary tuning of multiple SVM parameters,” *Neurocomputing*, 64:107-117, 2005.
  - [44] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.
  - [45] H. -P. Schwefel. *Evolution and Optimum Seeking*, Sixth-Generation Computer Technology Series, John Wiley & Sons, New York, 1995.
  - [46] H. -G. Beyer. *The Theory of Evolution Strategies*. Springer, Berlin, 2001.
  - [47] H. -G. Beyer and H.-P. Schwefel, “Evolution strategies: a comprehensive

- introduction,” *Natural Computing*, 1:3–52, 2002.
- [48] M. Stone, “Cross-validatory choice and assessment of statistical predictions,” *Journal of the Royal Statistical Society*, 36:111-147, 1974.
- [49] M. Stone, “An asymptotic equivalence of choice of model by cross-validation and Akaike’s criterion,” *Journal of the Royal Statistical Society*, 39:44-47, 1977.
- [50] H. Allen, “The relationship between variable selection and data augmentation and a method of prediction,” *Technometrics*, 16:125-127, 1977.
- [51] A. Luntz and V. Brailovsky, “On estimation of characters obtained in statistical procedure of recognition (in Russian),” *Technicheskaya Kibernetika*, 3, 1969.
- [52] G. C. Cawley and N. L. C. Talbot, “Fast exact leave-one-out cross-validation of sparse least squares support vector machines,” *Neural Networks*, 17:1467-1475, 2004.
- [53] T. Van Gestel, B. Baesens, J. Suykens, M. Espinoza, D. Baestaens, J. Vanthienen and D. De Moor, “Bankruptcy prediction with least squares support vector machine classifiers,” In *Proceedings of the International Conference on Computational Intelligence for Financial Engineering*, pages 1-8, Hong Kong, China, 2003.
- [54] Y. Bengio, “Gradient-based optimization of hyper-parameters,” *Neural Computation*, 12:1889–1900, 2000.
- [55] O. Chapelle, V. Vapnik, O. Bousquet and S. Mukherjee, “Choosing multiple parameters for support vector machines,” *Machine Learning*, 46:131–159, 2002.
- [56] S.S. Keerthi, “Efficient tuning of SVM hyper-parameters using radius margin

Bibliography

- bound and iterative algorithms,” *IEEE Transactions on Neural Networks*, 13 (5):1225–1229, 2002.
- [57] K. -M. Chung, W. -C. Kao, C. -L. Sun and C.-J. Lin, “Radius margin bounds for support vector machines with RBF kernel,” *Neural Computation*, 15:2643–2681, 2003.
- [58] C. Gold and P. Sollich, “Model selection for support vector machine classification,” *Neurocomputing*, 55:221–249, 2003.
- [59] L. Bo, L. Wang and L. Jiao, “Feature scaling for kernel Fisher discriminant analysis using leave-one-out cross validation,” *Neural Computation*, 18:961-978, 2006.
- [60] X. Yao and Y. Liu, “Fast evolution strategies,” *Control and Cybernetics*, 26:467-496, 1997.
- [61] J. Jägersküpper, “Analysis of a simple evolutionary algorithm for minimization in Euclidian spaces,” In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science*, vol. 2719, pages. 1068–1079, Springer, Berlin, 2003.
- [62] G. Rätsch, T. Onoda and K. -R. Müller, “Soft margins for AdaBoost,” *Machine Learning*, 42:3287–320, 2001.
- [63] S. A. Billings and K. L. Lee, “Nonlinear Fisher discriminant analysis using a minimum squared error cost function and the orthogonal least squares algorithm,” *Neural Networks* 15:263-270, 2002.
- [64] U. von Luxburg, O. Bousquet and B. Schölkopf, “A compression approach to support vector model selection,” *Journal of Machine Learning Research*, 5:293-323, 2004.

- [65] T. Glasmachers and C. Igel, "Gradient-based adaptation of general Gaussian kernels," *Neural Computation*, 17:2099-2105, 2005.
- [66] U. Alon, N. Barkai, D. Notterman, K. Gish, S. Ybarra, D. Mack, A. Levine, "Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays," *Proceedings of the National Academy of Sciences of the USA*, 96:6745-6750, 1999.
- [67] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield and E. S. Lander, "Molecular classification of cancer: class discovery and class prediction by gene expression monitoring," *Science* 286:531-537, 1999.
- [68] N. Iizuka, M. Oka, H. Yamada-Okabe, M. Nishida, Y. Maeda, N. Mori, T. Takao, T. Tamesa, A. Tangoku, H. Tabuchi, K. Hamada, H. Nakayama, H. Ishitsuka, T. Miyamoto, A. Hirabayashi, S. Uchimura and Y. Hamamoto, "Oligonucleotide microarray for prediction of early intrahepatic recurrence of hepatocellular carcinoma after curative resection," *The Lancet* 361:923-929, 2003.
- [69] C. L. Nutt, D. R. Mani, R. A. Bentensky, P. Tamayo, J. G. Cairncross, C. Ladd, U. Pohl, C. Hartmann, M. E. McLaughlin, T. T. Batchelor, P. M. Black, A. Von Deimling, S. L. Pomeroy, T. R. Golub and D. N. Louis, "Gene expression-based classification of malignant gliomas correlates better with survival than histological classification," *Cancer Research*, 63:1602-1607, 2003.
- [70] C. Ambroise and G. J. McLachlan, "Selection bias in gene extraction on the basis of microarray gene expression data," *Proceedings of the National*

Bibliography

- Academy of Sciences of the USA*, 99:6562-6566, 2002.
- [71] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, 97:245-271, 1997.
- [72] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, 97:273-324, 1997.
- [73] I. Guyon, J. Weston, S. Barnhill and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine Learning* 46:389-422, 2002.
- [74] A. Rakotomamonjy, "Variable selection using SVM-based criteria," *Journal of Machine Learning Research*, 3:1357-1370, 2003.
- [75] P. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach*. Prentice Hall, London, 1982.
- [76] I. Tsamardinos and C. F. Aliferis, "Towards principled feature selection: relevance, filters and wrappers," In *the Ninth International Workshop on Artificial Intelligence and Statistics*, Key West, Florida, USA, 2003.
- [77] A. R. Webb. *Statistical Pattern Recognition*. John Wiley & Sons, London, 2002.
- [78] M. L. Raymer, W. F. Punch, E. D. Goodman, L. A. Kuhn and A. K. Jain, "Dimensionality reduction using genetic algorithms," *IEEE Transactions on Evolutionary Computation*, 4:164-171, 2000.
- [79] L. Li, W. Jiang, X. Li, K. L. Moser, Z. Guo, L. Du, Q. Wang, E. J. Topol, Q. Wang and S. Rao, "A robust hybrid between genetic algorithm and support vector machine for extracting an optimal feature gene subset," *Genomics*, 85:16-23, 2005.

- [80] T. Jirapech-Umpai and S. Aitken, "Feature selection and classification for microarray data analysis: evolutionary methods for identifying predictive genes," *BMC Bioinformatics* 6:148, 2005.
- [81] L. Li, T. A. Darden, C. R. Weinberg, A. J. Levine and L. G. Pedersen, "Gene assessment and sample classification for gene expression data using a genetic algorithm/K-nearest neighbor method," *Combinatorial Chemistry & High Throughput Screening*, 4:727-739, 2001.
- [82] K. Duan, J. C. Rajapakse, "A variant of SVM-RFE for gene selection in cancer classification with expression data," In *Proceedings of International Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, 2004.
- [83] N. Pochet, F. De Smet, J. Suykens and B. De Moor, "Systematic benchmarking of microarray data classification: assessing the role of non-linearity and dimensionality reduction," *Bioinformatics* 20:3185-3195, 2004.
- [84] X. Zhou and K. Z. Mao, "LS bound based gene selection for DNA microarray data," *Bioinformatics*, 21:1559-1564, 2005.
- [85] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, the third edition, 1996.
- [86] R. Simon, M. D. Radmacher, K. Dobbin and L. M. McShane, "Pitfalls in the use of DNA microarray data for Diagnostic and Prognostic Classification," *Journal of National Cancer Institute*, 95:14-18, 2003.
- [87] U. M. Braga-Neto and E. R. Dougherty. "Is cross-validation valid for small-sample microarray classification?" *Bioinformatics* 20:374-380, 2004.
- [88] B. Efron and R. Tibshirani, "Improvements on cross-validation: the 632+

Bibliography

- bootstrap method,” *Journal of the American Statistical Association*, 78:548-560, 1997.
- [89] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.
- [90] B. Efron, “Bootstrap methods: another look at the jackknife,” *Annals of Statistics*, 7:1-26, 1979.
- [91] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, London, 1993.
- [92] X. Li, S. Rao, Y. Wang and B. Gong, “Gene mining: a novel and powerful ensemble decision approach to hunting for disease genes using microarray expression profiling,” *Nucleic Acids Research*, 32:2685-2694, 2004.
- [93] G. C. Cawley and N. L. C. Talbot, “A greedy training algorithm for sparse least squares support vector machines,” In *Proceedings of the International Conference on Artificial Neural Networks*, Springer Lecture Notes in Computer Science (LNCS), vol. 2415, pages 681-686, Madrid, Spain, 2002.
- [94] G. C. Cawley and N. L. C. Talbot, “Efficient formation of a basis in a kernel induced feature space,” In *Proceedings of the European Symposium on Artificial Neural Networks* pages 1-6, Bruges, Belgium, 2002.
- [95] J. A. K. Suykens, L. Lukas, and J. Vandewalle, “Sparse least squares support vector machine classifiers,” In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 37-42, Bruges, Belgium, 2000.
- [96] J. A. K. Suykens, J. De Brabanter, L. Lukas and J. Vandewalle, “Weighted least squares support vector machines: robustness and sparse approximation,” *Neurocomputing, Special issue on fundamental and information processing*

- aspects of neurocomputing*, 48:85-105, 2002.
- [97] B. J. de Kruif and T. J. de Vries, “Pruning error minimization in least squares support vector machines,” *IEEE Transactions on Neural Networks*, 14(3): 696-702, 2003.
  - [98] L. Hoegaerts, J. Suykens, J. Vandewalle and B. De Moor, “A comparison of pruning algorithms for sparse least squares support vector machines,” In *Proceedings of the 11th International Conference on Neural Information Processing*, Calcutta, India, 2004.
  - [99] L. Hoegaerts. *Eigenspace Methods and Subset Selection in Kernel Based Learning*. PhD thesis, Faculty of Engineering, K.U.Leuven (Belgium), 2005.
  - [100] Y. Lu, N. Sundararajan and P. Saratchandran, “A Sequential Learning Scheme for Function Approximation using Minimal Radial Basis Function Neural Networks,” *Neural Computation*, 9:461-478, 1997.
  - [101] A. J. Smola and B. Schölkopf, “Sparse greedy matrix approximation for machine learning,” In *Proceedings of 17th International Conference on Machine Learning*, pages 911-918, 2000.
  - [102] G. Baudat and F. Anouar, “Kernel-based methods and function approximation,” In *Proceedings of the International Joint Conference on Neural Networks*, pages 1244–1249, 2001.
  - [103] A. J. Smola and P. Bartlett, “Sparse greedy gaussian process regression” In *Advances in Neural Information Processing Systems 14*, pages 619–625, MIT Press, 2001.
  - [104] M. Seeger, C. K. I. Williams and N. Lawrence, “Fast forward selection to speed up sparse Gaussian process regression,” In *Workshop on AI and Statistics*

Bibliography

- 9, 2003.
- [105] P. Sun and X. Yao, "Greedy forward selection algorithms to Sparse Gaussian Process Regression," In *Proceedings of the International Joint Conference on Neural Networks*, 2006.
  - [106] B. Natarajan, "Sparse approximate solutions to linear systems," *SIAM Journal of Computing*, 25(2):227-234, 1995.
  - [107] D. Zongker and A. K. Jain, "Algorithms for feature selection: an evaluation," In *Proceedings of the International conference on Pattern Recognition*, pages 18-22, 1996.
  - [108] S. Chen, S. A. Billings and W. Luo, "Orthogonal least squares methods and their application to non-linear system identification," *International Journal of Control*, 50 (5):1873-1896, 1989.
  - [109] P. Sun, "Sparse kernel least squares classifier," In *Proceeding of the 4th IEEE International Conference on Data Mining*, pages 539-542, Brighton, UK, 2004.
  - [110] P. Pudil, F. J. Ferri, J. Novovicova and J. Kittler, "Floating Search methods in feature selection with nonmonotonic criterion function," In *Proceedings of the International conference on Pattern Recognition*, volume 2, pages 279-283, 1994.
  - [111] P. Pudil, J. Novovicova and J. Kittler, "Floating search methods in feature selection," *Pattern Recognition Letters*, 15:1119-1125, 1994.
  - [112] J. Q. Candela. *Learning with Uncertainty - Gaussian Processes and Relevance Vector Machines*. PhD thesis, Technical University of Denmark, 2004.
  - [113] S. Keerthi and W. Chu, "A matching pursuit approach to sparse Gaussian process regression," In *Advances in Neural Information Processing Systems 18*,

-
- The MIT Press, Cambridge, MA, 2006.
- [114] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," In *Advances in Neural Information Processing Systems 18*, Cambridge, MA, The MIT Press, 2006.
  - [115] M. Wu, B. Schölkopf and G. Bakir, "Building sparse large margin classifiers," In *Proceedings of the 22th International Conference on Machine Learning*, pages 1001–1008, ACM, 2005.
  - [116] M. Wu, B. Schölkopf, and G. Bakir, "A direct method for building sparse kernel learning algorithms," *Journal of Machine Research*, 7:603-624, 2006.
  - [117] J. Platt, "A resource-allocating network for function interpolation," *Neural Computation*, 3:213–225, 1991.
  - [118] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural Computation*, 5:954–975, 1993.
  - [119] Y. Lu, N. Sundararajan and P. Saratchandran, "Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm," *IEEE Transactions on Neural Networks*, 9(2):308–318, 1998.
  - [120] G. -B. Huang, P. Saratchandran and N. Sundararajan, "An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks," *IEEE Transactions on SMC part B*, 34(6):2284-2292, 2004.
  - [121] G. -B. Huang, P. Saratchandran and N. Sundararajan, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Transactions on Neural Networks*, 16(1):57--67, 2005.
  - [122] M. E. Tipping, "Sparse Bayesian learning and the relevance vector machine,"

Bibliography

- Journal of Machine Learning Research*, 1:211–244, 2001.
- [123] P. B. Nair, A. Choudhury and A. J. Keane, “Some greedy learning algorithms for sparse regression and classification with Mercer kernels,” *Journal of Machine Learning Research*, 3:781–801, 2002.
- [124] Y. Freund, “Boosting a weak learning algorithm by majority,” *Information and Computation*, 121(2):256–285, 1995.
- [125] A. Krogh, “Neural networks ensembles, cross validation, and active learning,” In *Advances in Neural Information Processing Systems 7*, The MIT Press, Cambridge, MA, 1995.
- [126] L. Breiman, “Bagging predictors,” *Machine Learning*, 24:123-140, 1996.
- [127] Y. Freund and R. Schapire, “Experiments with a new boosting algorithm,” In *Proceedings of the 13th International Conference on Machine Learning*, pages 148-156. Morgan Kaufmann, 1996.
- [128] L. Breiman, “Arcing classifiers,” *The Annals of Statistics*, 26:801-849, 1998.
- [129] T. Ho, “The random space method for constructing decision forests,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832-844, 1998.
- [130] Y. Liu and X. Yao, “Ensemble learning via negative correlation,” *Neural Networks*, 12(10):1399-1404, 1999.
- [131] L. Breiman, “Random forests,” *Machine Learning*, 45:5-32, 2001.
- [132] T. Dietterich, “An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization,” *Machine Learning*, 40:1-22, 2000.
- [133] L. Kuncheva and C. Whitaker, “Measures of diversity in classifier ensembles

-
- and their relationship with the ensemble accuracy,” *Machine Learning*, 51: 181-207, 2003.
- [134] L. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley, 2004.
- [135] T. Evgeniou, L. Perez-Breva, M. Pontil, and T. Poggio, “Bounds on the generalization performance of kernel machine ensembles,” In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 271–278. Morgan Kaufmann, 2000.
- [136] H. -C. Kim, S. Pang, H. -M. Je, D. Kim and S. Y. Bang, “Constructing support vector machine ensemble,” *Pattern Recognition*, 36(12):2757-2767, 2003.
- [137] G. Valentini and T. G. Dietterich, “Bias-variance analysis of support vector machines for the development of SVM-based ensemble,” *Journal of Machine Learning Research*, 5:725-775, 2004.
- [138] B. Sun and D. Huang, “Least squares support vector machine ensemble,” In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2013 – 2016, 2004.
- [139] L. Kuncheva, “That elusive diversity in classifier ensembles,” In *Proceedings of IbPRIA 2003, Lecture Notes in Computer Science*, vol. 2652, pages 1126–1138, *Mallorca, Spain*, 2003.
- [140] E. Bauer and R. Kohavi, “An empirical comparison of voting classification algorithms: Bagging, boosting, and variants,” *Machine Learning*, 36:105-142.
- [141] R. E. Schapire, Y. Freund, P. L. Bartlett and W. S. Lee, “Boosting the margin: a new explanation for the effectiveness of voting methods,” *Annals of Statistics*, 26(5):1651–1686, 1998.

Bibliography

- [142] L. Mason, P. L. Bartlett and J. Baxter, “Improved generalization through explicit optimization of margins,” *Machine Learning*, 38:243-255, 2000.
- [143] D. Skalak, “The sources of increased accuracy for two proposed boosting algorithms,” In *Proceedings of American Association for Artificial Intelligence, Integrating Multiple Learned Models Workshop*, 1996.
- [144] G. Giacinto and F. Roli, “Design of effective neural network ensembles for image classification processes,” *Image Vision and Computing*, 19:699-707, 2001.
- [145] R. Kohavi and D. Wolpert, “Bias plus variance decomposition for zero-one loss functions,” In *Proceedings of the 13th International Conference on Machine Learning*, pages 275–283, Morgan Kaufmann, 1996.
- [146] J. Fleiss. *Statistical Methods for Rates and Proportions*. John Wiley & Sons, 1981.
- [147] D. Patridge and W. J. Krzanowski, “Software diversity: Practical statistics for its measurement and exploitation,” *Information & Software Technology*, 39:707-717, 1997.
- [148] L. Hansen and P. Salamon, “Neural network ensembles,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.
- [149] C. J. C. Burges. “A tutorial on support vector machines for pattern recognition,” *Data Mining and Knowledge Discovery*, 2:121-167, 1998.
- [150] D. Margineantu and T. Dietterich, “Pruning adaptive boosting,” In *Proceedings of the 14th International Conference on Machine Learning*, pages 211-218, Los Altos, CA, 1997.
- [151] C. Tamon and J. Xiang. “On the boosting pruning problem,” In *Proceedings of*

- the 11th European Conference on Machine Learning, Lecture Notes in Computer Science*, Vol. 1810, pages 404-412, Springer, 2000.
- [152] G. Brown, J. Wyatt, R. Harris and X. Yao. "Diversity creation methods: a survey and categorization," *Information Fusion Journal (Special issue on Diversity in Multiple Classifier Systems)*, 6(1):5-20, 2005.
- [153] G. Brown, J. Wyatt and P. Tino, "Managing diversity in regression ensembles," *Journal of Machine Learning Research*, 6:1621-1650, 2005.
- [154] J. Weston and C. Watkins, "Support vector machines for multiclass pattern recognition," In *Proceedings of the 7th European Symposium on Artificial Neural Networks*, 1999.
- [155] J. A. K. Suykens and J. Vandewalle, "Multiclass least squares support vector machines," In *Proceedings of the International Joint Conference on Neural Networks*, 1999.
- [156] J. Platt, N. Cristianini and J. S. Taylor, "Large margin dags for multiclass classification," In *Advances in Neural Information Processing Systems 12*, The MIT Press, Cambridge, MA, 2000.
- [157] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multi-class support vector machines," *IEEE Transactions on Neural Networks*, 13: 415-425, 2002.
- [158] V. Vural and J. G. Dy, "A hierarchical method for multiclass support vector machines," In *Proceedings of the 21th International Conference on Machine Learning*, ACM, 2004.
- [159] X. Zhu, "Semi-supervised learning literature survey," Computer Sciences Technical Report 1530, University of Wisconsin-Madison.
- [160] O. Chapelle. *Semi-supervised Learning*. The MIT Press, 2006.