Studies of Model Selection and Regularization for Generalization in Neural Networks with Applications

GUO, Ping

Supervisor: Professor Michael R. Lyu

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

in Computer Science and Engineering

November 2001

Acknowledgement

To the many people who have helped me in my life and my research work, I must thank them. In particular, my heartfelt thanks to:

First I would like to thank my supervisor Professor Michael R. Lyu for giving me the opportunity to join his research group. He provided technical training, constructive criticism, guidance, and inspiration. He also taught me to organize my thoughts and communicate them in a coherent manner. I also thank Professor I. King, for giving me some useful suggestion and special comments on parts of my research work.

I would like to thank all the members of the research group for being good friends, thanks them for many useful discussions and lots of fun times.

I would also like to express my deep appreciation to Professor K. S. Leung, the Chairman of Department of Computer Science and Engineering, and Professor L. W. Chan, the head of graduate division at Department of Computer Science and Engineering, for their help and encouragement during my study toward to finishing this thesis.

Thanks are also due to the staff in the Department of Computer Science and Engineering for their assistance and advice, and my other friends and colleagues for their help and understanding while I was completing this thesis.

Finally, I am grateful to my wife for her understanding, encouragement and support during difficult times. I would like also to thank my parents and my daughter for their love and lifelong support.

Abstract

This thesis investigates the generalization problem in artificial neural networks, attacking it from two major approaches: regularization and model selection.

On the regularization side, under the framework of Kullback–Leibler divergence for feedforward neural networks, we develop a new formula for the regularization parameter in Gaussian density kernel estimation based on available training data sets. Experiments show that the estimated regularization parameter is valid for most cases. With the derived formula, all sample data sets can be used to estimate the smoothing parameter, which is less computationally expensive than using the leave-one-out cross-validation method. Furthermore, the new covariance matrix estimation formula is suitable for small sample data with high dimension setting in the regularized Gaussian classifier case.

On the model selection side, both theory and extensive experiments are conducted for investigating the Bayesian Ying-Yang (BYY) model selection criterion to determine the cluster number in small sample-size cases. We derive new formula for estimating the smoothing parameters with proper approximations in the Smoothed Expectation-Maximum (SEM) technique. Experimental results show that with improved mixture model parameters, the BYY model selection criterion performance is enhanced.

From the model selection viewpoint, generalization can also be improved by combining several nets to form ensemble networks. The relationship between Mixture of Experts (ME) and the ensemble networks is established in this thesis. In an approximation where ME reduces to ensemble neural nets, the ensemble nets can be globally optimized instead of being individual members. A new method is consequently proposed to average

ensemble networks in the parameter space.

The stacked generalization method provides a way of combining trained networks altogether. This approach uses partitioning of the data set to find an overall system which improves generalization performance. In order to investigate this scheme effectively, we develop a new learning algorithm called Pseudoinverse Learning algorithm (PIL). The efficiency of the PIL algorithm is demonstrated through several experiments.

Two applications are investigated in detail for model selection. The first application is in the image processing area. Automatic determination of the region number in image segmentation is a step to high level understanding and interpretation of an image by machine. A model selection criterion can be applied to determine the region number if the number of segments to be yielded is equal to the number of clusters in the image feature space. Experimental results show that with the model selection criterion, we can determine a reasonable region number for automatic image segmentation without *a priori* knowledge.

Another application is in the software reliability modeling area. We investigate the use of the mixture model analysis as a tool for early prediction of fault-prone program modules. The EM algorithm is engaged to build the model. By employing only software size and complexity metrics, this technique facilitates the development of an unsupervised model for predicting software quality without the prior knowledge of the number of faults in the modules. The technique is successful in classifying software into fault-prone and non fault-prone modules with a relatively low error rate, thus providing a reliable indicator for software quality prediction.

模型選擇與調整對神經網絡的歸納能力及其應用之研究

摘要

本論文探討人工神經網絡中的歸納能力問題。對改進歸納能力，有兩種主要的方法：模型選擇與調整，本論文包括了這兩方面的研究工作。

對前饋神經網絡，我們證明了系統熵用高斯分布函數表述這一種特殊情況下，在最大似然學習網絡參數時，可退化到一階 Tikhonov 調整器。調整參數等同于密度核估計的平滑參數，可基于訓練數據用新發展的公式來估計。實驗結果顯示了用新方法估計的調整參數與校驗方法估計的是同一數量級的。

在 Kullback-Leibler 信息度量的框架下，對調整的高斯分類器在高維小樣本情況下我們推出了新的協方差矩陣估計公式。同時還發展了一有效的平滑參數估計公式，實驗發現所用的近似在大多數情況下是適用的。採用所得到的公式，全部樣本可用於估計平滑參數，而且比用留一交叉校驗方法計算強度要小。

從理論和實驗上探討了小數量樣本情況下用 BYY 模型選擇確定類別數目的問題。在一些適當近似下，我們導出了新的估計平滑參數的公式。實驗結果表明，採用 Bootstrap 或 SEM 技術來估計混合模型參數，可以改善 BYY 模型選擇準則的準確率。

結合多個神經網絡構成組合網絡也可以改進歸納能力。本論文還建立了組合網絡與專家混合(ME)的關係。在特殊情況下，即軟−極大函數與輸入變量無關時，ME 退化到組合神經網絡。在這種近似下，對組合網絡是全局優化而不是對組合網絡中的各個網絡單獨優化。對組合網絡的權重平均系數也可用 EM 算法求得。此外，還提出了一種平均參數空間來組合網絡的新方法。

用劃分的數據可訓練一系列網絡，而 *堆疊歸納* 則提供了把這些網絡結合在一起的方法，這通常可改進整個系統的歸納能力。為了有效地探討這種技術，我們發展了一種新的學習算法(PIL)。實驗結果表明 PIL 算法有較高的效率。

自動確定要分割圖象的區域數目是通往較高級別上機器理解與解釋圖象其中的一步。當我們假定要分割得區域數目等於圖象特征空間的類別數目時，可用某些模型選擇準則來確定區域數目。實驗結果表明在大多數情況下，用模型選擇準則可選取合理的區域數目，這使在沒有先驗知識時自動分割圖象成為可能。

使用混合模型分析對早期預測易於出錯的程序模塊也進行了探討。僅採用軟件尺寸和複雜性度量，即使在沒有關於模塊中缺陷數目的先驗知識的情況下，用 EM 算法也可建立模型去預測軟件質量。此外，Akaike 信息準則可用於選擇模型數目，這個模型數目被認為是程序模塊應被分類的類別數目。我們所採用的技術成功地把軟件劃分成易於出錯和非易於出錯的模塊，而且劃分的相對誤差較低，這給出一種軟件質量預測的較可靠的指標。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Generalization in Artificial Neural Network

In recent years neural network computing has emerged as a practical technology, with successful applications in many fields. It is widely acknowledged that successful applications of neural computing require a systematic approach. The principle of neural learning research has experienced an explosive period and many theoretical issues have been studied and clarified [1, 2, 3, 4, 5].

In essence, almost all of the neural network applications can be summarized as follows: Given a set of randomly generated data, and a family of neural networks all sharing a common "architecture", construct a neural network from this family, that best approximates the data with high probability [6]. Stated in this form, the problem can be considered as nonlinear curve-fitting or nonlinear regression. What distinguishes the use of neural networks for this limited purpose, as opposed to many other standard techniques, is the widespread belief that "neural networks can generalize". In other words, it is believed that, after a neural network has been "trained" on a sufficiently large number of input-output pairs in a supervised learning manner, it can then correctly predict all future input-output pairs, even for those inputs that the network has not seen previously. It is shown that *perfect* generalization by a neural network is an impossibility. Rather, all that one can aspire to is that, after a sufficient amount of training, the trained neural network

can predict the correct output *with high probability* on a randomly selected test input.

Neural Networks, like other flexible nonlinear estimation methods such as kernel regression and smoothing splines, can suffer from either underfitting or overfitting [7]. Therefore, they exhibit poor generalization performance in these cases. A network that is not sufficiently complex can fail to detect fully the signal in a complicated data set, leading to underfitting. On the other hand, a network that is too complex may fit the noise, not just the signal, leading to overfitting. Overfitting is especially dangerous because it can easily lead to predictions that are far beyond the range of the training data in many common types of networks. Overfitting can also produce wild predictions in multilayer perceptrons even with noise-free data. The degree to which overfitting may happen is related to the number of training patterns and the number of parameters in the model. In general, with a fixed number of training patterns overfitting can occur when the model has too many parameters.

The best way to avoid overfitting is to use lots of training data. While in some real-world cases, it is impossible to obtain large enough number of training data. Given a fixed amount of training data, there are two main approaches to avoid underfitting and overfitting, and hence getting good generalization: model selection and regularization. Model selection is to select the model which "best explains" the given data from a set of models; Regularization is the procedure of allowing parameters bias towards what are considered to be more plausible values, which reduces the variance of the estimates at the cost of introducing bias. In the article of Geman *et. al* [8], a more rigorous approach on the trade-off between bias and variance is discussed. The statistical bias is the difference between the average value of an estimator and the correct value. Underfitting produces excessive bias in the outputs, whereas overfitting produces excessive variance.

In the literature, there exists some research work related to model selection or regularization, for examples, Moody [9] regarding weight decay and Weigend [10] regarding early stopping. Weight decay [9] and early stopping [10, 11] are the most popular meth-

ods of regularization. Combining networks [12] can be categorized as a special case of model selection, which selects all models to form ensemble networks. To estimate generalization error, Bartlett [13] obtains learning-theory results in which generalization error is related to the L_1 norm of the weights instead of the Vapnik-Chervonenkis (VC) dimension [14, 15]. But some problems still need to be studied in details, for examples, model selection criterion performance and regularization parameter estimation. The goal of this thesis is to investigate the generalization problem in both unsupervised and supervised learning cases. In addition to exploring the regularization in feedforward neural networks model and in Gaussian mixture model under the framework of the Kullback-Leibler divergence, this thesis investigate many other topics, including the Beyesian Ying-Yang (BYY) model selection criterion performance in small number samples case, the generalization with ensemble networks, and model selection in some practical applications for automatic image segmentation as well as software reliability engineering.

Figure 1.1 shows the thesis overview. More details are described as the following.

1.2 Thesis Overview

Chapter 2: Under the framework of the Kullback-Leibler divergence, we prove that one particular case of the system entropy with Gaussian probability density and kernel density estimation reduces into the first order Tikhonov regularizer when conducting the maximum likelihood learning for the network parameters for feedforward neural networks. The regularization parameter is the smooth parameter in kernel density estimation, which can be estimated by a newly derived formula. The formula is developed for online approximate estimation of the regularization parameter using training data. Experiments show that the estimated regularization parameter is the same order as that estimated by the validation method. The similarity and difference of the obtained results with other's work are also discussed.

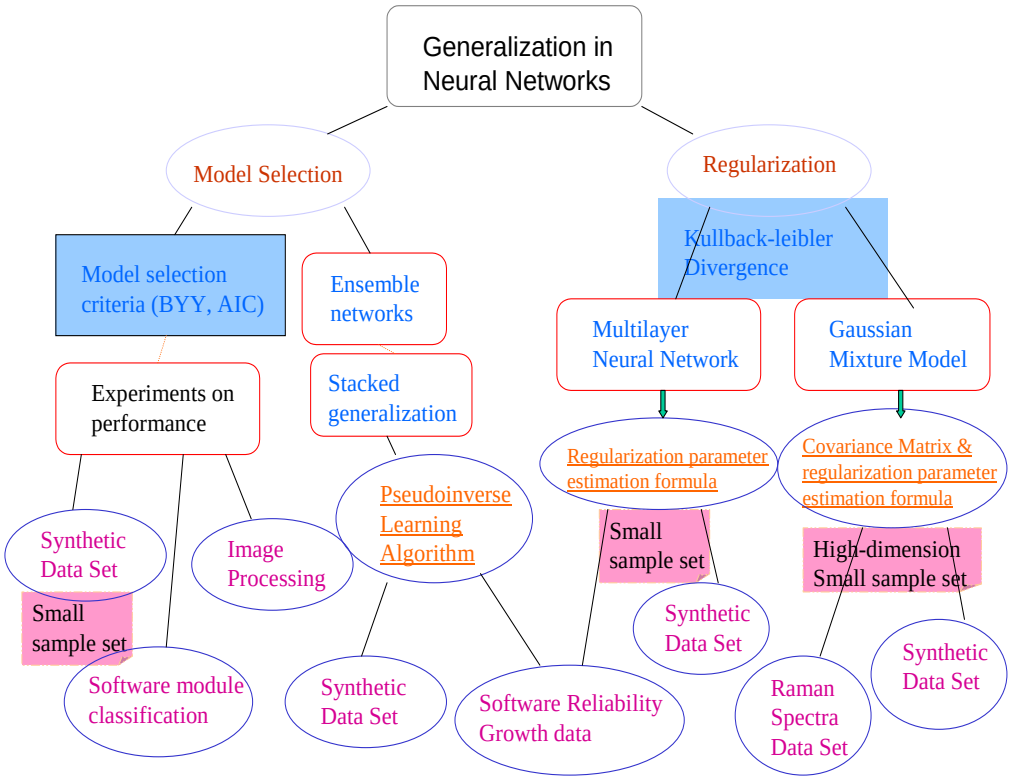


Figure 1.1: The thesis overview.

Chapter 3: For small sample with high dimension setting in Gaussian classification case, if the dimension d of variable $\mathbf{x}$ is comparable to the number of training samples n_j in class j, the problem becomes poorly-posed. Even worse, if the number n_j of training samples is less than the dimensionality, the problem becomes ill-posed. In this case, not all parameters can be properly estimated and classification accuracy is degraded. To solve these problems, one of the method is regularization. Under the framework of Kullback–Leibler information measure (divergence), the new covariance matrix estimation formula with regularized term is developed. An efficient smoothing parameter approximation formula is derived too, and the approximation is found from experiments to be valid for most cases. With Kullback–Leibler information measure, all samples can be used to estimate the smoothing parame-

ter without the need of partitioning data set into training and validation samples, which is less computation-expensive than using the leave-one-out cross-validation method.

Chapter 4: In this chapter, we describe the results of investigating the BYY data smoothing theory in the finite Gaussian mixture model case. Both theory and intensive experimental work are done for investigating BYY model selection for determining the cluster number in small number samples case. Taylor expansion approximation is used to approximate the integration in the cost functions. A new formula for estimating smoothing parameter h is derived under a proper approximation. Experimental results show that with Bootstrap or Smoothed EM technique estimated mixture model parameters, the BYY model selection criterion performance is improved.

Chapter 5: Generalization can be improved by combining neural networks as well. The relationship between the Mixture of Experts (ME) and the ensemble networks is established in this chapter. As a special case that soft-max function is independent of input variables, the ME reduces to ensemble neural networks. With this approximation, it is a global optimization of the ensemble networks instead of individual members. Simultaneously, the weighting average coefficient for the ensemble networks can be obtained through the EM-like algorithm. Experiments show that the ME is more general and powerful model than the ensemble networks in parameter estimation with maximum likelihood learning. Besides, by using a learning methodology to avoid networks falling into the different local minima, we make it possible to overcome the difficulty of averaging the ensemble networks in the parameter space. Experimental results show that the adopted strategy is efficient to improve network performance with finite training samples and the ensemble network architecture is much simpler than that in the functional space.

Chapter 6: The method of *stacked generalization* provides a way of combining trained networks together, which uses partitioning of the data set to find an overall system with improved generalization performance. However, this approach requires to train a lot of networks for level-1 training samples, which is very computation-time consuming when using back propagation algorithm to perform the required task. In order to efficiently investigate the performance of the stacked generalization, we develop a new learning algorithm called Pseudoinverse Learning Algorithm (PIL) for feedforward neural networks. The algorithm is based on generalized linear algebraic methods, and it adopts matrix inner products and pseudoinverse operations. Incorporating with a network architecture of which the number of hidden layer neurons is equal to the number of examples to be learned, the algorithm eliminates learning error by adding hidden layers and gives an exact solution (Perfect Learning). Unlike gradient descent algorithms, the PIL is a feed-forward only, fully automated algorithm, including no critical user-dependent parameters such as learning rate or momentum constant. The experimental results show the efficiency of the PIL algorithm.

Chapter 7: The model selection can be applied to image segmentation applications. Automatically determining the region number in an image segmentation is a step to higher level understanding and interpretation of an image by a machine. The BYY model selection criterion can be applied to determine the region number when we assume that the number of segments to be yield is equal to the number of clusters in an image feature space. The influence of the color space selection on region number determination is experimentally explored. Experimental results indicate that with the BYY model selection criterion, in most cases we can select the reasonable region number as long as the proper color space is selected. This approach makes it possible for automatically segmenting a given image without *a priori* knowledge.

Chapter 8: The use of the mixture model analysis as a tool for early prediction of fault-prone program modules is investigated in this chapter. The EM algorithm is engaged to build the model. By employing only software size and complexity metrics, this technique can develop a model for predicting software quality without the prior knowledge of the number of faults in the software modules. In addition, Akaike Information Criterion (AIC) is employed to select the model number, which is assumed to be the number of classes the program modules should be classified into. The technique is successful in classifying software into fault-prone and non fault-prone modules with a relatively low error rate, providing a reliable indicator for software quality prediction.

Chapter 2

Regularization: Feedforward Neural Networks Case

2.1 Introduction

It is well known that the goal of training neural networks is not to learn an exact representation of the training data itself, but rather to build a statistical model of the process which generates the data. In practical application of the feedforward neural networks, if the network is over-fitting to the noise on the training data, especially for the small number training samples case, it will give poor generalization. To control an appropriate complexity of the network can improve generalization. There are two main approaches for this purpose: model selection and regularization. Model selection for a feedforward neural network requires choosing the number of hidden neurons and thereof connection weights. The common statistical approach to model selection is to estimate the generalization error for each model and to choose the model minimizing this error [16, 17]. Regularization involves constraining or penalizing the solution of the estimation problem to improve network generalization ability by smoothing the predictions [18, 19]. Most common regularization methods include weight decay [20] and addition of artificial noise to the inputs during training [21, 22].

Regularization method is widely used for smoothing output [23, 24]. A value of the regularization parameter is determined by using the statistical techniques such as cross-

validation [25], booststrapping [26], and Bayesian method [27]. Most work uses a validation set to select the regularization parameter [28, 29, 30, 31]. This requires to split a given data set into training and validation sets. The optimal selection of the regularization parameter on the validation set sometimes depends on how to partition the data set. For a small number data set, we usually use leave-one-out cross-validation method. However, a recent study shows that cross-validation performance is not always good in the selection of linear models [32].

In this chapter, under the framework of the Kullback-Leibler divergence, we show that a particular case of the system entropy with Gaussian probability density and kernel density estimation for feedforward neural networks reduces into the first order Tikhonov regularizer. The smoothing parameter in the kernel density function plays the role of the regularization parameter. Under some approximation, an estimation formula can be derived for estimating the regularization parameter based on training data set. There is a lot of research work in smoothing parameter estimation of kernel density function; however, in this chapter we only focus on comparing the obtained result with maximum a *posteriori* (MAP) framework [27]. Experimental results show that the new derived estimation formula works well in the sparse and small training sample case.

2.2 System Probability Function

When given a data set $D = \{\mathbf{x}_i, \mathbf{z}_i\}_{i=1}^{N}$, we consider that the data can be modelled by a probability function. At one particular architecture design, we can let the kernel density of the given data set be $p_h(\mathbf{x}, \mathbf{z})$, and the network function mapping is denoted as a joint probability function $p(\mathbf{x}, \mathbf{z})$ on the data set D. The relative entropy or Kullback-Leibler divergence for this particular system denoted by $J(h, \Theta)$ cost function, where Θ stands for a parameter vector, then the quantity of interest is the "distance" of these two probability densities, which can be measured as follows [33, 34],

$$\begin{aligned} J(h,\Theta) &= \iint p_h(\mathbf{x},\mathbf{z}) \ln \frac{p_h(\mathbf{x},\mathbf{z})}{p(\mathbf{x},\mathbf{z})} d\mathbf{x}d\mathbf{z} \\ &= -\iint p_h(\mathbf{x},\mathbf{z}) \ln p(\mathbf{z}|\mathbf{x}) d\mathbf{x}d\mathbf{z} \\ &\quad + \iint p_h(\mathbf{x},\mathbf{z}) \ln \frac{p_h(\mathbf{x},\mathbf{z})}{p_0(\mathbf{x})} d\mathbf{x}d\mathbf{z} \\ &= J_1(h,\Theta) + J_2(h). \end{aligned} \tag{2.1}$$

(For simplicity, unless specified, the lower value is $-\infty$, the upper value is ∞ for those integrals involve in probability functions in the thesis.)

Where we use the notation and Bayes theorem,

$$p(\mathbf{x},\mathbf{z}) = p(\mathbf{z}|\mathbf{x},\Theta)p_0(\mathbf{x}). \tag{2.2}$$

$p(\mathbf{z}|\mathbf{x},\Theta)$ is a parameter conditional probability and $p_0(\mathbf{x})$ is a prior probability function.

$$J_1(h,\Theta) \equiv -\iint p_h(\mathbf{x},\mathbf{z}) \ln p(\mathbf{z}|\mathbf{x},\Theta) d\mathbf{x}d\mathbf{z} \tag{2.3}$$

is related to network parameter vector Θ, and smoothing parameter $h = \{h_x, h_z\}$,

$$\begin{aligned} J_2(h) &\equiv \iint p_h(\mathbf{x},\mathbf{z}) \ln p_{h0}(\mathbf{x},\mathbf{z}) d\mathbf{x}d\mathbf{z}, \\ p_{h0}(\mathbf{x},\mathbf{z}) &\equiv \frac{p_h(\mathbf{x},\mathbf{z})}{p_0(\mathbf{x})} \end{aligned} \tag{2.4}$$

only related to the smoothing parameter h.

We can assign a prefixed kernel function $K(\cdot)$ and smoothing parameters h_x, h_z for nonparametric density estimation [35, 36] of $p_h(\mathbf{x},\mathbf{z})$ for a given discrete training data set D, where the kernel density function [36] is

$$\begin{aligned} p_{h_x}(\mathbf{x}) &= \frac{1}{N}\sum_{\mathbf{x}_i \in D} K_{h_x}(\mathbf{x}-\mathbf{x}_i), \\ K_{h_x}(\mathbf{x}-\mathbf{x}_i) &= \frac{1}{h_x^d} K(\frac{\mathbf{x}-\mathbf{x}_i}{h_x}) \end{aligned} \tag{2.5}$$

Note N represents the number of samples in the data set D, d is the dimension of a random variable x, and the joint distribution $p_h(\mathbf{x}, \mathbf{z})$ in this work is designed as

$$p_h(\mathbf{x}, \mathbf{z}) = \frac{1}{N}\sum_{x_i, z_i \in D} K_{h_x}(\mathbf{x}-\mathbf{x}_i) K_{h_z}(\mathbf{z}-\mathbf{z}_i). \tag{2.6}$$

The mostly used kernel density function is Gaussian kernel,

$$K_h(r) = G(r, 0, h\mathbf{I}_d) = \frac{1}{(2\pi h)^{d/2}} \exp\{-\frac{||r||^2}{2h}\}. \tag{2.7}$$

In the kernel density function, $\mathbf{I}_d$ is a $d \times d$ dimensional identity matrix. In this chapter, we use $\{d_x, d_z\}$ to represent the dimension of input x and output z vector, respectively.

According to the principle of minimum description length (MDL) [37, 38], the best model class for a set of observed data is the one whose representative permits the shortest coding of the data, then the system should be optimized with optimal or *ideal* codelength. The parameter h_x, h_z should be chosen with minimized Kullback–Leibler divergence function based on the given data set according to,

$$\{h_x, h_z\} = \arg\min_h J(h, \Theta^*), \tag{2.8}$$

where Θ^* stands for learned parameter and $J(h, \Theta)$ is represented by Eq. (2.1).

In the following we will discuss the regularization problem with a finite training data set D.

2.3 Tikhonov Regularizer

When estimating network parameter by Maximum Likelihood (ML) learning, we minimize the function $J(h, \Theta)$ to find the network parameter Θ with a fixed parameter h. For a particular design, the conditional probability function can be written in the form

$$p(\mathbf{z}|\mathbf{x}, \Theta) = p(\mathbf{z}|f(\mathbf{x}, \Theta)). \tag{2.9}$$

where $f(\mathbf{x}, \Theta)$ is a function of input variable x and parameter Θ.

In the network parameter learning procedure, only J_1 is involved because J_2 dose not contain the parameter Θ.

To evaluate the function J_1, one of the techniques is the well-known *Monte Carlo integration* [39, 40]. In the *Monte Carlo integration* approximation, when substituting Eqs. (2.6) and (2.9) into Eq. (2.3), integration can be approximated by summation, and we obtain

$$J_1(h, \Theta) = -\frac{1}{N'} \sum_{i=1}^{N'} \ln p(\mathbf{z}'_i|f(\mathbf{x}'_i, \Theta)), \tag{2.10}$$

where

$$\mathbf{x}'_i = \mathbf{x}_i + e_x, \qquad \mathbf{z}'_i = \mathbf{z}_i + e_z. \tag{2.11}$$

e_x, e_z are data points drawn from distribution $p_h(\mathbf{x}, \mathbf{z})$. In this case, $J_1(h, \Theta)$ is equivalent to a negative likelihood function of the system.

In the *Monte Carlo integration* approximation, we need to generate a number of data set, which is very computation-intensive.

Another method is the Taylor expansion approximation for an integral, which we use

in this chapter,

$$J_1(h,\Theta) = -\iint p_h(\mathbf{x},\mathbf{z}) \ln p(\mathbf{z}|f(\mathbf{x},\Theta)) d\mathbf{x} d\mathbf{z}. \tag{2.12}$$

When we consider one special case, $p(\mathbf{z}|f(\mathbf{x},\Theta)) = G(\mathbf{z}, g(\mathbf{x},W), \sigma^2\mathbf{I}_{d_z})$ is Gaussian density function,

$$G(\mathbf{z}, g(\mathbf{x},W), \sigma^2\mathbf{I}_{d_z}) = \frac{1}{(2\pi\sigma^2)^{d_z/2}} \exp[-\frac{1}{2\sigma^2}||\mathbf{z} - g(\mathbf{x},W)||^2]$$

$$\begin{aligned} J_1(h,\Theta) &= -\iint p_h(\mathbf{x},\mathbf{z}) \ln G(\mathbf{z}, g(\mathbf{x},W), \sigma^2\mathbf{I}_{d_z}) d\mathbf{x} d\mathbf{z} \\ &= \iint p_h(\mathbf{x},\mathbf{z})[\frac{1}{2\sigma^2}||\mathbf{z} - g(\mathbf{x},W)||^2] d\mathbf{x} d\mathbf{z} + \frac{d_z}{2} \ln 2\pi\sigma^2. \end{aligned} \tag{2.13}$$

In this case, $\Theta = \{k, \sigma^2, W\}$ stands for a network parameter set, and $g(\mathbf{x},W)$ is a neural network mapping function. For example, in three-layer feedforward neural network with k hidden neurons case,

$$g(\mathbf{x},W) = S(\sum W_{z|y} S(\sum W_{y|x}\mathbf{x})) \tag{2.14}$$

$W = \{W_{z|y}, W_{y|x}\}$ is a network weight parameter vector, $W_{y|x}$ is a $d_x \times k$ matrix which connects the input space R_x and the hidden space R_y, $W_{z|y}$ is a $k \times d_z$ matrix which connects the hidden space R_y and the output space R_z. $S(\cdot)$ is a sigmoidal function,

$$S(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{x}}}. \tag{2.15}$$

Eq. (2.13) will result in the traditional sum-square-errors function in maximum likelihood learning case at the limit of $h \to 0$, when we omit some factors which are irrelevant to the network weight parameter W.

Based on consideration of that random noise is added to the input data only during training, Bishop [41] proved that in ML estimation case, Eq. (2.10) can be reduced to the first order Tikhonov regularizer [42] for feedforward neural network with approximations.

On the other hand, as we know, the input data points can be modelled as samples drawn from a delta distribution function $\delta(\mathbf{x} - \mathbf{x}_i)$. Intuitional speaking, when $h \to 0$, kernel density function $p_h(\mathbf{x})$ becomes a δ function. If adding random noise to the input data, the data distribution now can be described empirically by a density distribution function $p_\sigma(\mathbf{x})$ with σ controlling the noise level. Moreover, $p_\sigma(\mathbf{x})$ can take the similar function form to kernel density function $p_h(\mathbf{x})$. So addition of random noise to the input data is equivalent to smoothing in kernel density estimation, thus we can also obtain the Tikhonov regularizer directly from Eq. (2.12).

Let $f(\mathbf{x}, \mathbf{z}, W) = ||\mathbf{z} - g(\mathbf{x}, W)||^2$, $f(\mathbf{x}, \mathbf{z}, W)$ be a scale function of vector variable $\mathbf{x}$ and $\mathbf{z}$. When we expand $f(\mathbf{x}, \mathbf{z}, W)$ as a Taylor series in powers of $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_i$, $\Delta\mathbf{z} = \mathbf{z} - \mathbf{z}_i$ and denote $f'(\mathbf{x}_i, \mathbf{z}, W) = \nabla_x f(\mathbf{x}_i, \mathbf{z}, W)$. When taking only up to the second order term, then we obtain

$$\begin{aligned} f(\mathbf{x}, \mathbf{z}, W) &\approx f(\mathbf{x}_i, \mathbf{z}_i, W) + (f'_x)^T \Delta\mathbf{x} + \frac{1}{2}(\Delta\mathbf{x})^T f''_x \Delta\mathbf{x} \\ &\quad + (\Delta\mathbf{x})^T f''_{x,z} \Delta\mathbf{z} + (f'_z)^T \Delta\mathbf{z} + \frac{1}{2}(\Delta\mathbf{z})^T f''_z \Delta\mathbf{z} \end{aligned} \tag{2.16}$$

Eq. (2.13) becomes

$$\begin{aligned} J_1(h, \Theta) &= \iint p_h(\mathbf{x}, \mathbf{z})[\frac{1}{2\sigma^2} f(\mathbf{x}, \mathbf{z}, W)] d\mathbf{x} d\mathbf{z} + \frac{d_z}{2} \ln 2\pi\sigma^2 \\ &\approx \frac{1}{2N\sigma^2} \sum_{i=1}^{N} \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) [f(\mathbf{x}_i, \mathbf{z}_i, W) \\ &\quad + (f'_x)^T \Delta\mathbf{x} + \frac{1}{2}(\Delta\mathbf{x})^T f''_x \Delta\mathbf{x} + (f'_z)^T \Delta\mathbf{z} + (\Delta\mathbf{x})^T f''_{x,z} \Delta\mathbf{z} \\ &\quad + \frac{1}{2}(\Delta\mathbf{z})^T f''_z \Delta\mathbf{z}] d\mathbf{x} d\mathbf{z} + \frac{d_z}{2} \ln 2\pi\sigma^2 \end{aligned} \tag{2.17}$$

Notice that for any density function, the integration in the whole space should be equal to one, i.e.,

$$\iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) d\mathbf{x} d\mathbf{z} = 1 \tag{2.18}$$

$$\begin{aligned} & \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) f(\mathbf{x}_i, \mathbf{z}_i, W) d\mathbf{x} d\mathbf{z} \\ = \quad & f(\mathbf{x}_i, \mathbf{z}_i, W) = ||\mathbf{z}_i - g(\mathbf{x}_i, W)||^2 \end{aligned} \tag{2.19}$$

For Gaussian type function integrals[1], we can obtain

$$\begin{aligned} \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) [(f'_x)^T \Delta\mathbf{x} + (f'_z)^T \Delta\mathbf{z}] d\mathbf{x} d\mathbf{z} &= 0 \\ \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) [(\Delta\mathbf{x})^T f''_{x,z} \Delta\mathbf{z}] d\mathbf{x} d\mathbf{z} &= 0 \end{aligned} \tag{2.20}$$

$$\begin{aligned} & \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) [\frac{1}{2} (\Delta\mathbf{x})^T f''_x \Delta\mathbf{x}] d\mathbf{x} d\mathbf{z} \\ = \quad & \frac{h_x}{2} \text{trace}[f''_x] = h_x \{||g'(\mathbf{x}, W)||^2 - ||[\mathbf{z}_i - g(\mathbf{x}_i, W)] g''(\mathbf{x}_i, W)||\} \end{aligned} \tag{2.21}$$

$$\begin{aligned} & \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) [\frac{1}{2} (\Delta\mathbf{z})^T f''_z \Delta\mathbf{z}] d\mathbf{x} d\mathbf{z} \\ = \quad & \frac{h_z}{2} \text{trace}[f''_z] = d_z h_z \end{aligned} \tag{2.22}$$

With the above results, the integration becomes

$$\begin{aligned} J_1(h, \Theta) &= \iint p_h(\mathbf{x}, \mathbf{z}) [\frac{1}{2\sigma^2} f(\mathbf{x}, \mathbf{z}, W)] d\mathbf{x} d\mathbf{z} + \frac{d_z}{2} \ln 2\pi\sigma^2 \\ &\approx \frac{1}{2N\sigma^2} \sum_{i=1}^{N} \{||\mathbf{z}_i - g(\mathbf{x}_i, W)||^2 + \\ &\quad h_x [||g'(\mathbf{x}, W)||^2 - ||(\mathbf{z}_i - g(\mathbf{x}_i, W)) g''(\mathbf{x}_i, W)||]\} \\ &\quad + h_z \frac{d_z}{2\sigma^2} + \frac{d_z}{2} \ln 2\pi\sigma^2 \end{aligned} \tag{2.23}$$

Because the term $h_z d_z / 2\sigma^2$ in the above equation is not implicitly related to the network weight parameter W, we can omit this term in weight parameter learning. This also

[1]For mathematical method of Gaussian integrals, the reader is referred to Appendix B in Bishop's book[21].

illustrates that smoothing on output cannot improve network generalization, thus we can let $h_z \to 0$ without loss of generality. The last term in the above equation is irrelevant to the weight parameter, and can be neglected too [21]. Now the equation becomes

$$\begin{aligned} J_1(h,\Theta) &\approx \frac{1}{2N\sigma^2}\sum_{i=1}^{N}\{||\mathbf{z}_i - g(\mathbf{x}_i,W)||^2 + \\ &\quad h_x[||g'(\mathbf{x},W)||^2 - ||(\mathbf{z}_i - g(\mathbf{x}_i,W))g''(\mathbf{x}_i,W)||]\} \end{aligned} \tag{2.24}$$

Rewrite the equation in the form

$$J_1 \approx J_s + h_x J_r \tag{2.25}$$

where

$$\begin{aligned} J_s &= \frac{1}{2N\sigma^2}\sum_{i=1}^{N}||\mathbf{z}_i - g(\mathbf{x}_i,W)||^2 \\ J_r &= \frac{1}{2N\sigma^2}\sum_{i=1}^{N}[||g'(\mathbf{x}_i,W)||^2 - ||(\mathbf{z}_i - g(\mathbf{x}_i,W))g''(\mathbf{x}_i,W)||] \end{aligned} \tag{2.26}$$

In the above equation, J_s represents the traditional sum-square-error function, while J_r stands for a regularization term.

In Eq. (2.26), the second derivative term is the Hessian term. Reed [43] described it as an approximate measure of the difference between the average surrounding values and the precise value of the filed at a point, and assumed to be zero. While Bishop [41, 44] considered that when minimizing the cost function, the second term in J_r involving the second derivatives of the network function $g(x,W)$ vanishes to $\mathcal{O}(h_x)$. For sufficiently small values of the smooth parameter h_x, this leads to

$$\begin{aligned} J_1 &\approx J_s + h_x J_r \\ &= \frac{1}{2N\sigma^2}\sum_{i=1}^{N}\{||\mathbf{z}_i - g(\mathbf{x}_i,W)||^2 + h_x||g'(\mathbf{x}_i,W)||^2\} \end{aligned} \tag{2.27}$$

From the above we can easily see that under some approximation discussed above, one special case $J(h, \Theta)$ function is reduced to the first order Tikhonov regularizer in the sense of maximum likelihood learning.

Furthermore, from the above results it is easy to see that the parameter h_x controls the degree of smoothness of the network mapping, just the same as the problem of controlling the degree of smoothing in nonparametric estimation. The optimum value of h_x is problem-dependent. Using the traditional sum-square-error function can not select this parameter completely with a given data set; it needs to use separated training and validation data sets, and to be optimized by the cross-validation method or another validation data set.

In the next section we develop a formula to estimate this regularization coefficient based on the training data set.

2.4 Estimation of Regularization Parameter

When $h \neq 0$, according to the principle of MDL, the regularization coefficient h can be estimated according to Eq. (2.8) with the minimized KL distance.

In implementation, we can give a fixed h_x value, run optimizing algorithm such as Back-Propagation to obtain a series of network parameter Θ^*, then give another h_x value, so on and so forth. We choose h_x^* such that its corresponding value of $J(h_x^*, \Theta^*)$ is the smallest. This is an exhaustive search method, which is computation expensive, but it can give an exact solution for regularization parameter. From practical implementation consideration, in the following we will derive the formula which is approximately the estimation regularization parameter based on training data in the network parameter learning processing.

For some problems, e.g. function mapping, in special cases we can assume that $p_0(x)$ is a uniformly distributed function and regard it as h independent. With this assumption, from Eq. (2.1) with respect to $\frac{\partial}{\partial h_x} J(h, \Theta) = 0$, we can obtain the formula for estimating

regularization parameter.

To find the minimization of Eq. (2.1) corresponding to h_x, we conduct the following derivation. Considering $J_1(h,\Theta)$ approximation (Eq. (2.27)), from Eq. (2.1) we obtain,

$$\begin{aligned}\frac{\partial}{\partial h_x}J(h,\Theta) &= \frac{\partial}{\partial h_x}J_1(h,\Theta)+\frac{\partial}{\partial h_x}J_2(h)\\ &\approx J_r+\frac{\partial}{\partial h_x}J_2(h).\end{aligned} \tag{2.28}$$

From Eq. (2.4), when $J_2(h)$ is a continuous and differentiable function, the last term of the above equation becomes

$$\frac{\partial}{\partial h_x}J_2(h)=\iint\frac{\partial p_h(\mathbf{x},\mathbf{z})}{\partial h_x}[1+\ln p_h(\mathbf{x},\mathbf{z})]d\mathbf{x}d\mathbf{z} \tag{2.29}$$

While it can be proved that

$$\iint\frac{\partial p_h(\mathbf{x},\mathbf{z})}{\partial h_x}d\mathbf{x}d\mathbf{z}=0, \tag{2.30}$$

Proof. Because the joint kernel density $p_h(\mathbf{x},\mathbf{z})$ in this work is designed as Gaussian kernel function,

$$p_h(\mathbf{x},\mathbf{z})=\frac{1}{N}\sum_{i=1}^{N}G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z}). \tag{2.31}$$

We can compute the partial derivative of $p_h(\mathbf{x},\mathbf{z})$,

$$\frac{\partial}{\partial h_x}p_h(\mathbf{x},\mathbf{z})=-\frac{d_x}{2h_x}p_h(\mathbf{x},\mathbf{z})+\frac{1}{2Nh_x^2}[\sum_{i}^{N}G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})||\mathbf{x}-\mathbf{x}_i||^2] \tag{2.32}$$

$$\begin{aligned}\iint\frac{\partial p_h(\mathbf{x},\mathbf{z})}{\partial h_x}d\mathbf{x}d\mathbf{z} &= -\frac{d_x}{2h_x}\iint p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z} \qquad (2.33)\\ &\quad+\frac{1}{2Nh_x^2}\iint\sum_{i}^{N}G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})||\mathbf{x}-\mathbf{x}_i||^2d\mathbf{x}d\mathbf{z}\end{aligned}$$

The first term in the above equation is

$$\begin{aligned} -\frac{d_x}{2h_x}\iint p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z} &= -\frac{d_x}{2Nh_x}\sum_i^N \iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})d\mathbf{x}d\mathbf{z} \\ &= -\frac{d_x}{2h_x}. \end{aligned} \tag{2.34}$$

As the second term is also Gaussian type integration, it can be evaluated to

$$\begin{aligned} &\frac{1}{2Nh_x^2}\iint \sum_i^N G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})||\mathbf{x}-\mathbf{x}_i||^2 d\mathbf{x}d\mathbf{z} \\ = &\frac{d_x}{2h_x}. \end{aligned} \tag{2.35}$$

Then we have

$$\iint \frac{\partial p_h(\mathbf{x},\mathbf{z})}{\partial h_x}d\mathbf{x}d\mathbf{z} = -\frac{d_x}{2h_x}+\frac{d_x}{2h_x} = 0. \tag{2.36}$$

□

With the above results, Eq. (2.29) reduces to

$$\frac{\partial}{\partial h_x}J_2(h) = \iint \frac{\partial p_h(\mathbf{x},\mathbf{z})}{\partial h_x}\ln p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z} \tag{2.37}$$

That is,

$$\begin{aligned} \frac{\partial}{\partial h_x}J_2(h) &= -\frac{d_x}{2h_x}\iint p_h(\mathbf{x},\mathbf{z})\ln p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z} \\ &\quad +\frac{1}{2Nh_x^2}\sum_i^N \iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x}) \\ &\quad \times G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})||\mathbf{x}-\mathbf{x}_i||^2\ln p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z} \end{aligned} \tag{2.38}$$

For parameter optimization, the gradient descent rule becomes [45]

$$\delta h_x = -\frac{\partial J(h,\Theta)}{\partial h_x}. \tag{2.39}$$

When minimizing $J(h,\Theta)$ with respect to h_x , the following equation can be obtained

$$\delta h_x = -J_r + \frac{d_x}{2h_x} E_a(h), \tag{2.40}$$

or let $\delta h_x = 0$, we get

$$h_x = \frac{d_x E_a(h)}{2J_r} \tag{2.41}$$

where

$$\begin{aligned} E_a(h) &= \iint p_h(\mathbf{x},\mathbf{z}) \ln p_h(\mathbf{x},\mathbf{z}) d\mathbf{x} d\mathbf{z} \\ &- \frac{1}{N d_x h_x} [\sum_i^N \iint G(\mathbf{x},\mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z},\mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\ &\times ||\mathbf{x}-\mathbf{x}_i||^2 \ln p_h(\mathbf{x},\mathbf{z}) d\mathbf{x} d\mathbf{z}. \end{aligned} \tag{2.42}$$

This is a formula for estimating regularization parameter based on training data. It can be used to optimize h_x iteratively. The integration in the above equation can be evaluated by *Monte Carlo integration*.

In practical implementation, especially for the small training data set case, we can use sparse data approximation in Eq. (2.42). That is, if data i is not correlated with data j for sparse data distribution, we can consider integration at $\mathbf{x}$ around $\mathbf{x}_i$, $\mathbf{z}$ around $\mathbf{z}_i$ only, and ignore other data. With this approximation, now let us evaluate the integration in $E_a(h)$, in which

$$\begin{aligned} &\iint p_h(\mathbf{x},\mathbf{z}) \ln p_h(\mathbf{x},\mathbf{z}) d\mathbf{x} d\mathbf{z} \\ = &\frac{1}{N} \sum_{i=1}^N \{ \iint G(\mathbf{x},\mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z},\mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\ &\times \ln \sum_{j=1}^N G(\mathbf{x},\mathbf{x}_j, h_x \mathbf{I}_{d_x}) G(\mathbf{z},\mathbf{z}_j, h_z \mathbf{I}_{d_z}) d\mathbf{x} d\mathbf{z} \} - \ln N \end{aligned} \tag{2.43}$$

Applying sparse data approximation (SDA) and considering small h, we obtain,

$$\begin{aligned} & G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})\ln\sum_{j=1}^{N}G(\mathbf{x},\mathbf{x}_j,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_j,h_z\mathbf{I}_{d_z}) \\ \approx\ & G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})\{\ln G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})\} \qquad (2.44) \\ =\ & G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})\{-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h_x}-\frac{||\mathbf{z}-\mathbf{z}_i||^2}{2h_z} \\ & -\frac{d_x}{2}\ln(2\pi h_x)-\frac{d_z}{2}\ln(2\pi h_z)\} \end{aligned}$$

The integration is reduced to

$$\begin{aligned} & \iint p_h(\mathbf{x},\mathbf{z})\ln p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z} \\ \approx\ & -\frac{d_x}{2}[1+\ln(2\pi h_x)]-\frac{d_z}{2}[1+\ln(2\pi h_z)]-\ln N \qquad (2.45) \end{aligned}$$

$$\begin{aligned} & \frac{1}{Nd_xh_x}[\sum_i^N\iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})||\mathbf{x}-\mathbf{x}_i||^2\ln p_h(\mathbf{x},\mathbf{z})]d\mathbf{x}d\mathbf{z} \\ \approx\ & \frac{1}{Nd_xh_x}\sum_i^N\iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})||\mathbf{x}-\mathbf{x}_i||^2 \\ & \times[-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h_x}-\frac{||\mathbf{z}-\mathbf{z}_i||^2}{2h_z}-\frac{d_x}{2}\ln(2\pi h_x)-\frac{d_z}{2}\ln(2\pi h_z)]d\mathbf{x}d\mathbf{z}-\ln N \\ =\ & -d_x-d_x(d_x-1)^2-\frac{d_x}{2}[1+\ln(2\pi h_x)]-\frac{d_z}{2}[1+\ln(2\pi h_z)]-\ln N \qquad (2.46) \end{aligned}$$

Then

$$\begin{aligned} E_a(h) =\ & \iint p_h(\mathbf{x},\mathbf{z})\ln p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z}-\frac{1}{Nd_xh_x}\sum_i^N\iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z}) \\ & \times||\mathbf{x}-\mathbf{x}_i||^2\ln p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z} \\ \approx\ & -\frac{d_x}{2}[1+\ln(2\pi h_x)]-\frac{d_z}{2}[1+\ln(2\pi h_z)]-\ln N \\ & -[-d_x-d_x(d_x-1)^2-\frac{d_x}{2}[1+\ln(2\pi h_x)]-\frac{d_z}{2}[1+\ln(2\pi h_z)]-\ln N] \\ =\ & d_x[1+(d_x-1)^2] \qquad (2.47) \end{aligned}$$

Notice that in maximum likelihood estimation,

$$\sigma^2=\frac{1}{N}\sum_{i=1}^{N}||\mathbf{z}_i-g(\mathbf{x}_i,W)||^2 \qquad (2.48)$$

From the above discussion, with Eqs. (2.26), (2.47) and (2.48), in sparse data approximation case, from Eq. (2.41) we can obtain the following equation for rough estimation of h_x:

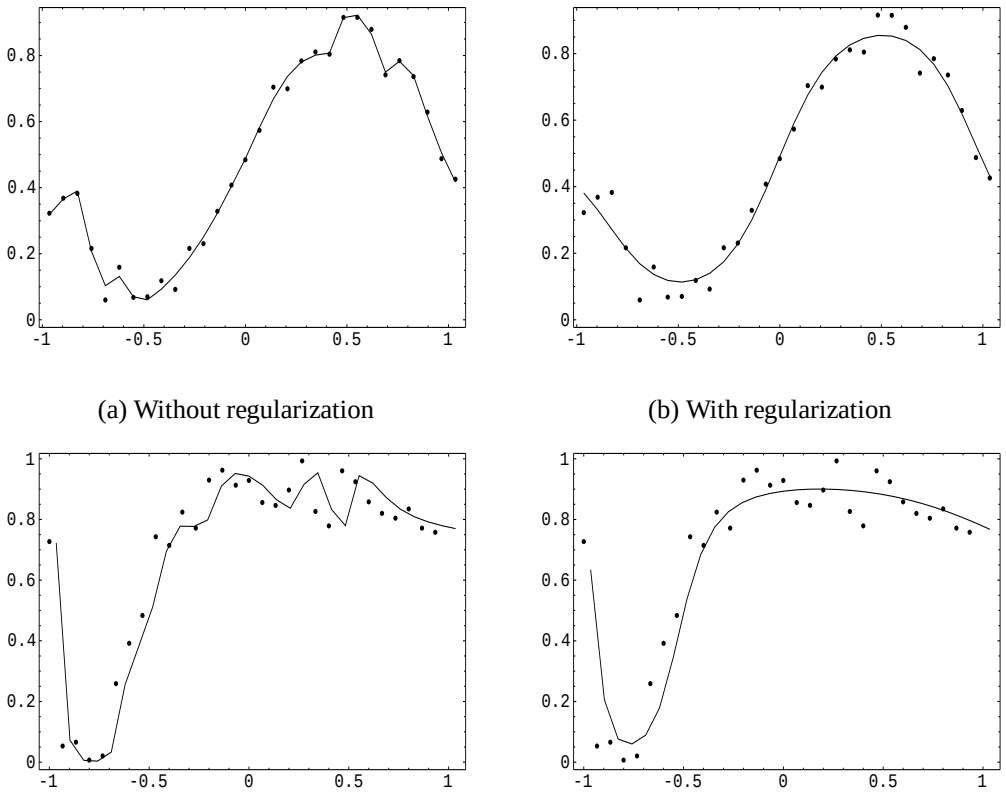
$$h_x \approx d_x^2[1 + (d_x - 1)^2]\frac{\sum_{i=1}^{N} ||\mathbf{z}_i - g(\mathbf{x}_i, W)||^2}{\sum_{i=1}^{N} ||g'(\mathbf{x}_i, W)||^2} \tag{2.49}$$

This is an approximate estimation of h_x by using the sum-square-error and penalty term, which is quite different with the equation obtained in paper [46]. In implementation, we need to find h_x and weight W by some adaptive learning algorithm. For example, we can first make an initial guess for a small non-zero value of h_x, and use this value to evaluate W, then periodically re-estimate the value of h_x by Eq. (2.49) in training processing. The advantage of this result is that only applying training data can be sufficient in estimating regularization coefficients, and h_x can be optimized on-line with minimized generalization error.

2.5 Experiments

Several experiments have been done with dynamically adjusting regularization parameter h_x. Some results are drawn in Figures 2.1–2.7. The results show that the optimal regularization parameter h_x can be found by seeking the minimum of $J(h, \Theta)$ by training data set only. We also apply the minimal generalization error method to validate the experiment results, and the same order of h_x has been obtained (see Figure 2.3). This confirms that the new parameter estimation formula is a good approximation. Unlike early stopping strategy, this new regularization parameter formula can work for overtraining network and does not need a validation set to guard when the training should stop.

The function mapping problem was considered in the experiments, and the sine and exponential functions were applied. In order to represent sufficient network complexity, we use 15 hidden neurons in three-layer network. Only 30 training samples were generated with Gaussian noise added to their output. With this kind of network architecture, if



(a) Without regularization

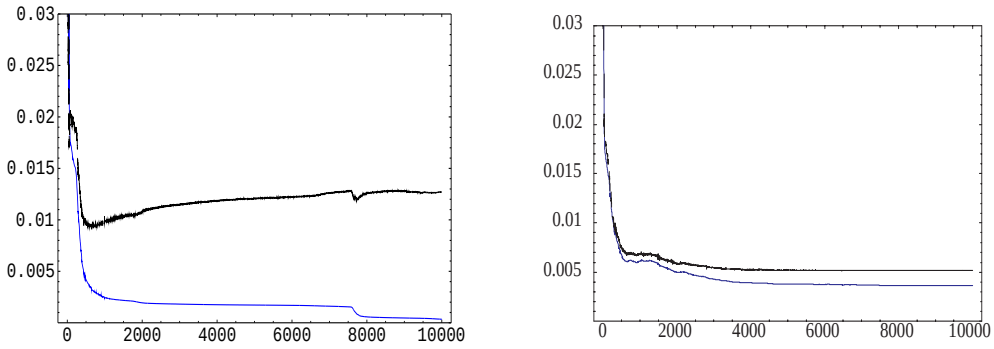
(b) With regularization

(c) Without regularization

(d) With regularization

Figure 2.1: Comparison of regularization in function mapping problem.

Dots are training samples, while solid line is the network output. (a, b) are for the sine function approximation problem. After the training is stopped, dynamically-estimated $h_x = 2.87 \times 10^{-4}$. (c, d) are for the exponential function approximation problem. After the training is stopped, dynamically-estimated $h_x = 1.27 \times 10^{-4}$.



(a) Without regularization (b) With regularization

Figure 2.2: Training epoch for the exponential function approximation problem.

Upper line represents validation error, while lower line depicts training error. Without regularization, training error is small while validation error is large. With regularization, validation error is reduced and training error is increased a little, illustrating that over-fitting does not occur.

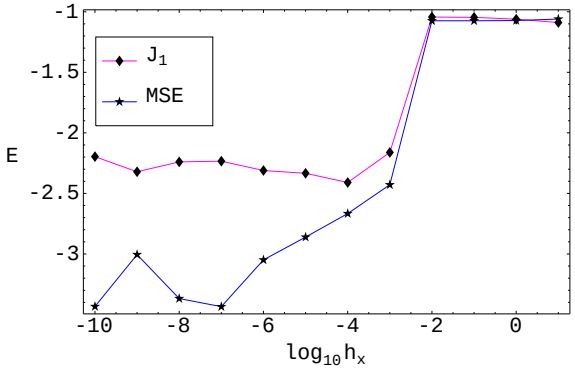
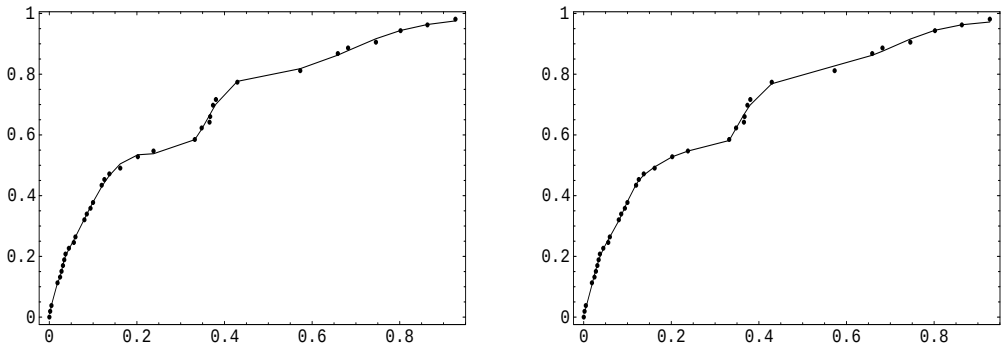


Figure 2.3: The training mean square error (MSE) on the training data set and J_1 on the validation data set, plotted versus the smooth parameter h_x.

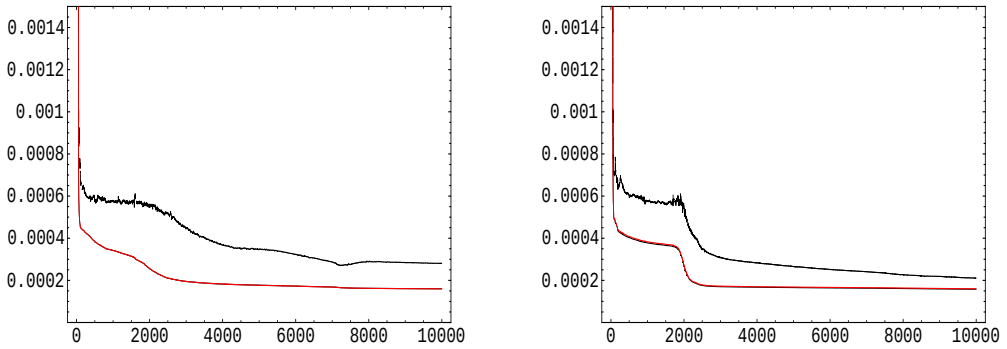
The network was trained by 30 samples which are drawn from the exponential function. We use a validation data set with 30 data points to calculate J_1 value again after the training is stopped. For each h_x value, the network was trained until the total error J_1 (Eq. 2.27) was minimized, measured by successive error difference being less than 10^{-8} and over 10^4 epoch being passed. The minimal J_1 indicates an optimal $\log_{10} h_x \approx -4$. Dynamically-estimated h_x value is 1.27×10^{-4} in this case.



(a) Without regularization (b) With regularization

Figure 2.4: Software reliability growth model approximation problem with data set sys1.

Dots are training samples, while solid line is the network output. After the training is stopped, dynamically-estimated $h_x = 1.17 \times 10^{-8}$. Because the noise is very small, the difference with and without regularization is not obvious.



(a) Without regularization (b) With regularization

Figure 2.5: Training epoch for the software reliability growth model data set sys1.

Upper line represents validation error, while lower line depicts training error. Without regularization, training error is small while validation error is a bit large. With regularization, validation error is reduced.

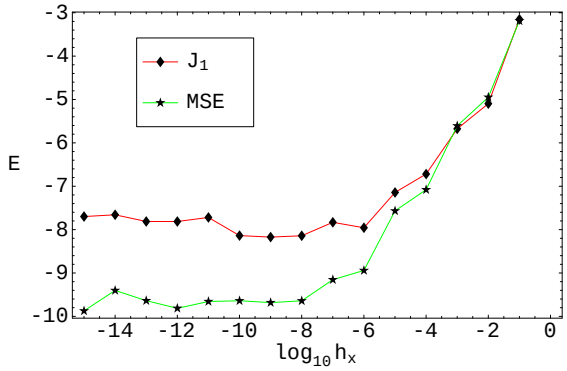


Figure 2.6: The MSE on the training data set and J_1 on the validation data set, plotted versus the h_x for sys1 data set.

The network was trained by 37 samples which are drawn from the sys1 data set. Use a validation data set with 17 data points to calculate J_1 value again after the training is stopped. For each h_x value, the network was trained until the total error J_1 was minimized, measured by over 10^4 epoch being passed. The minimal J_1 indicates an optimal value around $\log_{10} h_x \approx -9$. Dynamically-estimated h_x value is 1.17×10^{-8} in this case.

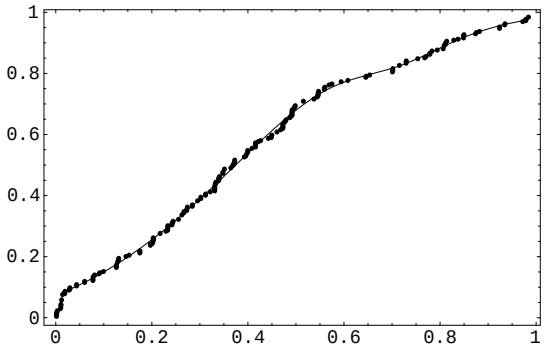


Figure 2.7: The neural network output for software reliability growth model approximation with data set sys3.

Dots are training samples, while solid line is the network output. For software reliability growth model data set sys3. Regularization does not make a significant difference.

without regularization, the phenomenon of over-fitting to noise can be observed as shown in Figure 2.1. In Figures 2.1 and 2.2, it is shown that with regularization, the network output is smoothed and generalization performance is improved. Figure 2.3 show the result of validation method estimation for regularization parameter.

Real-world data sets are used in the experiments too. The data sets are software failure data sys1 and sys3, which are contained in the attached Compact Disk of the *Handbook of software Reliability Engineering* [47]. The sys1 data set contains 54 data points. In order to validate the parameter estimation results, we partition the sys1 data into two parts: a training set and a validation set. The training set consists of 37 samples which are randomly drawn from the original data set. The remaining 17 samples comprise the validation set. The data sets are normalized to the range of values [0,1]. Normalization is a standard procedure for data preprocessing. In the software reliability investigation problem, the network input is the successive normalized failure occurrence times, and the network output is the accumulated failure number. During the training phase, each input sample x_t at time t is associated with the corresponding output value z_t at the same time t. The experimental results are shown in Figures 2.4–2.6. From Figure 2.5, it can be observed that with regularization, the validation error is less than that without regularization. Figure 2.6 shows that the minimal J_1 value indicates h_x in the range of 10^{-8} to 10^{-10}, while dynamically-estimated h_x value is 1.17×10^{-8}.

Another data set is sys3, which contains 278 data points. In the experiment, the number of training data is about $2/3$ of the total data number. That is, it consists of 186 randomly-drawn samples from the original data set. The remaining 92 samples form the test set. Because this data set is a bit large and the noise is small, it makes no obvious difference in the obtained results with respect to regularization. The trained network output is shown in Figure 2.7.

2.6 Discussion

In fact, the equation with regularization resulting from KL distance for feedforward networks is not completely equivalent to Tikhonov regularizer. Moreover, the starting point of deriving the regularization parameter estimation equation is different from the Mackey's Bayesian evidence or the MAP for hyper-parameters[27, 48]. For example, Mackey assumes the *prior* distribution of weight is Gaussian with hyper-parameter as the regularization parameter, and the penalty term is in the weight decay form. While we use nonparametric kernel density distribution, a particular approximation is equivalent to Tikhonov regularizer. The penalty term is the first derivation of sum-square-errors of a network mapping function. This form reduced to weight decay when the mapping function is in a generalized linear network, $g_j(\mathbf{x}, W) = \sum_l^{d_x} w_{j,l}\mathbf{x}_l$. Therefore,

$$\sum_{i=1}^{N} ||g'(\mathbf{x}_i, W)||^2 = N \sum_{j=1}^{M} w_j^2 \tag{2.50}$$

where M represents the number of network weight parameters and w_j is an element of the matrix W in a vector expression.

With the generalized linear network assumption, Eq. (2.49) becomes

$$h_x \approx d_x^2[1 + (d_x - 1)^2]\frac{\sum_{i=1}^{N} ||\mathbf{z}_i - g(\mathbf{x}_i, W)||^2}{N \sum_{j=1}^{M} w_j^2} \tag{2.51}$$

Now let us see the similarity of MAP approximation with our result in estimating the regularization parameter.

The cost function in Mackey's Bayesian inference is [27, 48],

$$S(w) = \frac{\beta}{2} \sum_{i=1}^{N} ||\mathbf{z}_i - g(\mathbf{x}_i, W)||^2 + \frac{\alpha}{2} \sum_{j=1}^{M} w_j^2 \tag{2.52}$$

In minimizing this cost function to find network weight parameter W, the effective value of the regularization parameter depends only on the ratio α/β, since an overall

multiplicative factor is unimportant. This means h_x should be equivalent to α/β under some approximation.

In Mackey's results[27, 48], a very rough approximation condition is $\gamma = M$ and $N \gg M$.

$$\gamma \equiv \sum_{j=1}^{M} \frac{\lambda_j}{\lambda_j + \alpha} \tag{2.53}$$

where $\{\lambda_j\}$ denote the eigenvalues of $\mathbf{H}$, the Hessian of unregularized cost function,

$$\mathbf{H} = \beta \nabla_w^2 E_D, \qquad E_D = \frac{1}{2} \sum_{i=1}^{N} ||\mathbf{z}_i - g(\mathbf{x}_i, w)||^2 \tag{2.54}$$

The matrix $\mathbf{A}$ is related to parameter α in the following form,

$$\mathbf{A} = \mathbf{H} + \alpha \mathbf{I}. \tag{2.55}$$

In order to compare with Mackey's formula, we rewrite the parameter α and β from [27, 48] in the following:

$$\beta = N/2E_D = N/\sum_{i=1}^{N} \{\mathbf{z}_i - g(\mathbf{x}_i, w)\}^2 \tag{2.56}$$

$$\alpha = M/2E_W = \frac{M}{\sum_{j=1}^{M} w_j^2} \tag{2.57}$$

Consequently,

$$\frac{\alpha}{\beta} = M \frac{\sum_{i=1}^{N} \{\mathbf{z}_i - g(\mathbf{x}_i, w)\}^2}{N \sum_{j=1}^{M} w_j^2} \tag{2.58}$$

Here we can clearly note the similarity between h_x in Eq. (2.51)) and α/β in Eq. (2.58)), where their difference is only at the constant coefficient. In h_x estimation, the constant coefficient is dependent on the dimension of input space, while in α/β estimation, the constant coefficient is the dimension of weight parameter vector. This explains that the Mackey's result is obtained in parameter space approximation, while our result

Table 2.1: Experimental results for regularization parameter estimation.

No Reg: No regularization; TE: Test Error; SDA: Sparse Data Approximation; MAP: Maximum A *Posteriori* approximation; $--$: Unstable value.

	$N(M)$	TE(No Reg.)	h_x (SDA)	TE (SDA)	h_x (MAP)	TE(MAP)
$sin(x)$	30					
$k=8$	(24)	0.00842	4.185×10^{-6}	0.00422	1.464×10^{-4}	0.0043
$k=15$	(45)	0.00695	2.87×10^{-4}	0.00418	$--$	0.097
Exp.	30					
$k=8$	(24)	0.0059	1.677×10^{-6}	0.0053	9.705×10^{-5}	0.0051
$k=15$	(45)	0.013	1.27×10^{-4}	0.0051	$--$	0.102
Sys1	37					
$k=9$	(27)	0.0003284	6.86×10^{-9}	0.000318	3.69×10^{-5}	0.000984
$k=15$	(45)	0.0000854	1.17×10^{-8}	0.000267	1.36×10^{-4}	0.001337
$k=20$	(60)	0.000311	4.29×10^{-9}	0.000258	1.12×10^{-2}	0.00636
Sys3	186					
$k=15$	(45)	0.0001535	3.892×10^{-10}	0.000104	1.118×10^{-4}	0.00032
$k=30$	(90)	7.55×10^{-5}	9.605×10^{-9}	0.0002227	3.612×10^{-4}	0.0003384
$k=60$	(180)	0.0001025	5.696×10^{-10}	0.0000675	2.83×10^{-4}	0.000347

is in data space approximation. Compared to the approximation condition, our approximation is based on the sparse data set, which is a reasonable approximation for the small number training data set case. While in Mackey's approximation, it requires $N \gg M$. In our function mapping experiments, we design that $N = 30$, $d_x = d_z = 1$, the hidden neuron number is $k = 15$, $M = (d_x + 1) \times k + k \times d_z = 45$. Because the experimental condition does not satisfy Mackey's very rough approximation condition $N \gg M$, it cannot be successful in estimating regularization parameter on-line with Eq. (2.58). In fact, the condition $N \gg M$ means that training sample number should be large enough compared to network complexity. If we have enough training samples, the generalization is also improved without regularization [21]. The experiment for data set sys3 confirms this observation.

Table 2.1 shows the experimental results for the comparison of regularization parameter estimation formula performance. It is seen that when $N > M$ or $N \sim M$, MAP

approximation based regularization parameter estimation formula performance is good, sometimes is better than SDA based. However, when we use lots of hidden neurons, for the case $N < M$, MAP approximation based formula performance becomes poor.

As we know, there is no free lunch for the optimization problem. To get the best regularization parameter value, the parameter numerical evaluation involves computation of Hessian matrix and log determinant of $\mathbf{A}^{-1}$, as well as eigenvalues of Hessian in Mackey's Bayesian inference. While in our approximation, it involves integration in data space. To save computational cost and on-line optimizing regularization parameter, a rough approximation is needed, but in this case the parameter value may not be the best one, and generalization error may not be the smallest with approximations.

2.7 Summary

In this chapter, we show that one particular case of the system entropy with Gaussian probability density reduces into the first order Tikhonov regularizer for feedforward neural networks in the maximum likelihood learning case, where the regularization parameter is the smoothing parameter h_x in the kernel density function. Under the framework of Kullback-Leibler divergence, we derive the formula for approximately estimating regularization parameter using training data only, without need to use validation data set. Experiments show that our estimated regularization parameter is in the same order as that estimated by the validation method. However, our method requires much less computation resource than the validation search method. The similarity and difference of the obtained results with others' work also discussed in this chapter.

Chapter 3

Classification for Small Sample Set with High Dimension

3.1 Introduction

The goal of the classification is to assign each sample in a given data set to a class according to some criterion of class membership. Classification has two aspects: supervised classification (discrimination) and unsupervised classification (clustering). In recent years, several classification algorithms have been developed to partition a data set into pre-defined classes. When the data are viewed as arising from two or more clusters mixed in varying proportions, we can use the finite Gaussian mixture distribution to analyze the data set. The Gaussian mixture distribution analysis method has been used widely in a variety of important practical situations, where the likelihood approach to the fitting of Gaussian mixture models has been utilized extensively [49, 50, 51, 52].

When classifying data with the Gaussian mixture model, the mean vector and covariance matrix of each component are not known in advance, and they have to be estimated from the given data set. While a large-size data set is desirable for estimating the parameters more accurately, in some real world situation, only a small-size data set can be obtained because of some restriction, e.g, high cost in collection such data set. For a relatively small-number sample data set, if the dimension d of variable x is comparable to the number of training samples n_j in class j, the problem may become poorly-posed. Worse,

if the number n_j of training samples is less than the dimensionality, the problem becomes ill-posed. In this case, not all parameters can be properly estimated and classification accuracy is degraded.

There are two possible solutions to solve this kind of problem: one is dimensionality reduction, and the other is regularization [53]. Regularization is the procedure of allowing parameters bias towards what are thought to be more plausible values, which reduces the variance of the estimates at the cost of introducing bias. The regularization techniques have been highly successful in classifying small number data with some heuristic approximations [53, 54, 55]. However, heuristic methods, for example RDA [54], require to select regularization parameters (or called Model) with some statistical techniques such as leave-one-out cross-validation, which is computation-expensive. Furthermore, a recent study shows that cross-validation performance is not always good in the selection of linear models [32]. Therefore, it is worthy to develop new techniques to deal with this problem.

Kullback-Leibler information measure [33, 34] can be considered as "distance" between two probability density models. This measure is also called *Kullback-Leibler divergence*. In this chapter, based on the mixture model analysis with the Kullback-Leibler information measure [56], we present the results of investigating covariance matrix estimation and regularization parameter selection in the Gaussian classifier for the small-sample set with high-dimension classification problem.

3.2 Classifications

3.2.1 Classification with Finite Gaussian Mixture Model

In supervised classification we have a set of data samples, each consisting of measurements on a set of variables, with associated labels, the class types. These are used as exemplars in the classifier design. In unsupervised classification we need to estimate *prior* probability and *posterior* probability in the classifier design. If these probabilities are known, it becomes supervised classification. So unsupervised classification is more

general than supervised classification in the mixture analysis case. Let us consider the general case first.

The data points $D = \{\mathbf{x}_i\}_{i=1}^{N}$ to be classified are assumed to be modelled by a mixture of k Gaussian densities with joint probability density of which the mathematical expressions are as the followings equations.

$$\begin{aligned} p(\mathbf{x}, \Theta) &= \sum_{j=1}^{k} \alpha_j G(\mathbf{x}, \mathbf{m}_j, \Sigma_j), \\ \text{with} \quad \alpha_j &\geq 0, \text{ and } \sum_{j=1}^{k} \alpha_j = 1 \end{aligned} \tag{3.1}$$

where

$$G(\mathbf{x}, \mathbf{m}_j, \Sigma_j) = \frac{\exp[-\frac{1}{2}(\mathbf{x} - \mathbf{m}_j)^T \Sigma_j^{-1} (\mathbf{x} - \mathbf{m}_j)]}{(2\pi)^{d/2} |\Sigma_j|^{\frac{1}{2}}} \tag{3.2}$$

is a general multivariate Gaussian density function, $\mathbf{x}$ denotes a random vector, d is the dimension of the $\mathbf{x}$, and parameter $\Theta = \{\alpha_j, \mathbf{m}_j, \Sigma_j\}_{j=1}^{k}$ is a set of finite mixture model parameter vectors. Here α_j is the *prior* probability, $\mathbf{m}_j$ is the mean vector, and Σ_j is the covariance matrix of the j-th component. Based on the given data set, these parameters can be estimated by the maximum likelihood (ML) method with Expectation-Maximum (EM) algorithm [57, 58].

In Gaussian mixture model case the Bayesian decision rule is used to classify the vector $\mathbf{x}$ into class j with the largest *posterior* probability. The *posterior* probability $P(j|\mathbf{x})$ represent the probability that sample $\mathbf{x}$ belongs to class j. Now we use Bayesian decision $j^* = \arg\max_j P(j|\mathbf{x})$ to classify $\mathbf{x}$ into class j^*. The probability $P(j|\mathbf{x})$ is usually unknown and have to be estimated from the training samples. With maximum likelihood estimation, the *posterior* probability can be written in the form

$$\begin{aligned} P(j|\mathbf{x}) &= \frac{\alpha_j G(\mathbf{x}, \mathbf{m}_j, \Sigma_j)}{p(x, \Theta)}, \\ \text{with } j &= 1, 2, \cdots, k, \end{aligned} \tag{3.3}$$

Taking the logarithm to above equation and omitting the common factors of the classes, the classification rule becomes,

$$j^* = \arg\min_j d_j(\mathbf{x}), \qquad j = 1, 2, \cdots, k \tag{3.4}$$

with

$$d_j(\mathbf{x}) = (\mathbf{x} - \mathbf{m}_j)^T \Sigma_j^{-1} (\mathbf{x} - \mathbf{m}_j) + \ln |\Sigma_j| - 2 \ln \alpha_j \tag{3.5}$$

This equation is often called the discriminant score for j-th class in the literature[53]. Furthermore, if the *prior* probability α_j is the same for all classes, it becomes discriminant function when omitting the term $2 \ln \alpha_j$.

3.2.2 Covariance Matrix Estimation

When the sample number N is small, the sample-base estimated class-specific covariance matrix becomes inaccurate, and hence results in lowered classification accuracy. To solve this problem, there are several techniques are proposed. In this chapter, we address this problem by using Kullback-Leibler divergence.

We consider that the given data can be modelled by a finite Gaussian mixture model, from the other side, the data set can be considered as samples drawn from a nonparametric density distribution $p_h(\mathbf{x})$ [35]. The same data set is described by two different function with some parameters. To make the two models consistency, we should estimate the parameters in the model based on given data. The "distance" of these two probability densities should be minimized in principle, then this quality usually is measured with the following Kullback-Leibler divergence[33, 34],

$$KL(h, k, \Theta) = \int p_h(\mathbf{x}) \ln \frac{p_h(\mathbf{x})}{p(\mathbf{x}, \Theta)} d\mathbf{x} \tag{3.6}$$

The above KL function, also called the system cost function, can be rewritten in the form,

$$KL(h,k,\Theta) = -\int p_h(\mathbf{x}) \ln p(\mathbf{x},\Theta) d\mathbf{x} + \int p_h(\mathbf{x}) \ln p_h(\mathbf{x}) d\mathbf{x} \tag{3.7}$$

where

$$p_h(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} G(\mathbf{x},\mathbf{x}_i,\mathbf{B}_h) = \frac{1}{N(2\pi)^{d/2}|\mathbf{B}_h|^{\frac{1}{2}}} \sum_{i=1}^{N} \exp[-\frac{1}{2}\sum_{j=1}^{d} \frac{(x_{,j}-x_{i,j})^2}{h_j}] \tag{3.8}$$

is assigned as Gaussian kernel density for given training samples D

Here $x_{i,j}$ represents the j-th component of the data point i, $\mathbf{B}_h$ is a $(d \times d)$ dimensional diagonal matrix with general form,

$$\mathbf{B}_h = \begin{pmatrix} h_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & h_d \end{pmatrix} \tag{3.9}$$

$h_i,\ i = 1,2,...,d$ are smoothing parameters in nonparametric kernel density. In the following we denote the set $h = \{h_i\}_{i=1}^{d}$.

The ordinary EM algorithm[57, 58] can be re-derived based on the minimization of the Kullback-Leibler divergence function (3.6) in the limit $h \to 0$, which we describe as the following [59],

E-step:

Calculate the *posterior* probability $P(j|\mathbf{x}_i)$ according to Eq. (3.3).

M-step:

$$\alpha_j^{new} = \frac{1}{N}\sum_{i=1}^{N} \frac{\alpha_j^{old} G(\mathbf{x}_i,\mathbf{m}_j,\Sigma_j)}{\sum_{j=1}^{k} \alpha_j^{old} G(\mathbf{x}_i,\mathbf{m}_j,\Sigma_j)} = \frac{1}{N}\sum_{i=1}^{N} P(j|\mathbf{x}_i) \tag{3.10}$$

$$\mathbf{m}_j = \frac{\sum_{i=1}^{N} P(j|\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^{N} P(j|\mathbf{x}_i)} = \frac{1}{n_j}\sum_{i=1}^{N} P(j|\mathbf{x}_i)\mathbf{x}_i \tag{3.11}$$

$$\widehat{\Sigma}_j = \frac{1}{n_j}\sum_{i=1}^{N} P(j|\mathbf{x}_i)(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T. \tag{3.12}$$

Here $n_j = \alpha_j N$ is an effective class sample number. The two steps are iterated until convergence to one of the local minima in the parameter space.

Unlike the supervised learning, the ML with EM algorithm can be used for total, unlabelled training data set. For small number samples, the ML estimated covariance matrix $\widehat{\Sigma}_j$ becomes singular when $n_j < d$, leading to unstable classification rate. To deal with this difficulty, one approach is regularization. In the following we address this problem based on Kullback-Leibler divergence with $h \neq 0$.

In an nonparametric kernel density function, the smoothing parameter h in Gaussian kernel density plays an important role in estimating mixture model parameters. The most concerned problem is covariance matrix estimation in classification, the mixture weight α_j and class mean $\mathbf{m}_j$ can be estimated with summation under the $h = 0$ approximation as in Eq. (3.12). Then we focus on covariance matrix estimation problem in the following. When minimizing cost function (3.6), that is, setting $\frac{\partial}{\partial \Sigma_j} KL(h, k, \Theta) = 0$, the following covariance matrix estimation formula can be obtained,

$$\Sigma_j = \frac{\int p_h(\mathbf{x}) P(j|\mathbf{x})(\mathbf{x} - \mathbf{m}_j)(\mathbf{x} - \mathbf{m}_j)^T d\mathbf{x}}{\int p_h(\mathbf{x}) P(j|\mathbf{x}) d\mathbf{x}}. \tag{3.13}$$

With above equation the parameters still need to be iterative estimating. There are several ways to evaluate this probability-type integration in each iterative estimating step.

One of the techniques is the well known *Monte Carlo integration*[39, 40], which is called *Stochastic Approximation* in paper [46]. In *Monte Carlo integration* approximation, we generate n' number of samples with Gaussian distribution around the data sample $\mathbf{x}_i$,

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{e}_x, \tag{3.14}$$

where $\mathbf{e}_x$ is the Gaussian noise drawn from the distribution $G(\mathbf{e}_x, 0, \mathbf{B}_h)$. With this approximation, the covariance matrix can be estimated according to

$$\Sigma_j = \frac{\sum_{i=1}^{N'} P(j|\mathbf{x}'_i)(\mathbf{x}'_i - \mathbf{m}_j)(\mathbf{x}'_i - \mathbf{m}_j)^T}{\sum_{i=1}^{N'} P(j|\mathbf{x}'_i)} \tag{3.15}$$

now total number of samples becomes $N' = (n' + 1)N$.

Another method to evaluate the integration is to use Taylor expansion approximation, which is used in this chapter. Because the $p_h(\mathbf{x})$ term is contained in the integration of Eq. (3.13), when x far away $\mathbf{x}_i$, the function value becomes very small when every component of h is small. In this case we can use Taylor expansion for $P(j|\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_i$ and take up to second order approximation.

$$P(j|\mathbf{x}) \approx P(j|\mathbf{x}_i) + (\mathbf{x} - \mathbf{x}_i)^T \nabla_x P(j|\mathbf{x}_i) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_i)^T \mathbf{H}_i(j)(\mathbf{x} - \mathbf{x}_i) \tag{3.16}$$

The integration of Eq. (3.13) now can be evaluated, leading to the following covariance matrix estimation formulae.

$$\Sigma_j^{(2)}(h) = (1 + \frac{1}{2}\text{Trace}[\mathbf{B}_h \mathbf{H}(j)])\mathbf{B}_h + \frac{\Sigma_\mathbf{e}}{(1+\eta)} + \frac{\widehat{\Sigma}_Q}{(1+\eta)} \tag{3.17}$$

where $\mathbf{H}(j) = \frac{1}{n_j}\sum_{i=1}^{N} \mathbf{H}_i(j)$, $\mathbf{H}_i(j)$ is the Hessian matrix, $\mathbf{H}_i(j) = \nabla_x^2 P(j|\mathbf{x}_i)$. Because this estimation is derived under the framework of Kullback-Leibler information measure, it is called as KLIM2 in our work.

If we only consider the first order approximation, the estimate becomes

$$\Sigma_j^{(1)}(h) = \mathbf{B}_h + \widehat{\Sigma}_j \tag{3.18}$$

This estimation is called as KLIM1 in this work.

The following notations are used in above equations,

$$\begin{aligned} \eta &= \frac{1}{2n_j} S(h,j), \qquad n_j = \alpha_j N, \\ S(h,j) &= \sum_{i=1}^{N} \text{Trace}[\mathbf{B}_h \mathbf{H}_i(j)], \\ \Sigma_\mathbf{e} &= \mathbf{B}_h \mathbf{H}_e \mathbf{B}_h \end{aligned} \tag{3.19}$$

$\mathbf{H}_e$ is a diagonal matrix in which the diagonal elements are the eigenvalues of $\mathbf{H}(j)$, and $\widehat{\Sigma}_j$ is represented by Eq. (3.12),

$$\widehat{\Sigma}_Q = \frac{1}{n_j}\sum_{i=1}^{N}[P(j|\mathbf{x}_i) + \frac{1}{2}\text{Trace}(\mathbf{B}_h\mathbf{H}_i(j))](\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T.$$

The Hessian matrix of the *posterior* probability function is computed as following,

$$\begin{aligned}
\mathbf{H}_i(j) \quad = \quad & P(j|\mathbf{x}_i)\{\Sigma_j^{-1}(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T\Sigma_j^{-1} \\
& - \sum_{j=1}^{k} P(j|\mathbf{x}_i)[\Sigma_j^{-1}(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T\Sigma_j^{-1}]\} \\
& + P(j|\mathbf{x}_i)\{\sum_{j=1}^{k} P(j|\mathbf{x}_i)\Sigma_j^{-1} - \Sigma_j^{-1}\} \\
& + 2P(j|\mathbf{x}_i)[\sum_{j=1}^{k} P(j|\mathbf{x}_i)\Sigma_j^{-1}(\mathbf{x}_i - \mathbf{m}_j) - \Sigma_j^{-1}(\mathbf{x}_i - \mathbf{m}_j)] \\
& \times \sum_{j=1}^{k} P(j|\mathbf{x}_i)(\mathbf{x}_i - \mathbf{m}_j)^T\Sigma_j^{-1} \qquad (3.20)
\end{aligned}$$

The quantity in the form $\sum_{j=1}^{k} P(j|\mathbf{x})Q(j)$, where $Q(j)$ stands for those terms following $P(j|\mathbf{x})$ in above equation, represents the weighted average value $Q(j)$ over all classes, the above Hessian equation reflects the difference between single class quantity and averaged quantity. If there is only one class, this Hessian matrix will become null matrix and $\Sigma_j^{(2)}(h)$ reduces into $\Sigma_j^{(1)}(h)$.

From above, we can see that new kinds of regularized covariance matrix, thereof regularized Gaussian classifier are obtained based on Kullback-Leibler information measure, where the smoothing parameter h controls the degree of regularization. Because multi-parameter optimization is more difficult than single parameter optimization, in this chapter we only consider one special case that the smoothing parameters are the same for all dimension, this means that

$$\mathbf{B}_h = h\mathbf{I}_d, \qquad (3.21)$$

where $\mathbf{I}_d$ is $d \times d$ dimensional identity matrix.

In the next section we discuss how an optimal value of smoothing parameter h can be selected based on the training samples.

3.3 Smoothing Parameter Selection

There are several ways to select smoothing parameter h, for examples, with training samples we can use statistical technique cross validation to select the optimal smoothing parameter. As we know, the goal in selecting smoothing parameter is to produce a model for the probability density which is as close as possible to the unknown density $p(\mathbf{x}, \Theta)$[21]. We can select the h by the following two methods.

3.3.1 Selecting h by *Monte Carlo method*

According to the principle of KL information measure, when $h \neq 0$, the smooth parameter h can be estimated with minimized KL divergence,

$$h^* = \arg\min J(h), \qquad J(h) = KL(k^*, \Theta^*, h) \tag{3.22}$$

In practical implementation, we can use exhaust search method. That is, for each h, we compute the $J(h)$ function values to search the minima of $J(h)$, and choose the h^* that minimize $J(h)$. Note in this approach all the samples can be used to estimate h^*. Therefore, it is different from cross-validation method which must split data into training set and validation set.

When selecting optimal h, we have to evaluate the integration equation of $J(h)$. The integral can be approximated by *Monte Carlo method* as mentioned above,

$$J(h) \approx -\frac{1}{N'} \sum_{i=1}^{N'} \{\ln p(\mathbf{x}'_i, \Theta) - \ln p_h(\mathbf{x}'_i)\} \tag{3.23}$$

In implementation, we generate n' number of samples around each original sample point, the accuracy of this approximation apparently depends on n'. When $n' \to \infty$, it evaluates the integration with arbitrarily small error. However, in practical evaluation of this integration, it is impossible to use very large number of samples to calculate Eq. (3.23), and only a limited number of generated samples is used, otherwise it is too computation expensive to implement. To get sufficient approximation accuracy without using very large generated samples set, here we can use extrapolation method. We can use an exponential function to regress n' with h, after get the mapping function of n' to h, we can extrapolate the h value at $n' \to \infty$. As an example, Figure 3.1 illustrates the result of this method. Using *Monte Carlo method* with extrapolation method we can compute h with higher accuracy. However, the cost is very computational intensive, especially in the high dimension case which we deal with in this chapter.

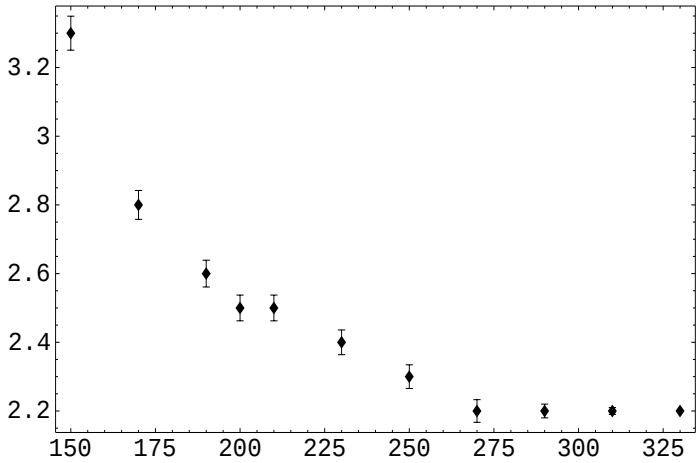


Figure 3.1: h vs. generated sample number n'.

These data points can be used to regress a nonlinear function, then applying the extrapolation method to determine the h value for $n' \to \infty$. When $n' > 300$, the h value tends to be stable. This figure is for $d = 6, n_j = 15$ and $k = 3$. Sample data points are drawn from a gaussian density function

3.3.2 Selecting h by *Taylor expansion Approximation*

Because computing the integration by *Monte Carlo method* is very time expensive, we proposed to use second order approximation for estimating smoothing parameter h.

Applying Taylor expansion for integration,

$$\begin{aligned} J(h) &= -\int p_h(\mathbf{x}) \ln p(\mathbf{x}, \Theta) d\mathbf{x} + \int p_h(\mathbf{x}) \ln p_h(\mathbf{x}) d\mathbf{x} \\ &= J_0(h) + J_e(h) \end{aligned} \tag{3.24}$$

where

$$\begin{aligned} J_0(h) &= -\int p_h(\mathbf{x}) \ln p(\mathbf{x}, \Theta) d\mathbf{x} \\ J_e(h) &= \int p_h(\mathbf{x}) \ln p_h(\mathbf{x}) d\mathbf{x}. \end{aligned}$$

Now we apply Taylor expansion to logarithm function at $\mathbf{x} = \mathbf{x}_i$ and only keep the second order term, the integration can be evaluated out. It results in the approximation of J_0,

$$J_0(h) \approx J_{01}(\mathbf{x}_i, \Theta) + h J_r(\mathbf{x}_i, \Theta) \tag{3.25}$$

where

$$J_{01}(\mathbf{x}_i, \Theta) = -\frac{1}{N} \sum_{i=1}^{N} \ln \sum_{j=1}^{k} \alpha_j G(\mathbf{x}_i, \mathbf{m}_j, \Sigma_j)$$

$$J_r(\mathbf{x}_i, \Theta) = -\frac{1}{2N} \sum_{i=1}^{N} \text{Trace}[\nabla_x^2 \ln p(\mathbf{x}_i, \Theta)] \tag{3.26}$$

While the logarithmic mixture density Hessian matrix can be computed as,

$$\begin{aligned} \nabla_x^2 \ln p(\mathbf{x}_i, \Theta) &= -\{\sum_{j=1}^{k} P(j|\mathbf{x}_i)\{\Sigma_j^{-1} - \Sigma_j^{-1}(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T \Sigma_j^{-1}\} \\ &\quad + \{\sum_{j=1}^{k} P(j|\mathbf{x}_i)[(\mathbf{x}_i - \mathbf{m}_j)^T \Sigma_j^{-1}]^T\}\{\sum_{j=1}^{k} P(j|\mathbf{x}_i)[(\mathbf{x}_i - \mathbf{m}_j)^T \Sigma_j^{-1}]\}\} \end{aligned} \tag{3.27}$$

Similar to J_0, we can obtain $J_e(h)$ term as following,

$$J_e(h) \approx \frac{1}{N}\sum_{i=1}^{N} R(\mathbf{x}_i, h) + \frac{h}{2N}\sum_{i=1}^{N} \text{Trace}[\nabla_x^2 R(\mathbf{x}_i, h)]$$

$$R(\mathbf{x}_i, h) = \ln p_h(\mathbf{x}_i)$$

While $\nabla_x^2 R(\mathbf{x}_i, h)$ is

$$\begin{aligned}\nabla_x^2 R(\mathbf{x}_i, h) &= \frac{1}{h^2}\{\sum_{j=1}^{N} \beta(\mathbf{x}_i, \mathbf{x}_j)[(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T - h\mathbf{I}_d] \\ &\quad -[\sum_{j=1}^{N} \beta(\mathbf{x}_i, \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)][\sum_{j=1}^{N} \beta(\mathbf{x}_i, \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)]^T\}\end{aligned}$$

where

$$\beta(\mathbf{x}, \mathbf{x}_i) = \frac{G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I}_d)}{\sum_{i=1}^{N} G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I}_d)},$$

and note that

$$\sum_{i=1}^{N} \beta(\mathbf{x}, \mathbf{x}_i) = \sum_{i=1}^{N} \frac{G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I}_d)}{\sum_{j=1}^{N} G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I}_d)} = 1.$$

With this relation,

$$\begin{aligned}\frac{h}{2N}\sum_{i=1}^{N} \text{Trace}[\nabla_x^2 R(\mathbf{x}_i, h)] &= \frac{1}{2Nh}\sum_{i=1}^{N}\{\sum_{j=1}^{N} \beta(\mathbf{x}_i, \mathbf{x}_j)[||\mathbf{x}_i - \mathbf{x}_j||^2] \\ &\quad -||\sum_{j=1}^{N} \beta(\mathbf{x}_i, \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)||^2\} - \frac{d}{2}\end{aligned}$$

Now the function $J(h)$ can be computed based on the original samples with summation instead of integration.

For very sparse data distribution, we can use the following approximation to estimate smoothing parameter.

$$\begin{aligned} p_h(\mathbf{x}) \ln p_h(\mathbf{x}) &\approx \frac{1}{N}\sum_{i=1}^{N} G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I}_d) \ln \frac{1}{N} G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I}_d) \\ &= \frac{1}{N}\sum_{i=1}^{N} G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I}_d)[-\frac{d}{2}\ln(2\pi h) - \frac{1}{2h}||\mathbf{x}-\mathbf{x}_i||^2 - \ln N] \end{aligned}$$

$$\begin{aligned} J_e(h) &= \int p_h(\mathbf{x}) \ln p_h(\mathbf{x}) d\mathbf{x} \\ &\approx -\frac{d}{2}\ln(2\pi h) - \frac{d}{2} - \ln N \end{aligned}$$

So we get the approximation formula for $J(h)$,

$$J(h) \approx J_{01}(\mathbf{x}_i, \Theta) + hJ_r(\mathbf{x}_i, \Theta) - \frac{d}{2}\ln(h) + C \tag{3.28}$$

where C is a constant irrelevant to h.

Taking partial derivative of $J(h)$ to h and let it be equal to zero,

$$\frac{\partial}{\partial h} J(h) = J_r(\mathbf{x}_i, \Theta) - \frac{d}{2h} = 0,$$

the very roughly estimation formula is obtained as

$$h = \frac{d}{2J_r} \tag{3.29}$$

3.4 Approximations for Regularization Term

In practice, the computation of η and Hessian matrix of equations (3.19) is still complex, some proper approximations should be adopted to simplify the calculation. Now let us consider several special cases.

Case 1: the mean and covariance of all classes are the same (All classes are overlapped together).

In this case, the quality $\sum_{j=1}^{k} P(j|\mathbf{x}_i)Q(j) = Q(j)$, it leads to $H_i(j) = 0$. Then

$$\eta = 0 \tag{3.30}$$

it reduces into the first order approximation KLIM1.

Case 2: the mean of all classes is equal to each other, or using averaged mean instead of individual class mean. Denote the weighted average $\sum_{j=1}^{k} P(j|\mathbf{x}_i)Q(j) = \overline{Q(j)}$, then,

$$\begin{aligned}
\mathbf{H}_i(j) &= P(j|\mathbf{x}_i)\{\Sigma_j^{-1}(\mathbf{x}_i - \overline{\mathbf{m}_j})(\mathbf{x}_i - \overline{\mathbf{m}_j})^T\Sigma_j^{-1} \\
&\quad - \sum_{j=1}^{k} P(j|\mathbf{x}_i)[\Sigma_j^{-1}(\mathbf{x}_i - \overline{\mathbf{m}_j})(\mathbf{x}_i - \overline{\mathbf{m}_j})^T\Sigma_j^{-1}]\} \\
&\quad + P(j|\mathbf{x}_i)\{\overline{\Sigma_j^{-1}} - \Sigma_j^{-1}\} \\
&\quad + 2P(j|\mathbf{x}_i)[\overline{\Sigma_j^{-1}} - \Sigma_j^{-1}](\mathbf{x}_i - \overline{\mathbf{m}_j})(\mathbf{x}_i - \overline{\mathbf{m}_j})^T\overline{\Sigma_j^{-1}}
\end{aligned} \tag{3.31}$$

with the approximation $\sum_{i=1}^{N} P(j|\mathbf{x}_i)(\mathbf{x}_i - \overline{\mathbf{m}_j})(\mathbf{x}_i - \overline{\mathbf{m}_j})^T \approx n_j\overline{\Sigma_j}$, and omit the cross term between classes,

$$\eta \approx h\text{Trace}[\overline{\Sigma_j^{-1}} - \Sigma_j^{-1}] \tag{3.32}$$

Case 3: the class covariance matrices are all the same, $\Sigma_j = \overline{\Sigma}$

$$\begin{aligned}
\mathbf{H}_i(j) &= P(j|\mathbf{x}_i)\{\Sigma^{-1}(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T\Sigma^{-1} \\
&\quad - [\Sigma^{-1}\overline{(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T}\Sigma^{-1}]\} \\
&\quad + 2P(j|\mathbf{x}_i)\Sigma^{-1}[\overline{(\mathbf{x}_i - \mathbf{m}_j)} - (\mathbf{x}_i - \mathbf{m}_j)] \\
&\quad \times \overline{(\mathbf{x}_i - \mathbf{m}_j)^T}\Sigma^{-1}.
\end{aligned} \tag{3.33}$$

When dropping the first two term in above equation if we assume that the difference is very small, then,

$$\eta \approx h\sum_{i=1}^{N}\text{Trace}[P(j|\mathbf{x}_i)\Sigma^{-1}[\overline{(\mathbf{x}_i - \mathbf{m}_j)} - (\mathbf{x}_i - \mathbf{m}_j)]\overline{(\mathbf{x}_i - \mathbf{m}_j)^T}\Sigma^{-1}]. \tag{3.34}$$

When above approximations are used, the computation cost will be significantly reduced.

From above approximations, it is known that second order regularization involves in calculation of the inverse covariance matrix. In the case of $n_j > d$, we can use Eq. (3.40) to estimate initial value of Σ_j. While for the case $n_j < d$, $\widehat{\Sigma}_j$ becomes singular, but with KLIM1 estimator, Σ_j is not singular as long as h is not too small. In this case, we adopt KLIM1 estimated covariance matrix as initial value to calculate $\mathbf{H}(j)$ and η.

In fact, if we let the eigenvector and eigenvalue of a covariance matrix $\widehat{\Sigma}_j$ be $\mathbf{u}_i$ and ν_i, respectively,

$$\widehat{\Sigma}_j \mathbf{u}_i = \nu_i \mathbf{u}_i, \qquad \mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}. \tag{3.35}$$

The inverse matrix of Σ_j in KLIM1 can be expressed as

$$\Sigma_j^{-1} = (h I_d + \widehat{\Sigma}_j)^{-1} = \sum_{i=1}^{d} \frac{\mathbf{u}_i \mathbf{u}_i^T}{\nu_i + h} \tag{3.36}$$

then,

$$\text{Trace}[\Sigma_j^{-1}] = \sum_{i=1}^{d} \frac{1}{\nu_i + h} \tag{3.37}$$

If $\widehat{\Sigma}_j$ is singular, then $|\widehat{\Sigma}_j| = 0$ and hence $|\nu I - \widehat{\Sigma}_j| = 0$. This means a singular matrix has at least one zero eigenvalue, and resulting in one term is proportional to h^{-1} in the above equation. It is clearly seen that as long as h is not too small, Σ_j^{-1} exists with finite value and the estimated classification rate will be stable.

3.5 Comparison of KLIM with Other Discriminant Analysis Methods

3.5.1 Review of previous work

When the class membership of given training samples is known, the "hard-cut" version of $P(j|\mathbf{x})$ is used in the mean vector and covariance matrix estimation,

$$P(j|\mathbf{x}_i) = \begin{cases} 1, & \text{If } \mathbf{x}_i \in \text{ class } j \\ 0, & \text{If } \mathbf{x}_i \notin \text{class } j \end{cases} \tag{3.38}$$

In this case,

$$\mathbf{m}_j = \frac{1}{n_j} \sum\nolimits_{i=1}^{n_j} \mathbf{x}_i \tag{3.39}$$

$$\widehat{\Sigma}_j = \frac{1}{n_j} \sum\nolimits_{i=1}^{n_j} (\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T. \tag{3.40}$$

now the $\mathbf{x}_i$ is a sample from class j with probability one, and n_j is the training sample number of class j. This is the traditional ML estimator. When using unbiased estimation,

$$\widehat{\Sigma}_j = \frac{1}{n_j - 1} \sum\nolimits_{i=1}^{n_j} (\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T, \tag{3.41}$$

this is called sample covariance matrix in the literature[35].

Using the classification rule Eq. (3.4) and Eq. (3.5) with above covariance estimation is called quadratic discriminant analysis (QDA). When class sample size n_j is approximately equal to or small compared with the dimension d, the covariance estimation with Eq. (3.40) will become highly variable, in this case it becomes ill- or poorly-posed classification problem. To improve classification performance, we can apply regularization as mentioned.

One of the regularization methods to deal with the poorly-posed problem is linear discriminant analysis (LDA)[60]. In LDA, the $\widehat{\Sigma}_j$ in Eq. (3.5) is replaced with following pooled covariance matrix, also called common covariance matrix,

$$\widehat{\Sigma} = \frac{1}{N}\sum_{j=1}^{k} n_j \widehat{\Sigma}_j \tag{3.42}$$

This applies a considerable degree of regularization by substantially reducing the number of parameters to be estimated.

Regularized discriminant analysis (RDA) is another regularization method which was proposed by Friedman in 1989[54]. RDA is designed for small number samples case, the covariance matrix takes the following form:

$$\Sigma_j(\lambda,\gamma) = (1-\gamma)\Sigma_j(\lambda) + \gamma\frac{Trace[\Sigma_j(\lambda)]}{d}\mathbf{I}_d \tag{3.43}$$

where

$$\Sigma_j(\lambda) = \frac{(1-\lambda)n_j\widehat{\Sigma}_j + \lambda N\widehat{\Sigma}}{(1-\lambda)n_j + \lambda N} \tag{3.44}$$

The two parameters λ and γ, which are restricted to the range 0 to 1, are regularization parameters to be selected according to maximum the leave-one-out classification accuracy. λ controls the amount of the $\widehat{\Sigma}_j$ that are shrunk towards $\widehat{\Sigma}$, while γ controls the shrinkage of the eigenvalues towards equality as Trace$[\Sigma_j(\lambda)]/d$ is equal to the average of the eigenvalues of $\Sigma_j(\lambda)$.

There exists another covariance matrix estimation formula which was proposed by Hoffbeck and Landgrebe in 1996.[55] They examine the diagonal sample covariance matrix, the diagonal common covariance matrix, and some pair-wise mixtures of those matrices. The proposed estimator has the following form:

$$\Sigma_j(\xi_j) = \xi_{j1} diag(\widehat{\Sigma}_j) + \xi_{j2}\widehat{\Sigma}_j + \xi_{j3}\widehat{\Sigma} + \xi_{j4} diag(\widehat{\Sigma}) \tag{3.45}$$

The elements of the mixing parameter $\xi_j = [\xi_{j1}, \xi_{j2}, \xi_{j3}, \xi_{j4}]^T$ are required to sum to unity: $\Sigma_{l=1}^4 \xi_{jl} = 1$. In order to reduce the computation cost, they only consider three cases: $(\xi_{j3}, \xi_{j4}) = 0$, $(\xi_{j1}, \xi_{j4}) = 0$, and $(\xi_{j1}, \xi_{j2}) = 0$. They called the covariance matrix estimator as LOOC because the mixture parameter ξ was optimized by Leave-One-Out Cross validation method.

3.5.2 Comparison of KLIM with RDA and LOOC

Now let us consider a special approximation of matrix $\mathbf{H}(j)$ in order to compare KLIM with other discriminant analysis methods.

The estimation of eigenvalue matrix of *posterior* probability Hessian is iterative procedure, where initialization is necessary. At the beginning, we don't know the true distribution of the samples, therefore the form of matrix Σ_j is also unknown. One of the assumptions is that let $\Sigma_j = \mathbf{I}_d$ be the initial value. Under this assumption, the *posterior* probability Hessian matrix becomes:

$$\begin{aligned}
\mathbf{H}(j) \quad = \quad & \frac{1}{n_j}\sum_{i=1}^{N}\{P(j|\mathbf{x}_i)\{(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T \\
& - \sum_{j=1}^{k} P(j|\mathbf{x}_i)[(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T]\} \\
& + 2P(j|\mathbf{x}_i)[\sum_{j=1}^{k} P(j|\mathbf{x}_i)(\mathbf{x}_i - \mathbf{m}_j) - (\mathbf{x}_i - \mathbf{m}_j)] \\
& \times \sum_{j=1}^{k} P(j|\mathbf{x}_i)(\mathbf{x}_i - \mathbf{m}_j)^T\}
\end{aligned} \tag{3.46}$$

Furthermore, if we regard that $\sum_{j=1}^{k} P(j|\mathbf{x}_i)\mathbf{m}_j = \mathbf{m}$ as averaged mean, the above equation becomes,

$$\mathbf{H}(j) = \widehat{\Sigma}_j - \widehat{\Sigma} + \frac{2}{n_j}\sum_{i=1}^{N} P(j|\mathbf{x}_i)(\mathbf{x}_i - \mathbf{m})(x_i - \mathbf{m})^T - \frac{2}{n_j}\sum_{i=1}^{N} P(j|\mathbf{x}_i)(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m})^T. \tag{3.47}$$

The last term represents the cross variance between individual class and common class, if classes overlapped part is very small, this term had very little significance, so it can be omitted under some cases. The third term can be regarded as approximately equivalent to the two times of the common covariance matrix, under this approximation, the Hessian matrix can be written as

$$\mathbf{H}(j) = \widehat{\Sigma}_j + \widehat{\Sigma} \tag{3.48}$$

When we rearrange the term in KLIM2, it leads to,

$$\Sigma_j^{(2)}(h) = (1 + \frac{h}{2}\text{Trace}[\mathbf{H}(j)])h\mathbf{I}_d + \frac{h^2}{(1+\eta)}ev(\widehat{\Sigma}_j) + \frac{h^2}{(1+\eta)}ev(\widehat{\Sigma}) + \frac{\widehat{\Sigma}_Q}{(1+\eta)} \tag{3.49}$$

where $ev(\Sigma)$ stands for a diagonal matrix in which the diagonal elements are the eigenvalues of Σ.

This result is interesting when compared to other's regularized matrix estimator. The term $h^2/2\text{Trace}[\mathbf{H}(j)])\mathbf{I}_d$ is very similar with term $\gamma/d\text{Trace}[\Sigma_j(\lambda)]\mathbf{I}_d$ of RDA in form, but the coefficient is different. While the terms of diagonal matrices are similar with LOOC, especially if $\widehat{\Sigma}$ and $\widehat{\Sigma}_j$ are diagonal, the eigenvalue matrix $ev(\Sigma)$ is exactly equal to diagonal(Σ), here again the coefficients are quite different. The most important difference of KLIM with RDA is that RDA uses two parameters control regularization, while KLIM uses the single parameter h. LOOC uses single parameter to control the mixing of two covariance matrix, while the KLIM just use the same portion to add eigenvalue matrices of sample and common covariance matrix. From above equations, we can regard

that the KLIM includes parts of RDA and parts of LOOC, it is an integration of RDA and LOOC.

KLIM is derived under the framework of Kullback-Leibler information measure, while RDA and LOOC are heuristically proposed. KLIM, RDA and LOOC are similar in that they all consider ML estimated covariance matrix and addition of extra matrices, KLIM and RDA both have an identity matrix multiplied by a scalar, but scalar term is different from each other. There is also a term in KLIM2 which is the eigenvalue matrix of *posterior* probability Hessian, while RDA considers it with LDA estimation, and LOOC considers it with the diagonal sample or common covariance matrix. Moreover, the ML estimate in KLIM2 has the regularization coefficient which related to difference between averaged classes quantities and single class quantities, while RDA is simply related to sample covariance matrix estimation.

KLIM is different with LOOC in that LOOC considers mixing sample covariance matrix and its diagonal, or common covariance. However, the value of mixing parameter η_j in LOOC is still selected by using leave-one-out cross validation statistical methods. In KLIM, the regularization parameter is the smoothing parameter in kernel density estimation, which can be selected based on KL divergence with all samples. While in RDA, regularization parameter is heuristically proposed, which must use some statistical method, such as bootstrap, leave-one-out cross validation, to optimize. In this point, RDA requires much more computation than KLIM.

In the following we show that at one special approximation, when covariance matrices are identical to each other, that is $\Sigma_j = \sigma \mathbf{I}_d$, we can get $h =$Trace$[\Sigma_j(\lambda)]/d$.

From Eqs. (3.26) and (3.29),

$$J_r(\mathbf{x}_i, \Theta) = \frac{1}{2N}\text{Trace}\sum_{i=1}^{N}[-\nabla_x^2 \ln p(\mathbf{x}_i, \Theta)] \tag{3.50}$$

while

$$
\begin{aligned}
-\sum_{i=1}^{N} \nabla_x^2 \ln p(\mathbf{x}_i, \Theta) &= \sum_{i=1}^{N} \{ \sum_{j=1}^{k} P(j|\mathbf{x}_i)[\Sigma_j^{-1} - \Sigma_j^{-1}(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T \Sigma_j^{-1}] \\
&\quad + [\sum_{j=1}^{k} P(j|\mathbf{x}_i)[(\mathbf{x}_i - \mathbf{m}_j)^T \Sigma_j^{-1}]^T][\sum_{j=1}^{k} P(j|\mathbf{x}_i)[(\mathbf{x}_i - \mathbf{m}_j)^T \Sigma_j^{-1}]] \}
\end{aligned}
$$

Using approximation $\sum_{i=1}^{N} P(j|\mathbf{x}_i)(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T = n_j \widehat{\Sigma}_j$, $\sum_{i=1}^{N} P(j|\mathbf{x}_i) = n_j$, and $\Sigma_j^{-1} \widehat{\Sigma}_j = \mathbf{I}_d$, the first term in above equation is equal to zero. In considering hard-cut case, the cross terms in second term of above equation can be omitted. The equation reduced into

$$
\begin{aligned}
J_r(\mathbf{x}_i, \Theta) &= \frac{1}{2N} \text{Trace} \sum_{i=1}^{N} [-\nabla_x^2 \ln p(\mathbf{x}_i, \Theta)] \\
&\approx \frac{1}{2N} \text{Trace} \sum_{j=1}^{k} \sum_{i=1}^{n_j} \Sigma_j^{-1} (\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T \Sigma_j^{-1} \\
&= \frac{1}{2} \text{Trace} \{ \sum_{j=1}^{k} \alpha_j \Sigma_j^{-1} \}
\end{aligned}
$$

From the Eq. (3.29),

$$
h = \frac{d}{2J_r(\mathbf{x}_i, \Theta)} \approx \frac{d}{\text{Trace}[\sum_{j=1}^{k} \alpha_j \Sigma_j^{-1}]} \tag{3.51}
$$

In a special case,

$$
\Sigma_j = \sigma \mathbf{I}_d, \qquad h = \sigma = \text{Trace}[\Sigma_j]/d \tag{3.52}
$$

In RDA, if we let $\lambda = 1$ and $\gamma = 1$, the covariance matrix will reduce to the same equation as above. Here we can see that even in the first order approximation, there still exists some relationship of KLIM1 with RDA.

3.6 Experiment Results

In order to investigate the performance of KLIM compared with RDA and LDA, we use both synthetic data and the real world Raman Spectra data set to conduct experiments.

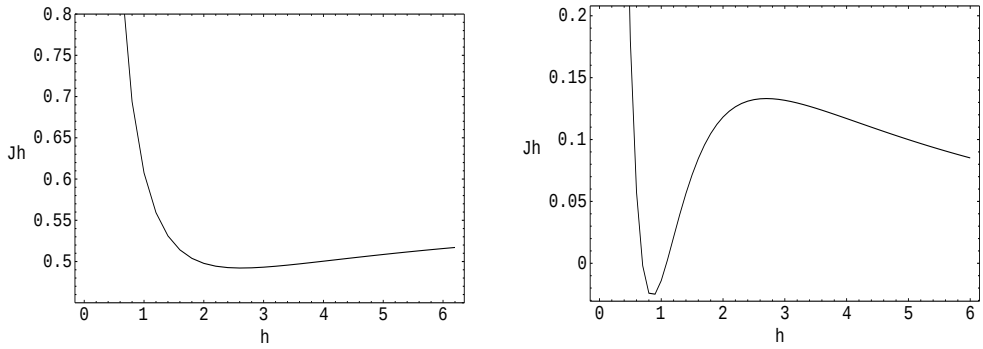
3.6.1 Synthetic data

In the experiments, the synthetic data set have been generated under different conditions. Three experiments with various distributions adapted from Friedman's paper[54] and four dimension, ($d = 6, 10, 20, 40$), were carried out. 15 training samples in each class were randomly drawn from three different Gaussian distribution, the mean and covariance matrix were estimated based on these training samples. Additional 100 independent test samples from each class were generated to be used as verifying classification accuracy. Each experiment is repeated 26 times and the mean of classification accuracy as well as standard deviation were reported.

In the experiments, the smoothing parameter h was estimated using Eqs. (3.22) and (3.24). This estimated value is smaller than that obtained by using Eq. (3.23). Figures 3.2a and 3.2b are typical $J(h)$ vs. h curves. We select the smallest h value corresponding to local minima of the $J(h)$. In the case of $n_j > d$, we can use Eq. (3.29) for fast estimation of h as an initial value. In RDA, the values of both λ and γ were sampled over a very coarse grid, (0.0, 0.25, 0.50, 0.75, 1.0), resulting in 25 data points.

In experiment 1, the covariance matrices of all three classes were equal to the identity matrix, that is, the equal spherical covariance matrices. The mean of the first class was at the origin, the mean of second class was 3.0 in the first variable and 0 in the other variables, and the mean of third class was 3.0 in the second variable and 0 in the other variables. Table 3.1 is the results for this experiment. In the tables presented at this chapter, the value in parentheses represents the standard deviation and dash lines represent the covariance matrix is singular in which case reliable results can't be obtained.

In experiment 2, all three classes had identical, highly ellipsoidal covariance matrices.



(a) (b)

Figure 3.2: The $J(h)$ function with some approximation.

(a) $J(h)$ vs. h curve computed by Monte Carlo integration approximation with Eq. (3.23). (b) $J(h)$ vs. h curve computed by Second order approximation with Eq. (3.24).

Table 3.1: Mean classification accuracy for experiment 1

	$d = 6$	$d = 10$	$d = 20$	$d = 40$
LDA	84.5(3.58)	75.3(6.86)	- - -	- - -
QDA	84.5(3.58)	75.3(6.88)	- - -	- - -
RDA	90.2(1.43)	88.57(5.37)	87.16(2.58)	91.2(2.09)
KLIM I	90.2(1.43)	91.73(1.29)	88.4(1.4)	91.26(1.29)
KLIM II	90.5(1.39)	92.17(1.61)	63.9(2.07)	63.0(3.61)

Table 3.2: Mean classification accuracy for experiment 2

	$d = 6$	$d = 10$	$d = 20$	$d = 40$
LDA	98.1(0.9)	100(0.01)	- - -	- - -
QDA	98.1(0.9)	100(0.01)	- - -	- - -
RDA	98.9(0.8)	100(0.0)	100(0.0)	100(0.0)
KLIM I	99.88(0.16)	100(0.0)	100(0.0)	100(0.0)
KLIM II	99.88(0.16)	100(0.0)	100(0.0)	100(0.0)

The covariance matrix for all three classes was a diagonal matrix whose diagonal elements were given by

$$\sigma_i = [9(i-1)/(d-1)+1]^2, \qquad 1 \leq i \leq d \tag{3.53}$$

The mean vector of the first class was at the origin, the elements of the mean vector of the second class were given by

$$\mu_{2,i} = 2.5\sqrt{\sigma_i/d}[(d-i)/(d/2-1)],$$

and the mean of last class was defined by $\mu_{3,i} = (-1)^i \mu_{2,i}$. The results of this experiments are listed in Table 3.2.

In experiment 3, all of the mean vectors of the three classes were at the origin, but the class covariance matrices were the unequal highly ellipsoidal. The diagonal elements of the covariances for first class was defined by Eq. (3.53), the other two were defined by

$$\sigma_{2,i} = [9(d-i)/(d-1)+1]^2, \tag{3.54}$$

and

$$\sigma_{3,i} = [9(i-(d-1)/2)/(d-1)]^2 \tag{3.55}$$

Table 3.3: Mean classification accuracy for experiment 3

	$d = 6$	$d = 10$	$d = 20$	$d = 40$
LDA	38.8(4.79)	42.2(4.25)	43.16(4.5)	39.64(5.2)
QDA	84.2(3.77)	84.1(6.3)	- - -	- - -
RDA	84.0(3.27)	84.9(5.78)	89.73(2.62)	74.2(8.6)
KLIM I	85.8(2.26)	92.7(2.65)	85.84(3.15)	81.75(3.47)
KLIM II	85.6(2.27)	92.1(4.46)	84.98(3.35)	55.67(2.62)

In the experiment 1 and 2, in most cases, KLIM led to higher classification accuracy than that by LDA and QDA. In experiment 3, the KLIM1 classification accuracy was higher than other covariance matrix estimation except in one case ($d = 20$). Table 3.3 shows the experiment results.

3.6.2 Raman Spectra Data

This real world data set was collected in the laboratory experiments from the physics department at Peking university. It includes 37 Raman spectra whose wave number ranges are from 320 to 1640 cm^{-1}, where each spectrum measured by the Charge-Coupled Device (CCD) detector with 1340 effective channel at the same time. The raw data set consists of three classes: 3 Raman spectra for Ethanol, 5 for Acetic acid, and 29 different time measures in synthesizing Ethyl acetate. The dimension of the raw data set is 1340 and the sample number is 37.

Because the dimension of the raw data set is very high and number is too small, we first divide each sample into 10 samples. From every 10 variables of the raw spectrum vector one point is drawn and used to construct a new sample vector. By this method, each original sample is subdivided into 10 samples, and the dimension d from 1340 is reduced to 134. We also scale the intensity of Raman spectra to the range of [0,1].

The data set we used for classification experiments is $d = 134$ now, where class 1 has 30 samples, class 2 has 50 samples and class 3 has 290 samples after preprocessing the data set. In order to study the performance of the regularized classifiers, we use bootstrap

technique [26] to conduct the experiments. 20 training samples are randomly drawn from each class and used to estimate the mean and covariance matrices. The remaining samples are used as test samples to verify classification accuracy. The experiments is repeated 26 times, the obtained result is averaged value. The same data set is used with different classification methods. This is still an ill-posed problem because of $n_j < d$. In this case, when we apply LDA and QDA to this Raman spectra data set, they are failed to give reliable classification results because the covariance matrix is singular.

On the other hand, the bootstrap experiments show that for RDA gives an averaged classification accuracy 99.27% , the standard deviation is 0.43, while the classification accuracy for KLIM1 and KLIM2 both reaches 99.81%, the standard deviation is 0.28. The results also illustrate that these three classes are well separated from each other in the high-dimension space. Here we should indicate that the classification results are comparable because we use the same data set for LDA, QDA, RDA, KLIM1 and KLIM2, it is not dependent on preprocessing of the data set.

3.6.3 Discussions

From these experiments we see that the performance of various classification schemes is generally data-dependent. For example, if all the classes have the same covariance matrix, LDA will lead to a higher classification accuracy than that of QDA. In experiment 1, the true covariance matrix is in equal sphere, which is a situation that may favor RDA as well as KLIM1. However, KLIM1 led to either the same or a little higher classification accuracy than RDA. In experiment 2, the classes are highly equal ellipsoidal distribution, and their mean positions are separated well with little overlapping. Consequently, all classification estimators produced high classification accuracy. This example also illustrates that the classification accuracy strongly depends on the degree of overlapping between classes. In experiments 3, the classes are heavily overlapped with highly unequal ellipsoidal distribution, in which case LDA performance is very poor. With properly selected

smoothing parameter, KLIM1 is better than RDA except in one case ($d = 20$). These experimental results indicate that KLIM1 covariance matrix estimator can lead to a higher classification accuracy, suggesting that KLIM1 is simple and good-enough for most cases. From these experiments, we also find that smoothing parameter values for KLIM were not an accurate requirement, as there exists a range of values in which a higher classification accuracy can be obtained. This range depends on training samples distribution. In most cases, however, smoothing parameter selection methods used in this chapter work quite well.

In comparing KLM1 and KLIM2, KLIM2 estimator led to the same or a higher classification accuracy than KLIM1 in poorly-posed problem, but its performance is not as good in ill-posed problem. One of the possible reason is that in ill-posed case, the computation of covariance matrix $\widehat{\Sigma}_j$ is highly variable, resulting in a large difference value between averaged quantity and single class quantity. This leads to strong regularization in estimating $\widehat{\Sigma}_Q$, consequently deteriorating KLIM2 estimation. Nevertheless, this phenomenon occurred only in cases in which classes are overlapped. When the classes are well separated from each other, the probability of $\mathbf{x}_i$ belonging to only one class will approach 1, resulting in $\eta \to 0$ and KLIM2 automatically reduces to KLIM1. In the case that we hope to take advantage of KLIM2, one possible method is to use pre-processed training samples, e.g., by a data dimension reduction technique such as Principal Component Analysis (PCA) [61] to reduce the dimension of the data set.

Another advantage of KLIM1 and KLIM2 is that they can be used to classify total un-labelled samples in small number and high dimension case, since they were derived based on minimizing KL divergence in which EM algorithm can be re-derived [59].

3.7 Summary

In this chapter, based on KL information measure, the KLIM covariance matrix estimation was derived and investigated for the classification problem. An efficient smoothing

parameter approximation formula was derived, and the approximation was found to be accurate for most cases in our experiments. With the KL information measure, total samples can be used to estimate smoothing parameter, making it less computation-expensive than using leave-one-out cross-validation method proposed in the literature. With the KL information measure based estimation method, more than half of the experiments show that the obtained KLIM estimator works well, as they show a little higher classification accuracy than RDA. Besides, in all experiments KLIM estimators are consistently better than QDA and LDA estimators.

Chapter 4

Cluster Number Selection in Small Sample Set Case

4.1 Introduction

In intelligent statistical data analysis or unsupervised classification, cluster analysis is to determine the cluster number or cluster membership of a set of given samples, $\{\mathbf{x}_i\}_{i=1}^N$ [62, 52, 63], by its mean vector, $\{\mathbf{m}_y\}_{y=1}^k$. In most cases, the first step of the clustering is to determine the cluster number. The second step is to design a proper clustering algorithm. In recent years, several clustering analysis algorithms have been developed to partition samples into several clusters, in which the number of clusters is *pre-determined*. The most notable approaches are, for example, the Mean Square Error (MSE) clustering and finite mixture model analysis.

The MSE clustering method typically is implemented by the well-known k-mean algorithm [62]. This method requires specifying the number of clusters, k, in advance. If k is correctly selected, then it can produce a good clustering result; otherwise, data sets cannot be grouped into appropriate clusters. However, in most cases the number of clusters is unknown in advance. Because it is difficult to select appropriate number of clusters, some heuristic approaches have been used to tackle this problem. The Rival Penalized Competitive Learning (RPCL) [64] algorithm has demonstrated a very good result in finding the cluster number. However, there is still no appropriate theory being developed [65, 66, 67].

In the mixture model cluster analysis, the sample data are viewed as two or more mixtures of normal (Gaussian) distribution in varying proportion. The cluster is analyzed by means of mixture distribution. The likelihood approach to the fitting of mixture models has been utilized extensively [68, 58, 49, 69, 70, 71, 50, 72, 73]. However, the determination of the appropriate cluster number still remains one of the most difficult problems in cluster analysis [51, 74].

The Bayesian-Kullback Ying-Yang (BYY) learning theory has been proposed in [75]. The BYY learning is a unified algorithm for both unsupervised and supervised learning which provides us a reference for solving the problem of selecting cluster number. The experimental results worked very well for a large set of samples when the smoothing parameter $h \to 0$ [76, 77]. However, for a relatively small set of samples, the Maximum Likelihood (ML) method with the Expectation-Maximization (EM) algorithm [57] for estimating mixture model parameters will not adequately reflect the characteristics of the cluster structure. In this way, the selected cluster number is incorrect. To solve the problem for the small set of samples, the BYY theory for data smoothing is developed in [46] is approach considers the nonparametric density estimation and the smoothing factor in the Parzen window.

In this chapter, we investigate the problem of determining the smoothing parameter and the model selection in clustering. We compare the parameter estimation result by using the bootstrap technique [78] and by using the smoothing EM algorithm. With this approach, the performance of the BYY model selection criterion for determining cluster number is greatly improved. Finally, we propose an efficient gradient descent smoothing parameter estimation approach that not only reduces the complicated computation procedure but also gives the optimal result.

4.2 Cluster Number Selection

First, we briefly review the finite mixture model and the BYY theory for model selection [76, 79].

4.2.1 Finite Mixture Model

Let us consider a Gaussian mixture model, the joint probability density which consists of k component Gaussians of which the mathematical expressions are the Eqs. (3.1-3.2) in Chapter 3.

Based on the given data set D, these mixture parameters can be estimated by maximum likelihood learning with EM algorithm.

4.2.2 BYY theory for finite mixture model and EM algorithm

As mentioned in [79, 80], unsupervised learning problems can be summarized into the problem of estimating joint distribution $P(\mathbf{x}, \mathbf{y})$ of patterns in the input space $\mathbf{X}$ and the representation space $\mathbf{Y}$. By the Bayesian Kullback Ying-Yang theory, we have the following Kullback–Leibler divergence [79]:

$$KL(M_1, M_2) = \iint P_{M_1}(y|\mathbf{x}) P_{M_1}(\mathbf{x}) \ln \frac{P_{M_1}(y|\mathbf{x}) P_{M_1}(\mathbf{x})}{P_{M_2}(\mathbf{x}|y) P_{M_2}(y)} d\mathbf{x} dy \tag{4.1}$$

where M_1 and M_2 are two different models.

The minimization of $KL(M_1, M_2)$ can be implemented by the *Alternative Minimization* procedure which alternatively minimizes one model while keeping other models temporarily fixed [79].

We can obtain a general form of KL function in the Gaussian mixture model case as:

$$KL(M_1, M_2) = \iint P(y|\mathbf{x}) p_{h_x}(\mathbf{x}) \ln \frac{P(y|\mathbf{x}) p_{h_x}(\mathbf{x})}{\alpha_y G(\mathbf{x}, \mathbf{m}_y, \Sigma_y)} d\mathbf{x} dy \tag{4.2}$$

To match two model M_1 and M_2 will lead to

$$P(y|\mathbf{x}) = \frac{\alpha_y G(\mathbf{x}, \mathbf{m}_y, \Sigma_y)}{p_{M_2}(\mathbf{x}, \Theta)}, \quad p_{M_2}(\mathbf{x}, \Theta) = \sum_{y=1}^{k} \alpha_y G(\mathbf{x}, \mathbf{m}_y, \Sigma_y) \tag{4.3}$$

the KL function becomes

$$KL(k, h, \Theta) = -\int p_{h_x}(\mathbf{x}) \ln p_{M_2}(\mathbf{x}, \Theta) d\mathbf{x} + \int p_{h_x}(\mathbf{x}) \ln p_{h_x}(\mathbf{x}) d\mathbf{x} \tag{4.4}$$

This function in fact is the system relative entropy function.

For mixture model parameter learning,

$$\Theta = \arg\min_{\Theta} KL(\Theta), \quad KL(\Theta) = KL(k, h, \Theta) \tag{4.5}$$

If KL function is minimized with respect to parameter Θ, the EM algorithm[57, 58] can be re-derived in the limit $h \to 0$. The mathematical expression of the EM algorithm are written as in Chapter 3 as Eqs. (3.3) and (3.12).

4.2.3 Model Selection Criterion

The determination of an appropriate number of clusters in a data set is one of the most difficult problems in clustering analysis [51, 74]. In the literature, there are several heuristically proposed information theoretical criteria. Following Akaike's pioneering work [81] in which an information criterion was first proposed for use in selecting the number of clusters in the mixture model cluster analysis. Similar studies include AICB [82], CAIC [83], and SIC [84]. These criteria combine the maximum value of the likelihood with the number of parameters.

The cluster number, k, is actually a structural scale parameter of the BYY system. From the BYY system, the BYY model selection criterion for determining the correct cluster number is derived in [76] as follows:

$$k = \arg\min_{k} J(k), \qquad J(k) = \begin{cases} J_1(k), \\ J_2(k), \end{cases} \tag{4.6}$$

$$J_1(k) = H_1(k) + J_2^g(k), \quad J_2(k) = \gamma_r H_1(k) + J_2^g(k) \tag{4.7}$$

$$0 \leq \gamma_r \leq 1 \tag{4.8}$$

where

$$J_2^g(k) = \sum_{y=1}^{k} \alpha_y \ln \sqrt{|\Sigma_y|} - \sum_{y=1}^{k} \alpha_y \ln \alpha_y \tag{4.9}$$

$$H_1(k) = \frac{1}{N} \sum_{i=1}^{N} \sum_{y=1}^{k} P(y|\mathbf{x}_i) \ln P(y|\mathbf{x}_i) \tag{4.10}$$

If taking $\gamma_r = 0$, we have $J_2(k) = J_2^g(k)$. In practice, we start with $k = 1$, estimate the parameter Θ by the EM algorithm based on the given samples, and compute $J(k)$. Then, we proceed to $k \to k + 1$, and compute $J(k)$ again. We continue this process after we gather a series of $J(k)$. The appropriate cluster number, k, is selected from the one with minimal $J(k)$.

4.3 Parameter Estimation with Bootstrap Technique

Although the model selection approach discussed above works well for a good size of data samples, we found out, from several experimental results, that the selected cluster number was not correct for a relatively small set of samples. The results are also incorrect with other theoretical information criteria mentioned above. The reason is that the MLE with the EM algorithm that estimates mixture model parameters will not reflect the characteristics of cluster structures adequately. As a result, it affects the correctness of determining the cluster number. In order to study the effect of parameter estimation on the BYY model selection, we incorporate the bootstrap technique with the EM algorithm in the MLE of mixture parameters as described in this section [78].

4.3.1 Bootstrap Technique

The bootstrap is a random resampling technique. In principle, the sampling process is described as following [26]:

Suppose the original data set is $D_o = \{\mathbf{x}_i\}_{i=1}^N$. We randomly draw a sample from this data set with probability $1/N$. The definition of random sampling allows a single sample $\mathbf{x}_i$ to appear more than once in the sample set. By this method, a bootstrap sample set, $D_1 = \{\mathbf{x}_{1i}^*\}_{i=1}^N$, is obtained by randomly sampling N times from the samples. Using the above resampling technique M times, we obtain M groups of data sets, $D_1, D_2, \cdots, D_M$. These are called bootstrap data sets. For each bootstrap data set, we calculate the maximum likelihood estimation of the finite mixture model parameter $\widehat{\Theta}_j$ using the EM algorithm. In this way, we can obtain the corresponding M sets of mixture model parameters, $\widehat{\Theta}_1, \widehat{\Theta}_2, \cdots, \widehat{\Theta}_M$, for M groups of data sets. This process is called the *parametric bootstrap* method.

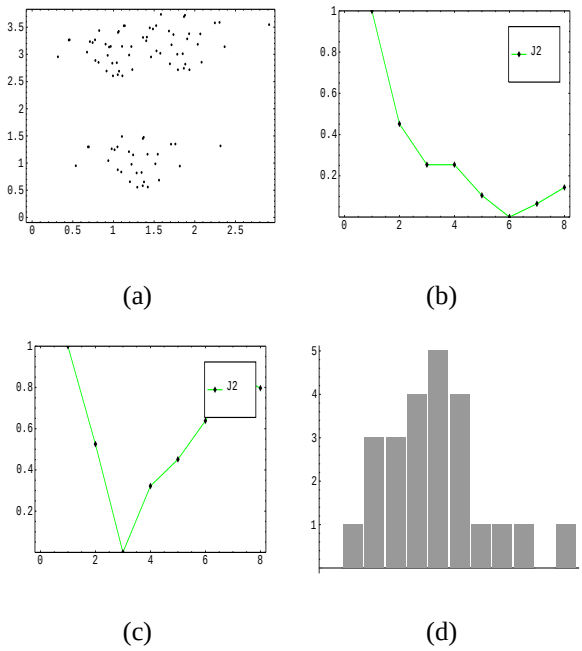
With maximum likelihood estimation, the parameter Θ has a normal distribution. The mean of estimated parameters is

$$\overline{\Theta} = \frac{1}{M} \sum_{j=1}^{M} \widehat{\Theta}_j \tag{4.11}$$

The true value of Θ is approximated by the mean value $\overline{\Theta}$.

4.3.2 Parameter Estimation with Bootstrap

In practical bootstrap implementation, the first step is to initialize the mixture parameters randomly. Since no *a priori* information is available, with random method the EM algorithm often converges to an undesirable local minima. In this work we propose a ”seed generation” method for initializing the mixture parameters to avoid the undesirable local minima. The ”seed generation” method generates the parameters systematically instead of randomly. Namely, we start with the whole data set to estimate the mixture parameters



(a) (b)

(c) (d)

Figure 4.1: The 2-D synthetic data set with three clusters.

The 2-D synthetic data set and the comparison of $J_2 - k$ curves with bootstrap to without bootstrap for three clusters. (a) data set, (b) $J_2 - k$ curve without bootstrap approach, (c) $J_2 - k$ curve with bootstrap approach, (d)A histogram plot of mean component of a cluster, which is similar to the normal distribution as expected.

with the EM algorithm. After repeating this procedure several times, we take the most probable parameters at each test. The obtained parameters are called "seeds" and are used as the initial values in the bootstrap sample estimation. Because the bootstrap data set is part of original data samples, the estimated value will be slightly different from the "seeds" value. In this way, the mixture parameters estimated with the bootstrap data set will be very close to the same local minima. There are two advantages for this "seeds" initialization approach. First, it guarantees to converge to close the same local minima. Secondly, the convergence speed is fast because only small variation is made.

Another problem in maximum likelihood estimation is that there is no label was given to clusters in the case $k \geq 2$ test. To avoid calculating the mean using different clusters parameters, we label clusters according to their mean center position. This guarantees that

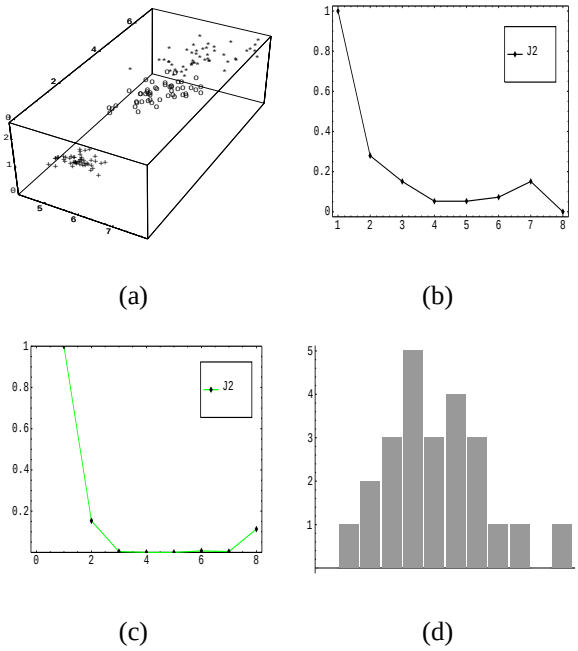
averaging the obtained parameters according to same label to compute the mean value become meaningful. Otherwise, it will get wrong results if different cluster parameters are used to compute the mean of a cluster.

We use some synthetic data sets and the real world IRIS data set to study the BYY criterion with bootstrap technique.

In the synthetic data set, we use $30 \times k$ samples which are randomly generated from k-Gaussian mixtures, in which each cluster consists of 30 data points. During the experiment, we vary k between 1 to 8. For each generated data set D, we resample it M times and generate M bootstrap data sets. In our work, $M = 25$ is used. Parts of the data sets and simulation results are shown in figures 4.1-4.2.

Figure 4.1 shows the experimental result of cost function $J_2(k)$ versus k for 3 different Gaussian mixture synthetic data sets. The second example is the IRIS data set. This is perhaps the best known database to be found in the pattern recognition literature. The data set consists of 3 classes of 50 samples, each with 4-dimension, where each class refers to a type of Iris plant. Within the data set, one class is linearly separable from the other two, and the other two are not linearly separable. Using the Gaussian mixture model approximation to investigate the structure of the IRIS data set, we found out that the approximation is somewhat difficult but is possible to get a good approximations. Figure 4.2 is the results of the IRIS data set. Although there are two overlapping clusters, the BYY criterion with bootstrap technique can determine the correct cluster number.

From the experiments, we find that if the parameters are estimated with the EM algorithm based on a given small data set without any correlation, the BYY criterion as well as other information criteria fail to detect correct cluster number. When the bootstrap technique together with ”seeds generation” is used to estimate the mixture parameters, the BYY criterion can select the correct cluster number. In most cases, reasonable results can be obtained with this combined technique.



(a) (b)

(c) (d)

Figure 4.2: Bootstrap experiment for Iris data set

The Iris data set and the comparison of $J_2 - k$ curves with bootstrap to without bootstrap estimated mixture parameters. (a) data set in x_1, x_3, x_4 axis view, (b) $J_2 - k$ curve without bootstrap approach, (c) $J_2 - k$ curve with bootstrap approach, (d) A histogram plot of mean component of a cluster, which is similar to the normal distribution as expected.

4.3.3 Summary for Bootstrap Technique

For small set of samples, by incorporating the bootstrap technique with the EM algorithm, we obtain a relatively robust performance for determining the cluster number with the BYY criterion and clustering. This illustrates that the proposed approach together with BYY criterion works well for small number sample case as long as mixture model parameters are properly estimated.

In the next section, we investigate the BYY data smoothing theory for parameter estimation.

4.4 BYY Data Smoothing Theory

Under the *conditional mean field approximation*, minimizing KL function corresponding parameter Θ will lead to the Smoothing EM (SEM) algorithm [80], where the updates in E-step and M-step are given as follows:

E-step:

$$P(y|\mathbf{x}_i) = \frac{\alpha_y G(\mathbf{x}_i, \mathbf{m}_y, \Sigma_y)}{\sum_{y=1}^{k} \alpha_y G(\mathbf{x}_i, \mathbf{m}_y, \Sigma_y)} \tag{4.12}$$

M-step:

$$\alpha_y^{new} = \frac{1}{N} \sum_{i=1}^{N} \frac{\alpha_y^{old} G(\mathbf{x}_i, \mathbf{m}_y, \Sigma_y)}{\sum_{y=1}^{k} \alpha_y^{old} G(\mathbf{x}_i, \mathbf{m}_y, \Sigma_y)} \tag{4.13}$$

$$\mathbf{m}_y^{new} = \frac{\sum_{i=1}^{N} P(y|\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^{N} P(y|\mathbf{x}_i)} = \frac{1}{\alpha_y N} \sum_{i=1}^{N} P(y|\mathbf{x}_i)\mathbf{x}_i \tag{4.14}$$

$$\widehat{\Sigma}_y^{new} = h\mathbf{I}_d + \frac{1}{\alpha_y N} \sum_{i=1}^{N} P(y|\mathbf{x}_i)[(\mathbf{x}_i - \mathbf{m}_y)(\mathbf{x}_i - \mathbf{m}_y)^T]. \tag{4.15}$$

We can find that the SEM algorithm is different from the ordinary EM algorithm in that it employs covariance estimation correction.

According to the principle of minimizing KL function, when $h \neq 0$, the smoothing parameter h should be estimated as

$$h = \arg\min J(h), \qquad J(h) = KL(k^*, \Theta^*, h) \tag{4.16}$$

4.5 Practical Implementation Consideration

The BYY data smoothing is a quite new technique. Two aspects for implementing BYY data smoothing should be discussed. One aspect is that we need to verify if the estimated

parameter for determining the cluster number with data smoothing. The other aspect is the selection of a proper smoothing parameter to estimate the mixture parameter.

Without loss of generality, we use a heuristic estimation of smoothing parameter h for fast implementation. For example, we can use $1/N$ of average distance approximation to estimate h value as follows:

$$h = \frac{1}{dNN^2} \sum_{i=1}^{N} \sum_{j=1}^{N} ||\mathbf{x}_j - \mathbf{x}_i||^2 \tag{4.17}$$

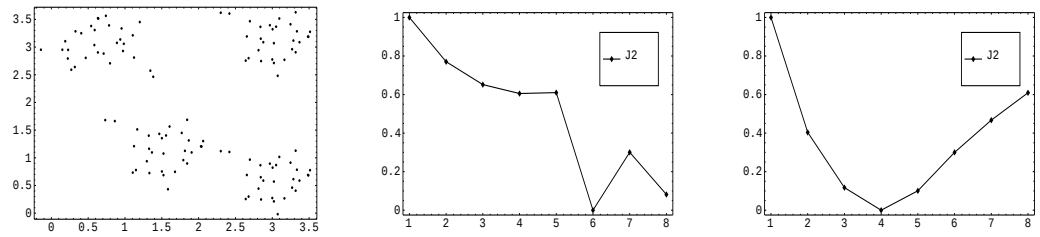
4.5.1 Experiments for Data Smoothing

In order to investigate the data smoothing effect, we first use some synthetic data sets to conduct the experiments.

The data sets have been generated under different conditions, such as different Gaussian mixtures, different mean $\mathbf{m}_y$, and different covariance Σ_y of each cluster. In order to eliminate the influence of the EM algorithm that converges to different local minima, we repeat the experiments with the same condition but with different initial parameter values for each test.

In computer experiments, we randomly generate $30\times k$ two-dimensional samples and $50\times k$ three-dimensional samples, where k is the number of Gaussian mixtures, varying from 1 to 8. Three data sets and their experimental results are shown in Figures 4.3- 4.5.

The cluster number selection criterion is when the cost function $J(k, \Theta)$ versus k reaches its global minimum point at $k = k^*$, where k is the candidate cluster number and k^* is the actual number of Gaussians in the finite Gaussian mixture model. Figure 4.3 shows the experimental result of the cost function $J_2(k)$ versus k for two dimensional Gaussian mixture data set. From Figure 4.3(b), we find that the ordinary EM algorithm over-estimates the actual cluster number (which gives us 6 clusters), while the data smoothing SEM algorithm gives a reasonable result. In Figure 4.3(c) the best cluster



(a) Data set (b) Without data smoothing (c) With data smoothing, $h = 0.0207$

Figure 4.3: The synthetic data set, 4 clusters

The 2-D synthetic data set and the comparison of J versus k (b) The result "without data smoothing" approach, (c) The result "with data smoothing" approach. The results show that 4 clusters is the best number.

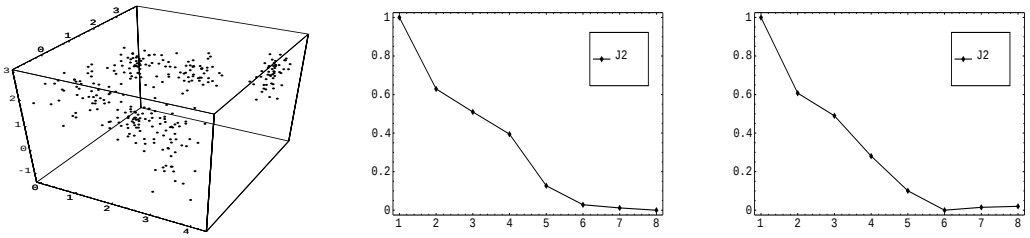
number is 4 from the J_2 versus k plot. Similarly in Figure 4.4(c), the best cluster number is 6, while the result of the ordinary EM algorithm shown in Figure 4.4(b) is 8. As for large samples case, the experiments show no obvious difference between $h = 0$ and $h \neq 0$ in search of the correct cluster numbers [77].

Another example is the same IRIS data set that was used in the bootstrap experiment previously. Figure 4.5 depicts the results of the IRIS data set. The experimental results show that the correct cluster number is 3 from Figure 4.5(c). We see that with data smoothing, the performance of cluster number selection is improved.

4.5.2 Smoothing Parameter Estimation

According to the principle of minimizing KL function, the optimal smoothing parameter can be obtained from Eq. (4.16). However, the evaluation of integration is computation-expensive. Therefore, we propose an approximation scheme in order to avoid the integration. In the following, we first review the quantized method which is recommended in [46]; then we derive a new gradient descent approximation for estimating this smoothing parameter h.

The Quantized Method:



(a) Data set (b) Without data smoothing (c) With data smoothing, $h = 0.00799$

Figure 4.4: The synthetic data set, 6 clusters

The 3-D synthetic data set and the comparison of J versus k (b) The curve "without data smoothing", (c) The curve "with data smoothing". The results show that 6 clusters is the best number.

On each of the quantized levels $h_r, r = 1, 2, ..., n_h$, we run the SEM algorithm to obtain a series of mixture parameter Θ^*. We then choose one h_r such that its corresponding value of $KL(\Theta^*, k, h_r)$ is the smallest. This approach is an exhaustive search method and usually is computation-expensive.

Gradient descent approach

For the gradient descent approach, we need to find an approximation for estimating parameter h. Referring to [46][1], we first briefly review the following equation.

$$h_x^2 = \frac{1}{d_x N N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \beta_i(\mathbf{x}'_j) ||\mathbf{x}'_j - \mathbf{x}_i||^2 |_{h_x^{old}} \qquad [46, eq.14b]$$

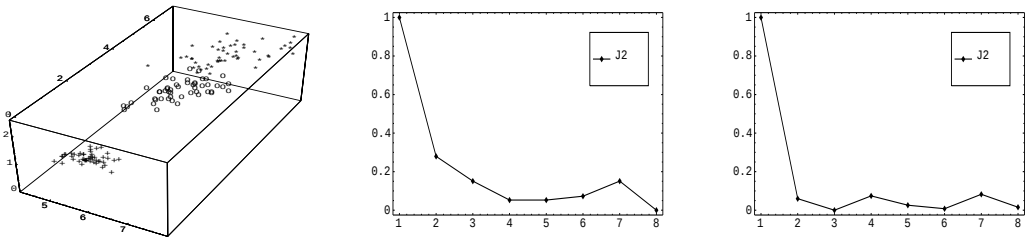
where

$$\beta_i(\mathbf{x}) = \frac{G(\mathbf{x}, \mathbf{x}_i, h_x^2 I_{d_x})}{\sum_{i=1}^{N} G(\mathbf{x}, \mathbf{x}_i, h_x^2 I_{d_x})}$$

Note $G(\mathbf{x}, \mathbf{x}_i, h_x^2 I_{d_x})$ is a Gaussian density function.

Now let us denote

[1]The h^2 in Eq. [46, eq.14b] is equal to h used in this report.



(a) 3-D view base x_1, x_3, x_4 axis

(b) Without data smoothing

(c) With data smoothing, $h = 0.0153$

Figure 4.5: The Iris data set, 3 clusters

The IRIS data set and the comparison of J versus k (a) IRIS data set in x_1, x_3, x_4 axis view. (b) The curve "without data smoothing", (c) The curve "with data smoothing". The results show that 3 clusters is the best number.

$$I_1 = \frac{1}{d_x N} \sum_{i=1}^{N} \int \beta_i(\mathbf{x}) ||\mathbf{x} - \mathbf{x}_i||^2 d\mathbf{x}$$

$$I_2 = \frac{1}{d_x N} \sum_{i=1}^{N} \int G(\mathbf{x}, \mathbf{x}_i, h_x^2 I_{d_x}) ||\mathbf{x} - \mathbf{x}_i||^2 d\mathbf{x}$$

Integrate I_2, we get $I_2 = h_x^2$.

Because $\beta_i(\mathbf{x})$ is positive and $\beta_i(\mathbf{x}) \leq G(\mathbf{x}, \mathbf{x}_i, h_x^2 I_{d_x})$ for $\forall\ \mathbf{x}$, it lead to

$$\beta_i(\mathbf{x}) ||\mathbf{x} - \mathbf{x}_i||^2 \leq G(\mathbf{x}, \mathbf{x}_i, h_x^2 I_{d_x}) ||\mathbf{x} - \mathbf{x}_i||^2$$

This indicates $I_1 \leq I_2 = h_x^2$, no matter how $\mathbf{x}_i$ distributed.

As we know, for any finite number of samples N', the summation value will be less than the integration value when the function is positive, i.e.,

$$(h_x^2)^{new} < I_1 \leq I_2 = (h_x^2)^{old}$$

From the above inequality, we can see that the approach always finds a h_x^2 regardless the data distribution and initialization. Because h_x^2 is non-negative, the value of h_x^2 will approach to zero eventually. Our experimental results verified this conclusion.

In order to cope with the above-mentioned efficiency, we derive a new equation for estimating smoothing parameter h based on Kullback-Leibler divergence.

Rewrite Eq. (4.4) in the following form:

$$\begin{aligned} KL(\Theta) &= \int p_{h_x}(\mathbf{x}) g(\mathbf{x}, \Theta) d\mathbf{x} + \int p_{h_x}(\mathbf{x}) \ln p_{h_x}(\mathbf{x}) d\mathbf{x} \\ &= J_0 + J_h \end{aligned} \tag{4.18}$$

where

$$J_0 \equiv \int p_{h_x}(\mathbf{x}) g(\mathbf{x}, \Theta) d\mathbf{x} \tag{4.19}$$

$$J_h \equiv \int p_{h_x}(\mathbf{x}) \ln p_{h_x}(\mathbf{x}) d\mathbf{x} \tag{4.20}$$

$$g(\mathbf{x}, \Theta) \equiv -\ln P_{M_2}(\mathbf{x}, \Theta) \tag{4.21}$$

If we use Gaussian kernel density

$$p_{h_x}(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I}_d) \tag{4.22}$$

then we have

$$J_0 = \int p_{h_x}(\mathbf{x}) g(\mathbf{x}, \Theta) d\mathbf{x} = \frac{1}{N} \sum_{i=1}^{N} \int G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I}_d) g(\mathbf{x}, \Theta) d\mathbf{x} \tag{4.23}$$

Because the $G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I}_d)$ term is inside the integral in above equation, when $\mathbf{x}$ moves away from $\mathbf{x}_i$, the function value becomes very small. So we can use Taylor expansion for $g(\mathbf{x}, \Theta)$ at $\mathbf{x} = \mathbf{x}_i$. When h is small, we can omit the higher order terms and only keep the first order term. By doing this, we have the following approximation of J_0 (detailed derivations are given in Appendix A):

$$\begin{aligned} J_0(\mathbf{x}, \Theta, h) \quad &\approx J_{01}(\mathbf{x}_i, \Theta) + h\frac{1}{2N}\sum_{i=1}^{N} \text{trace}[\nabla\nabla g(\mathbf{x}_i, \Theta)] \\ &= J_{01}(\mathbf{x}_i, \Theta) + hJ_r(\mathbf{x}_i, \Theta) \end{aligned} \tag{4.24}$$

$$KL(\mathbf{x}, \Theta, h) \approx J_0(\mathbf{x}_i, \Theta) + hJ_r(\mathbf{x}_i, \Theta) + \int p_{h_x}(\mathbf{x}) \ln p_{h_x}(\mathbf{x}) d\mathbf{x} \tag{4.25}$$

$$\frac{\partial}{\partial h} KL(\mathbf{x}, \Theta, h) \approx J_r(\mathbf{x}_i, \Theta) + \frac{\partial}{\partial h} \int p_{h_x}(\mathbf{x}) \ln p_{h_x}(\mathbf{x}) d\mathbf{x} \tag{4.26}$$

We know that

$$\frac{\partial}{\partial h} \int p_{h_x}(\mathbf{x}) \ln p_{h_x}(\mathbf{x}) d\mathbf{x} = \int \ln p_{h_x}(\mathbf{x}) \frac{\partial p_{h_x}(\mathbf{x})}{\partial h} d\mathbf{x} + \int \frac{\partial p_{h_x}(\mathbf{x})}{\partial h} d\mathbf{x} \tag{4.27}$$

where

$$\frac{\partial}{\partial h} p_{h_x}(\mathbf{x}) = -\frac{1}{2h} d_x p_{h_x}(\mathbf{x}) + \frac{1}{2N(h)^2} \sum_{i}^{N} G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I}_d) ||\mathbf{x} - \mathbf{x}_i||^2 \tag{4.28}$$

the last term in Eq. (4.27) can be calculated,

$$\int \frac{\partial p_{h_x}(\mathbf{x})}{\partial h} d\mathbf{x} = 0 \tag{4.29}$$

So equation (4.27) becomes

$$\begin{aligned}\frac{\partial}{\partial h}J_h &= \int \ln p_{h_x}(\mathbf{x})\frac{\partial}{\partial h}p_{h_x}(\mathbf{x})d\mathbf{x} \\ &= -\frac{d}{2h}\int p_{h_x}(\mathbf{x})\ln p_{h_x}(\mathbf{x})d\mathbf{x} \\ &+ \frac{1}{2N(h)^2}\int \ln p_{h_x}(\mathbf{x})\sum_i^N G(\mathbf{x},\mathbf{x}_i,h\mathbf{I}_d)||\mathbf{x}-\mathbf{x}_i||^2 d\mathbf{x}\end{aligned} \tag{4.30}$$

From

$$\frac{\partial}{\partial h}KL(\mathbf{x},\Theta,h) = 0 \tag{4.31}$$

and with mean center approximation (see Appendix A) we can obtain the new gradient decent formula for estimating h as:

$$h^{new} = h^{old} + \eta\delta h \tag{4.32}$$

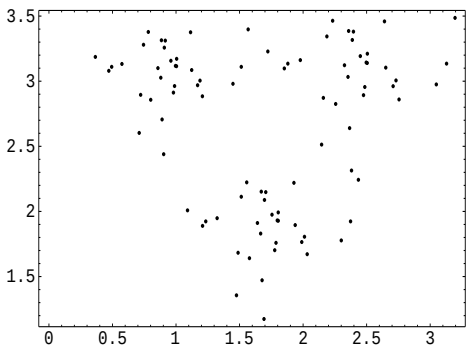
where η is a learning parameter and

$$\delta h \approx hJ_r - \frac{1}{2N}\sum_j^N (p_{h_x}(\mathbf{x}_j) - 1)\ln p_{h_x}(\mathbf{x}_j) \tag{4.33}$$

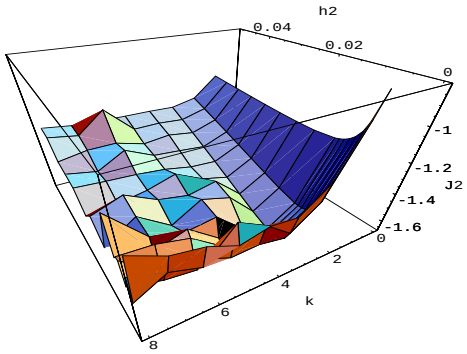
$$J_r(\mathbf{x}_i,\Theta) = \frac{1}{2N}\sum_{i=1}^N ||\sum_{y=1}^k P(y|\mathbf{x}_i)(\mathbf{x}_i - \mathbf{m}_y)^T\Sigma_y^{-1}||^2\}. \tag{4.34}$$

Let $\delta h = 0$, we obtain the following estimation equation for h:

$$h = \frac{\frac{1}{2N}\sum_j^N [p_{h_x}(\mathbf{x}_j) - 1]\ln p_{h_x}(\mathbf{x}_j)}{J_r(\mathbf{x}_i,\Theta)} \tag{4.35}$$



(a) The data set, 3 clusters, each cluster 30 points.



(b) The 3-D view of J_2 vs. h and k

Figure 4.6: The quantized method for the synthetic data set with 3 clusters.

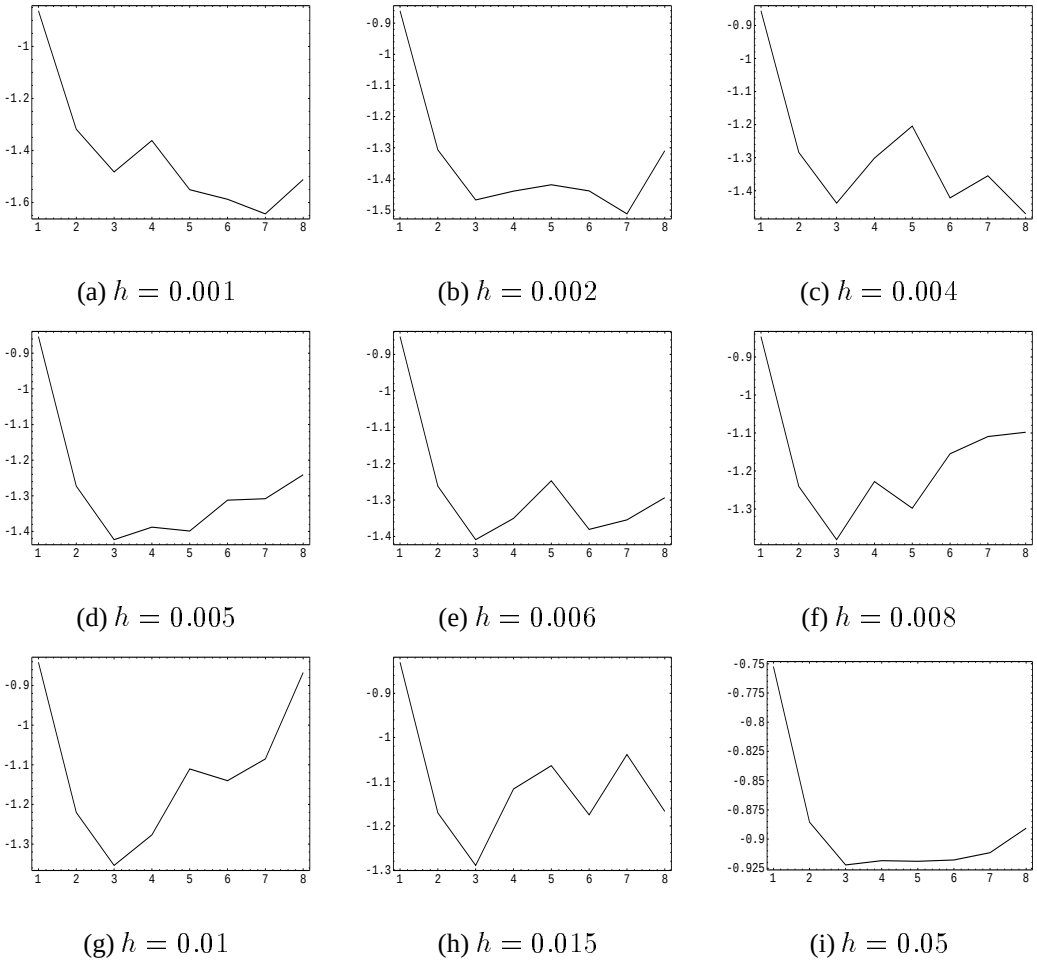
From the 3-D view of J_2 versus h, and k, we find that a local minima occurs at $k = 3$ and h is around 0.006.

4.5.3 Experiments

Now we present the experimental results for both the quantized and the gradient descent approximations of h.

In the experiments, we vary h value from 0.001 to 0.05 and k from 1 to 8. From Figure 4.6, we see that using the quantized method, k and h can be determined simultaneously. From these results, we obtain h value at minimal $KL(h, k^*, \Theta^*)$. In fact, with a range of h value, we can determine k^* from the J_2 plot. Usually, in the range of h value, we choose h to be the smallest one. Figure 4.7 shows that when h value is between 0.005 to 0.05, the correct k^* can be determined.

From these experiments, we know that by using the gradient method, the searching range is limited in a small region of h value compared to the quantized level method. Different k will result in different mixture model parameters, Θ; therefore it produces different h estimations. To find the optimal one, we can use the properties of J_1 and J_2 [46] to analyze the results and to determine k and h. We know that if $h = 0$ or h is too small, k will be over-estimated. If h is too large, the curve will be too smooth and



(a) $h = 0.001$ (b) $h = 0.002$ (c) $h = 0.004$

(d) $h = 0.005$ (e) $h = 0.006$ (f) $h = 0.008$

(g) $h = 0.01$ (h) $h = 0.015$ (i) $h = 0.05$

Figure 4.7: The J_2 versus k plots for different h values.

A correct cluster number can be detected from a range of h values. The data set used is the same as the one in Figure 4.6 (a).

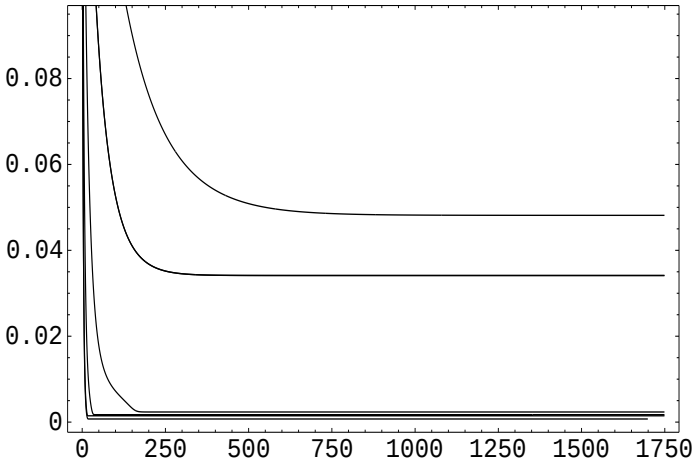
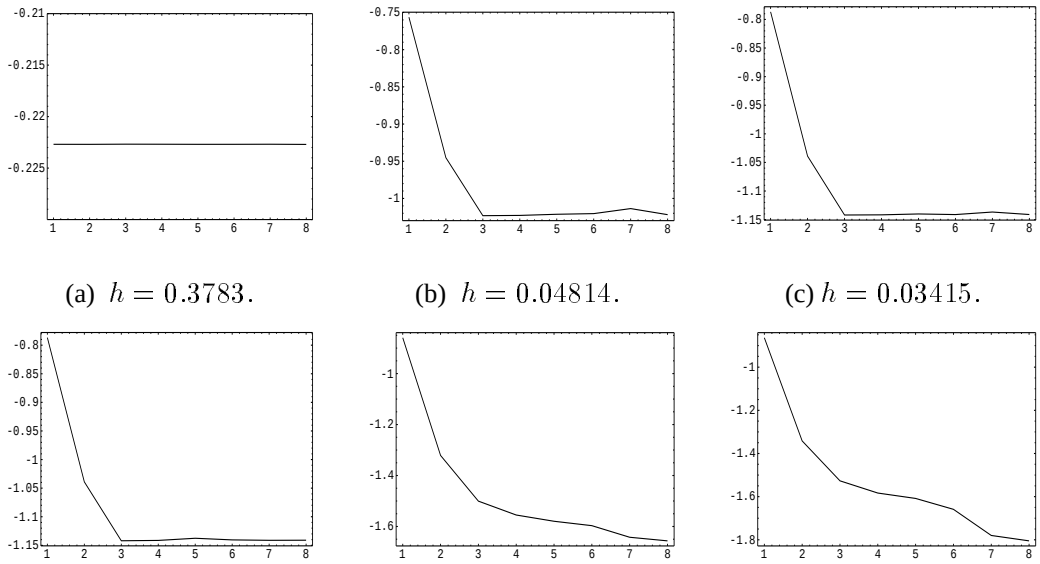


Figure 4.8: Gradient descent approximation of h.

Iterative epochs of finding h using different $J_r(k, \Theta^*)$ for different k. From the top to the bottom: curves are for $k = 2$, 3, 4, 5, 6, 7, respectively. Learning factor, η, for this experiment is 0.0001.

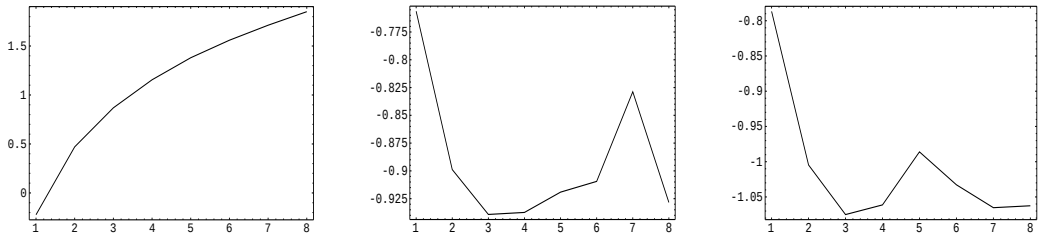


(a) $h = 0.3783$. (b) $h = 0.04814$. (c) $h = 0.03415$.

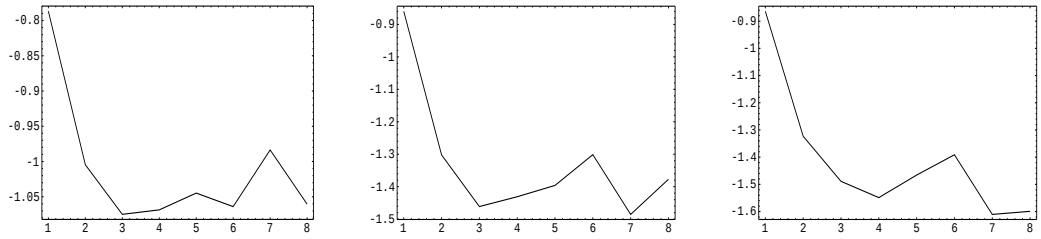
(d) $h = 0.03413$. (e) $h = 0.0024$. (f) $h = 0.0006$.

Figure 4.9: The results for the gradient descent approach that estimates h and the corresponding $J_1(k)$ curves.

If $h = 0.3783$, k is under-estimated, while for h less than 0.0024, the k is over-estimated.



(a) $h = 0.3783$. (b) $h = 0.04814$. (c) $h = 0.03415$.



(d) $h = 0.03413$. (e) $h = 0.0024$. (f) $h = 0.0006$.

Figure 4.10: The results for the gradient descent approach that estimates h and the corresponding $J_2(k)$ curves.

From these figures, same phenomenon was observed as in Figure 4.9.

k will be under-estimated (see Figure 4.9 for the comparison). In most cases, we can easily determine k and h from the experimental results. For example, in the experiments, the effect of data smoothing is somehow similar to increase of the number of samples. According to the theorem in [76], $J_1(k^*) < J_1(k)$ if $k < k^*$, and $J_1(k^*) = J_1(k)$ if $k \geq k^*$. Figures 4.8-4.10 show the results of the gradient descent approach. From Figure 4.10, we can easily find the possible k^* is 3. From these figures, we obtain the optimal h values, as 0.04814, 0.03413 and 0.03415, respectively, through the gradient descent approach.

4.6 Summary

In this chapter, we first review the BYY learning theory scheme for data smoothing. For a small set of samples, by combining the bootstrap technique with the EM algorithm, we ob-

tain a relatively robust performance for determining the cluster number. The experimental results, both with the bootstrap and with the SEM technique, show that the BYY-based model selection algorithm performs quite well in cluster number determination provided that the mixture model parameters are properly estimated.

The selection of the smoothing parameter h is a crucial problem. In this study, we derive an estimating formula for the smoothing parameter h. Often with the estimated h parameter, we can obtain a correct cluster number. Based on Kullback–Leibler divergence, we derive the gradient descent approach for estimating the smoothing parameter. The experiments indicate that the proposed approach works very well, and it is less computation-intensive compared to the exhausted search methods.

In fact, under the circumstance of different models, different sample sizes, and different data distributions, the determination of an appropriate cluster number using the Gaussian mixture model is very difficult. From our derivations and experiments, the BYY-based model selection criterion can select a reasonable cluster number even in a small set of samples.

Chapter 5

Ensemble Neural Networks

5.1 Introduction

For a given finite data set we usually train several neural networks with different architectures and different initial weights. Combining these neural networks together forms a committee of networks, which is called ensemble neural networks. This approach gives the advantage that prediction using the average of the committee gives a better generalization than the single network in the committee[85]. In the literature, it is a common practice to form ensemble networks with two steps[86]. First, several individual networks were generated by various methods. For example, we can use identical training data to train networks with different architectures, with different initial conditions, or various training algorithms [87, 85]. We can also use statistical techniques, such as bootstrap and cross validation[26], to partition training data set, then use the partitioned data set to train the networks. It is called *bagging predictors* if using bootstrap samples[88]. In the application of neural networks to a classification problem, some authors pay particular attention to this first step[89, 90, 91, 92, 93]. For the second step, we combine trained networks together with proper weighting coefficients to form the ensemble networks. The optimal ensemble averaging of the neural networks problem is also noted by some authors [87, 94, 95, 96]. In this chapter, we show a more general case based on the mixture models, and address the practical implement problem encountered in averaging the ensemble

networks in the parameter space.

The mixture model, or called mixture of experts (ME) [97] as well as its alternative model [98], employs a modular network structure and applies the soft-max gating network to control the individual network output. This approach will make the individual network to become an expert at some local input region, and produce biased estimations in the special input region. When the soft-max gating network is assumed irrelevant to input variables, ME reduces to the ensemble neural networks model. This model will globally optimize the ensemble networks instead of its separate steps. In this model, the weighting coefficient is based on the probability of members in the ensemble networks. Under some approximation, it reduces to the least-square-error-based weighting coefficient.

The relationship between mixture of experts and Combining Multiple Classifiers (CMC) has been addressed in paper [87] and [94]. In recent years, there are some new developments in the ensemble nets [99]. In the following section, we show that the ME is a more general model than the ensemble networks, and the result in paper [99] can be obtained as a special case of the ME.

5.2 Relationship between ME and Ensemble Neural Networks

5.2.1 Review of Mixture of Experts

Here we briefly review mixture of experts (ME) [97].

The ME can be described using the following conditional probabilities of network output $\mathbf{z}$ on given input $\mathbf{x}$ at parameter θ:

$$p(\mathbf{z}|\mathbf{x},\theta) = \sum_{i}^{K} g_i(\mathbf{x},\nu) p(\mathbf{z}|\mathbf{x},\theta_i). \tag{5.1}$$

In network output is the real case, $p(\mathbf{z}|\mathbf{x},\theta_i) = G(\mathbf{z}, g(\mathbf{x},W^i), \sigma_i^2 \mathbf{I}_{d_z})$ is Gaussian density,

$$G(\mathbf{z}, g(\mathbf{x}, W_i), \sigma_i^2 \mathbf{I}_{d_z}) = \frac{\exp\{-\frac{1}{2\sigma_i^2}||\mathbf{z} - g(\mathbf{x}, W_i)||^2\}}{[2\pi(\sigma_i^2)]^{d_z/2}} \tag{5.2}$$

where $\mathbf{x} \in R^{d_x}$, $\mathbf{z} \in R^{d_z}$, θ consists of ν and $\{W_i, \sigma_i^2\}_{i=1}^K$. The vector $g(\mathbf{x}, W_i)$ is the ith expert network output. The $g_i(\mathbf{x}, \nu)$ is scalar soft-max function given by

$$g_i(\mathbf{x}, \nu) = \frac{e^{\beta_i(\mathbf{x},\nu)}}{\sum_j^K e^{\beta_j(\mathbf{x},\nu)}} \tag{5.3}$$

In the above equation, $\{\beta_i(\mathbf{x}, \nu)\}_{i=1}^K$ are the outputs of the gating network.

In the alternative model of ME (called AME in the remaining text) [98], the soft-max function and mixture density are

$$\begin{aligned} g_i(\mathbf{x}, \nu) &= \frac{\alpha_i P(\mathbf{x}|v_i)}{\sum_j^K \alpha_j P(\mathbf{x}|v_j)} \\ P(\mathbf{x}|v_j) &= \alpha_j(v_j)^{-1} b_j(\mathbf{x}) \exp\{c_j(v_j)^T t_j(\mathbf{x})\} \end{aligned} \tag{5.4}$$

where $P(\mathbf{x}|v_j)$'s are density functions from the exponential family. The AME soft-max function represents a more general case of the gating network.

When we assume that training data set $D = \{\mathbf{x}_i, \mathbf{z}_i\}_{i=1}^N$, the log likelihood function for ME is,

$$\begin{aligned} L(\theta, K) &= \sum_{\mathbf{x},\mathbf{z} \in D} \ln p(\mathbf{z}|\mathbf{x}, \theta) \\ &= \sum_{i=1}^N \ln\{\sum_j^K g_j(\mathbf{x}_i, \nu) p(\mathbf{z}_i|\mathbf{x}_i, \theta_j)\} \end{aligned} \tag{5.5}$$

The parameter θ is estimated by the Maximum Likelihood(ML) learning method, where the ML estimation of parameter θ can be found using EM algorithm[57].

The idea of using a mixture model to address the ensemble nets problem can be traced back early in [94]. At that time, the ensemble nets is called Combining Multiple Classifiers (CMC). Later in [98], as a special case of the ME, CMC was further discussed.

Recently, some new results are presented in the ensemble nets study [99]. In the following we will show that the obtained result in [99] is still a special case of the ME.

5.2.2 Ensemble Networks

When we simply assume that gating network outputs $\beta_i(\mathbf{x}, \nu), i = 1, 2, \cdots, K$, are constants in the ME, Eq. (5.5) become,

$$\begin{aligned} L(\theta, K) &= \sum_{i=1}^{N} \ln\{\sum_{j}^{K} g_j(\mathbf{x}_i, \nu) p(\mathbf{z}_i|\mathbf{x}_i, \theta_j)\} \\ &= \sum_{i=1}^{N} \ln\{\sum_{j}^{K} \alpha_j p(\mathbf{z}_i|\mathbf{x}_i, \theta_j)\} \end{aligned} \tag{5.6}$$

where α_i's are combining coefficients, $\alpha_i \geq 0, \sum_{i=1}^{K} \alpha_i = 1$.

This equation represents the log likelihood for a distribution of K mixture density functions, or the so-called ensemble networks. Here we can see that the mixture model naturally includes the ensemble nets, and the weighting parameter is a special case of the soft-max network assumed to be constant. When the soft-max network is independent of input variables, each network in the ensemble networks is effective in the whole input region. The difference of each model in the ensemble networks may be caused by network structures, initial conditions, *et al*. Especially, when the weighting coefficient α_i equals to $1/K$, it becomes a simple average ensemble network.

We can rewrite Eq. (5.6) as

$$L(\theta, K) = \sum_{i=1}^{N} \ln\{\sum_{j}^{K} \alpha_j G(\mathbf{z}_i, g(\mathbf{x}_i, W_j), \sigma_j^2)\} \tag{5.7}$$

As a special case of the ME, the EM-like algorithm can be used to find the maximum likelihood solution for these specific parameters. This is shown as follows:

E-step: fix α_i^{old}, W_i^{old} and $(\sigma_i^2)^{old}$, and compute $h(i, j)$ by

$$h(i,j) = \frac{\alpha_i^{old} G(\mathbf{z}_j, g(\mathbf{x}_j, W_i), \sigma_i^2)}{\sum_l^K \alpha_l^{old} G(\mathbf{z}_j, g(\mathbf{x}_j, W_l), \sigma_l^2)} \tag{5.8}$$

M-step: find new estimations of parameter $\alpha_i^{new}, W_i^{new}$ and $(\sigma_i^2)^{new}$ with :

$$\begin{aligned} \alpha_i^{new} &= \frac{1}{N}\sum_{j=1}^{N} h(i,j) \\ W_i^{new} &= \max L(W_i, (\sigma_i^2)^{old}, K) \\ (\sigma_i^2)^{new} &= \max L(W_i^{old}, \sigma_i^2, K) \end{aligned} \tag{5.9}$$

For example, moving one step along the gradient descent direction, we get

$$\begin{aligned} W_i^{new} &= W_i^{old} + \gamma \frac{\partial L}{\partial W_i}|_{W_i = W_i^{old}} \\ (\sigma_i^2)^{new} &= (\sigma_i^2)^{old} + \gamma \frac{\partial L}{\partial (\sigma_i^2)}|_{(\sigma_i^2) = (\sigma_i^2)^{old}} \end{aligned} \tag{5.10}$$

where γ is a learning constant.

In this way, we can find network parameter W_i and variances σ_i^2, as well as weighting parameter α_i by adaptive learning. This is global optimizing ensemble networks instead of separate steps by using the ML estimation. Here we can see that ME is a more general and powerful model than the ensemble networks. Even in a simple case of ME-based model,(i.e, when network parameters are known or obtained by some other optimizing algorithm, and only α_i is learned [98]), we can also obtain the optimal weighting-average parameter. Following we will discuss some approximation cases for the weighting parameter.

One of the approximation for the weighting parameter α_i is a simple calculation with obtained parameter values instead of an adaptive computation. Suppose we have obtained every network parameters with the above EM-like algorithm or some other optimizing algorithms, then ${\sigma_i}^2$ can be obtained with the maximum likelihood estimation,

$$\sigma_i^2 = \frac{1}{N}\sum_{j=1}^{N}[\mathbf{z}_j - g(\mathbf{x}_j, W_i)]^2 \tag{5.11}$$

and let $\alpha_i^{old} = 1/K$. From Eqs. (5.8) and (5.9) we can obtain

$$\begin{aligned}\alpha_i &= \frac{1}{N}\sum_{l=1}^{N}\frac{G(\mathbf{z}_l, g(\mathbf{x}_l, W_i), \sigma_i^2)}{\sum_j^K G(\mathbf{z}_l, g(\mathbf{x}_l, W_j), \sigma_j^2)} \\ &= \frac{1}{N}\sum_{l=1}^{N}\frac{(\sigma_i^2)^{-d_z/2}\exp\{-\frac{1}{2\sigma_i^2}||z_l - g_{i,l}||^2\}}{\sum_j^K (\sigma_j^2)^{-d_z/2}\exp\{-\frac{1}{2\sigma_j^2}||z_l - g_{j,l}||^2\}}\end{aligned} \tag{5.12}$$

This is another form of weighting combination parameter in the maximum likelihood sense for the ensemble networks.

If network output is binary, for example, a two-class situation, we specify $p(\mathbf{z}|\mathbf{x}, \theta_i) = g_i^z(1-g_i)^{1-z}$, $z = \{0, 1\}$. In this case, with $\alpha_i^{old} = 1/K$, α_i becomes

$$\begin{aligned}\alpha_i &= \frac{1}{N}\sum_{l=1}^{N} h(i,l) \\ &= \frac{1}{N}\sum_{l=1}^{N}\frac{g_i^{z_l}(1-g_i)^{1-z_l}}{\sum_j^K g_j^{z_l}(1-g_j)^{1-z_l}}\end{aligned} \tag{5.13}$$

From this equation we can see that the dynamically weighted ensemble neural network [99] is ME's hard-cut approximation.

Now let us consider a special approximation for Eq. (5.7).

Suppose we have obtained the network parameters by using the above EM-like algorithm or some other optimizations procedure. We substitute averaged network function $f(\mathbf{x}, W) = \sum_i^K \beta_i g(\mathbf{x}, W_i)$ into Eq.(5.7) and regard variance the same in each model,

The Eq. (5.7) becomes

$$\begin{aligned} L(\beta) &= \sum_{l=1}^{N} \ln\{\sum_{i}^{K} \alpha_i G(\mathbf{z}_l, g(\mathbf{x}_l, W_i), \sigma_i^2) \\ &\approx \sum_{l=1}^{N} \ln\{\sum_{i}^{K} \alpha_i G(\mathbf{z}_l, f(\mathbf{x}_l, W), \sigma^2) \\ &\approx -\sum_{l=1}^{N} [\frac{1}{2\sigma^2}(\mathbf{z}_l - f(\mathbf{x}_l, W))^2 + \frac{d_z}{2} \ln \sigma^2] \end{aligned} \tag{5.14}$$

When omitting some constants irrelevant to β, we have

$$E(\beta) = \frac{1}{2} \sum_{l=1}^{N} [\mathbf{z}_l - \sum_{i}^{K} \beta_i g(\mathbf{x}_l, W_i)]^2 \tag{5.15}$$

This form represents the traditional ensemble networks with weighting coefficient β_i. Minimizing $E(\beta)$ can obtain the corresponding optimal weighting parameter β_i for averaging in functional space which was directly obtained by [85]:

$$\beta_i = \frac{\sum_j (\sigma_i \sigma_j)^{-1}}{\sum_l \sum_j (\sigma_l \sigma_j)^{-1}} \tag{5.16}$$

Here we can see that the traditional ensemble network is just a special case of the mixture model. However, the optimizing parameter algorithm is also presented in our model. The algorithm provides us not only every network function parameter, but also the ensemble weighting parameter.

The advantage of using the mixture-density-based model for ensemble networks is that it gives a global consideration, which results the optimal combining networks. Since the model includes the ordinary committee network, it also provides a better generalization

than single network. In fact, the expectation output for this mixture model is:

$$\begin{aligned} \overline{\mathbf{z}} &= \int \mathbf{z} p(\mathbf{z}|\mathbf{x}, \theta) d\mathbf{z} \\ &= \int \mathbf{z} \sum_{i}^{K} \alpha_i G(\mathbf{z}, g(\mathbf{x}, W_i), \sigma_i^2) d\mathbf{z} \\ &= \sum_{i}^{K} \alpha_i g(\mathbf{x}, W_i) \end{aligned} \tag{5.17}$$

This is the averaging ensemble networks. From the statistical point of view, it is obvious that Gaussian mixtures can provide a more accurate model for estimating generalization error than a single Gaussian.

Table 5.1: The ensemble network weighting parameter α_i, β_i and individual network σ_i^2 .

Network i	1	2	3	4	5	6	7	8
σ_i^2 ($\times 10^{-3}$)	5.5437	5.5721	5.4374	5.7276	5.1445	5.5665	5.6398	5.3832
α_i	0.0561	0.1926	0.2156	0.0484	0.3866	0.0249	0.0219	0.0534
β_i	0.1245	0.1242	0.1257	0.1225	0.1291	0.1242	0.1234	0.1263

5.2.3 Experiments for Averaging in the Functional Space

In the experiments, we use the three-layer neural network architecture. That is, $g(\mathbf{x}, W) = S(\sum W_{ij}^o S_j(\sum W_{jt}^h x_t))$, where $S(\cdot)$ is sigmoid activation function, and $W = \{W_{ij}^o, W_{jt}^h\}$ stands for network hidden and output connecting weights. The target sample can be viewed as being generated according to the signal-plus-noise model,

$$z_i = h(x_i) + \epsilon_i. \tag{5.18}$$

In our experiments, we construct $h(x) = \sin(x)\cos(3x) + x/3$ nonlinear function as the underlying function. 30 training samples are uniformly chosen from this function, and input region is restricted in $0 \leq x \leq \pi$, then scale this region to $0 \leq x \leq 1$. With Gaussian noise added to the output, the target function is $y(x) = \sin(x)\cos(3x) + x/3 + \epsilon$,

where ϵ is the noise with mean $m_e = 0$ and variance σ =0.1. In addition, another 30 data pairs are randomly generated with noise and used as test data.

We train the neural network with 10+i hidden neurons, $i = 0, 1, ..., K$, and the initial weight values are randomly initialized in the range of $[-1.5, 1.5]$. Namely, the architecture of each network is different from each other, and the connecting weight parameter is different from each other also. In practice, in order to speed up the convergence, only re-estimating values of α_i periodically in the M-step of the adaptive algorithm is needed.

The weighting parameters are shown in Table 5.1. In Table 5.1, all parameters are obtained based on the training data set. β_i value is computed according to Eq. (5.16).

When errors are uncorrelated, $\sigma_i \sigma_j \simeq 0$ if $i \neq j$. β_i will then become simple GEM form[85],i.e.,

$$\beta_i = \frac{{\sigma_i}^{-2}}{\sum_j {\sigma_j}^{-2}} \tag{5.19}$$

We can use maximum likelihood estimation for ${\sigma_i}^2$, regard $\frac{1}{\sigma_i^2}\{\mathbf{z}^n - g(\mathbf{x}^n, W_i)\}^2$ near a constant, and consider the case of $d_z = 2$ in Eq. (5.12). Under this rough approximation, we have

$$\alpha_i = \frac{{\sigma_i}^{-2}}{\sum_j {\sigma_j}^{-2}} \tag{5.20}$$

Which reduces into the form of a simple GEM.

Figure 5.1 displays the total test error vs. the network number K. In Figure 5.1, the sum-of-square errors are computed by using the test data set, which can be regarded as an estimated generalization error. It is seen that the ensemble networks total test error is small when using weighting parameter α_i than using β_i for most cases. This implies that α_i is somewhat optimal comparing with β_i. Similar results are obtained for several different data sets and for different network architectures.

The experimental results demonstrate that with an adaptive learning, better ensemble averaging parameters can also be obtained. The generalization error is less than using the least-square error based on an averaging parameter.

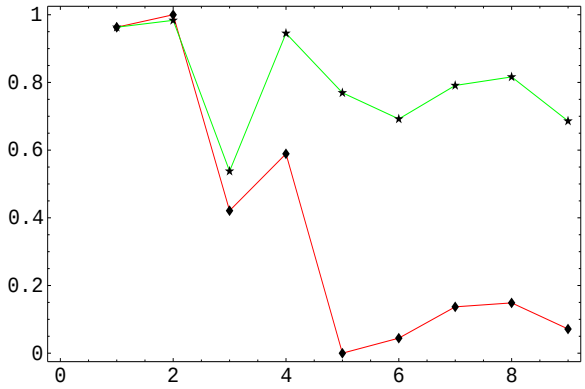


Figure 5.1: The scaled test errors vs. network number K

The lines are for ensemble nets weighted by α_i and β_i, respectively. The line with diamond is for α_i, and the line with star is for β_i

5.3 Averaging Connecting Weights

In the above experiments, we use different network structure to average ensemble networks. This can only be combined in the functional space. If we hope to make an efficient finite data usage, we can adopt some statistical resampling techniques, such as jackknifing, bootstraps and cross-validation[26]. That is, using resample techniques, we get m group data sets, D_1, D_2, ... , D_m, to train network, with corresponding parameters $\widehat{\Theta}_1, \widehat{\Theta}_2, ..., \widehat{\Theta}_m$. Among these parameters if only one is chosen and the others are discarded, the chosen network should have the best performance on the validation data set. However, it may not be the one with the best approach to Θ^* and it may not get the best performance on new test data.

In real applications, if we fix the network architecture, the parameters in a feedforward neural network take the form of connecting weights, which usually have many different

local minima. Therefore, the simply averaging the weights $\widehat{W}_1, \widehat{W}_2, ..., \widehat{W}_m$ cannot expect to improve the network performance. Thus, the method of combining the networks usually lies in network functional space instead of in parameter space. For Generalized Ensemble Method network function[85], $g_{fav}(\mathbf{x}, W)$ is

$$g_{fav}(\mathbf{x}, W) \equiv \sum_{i=1}^{m} \beta_i g_i(\mathbf{x}, W) = \sum_{i=1}^{m} \beta_i g(\mathbf{x}, \widehat{W}_i) \tag{5.21}$$

This expression is for averaging in functional space, applying a linear combination of trained networks. The advantage of the functional average is that networks with different architectures can then be combined together. But the combined network requires storing all trained network functions for ensemble purpose. While for averaging in parameter space, we have,

$$\overline{W} = \sum_{i=1}^{m} \beta_i \widehat{W}_i, \quad g_{pav}(\mathbf{x}, W) = g(\mathbf{x}, \overline{W}) \tag{5.22}$$

Apparently the last network architecture $g_{pav}(\mathbf{x}, W)$ is simpler than $g_{fav}(\mathbf{x}, W)$. When we make prediction using new data by $g_{fav}(\mathbf{x}, W)$, we need to input new data to every single network, and average all output data of these networks. While in $g_{pav}(\mathbf{x}, W)$, because the obtained network is still in the original model space, we only need to store the averaged weight parameter, and do not need to average all output of networks. From this point of view $g_{pav}(\mathbf{x}, W)$ is superior to $g_{fav}(\mathbf{x}, W)$ both in memory requirement and in recall speed.

5.3.1 Problems of Weights Average

There are two problems we should be dealt with weight average. One is how to get the proper weights, and the other is how to combine these weights.

Let us first consider the proper weights problem. Because the Back Propagation as well as other gradient decent algorithms sometimes fall into local minima, the value of $\widehat{W}$ not only depends on training data, but also depends on initial starting value. Even for the same data set, different starting values might result in different $\widehat{W}$, sometimes

far away from $\overline{W}$. If we use the gradient decent algorithm to find the connecting weight parameters, averaging the parameter cannot improve network performance. Figure 5.2 is an example to illustrate the network connecting weight tracking in a training processing. To our knowledge, so far ensemble network just averages in the functional space, and no one has investigated and implement parameter average to get a good generalization performance.

As for combining the weights, when $\overline{W} = \sum_{i=1}^{m} \beta_i \widehat{W_i}$ is used, we have an error function for finite data set

$$\begin{aligned} J &= \frac{1}{2N} \sum_{i=1}^{N} ||\mathbf{z}_i - g(\mathbf{x}_i, \overline{W})||^2 \\ &= \frac{1}{2N} \sum_{i=1}^{N} ||\mathbf{z}_i - g(\mathbf{x}_i, \sum\nolimits_{i=1}^{m} \beta_i \widehat{W_i})||^2 \end{aligned} \tag{5.23}$$

In principle, we can determine the optimal ensemble coefficient β_i by

$$\beta_i = \arg\min J(\mathbf{x}, \sum\nolimits_{j=1}^{m} \beta_j \widehat{W_j}), \text{ with } \sum_{i=1}^{m} \beta_i = 1$$

Because $g(x_i, \sum_{j=1}^{m} \beta_j \widehat{W_j})$ usual takes a nonlinear form, it is impossible to get explicit expression for β_i, and we must seek some approximate solution for β_i.

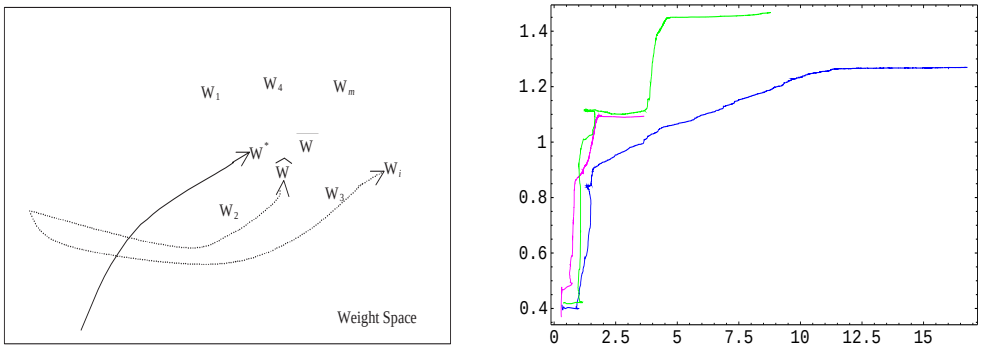
One special case is using linear approximation. When assuming square error of each network is uncorrelated and zero mean, we can obtain Eq. (5.19),

$$\beta_i = \frac{\sigma_i^{-2}}{\sum_j \sigma_j^{-2}} \tag{5.24}$$

To focus the problem in the parameter space, in the following we use this average weighting coefficient β_i instead of α_i

5.3.2 Solutions for Weights Average

As illustrated in Figure 5.2, there are many local minima in the parameter space. The difficulty of making parameter average becomes practically notable in that the weight for

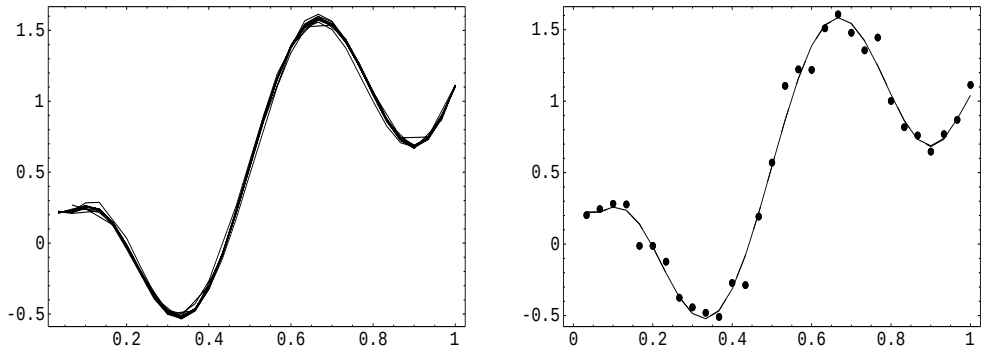


(a) (b)

Figure 5.2: Weight space with local minima.

(a) The relation of weight average with generalization. W^* stands for the network trained with large scale data. With W^* the network has good generalization ability. With finite data set trained network, the weight approaches to W_i. While regularizer forces weight approach to $\widehat{W}$. With resample techniques such as bootstrapping to train the network, we can get a set of weights $W_1, W_2, ..., W_m$. Averaging these weights we get the average weight $\overline{W}$ to approach W^*. In this way we improve network generalization ability. (b) The tracks of the weight vector when training network with BP algorithm. Horizontal axis stand for $|W_h|$, the length of hidden layer weight vector, while the vertical axis is $|W_o|$, the length of output layer weight vector. Three lines represent the three times of training network using the same data set, respectively. Each line represents that weight vector starting at small values and ending at different point with others in the weight space.

each network is not around the same local minima. How to guarantee the weight in the newly trained network is not far away from the other local minima? In our view, the possible solution is to develop a special methodology so that as long as the data set does not change a lot, the weights also does not change a lot. That is, to lock it at one minima. Based on this idea, we developed a novel method called PIL to train the feedforward neural network [100, 101]. This algorithm can find the global minima $\widehat{W}$, which only depend on the training data set. That is, as long as the training data is the same, the weight will be the same. By using PIL, every time we take one group partitioned data set to train the network, and get one $\widehat{W}_i$. The algorithm adopts in a batch way to train the



(a) (b)

Figure 5.3: The individual networks output and the ensemble network output with averaged weight parameter.

(a) The individual networks output. Because in each CV training data set there is only one data point different from others, the weight changes a little, and output also changes a little. This illustrates that our method indeed locks weights near one local minimum; (b) Dots represent training data, and the solid line is the ensemble network output with averaged weight parameter.

network, and calculate the pseudoinverse of the hidden output matrix. The details of the PIL algorithm will be presented in the next chapter. In the following section we would explain our method through an example and show the experimental results of the weights averaging ensemble network.

5.4 Experimental Illustration

One necessary condition to get good generalization is that the underlying function should be, in some sense, smooth. In the experiments, we adopt the $\sin(x)\cos(3x) + x/3$ nonlinear function as used in the last section, and one exponential function as training patterns.

Different from averaging in the above functional space experiments, leave-one-out cross-validation method is now used to partition 30 training samples, and 30 group-replicated training data sets, called CV samples, can be obtained. Each CV sample set has one validation data pair and 29 training data pairs. Therefore 30 networks are ob-

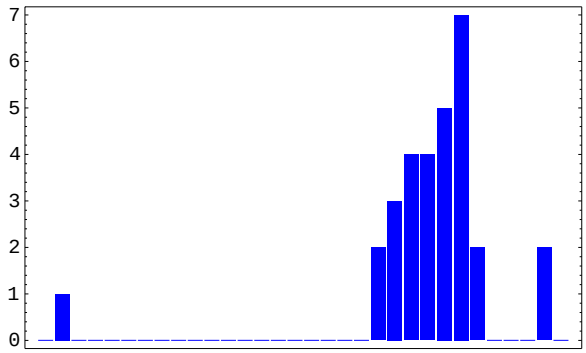


Figure 5.4: The output layer weights distribution of 30 networks.

tained with the CV training set.

We train the neural network with 29 hidden neuron numbers, which is equal to the training sample number used in training the expected network. After input matrix is multiplied with hidden weight matrix, nonlinear sigmoid transformation is applied to produce hidden output. The output layer weight is the multiplication of pseudoinverse of the hidden output and the desired output data.

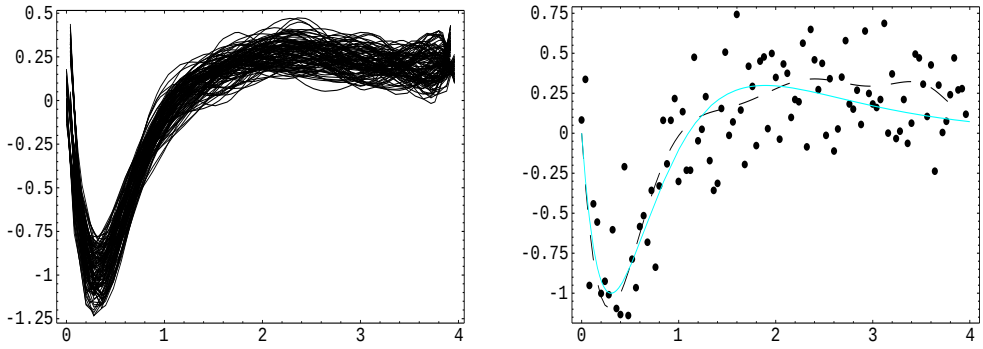
In order to reduce the fluctuation of the connecting weight, we use fixed hidden weights, This will not only reduce the complexity of the network, but also get good generalization. The hidden weight values are randomly initialized only once in the range of $[-0.5, 0.5]$.

We define the generalized error as the training error plus the test error. That is, for leave-one-out cross-validation, the single network generalized error is mean square error;

$$\begin{aligned} GE &= \frac{1}{2N}[\sum_{i \neq j}^{N}(\mathbf{z}_i - g(\mathbf{x}_i, W))^2 + (\mathbf{z}_j - g(\mathbf{x}_j, W))^2] \\ &= \frac{1}{2N}\sum_{i}^{N}(\mathbf{z}_i - g(\mathbf{x}_i, W))^2 \end{aligned} \tag{5.25}$$

For computing PAV, errors for averaging ensemble networks in parameter space, $g(\mathbf{x}, W)$ uses Eq. (5.22) instead. For computing FAV, errors for averaging ensemble networks in functional space, $g(\mathbf{x}, W)$ uses Eq. (5.21) instead.

The comparison of generalized errors are shown in Table 5.2. These results indicate



(a) (b)

Figure 5.5: Experiment for exponential function mapping

(a) The 100 individual networks output. (b) Dots are training data, the light gray line is the underlying function, and the dash line is the ensemble network output with averaged weight parameter. The ensemble net output is close to the underlying function.

Table 5.2: The networks Errors

	Min	Max	Mean	PAV	FAV
GE	0.00384	0.00494	0.00391	0.00392	0.00389

that the performance of our model is near the same as functional averaging. The reason is that we use partial linear approximation in our approach.

The individually trained network as well as the weight average ensemble networks output are shown in Figure 5.3a and 5.3b. We can see that the generalization indeed improves. Figure 5.4 is the distribution of output weights, in which we can easily see that our method indeed locks at one local minimum. In considering only 30 training samples, this is the best obtainable results based on a small training set.

Another example, using an exponential function, is shown in Figure 5.5.

The results demonstrate that our method not only can keep nonlinear mapping properties of neural networks, but also can realize the parameter space average. It is obvious that the results are dependent on training data and validation data. For this training data

set one may find a single network with minimal test error, but this is only for a particular data set, while the ensemble networks approach is more general and robust.

5.5 Summary

In this chapter, we discuss the relationship between mixture of experts and the ensemble networks. In a special case that soft-max function is independent of input variables, the ME reduces to ensemble neural networks. With this approximation, it is a global optimization of ensemble nets instead of individual members. Simultaneously, the weighting average coefficient for ensemble nets can be obtained through the EM algorithm. Experiments show that the ME is a more general and powerful model than ensemble networks in parameter estimation with maximum likelihood learning.

In the average parameter space experiments, we use the cross-validation-based partition training data set to train neural networks. By using a learning methodology to avoid networks problems falling into different local minima, we overcome the difficulty of averaging ensemble networks in the parameter space. Experimental results show that the adopted strategy is efficient in improving networks performance with finite training samples, and the ensemble network architecture is much simple than that in the functional space.

Chapter 6

Pseudoinverse Learning Algorithm

6.1 Introduction

Multilayer feedforward neural networks have already been found to be successful for various supervised learning tasks. Both theoretical and empirical studies have shown that the networks are of powerful capabilities for pattern classification and universal approximation [21, 102, 103]. Several adaptive learning algorithms for multilayer feedforward neural networks have recently been proposed [104, 45, 105, 106]. Most of these algorithms are based on variations of the gradient descent algorithm, for example, Back Propagation (BP) algorithm [104]. These algorithms usually have a poor convergence rate and sometimes fall into local minima instead of global minima [107]. Convergence to local minima can result from the insufficient number of hidden neurons as well as improper initial weight settings. However, slow convergence rate is a common problem of the gradient descent methods, including the BP algorithm. Various attempts have been made to speed up learning, such as proper initialization of weights to avoid local minima, and an adaptive least-square algorithm using the second order terms of error for weight updating [108]. There is another drawback for most gradient descent algorithms, namely, "learning factors problems", such as learning rate and momentum constant. The values of these parameters are often crucial for the success of the algorithm. Most gradient descent methods depend on these parameters which have to be specified by the user, as no

theoretical basis for choosing them exists. Furthermore, for applications which require high precision output, such as the prediction of chaotic time series, the known algorithms are often too slow and inefficient. In some cases, for example, like *stacked generalization* [109] which requires training a lot of networks to get level-1 training samples, it is very computation-time consuming when applying BP algorithm to perform the required task. Therefore, it is worthwhile to seek new algorithms which are suitable for the applications that require high precision output, whereas the network structure is less important.

In order to reduce training time and investigate the generalization properties of learned neural networks, this chapter presents a *Pseudoinverse Learning algorithm* (PIL), which is a feedforward-only algorithm. Learning errors are transferred forward and the network architecture is established. The previously trained weights in the network are not changed. Hence, the learning errors are minimized separately on each layer instead of globally for the network as a whole. The learning accuracy is determined by the number of layer. By adding layers to eliminate errors, all examples of a training set can be exactly learned. From a mathematical computational point of view, the algorithm is based on generalized linear algebraic method and employs matrix inner products and pseudoinverse operations.

6.2 The Network Structure and Learning Algorithm

6.2.1 The network Structure

Let us consider a multilayer feedforward neural network. The network has one input layer, one output layer and several hidden layers. The first layer with n neurons is the input layer including last neuron being a bias neuron of constant output. The last layer with m neurons is the output layer. The number of hidden layers depends on the desired learning accuracy and the used data set.

The weight matrix $\mathbf{W}^l$ connects layer l and layer $l+1$ with elements $w^l_{i,j}$. Element $w^l_{i,j}$ connects neurons i of layer l with neurons j of layer $l+1$. Note that the $\mathbf{W}^0$ matrix connects the input layer and the first hidden layer, whereas the $\mathbf{W}^L$ matrix connects the

last hidden layer and the output layer. We assume only the input layer has the bias neuron, while the hidden layer(s) and the output layer have no bias neuron. The nonlinear activation function is denoted as $\sigma(\cdot)$. For example, we can use the so-called sigmoidal function,

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{6.1}$$

whose output is in the range of (0,1), or a hyperbolic function

$$\sigma(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{6.2}$$

whose output is in the range of (-1,1) as an activation function.

Given a training data set $D = \{\mathbf{x}_i, \mathbf{o}_i\}_{i=1}^{N}$, let $(\mathbf{x}_i, \mathbf{o}_i)$ be the ith input-output training pair, where $\mathbf{x}_i = (x_1, x_2, \ldots, x_n) \in R^n$ is the input signal vector and $\mathbf{o}_i = (o_1, o_2, \ldots, o_m) \in R^m$ is the corresponding target output vector. For given N sets of input–output vector pairs as examples to be learned, we can summarize all given input vectors into a matrix $\mathbf{X}^0$ with N rows and $n + 1$ columns. Here the last column of $\mathbf{X}^0$ is a bias neuron of constant value θ. Each row of $\mathbf{X}^0$ contains the signals of one input vector. Note $\mathbf{X}^0 = [\mathbf{X}|\theta]$, where matrix $\mathbf{X}$ consists of all signal $\mathbf{x}_i$ as row vectors. All desired target output vectors are summarized into a matrix $\mathbf{O}$ with N rows and m columns. Each row of the matrix $\mathbf{O}$ contains the signals of one output vector $\mathbf{o}_i$.

The described networks are of multilayer perception type: They first compute an inner product of the incoming signals matrix with their respective weight matrix. Afterwards, an activation function is applied, producing the output of the neuron which is sent to all neurons of the following layer. In this designed network structure, the activation function is not applied to the output layer, so the last layer is linear.

Basically, the task of training the network means trying to find the weight matrix

which minimizes the sum-square-error function,

$$E = \frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1}^{m}||g_j(\mathbf{x}_i,\Theta)-o_{j,i}||^2, \tag{6.3}$$

where $g(\mathbf{x},\Theta)$ is a network mapping function and Θ is the network parameter set. Θ includes connection weight $\mathbf{W}$ and a bias parameter. In a three-layer structure case,

$$g_j(\mathbf{x},\Theta) = \sum_{i=1}^{N} w_{i,j}\sigma_i(\sum_{l=1}^{n} w_{i,l}x_l + \theta_i). \tag{6.4}$$

where θ_i is a bias value for the network input.

For simplifying, we can write the system error function in the matrix form,

$$E = \frac{1}{2N}\text{Trace}[(\mathbf{G}-\mathbf{O})^T(\mathbf{G}-\mathbf{O})]. \tag{6.5}$$

Propagating the given examples through the network, multiplying the output of layer l with the weights between layers l and $l+1$, and applying the nonlinear activation function to all matrix elements, we get:

$$\mathbf{Y}^{l+1} = \sigma(\mathbf{Y}^l\mathbf{W}^l), \tag{6.6}$$

and the network output should be

$$\mathbf{G} = \mathbf{Y}^L\mathbf{W}^L, \tag{6.7}$$

where we use superscript L to donate the last layer.

By examining the above equations and reformulating the task of training, the problem becomes

$$\text{minimize } ||\mathbf{Y}^L\mathbf{W}^L - \mathbf{O}||^2. \tag{6.8}$$

This becomes a linear least-square problem. If we can find the network weight parameter such that $||\mathbf{Y}^L\mathbf{W}^L - \mathbf{O}||^2 = 0$, we will have trained the neural network to learn all given examples exactly, that is, a perfect learning.

We focus our discussion on the last hidden layer now. For the sake of convenience, in the remaining of the chapter we drop superscript index L in Eq. (6.7).

6.2.2 Existence of the Solution

Now let us discuss the equation

$$\mathbf{YW} = \mathbf{O}, \quad \mathbf{W} \in R^{p\times m}, \quad \mathbf{Y} \in R^{N\times p}, \quad \mathbf{O} \in R^{N\times m} \tag{6.9}$$

When $p < N$, the system is an *underdetermined* system. Notice that such a system either has no solution or has an infinitive number of solutions.

If $\mathbf{Y} \in R^{N\times N}$ is invertible and has been learned in $L-1$ layer, then the system of Eq. (6.9) is, in principle, easy to solve. The unique solution for the last layer weight matrix is $\mathbf{W} = \mathbf{Y^{-1}O}$. If $\mathbf{Y}$ is an arbitrary matrix in $R^{N\times p}$, then it becomes more difficult to solve Eq. (6.9). There may be none, one or an infinite number of solutions depending on where $\mathbf{O} \in R(\mathbf{Y})$ space and whether $N-$ rank$(\mathbf{Y}) > 0$.

One would like to be able to find a matrix (or some matrices) $\mathbf{C}$, such that solution of Eq. (6.9) are of the form $\mathbf{CO}$. But if $\mathbf{O} \notin R(\mathbf{Y})$, then Eq. (6.9) has no solution.

In order to make our approach self-contained, we rewrite the relative linear algebra theorem in the following. The corresponding proof is from the reference book [110].

Theorem 1: The system $\mathbf{YW} = \mathbf{O}$ has a solution if and only if

$$\text{rank}([\mathbf{Y}, \mathbf{O}]) = \text{rank}(\mathbf{Y}). \tag{6.10}$$

Proof: Let $\mathbf{S}$ denote the column space of $\mathbf{Y}$, and let $\mathbf{S}^*$ denote the column space of $[\mathbf{Y}, \mathbf{O}]$, then $\mathbf{YW} = \mathbf{O}$ has a solution if and only if $\mathbf{O}$ is in $\mathbf{S}$. But $\mathbf{O}$ is in $\mathbf{S}$ if and only if $\mathbf{S}$ and $\mathbf{S}^*$ have the same dimension, i.e., $\mathbf{Y}$ and $[\mathbf{Y}, \mathbf{O}]$ have the same rank. (The rank of a matrix is equals to the maximal number of independent rows or columns.)

6.2.3 Pseudoinverse Solution is the Best Approximation

We intend to use the pseudoinverse solution for finding weight matrices, as the theorem from linear algebra states that pseudoinverse solution is the best approximation solution

for Eq. (6.9). It achieves a global minimum in the weight parameter space if the exact solution is reached.

Theorem 2: Suppose that $\mathbf{X} \in R^{p\times m}, \quad \mathbf{A} \in R^{N\times p}, \quad \mathbf{B} \in R^{N\times m}$, then the best approximate solution of the equation $\mathbf{AX} = \mathbf{B}$ is $\mathbf{X}_0 = \mathbf{A}^+\mathbf{B}$ (we use superscript "+" to denote the pseudoinverse form of a matrix).

Theorem 2 can be similarly derived from [110]). From the theorem 2 we get:

Corollary 1: The best approximate solution of $\mathbf{AX} = \mathbf{I}$ is $\mathbf{X} = \mathbf{A}^+$.

Based on the above analysis, we try to find the output layer weight in the following way. Let $\mathbf{W} = \mathbf{Y}^+\mathbf{O}$, the learning problem becomes $||\mathbf{YY}^+\mathbf{O} - \mathbf{O}||^2 = 0$, where $\mathbf{Y}^+$ is the pseudoinverse of $\mathbf{Y}$. This is equal to finding the matrix $\mathbf{Y}$ so that $\mathbf{YY}^+ - \mathbf{I} = \mathbf{0}$, where $\mathbf{I}$ is the identity matrix. Now the task of training the network becomes that of managing to raise the rank of matrix $\mathbf{Y}$ up to a full rank. As soon as $\mathbf{Y}$ becomes a full rank matrix, $\mathbf{YY}^+$ will be equal to the identity matrix $\mathbf{I}$. Note that since we multiply $\mathbf{Y}$ on the right side by $\mathbf{Y}^+$, it only requires the right inverse of $\mathbf{Y}$ to exist, and $\mathbf{Y}^+$ is not necessary to be a two-sided inverse of $\mathbf{Y}$. This means that $\mathbf{Y}$ needs not be a square matrix, but its number of columns should not be less than its number of rows. This condition requires that hidden neuron numbers be greater than or equal to N. If the condition is satisfied, we can find an exact solution for the weight matrix. In our network architecture design, we set the hidden neuron number to be equal to N. With this network structure, we can find the weight matrix which can exactly map to the training set.

6.2.4 The Pseudoinverse Learning Algorithm

According to the above discussion, we first let the weight matrix $\mathbf{W}^0$ be equal to $(\mathbf{Y}^0)^+$ which is an $(n+1) \times N$ matrix. Then we apply a nonlinear activation function, that is, to compute $\mathbf{Y}^1 = \sigma(\mathbf{Y}^0\mathbf{W}^0)$, then compute $(\mathbf{Y}^1)^+$, the pseudoinverse of $\mathbf{Y}^1$, and so on. Because the algorithm is feedforward only, no error will propagate back to the preceding layer of the neural network, and we cannot use a standard error form $E = \frac{1}{2N}\text{Trace}[(\mathbf{G} -$

$\mathbf{O})^T(\mathbf{G}-\mathbf{O})]$ to judge whether the trained network has reached the desired accuracy during the training procedure. Instead, we use the criterion $||\mathbf{Y}^l \cdot (\mathbf{Y}^l)^+ - \mathbf{I}||^2 < \mathbf{E}$. At each layer, we compute $||\mathbf{Y}^l\mathbf{Y}^{l+} - \mathbf{I}||^2$. If it is less than the desired error, we set $\mathbf{W}^L = (\mathbf{Y}^L)^+\mathbf{O}$ and stop the training procedure. Otherwise, let $\mathbf{W}^l = (\mathbf{Y}^l)^+$, add another layer, and feed forward previous layer output to the next layer again, until we reach the required learning accuracy.

To use any nonlinear activation function in the hidden nodes is to utilize the nonlinearity of the function, and to increase the linear independency among the column (row) vectors or, equivalently, the rank of the matrix. It is proven that sigmoid functions of a hidden layer of the network can raise the dimension of the input space up to the number of the hidden neurons [111]. So through a nonlinear activating action, the rank of the transformed matrix will be raised layer by layer.

In this way, we get a feedforward-only algorithm which reduces learning error on every layer. First we establish a two-layer neural network. If the given precision cannot be reached, a third layer is added to eliminate the remaining error. If the third added layer still cannot satisfy the desired accuracy, then another hidden layer is added again to reduce the learning errors, so on and so forth until the required accuracy is achieved. Mathematically, we can summarize the algorithm to the following steps:

Step 1. Set hidden neuron number as N, and let $\mathbf{Y}^0 = \mathbf{X}^0$.

Step 2. Compute $(\mathbf{Y}^0)^+ =$Pseudoinverse($\mathbf{Y}^0$).

Step 3. Compute $||\mathbf{Y}^l \cdot (\mathbf{Y}^l)^+ - \mathbf{I}||^2$. If it is less than the given error E, go to step 6. If not, go on to the next step.

Step 4. Let $\mathbf{W}^l = (\mathbf{Y}^l)^+$. Feed forward the result to the next layer, and compute $\mathbf{Y}^{l+1} = \sigma(\mathbf{Y}^l\mathbf{W}^l)$.

Step 5. Compute $(\mathbf{Y}^{l+1})^+ =$Pseudoinverse($\mathbf{Y}^{l+1}$), set $l \leftarrow l+1$, and go to step 3.

Step 6. Let $\mathbf{W}^L = (\mathbf{Y}^L)^+\mathbf{O}$.

Step 7. Stop training. The network mapping function is $\mathbf{G} = \sigma(\ldots\sigma(\sigma(\mathbf{Y}^0\mathbf{W}^0)\mathbf{W}^1)\ldots)\mathbf{W}^L$.

6.3 Adding and Deleting Samples

The proposed algorithm is a batch-way learning algorithm, in which we assume that all the input signals are available at the time of training. However, in real-time applications, as a new input vector is given to the network, the weight matrix must be updated. Or, we need to delete a sample from the learned weight matrix. It is not efficient at all if we recompute the pseudoinverse function of a new weight matrix in the PIL algorithm. When we assign the hidden neuron number to be equal to the number of training samples, adding or deleting samples is equivalent to adding or deleting hidden neuron number. Here we use neuron addition or deletion algorithms to efficiently compute the pseudoinverse matrix.

According to Griville's theorem [112], the first k columns of $\mathbf{Y}$ matrix consist of a submatrix, and the pseudoinverse function of this submatrix can be calculated from the previous $(k-1)$th pseudoinverse submatrix. That is,

$$\mathbf{Y}_k^+ = \begin{bmatrix} \mathbf{Y}_{k-1}^+(\mathbf{I} - \mathbf{y}_k\mathbf{b}^T) \\ \mathbf{b}^T \end{bmatrix} \tag{6.11}$$

where the vector $\mathbf{y}_k$ is the $k-$th column vector of the matrix $\mathbf{Y}$, while

$$\mathbf{b} = \begin{cases} (\mathbf{I} - \mathbf{Y}_{k-1}\mathbf{Y}_{k-1}^+)\mathbf{y}_k, & \text{if } ||\mathbf{I} - \mathbf{Y}_{k-1}\mathbf{Y}_{k-1}^+\mathbf{y}_k|| \neq 0 \\ \frac{(\mathbf{Y}_{k-1}^+)^T\mathbf{Y}_{k-1}^+\mathbf{y}_k}{1+||\mathbf{Y}_{k-1}^+\mathbf{y}_k||^2}, & \text{otherwise.} \end{cases} \tag{6.12}$$

It needs at most N times iterative cycles to obtain the pseudoinverse function of a matrix if there are N columns in this matrix. With this theorem, we can add the hidden neurons relatively easy to calculate the pseudoinverse matrix.

When a hidden neuron is deleted, the matrix needs to be updated. It is not efficient at all if we compute the pseudoinverse matrix from the beginning. Here we consider using bordering algorithm[113] to compute the inverse of the matrix[1]. The formula for

[1]For some cases if the matrix is singular, we can add the same dimension gaussian noise matrix to perturb the matrix. Because the noise is identical and independent distribution, the perturbed matrix will have the inverse function with probability one.

the pseudoinverse matrix can be obtained also from a partitioned matrix multiplication. Given the inverse of a $k \times k$ matrix, the method shows how to find the inverse of a $(k+1) \times (k+1)$ matrix, which is the same $k \times k$ matrix with an additional row and an additional column at its borders.

If the column vectors $\mathbf{y}_i$ in $\mathbf{Y}$ is linearly independent to each other, then by definition,

$$\mathbf{Y}^{+} = (\mathbf{Y}^T\mathbf{Y})^{-1}\mathbf{Y}^T. \tag{6.13}$$

Let $\mathbf{V} = \mathbf{Y}^T\mathbf{Y}$, and we can calculate $\mathbf{V}_{k+1}^{-1}$ from the prior $\mathbf{V}_k^{-1}$ without inverting a matrix. The algorithm is

$$\mathbf{V}_{k+1}^{-1} = \begin{pmatrix} \mathbf{V}_k^{-1} + \frac{1}{\alpha}\mathbf{v}\mathbf{v}^T & -\frac{1}{\alpha}\mathbf{v} \\ -\frac{1}{\alpha}\mathbf{v}^T & \frac{1}{\alpha} \end{pmatrix} \tag{6.14}$$

where $\mathbf{v} = \mathbf{V}_k^{-1}\mathbf{Y}_k^T\mathbf{y}_{k+1}$, and $\alpha = \mathbf{v}_k^T\mathbf{Y}_k^T\mathbf{y}_{k+1}$.

When deleting a vector from the matrix, consider the original matrix containing $k+1$ vector pairs. The key step is to compute $\mathbf{V}_k^{-1}$ from $\mathbf{V}_{k+1}^{-1}$. When the $(k+1)$th pair is deleted from the matrix, we rewrite $\mathbf{V}_{k+1}^{-1}$ as four partitions:

$$\mathbf{V}_{k+1}^{-1} = \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{b}^T & c \end{pmatrix} \tag{6.15}$$

where $\mathbf{A}$ is $k \times k$, $\mathbf{b}$ is $k \times 1$, and c is a scalar. By comparing with Eq. (6.14), it is apparent that $\mathbf{A} = \mathbf{V}_k^{-1} + \frac{1}{\alpha}\mathbf{v}\mathbf{v}^T$, $\mathbf{b} = (1/\alpha)\mathbf{v}$, and $c = 1/\alpha$. From these expressions, we find that the desired result is

$$\mathbf{V}_k^{-1} = \mathbf{A} - \frac{1}{c}\mathbf{b}\mathbf{b}^T. \tag{6.16}$$

The inverse of the $k \times k$ matrix can now be calculated from the $(k+1) \times (k+1)$ matrix. This is equivalent to deleting the last hidden neuron and updating the weight matrix.

This algorithm is very effective in the case of leave-one-out cross-validation partition training samples (CVPS). Because in each CVPS data set only one sample is different

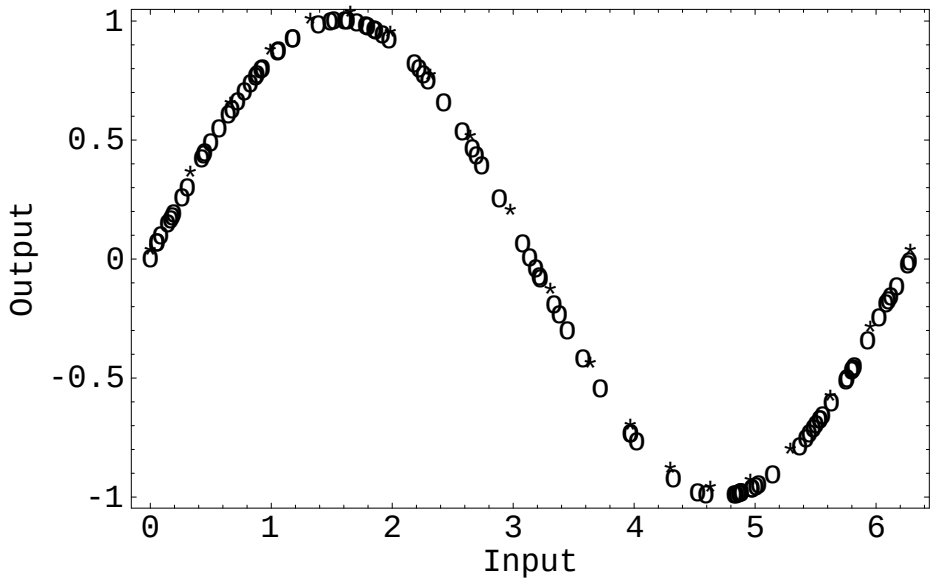


Figure 6.1: The trained network output for $y = sin(x)$ function mapping problem.

Where the "$*$" stands for training data output, while "o" stands for test data.

from the total sample set. We can first compute the inverse of the matrix which is obtained based on the full sample set, then at each time, move only one sample to the last column position, and use the above algorithm to delete this sample. In this way we can obtain the desired weight matrices on CVPS data sets efficiently.

6.4 Numerical Examples

6.4.1 Function Mapping Examples

The algorithm is tested with the following function mapping examples. The total learning error is defined as in Eq. (6.3), while average learning error is:

$$RMSE = \frac{1}{N}\frac{1}{m}\sqrt{\sum_{j}^{N}\sum_{i}^{m}(g_{j,i} - o_{j,i})^2} \tag{6.17}$$

where $o_{j,i}$ and $g_{j,i}$ are the desired network output and the actual output, respectively.

Example 1. Consider a nonlinear mapping problem of $Sine$ function by neural networks. For the training set, 50 input-output signals (x_i, y_i) pairs are generated with

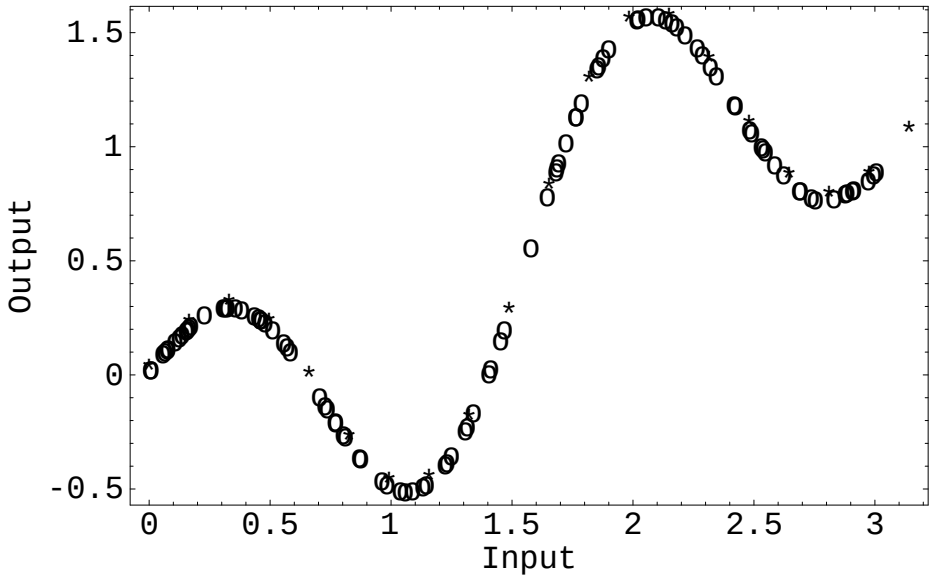


Figure 6.2: The trained network output for $y = sin(x)cos(x) + x/3$ function mapping problem.

Where the "$*$" stands for training data, while "o" stands for test data.

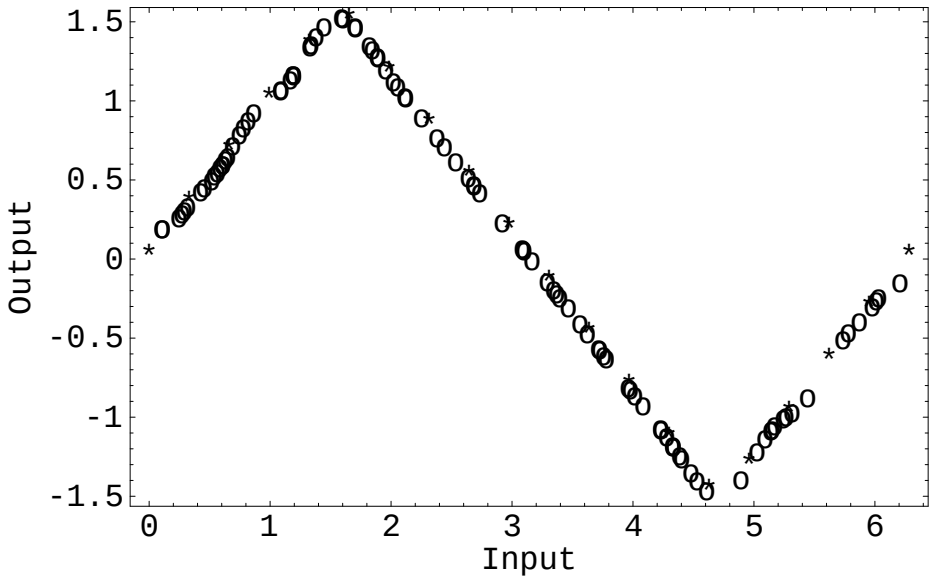


Figure 6.3: The trained network output for function defined in Eq. (6.19).

With 20 learning examples trained network output. The "$*$" stands for training data, while "o" stands for test data.

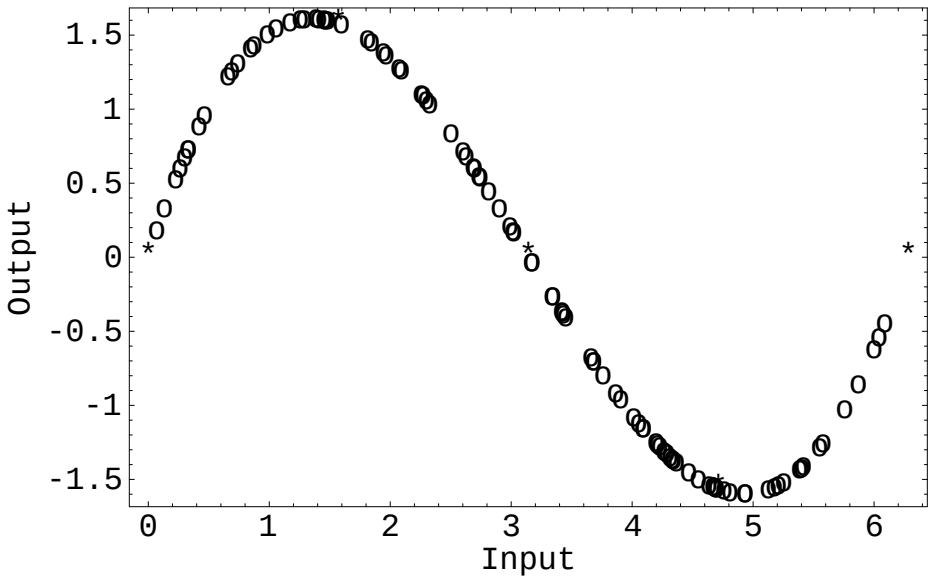


Figure 6.4: The trained network output for function defined in Eq. (6.19).

With only 5 learning examples trained network output. The "$*$" stands for training data, while "o" stands for test data.

$x_i = 2\pi * i/49$, for $i = 0, 1, 2, \cdots, 49$, and the corresponding y_i's are computed using $y_i = sin(x_i)$. The given learning error is $E = 10^{-7}$. If the learning error is $E < 10^{-7}$, we regard that perfect learning has been reached. For this problem, input neuron number is $n + 1 = 2$ including the bias one, output neuron is $m = 1$, and hidden layer neuron number is $N = 50$. After using the pseudoinverse learning algorithm proposed above, we reach the perfect learning when two hidden layers are added. The trained network altogether has four layers including input and output layers. The actual learning error is $E = 7.533 \times 10^{-18}$.

Example 2. This is the nonlinear mapping of eight input quantities x_i into three output quantities y_i problem, defined by Biegler-König and Bärmann in [114]:

$$\begin{aligned} y_1 &= (x_1 * x_2 + x_3 * x_4 + x_5 * x_6 + x_7 * x_8)/4.0 \\ y_2 &= (x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8)/8.0 \\ y_3 &= (1 - y_1)^{0.5} \end{aligned} \tag{6.18}$$

Table 6.1: Generalization ability test results. Given training error is 10^{-7}.

	Input range	N	Test Set N_1	Generalized E	Generalized $RMSE$	Max deviation
Example 1	0–2π	20	100	0.00049	0.00031	0.0121
Example 2	0–1	20	100	0.23481	0.00228	0.1838
Example 3	0– π	20	100	0.00452	0.00095	0.0899

Table 6.2: Generalization ability comparison of two examples.

	Input range	N	Test Set N_1	Generalized E	Generalized $RMSE$	Max deviation
Example 1	0–2π	5	100	0.47846	0.00971	0.16118
		50	100	2.439×10^{-9}	0.9843×10^{-7}	2.5148×10^{-5}
Example 4	0– 2π	5	100	5.00536	0.03164	0.56247
		50	100	0.41345	0.00455	0.21708

All three functions are defined for values between 0 and 1 and they produce values in this range. For the training set, 50 sets of input signals x_i are randomly generated in the range of 0 to 1, and the corresponding y_i's are computed using the above equation. The desired learning error we require is $E = 1.0 \times 10^{-7}$. When training is finished, only one hidden layer is added, and the actual learning error is $E = 3.573 \times 10^{-25}$ for this problem.

Example 3. Another functional mapping problem is $y = sin(x)cos(3x) + x/3$. Similar to Example 1, we use 50 examples with x_i in the region of 0 to π to train the network. Perfect learning is reached after two hidden layers are added. Actual learning error is $E = 4.734 \times 10^{-17}$.

6.4.2 Generalization

We also tested the generalization ability of trained networks to forecast function values of examples not belonging to the training set. For $Sine$ functional mapping, we train the network using 20 examples with $x_i = 2\pi * i/19$, for $i = 0, 1, 2, \cdots, 19$, and the corresponding y_i computed using $y_i = sin(x_i)$. After the network is trained, $N_1 = 100$ input signals x_i's are randomly generated within the range of 0 to 2π for testing the network, and the corresponding y_i's are computed using the trained network. Figure 6.1 shows the result, which is reasonably good. We have also tested Examples 2 and 3 with 20 examples training network and using 100 randomly generated input signals for testing. The results are shown in the Table 6.1 and Figure 6.2. In the tables of this chapter,

Max deviation is defined as the maximum value of the difference between real network output values and the desired values. Namely, Max deviation = $maximum|g_{j,i} - o_{j,i}|$, for $i = 1, 2, \cdots m$ and $j = 1, 2, \cdots, N_1$.

From Table 6.1, we see that $Sine$ function mapping problem has the least generalized errors. For further investigating the proposed network architecture and learning algorithm's response to unlearned data, we present the following example.

Example 4. A $Sine$-like piecewise linear function is defined by:

$$y = \begin{cases} x, & \text{if } 0 \leq x < \pi/2, \\ \pi - x, & \text{if } \pi/2 \leq x < 3\pi/2, \\ x - 2\pi, & \text{if } 3\pi/2 \leq x \leq 2\pi. \end{cases} \tag{6.19}$$

First, 20 examples with $x_i = 2\pi * i/19$, for $i = 0, 1, 2, \cdots, 19$, and the corresponding y_i's computed based on the above equation are used to train the network. Then 100 input signals randomly generated in the range of 0 to 2π are used to test the trained network. The result is shown in Figure 6.3.

When using five set examples $\{(0, 0), (\pi/2, 1), (\pi, 0), (3\pi/2, -1), (2\pi, 0)\}$ to train the network, we get a network structure which has one hidden layer with five hidden neurons. The learning error is $E = 3.314 \times 10^{-26}$. Afterward, 100 sets of input signals x_i randomly generated within the range of 0 to 2π are applied to test the network. The result is shown in Figure 6.4. From Figure 6.4, it can be seen that the network acts like a $Sine$ function. It should be reminded that the architecture and weight matrices are the same for Example 1 and Example 4 when using the above five examples. This result shows that the network forecast ability is better for smooth function when the data are in the range of training input space. When 50 examples with $x_i = 2\pi * i/49$, for $i = 0, 1, 2, \cdots, 49$, and the corresponding y_i computed with corresponding equation are used to train the network, 100 randomly-generated input signals in the range of 0 to 2π are applied to test the trained network, and the results are shown in Table 6.2. In Example 1 and Example 4, only W^L

matrix is different, and the other matrices are the same whereas 50 set examples are used to train the network. But the network's response to the same input matrix is totally different.

One point that needs to be mentioned is that the computation accuracy is machine-dependent due to recision of internal data representation. Furthermore, randomly-generated test data may vary when re-computed. Therefore, only the order of magnitude of the learning error is of practical interest.

6.5 Stacked Generalization

As we know, one of the important purpose to train a neural network is for generalization. When training samples set is small and deteriorates by random noise, the network is sometimes overtrained and becomes fitted to the noise, while overfitting the noisy data will degrade the prediction accuracy of the network.

The method of *stacked generalization* [109] provides a way of combining trained networks together, engaging partitioning of the data set to find an overall system with improved generalization performance. The idea is to train the level-0 networks first and then examine their behavior when generalizing. This provides a new training set for training the level-1 network.

The specific procedure for setting up the stacked generalization system is as follows. Let the complete set of available data be denoted by D. We first leave aside a single data point from D as a validation point, and treat the remainder of D as a training set. All level-0 networks are then trained by the training partition and their outputs are measured using the validation data point. This generates a single pattern for a new data set which will be used to train the level-1 network. The inputs of this pattern consist of the outputs of all the level-0 networks, and the target value is the corresponding target value from the original full data set. This process is repeated with a different choice for the data point which is kept aside. After cycling through the full data set of N points we have N patterns in the new data set, which is now used to train the level-1 network. Finally, all

of the level-0 networks are re-trained using the full data set D. Predictions on new data can now be made by presenting new input vector to the level-0 networks and taking their outputs as the inputs to the level-1 network, whose output constitutes the predicted output.

Mathematical expression is as the following for CVPS of stacked generalization. Given a training data set $D = \{\mathbf{x}_i, \mathbf{o}_i\}_{i=1}^N$, we randomly partition the data into K almost-equal subset $Ds_1, Ds_2, \cdots, Ds_K$. Define Ds_j and $Ds_{(-j)} = D - Ds_j$ to be the validation and training sets for the jth fold of a K-fold cross-validation. These are called level-0 models. Especially, if $K = N$, the validation set only has one sample, while training set contains $N - 1$ samples. This is called leave-one-out cross-validation.

Let $\mathbf{z}_i$ denote the validation output of the model M_j on $\mathbf{x}_i$. At the end of the entire cross-validation process, the data set assembled from the outputs of the models is

$$D_{cv} = \{\mathbf{z}_i, \mathbf{o}_i\}_{i=1}^N. \tag{6.20}$$

This is the level-1 data set used to train level-1 model. To complete the training process, the final level-0 model is derived using all the data in D.

The experiments show that with smooth function or piecewise smooth function, the trained network generalization performance is good with stacked generalization. The examples also illustrate that generalization can be expected when the underlying function is sufficiently smooth. However, for noisy data, if the network is overtrained (overfitting to noise), the generalization will be poor. Using stacked generalization can not improve the network performance when overtrained networks are engaged. The reason is that the overtrained network is biased to particular training samples; therefore, forecasting the values which are not in the training set will be far away from the expected values.

In order to investigate the properties of the stacked generalization technique in noisy data case, we adopt real world data sets in further experiments. The data sets are Sys1 and Sys3 software failure data applied for software reliability growth modelling in [47].

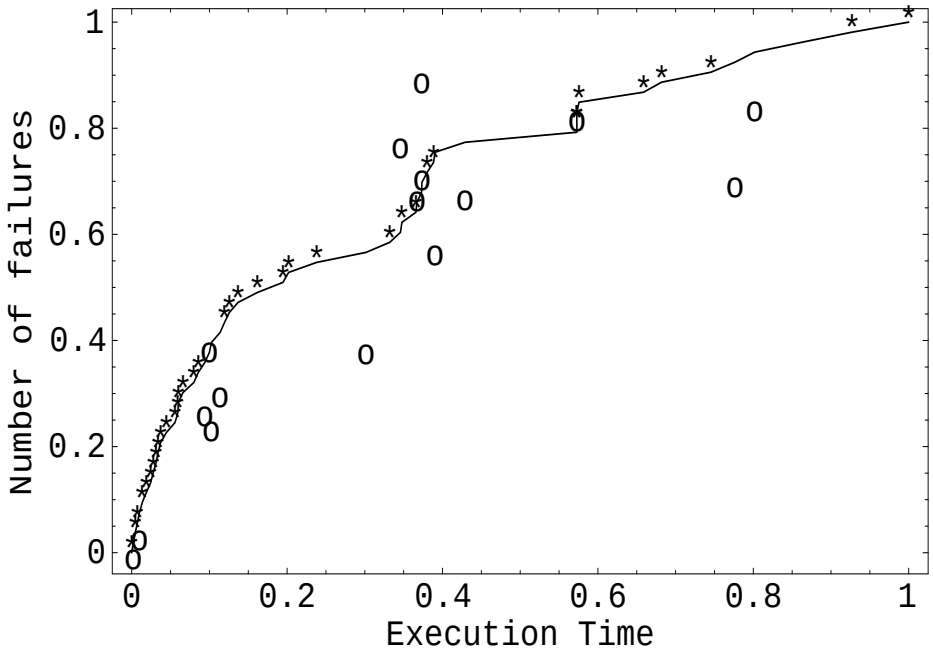


Figure 6.5: The neural network model trained with software reliability Sys1 data set (normalized).

Solid line is original data, "$*$" stands for training data samples, while "o" stands for test data samples. Because of overfitting the training samples, the network generalization is poor.

Sys1 data set contains 54 data pairs. In the experiment, we partition the data into two parts: training set and test set. The training set consists of 37 samples which are randomly drawn from the original data set. The remaining 17 samples consist the test set. The data set are normalized to the range of [0,1]. Normalizing is a standard procedure for data preprocessing. In this problem, the network input is normalized successive failure occurrence times, and the network output is the accumulated failure number. During training, each input sample x_t at time t is associated with the corresponding output value o_t at the same time t. This kind of training is called generalization training [115].

Figure 6.5 shows the experimental result for software reliability growth modelling trained by using data set Sys1, which is one of the level-0 network output. Figure 6.6 shows the stacked generalization output for Sys1 data set. Because of overfitting the training samples, the level-0 output strays away. These samples are not in level-1 training

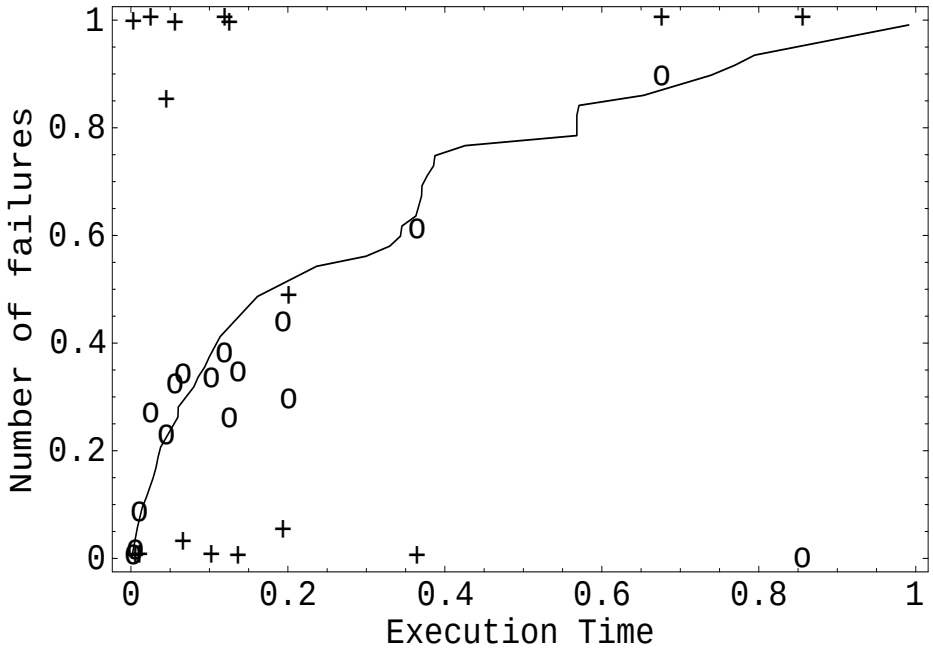


Figure 6.6: The stacked generalization output for Sys1 data set (normalized).

Solid line is the original data, "o" stands for test data output of level-0 neural network, while "+" stands for test data output of level-1 network output. The results are also poor.

data either, and the level-1 network outputs are further away from the desired values. The generalization ability is not improved by stacked generalization because of overfitting to the noise. Here we can see that when overfitting to the noise occurs, stacked generalization is not a suitable technique for improving network generalization performance. Poor generalization ability is not what we expected, so we should seek for the methods that can avoid the overfitting in noisy data cases.

As we have mentioned early in this chapter, PIL algorithm is eliminating learning errors layer by layer. For the generalization problem, we do not expect to realize the perfect learning. Therefore, we may adopt the strategy like early stopping to employ a three-layer neural network structure.

Figure 6.7 shows the experimental result by adopting three-layer structure trained with data set Sys1, which is one of the level-0 network output. To avoid overfitting, the training

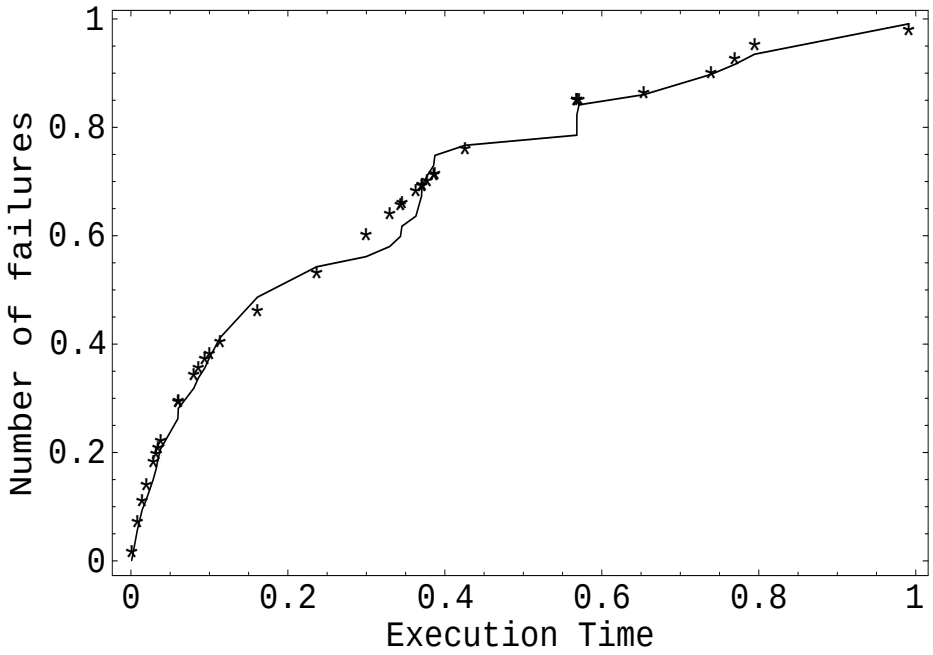


Figure 6.7: The three-layer network trained with software reliability Sys1 data set (normalized).

Solid line is the original data and "$*$" stands for training data samples. Training accuracy is not very high and overfitting is avoided.

error is not small, and the network outputs for training samples are not completely fitting the target values. Compared with perfect leaning error which is 2.3×10^{-9}, the training error is now 0.0034. This introduces the bias to the training samples, and the output tend to be a smooth curve.

Figure 6.8 shows the stacked generalization output for Sys1 data set. In this case, with stacked generalization, the total sum-of-square test error is 0.0152. While without stacked generalization, the total sum-of-square test error is 0.0434. Therefore, generalization ability is improved by stacked generalization.

Another data set is Sys3. In this data set, altogether there are 278 data pairs. In the experiment, we partition the data into a training set and a test set. The number of training data is about $2/3$ of the total data number, consisting of randomly drawn 186 samples from the original data set. The remaining 92 samples form the test set.

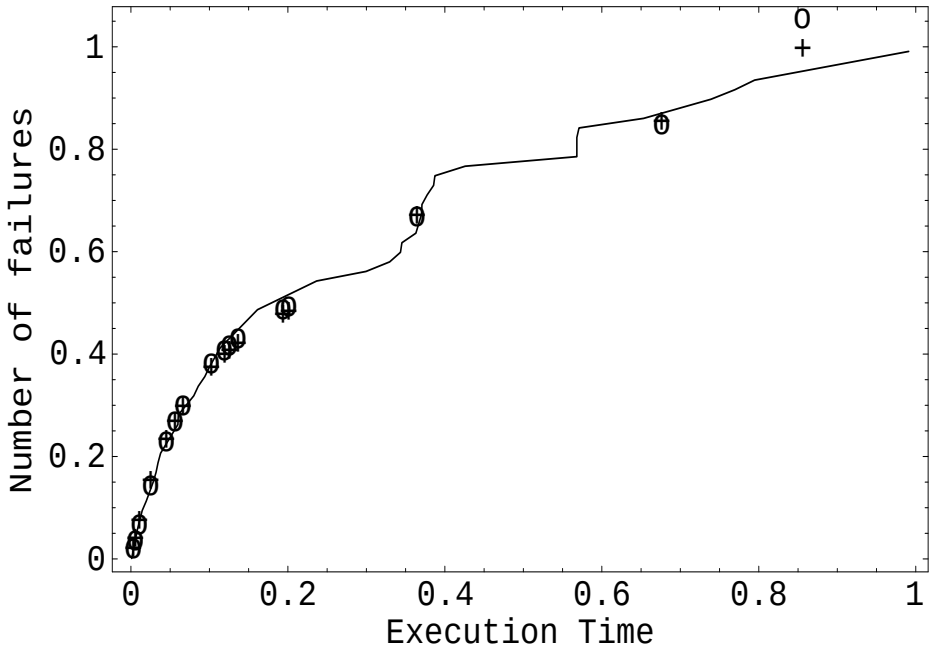


Figure 6.8: The stacked generalization output for Sys1 data set (normalized).

Solid line is the original data, "o" stands for test data output of level-0 neural network, while "+" stands for test data output of level-1 network output. Generalization is improved at the cost of introducing training bias.

If we assign the training error as 10^{-7}, after two hidden layers are added, the final training error reaches the order of 10^{-14}. But with this trained network, the test error (20.329) is large. Figure 6.9 shows the results.

Now we still use leave-one-out CVPS to train level-0 neural networks for stacked generalization. At this time, the three-layer network structure is adopted. For individual network, the training error is about 0.0442, while the test error is 0.0221. Figure 6.10 shows the individual network training output, while Figure 6.11 is for stacked generalization results.

From these real-world experimental results, we can see that it is at the cost of introducing the bias (training error) to reduce the variance (generalization error) [8]. For most generalization problems the stacked generalization can be expected to reduce the generalization error rate. For example, in the Sys1 experiment, the test error is 0.0434

Table 6.3: Training error and generalization error for software reliability growth model data set

Data Set		Sys1	Sys3
Training number		37	186
Test number		17	92
Individual net (4-layers)	Training error	3.49×10^{-4}	1.47×10^{-14}
	Test error	58.003	20.329
Individual net (3-layers)	Training error	0.0181	0.0442
	Test error	0.0434	0.0215
Stacked (level-1)	Test error	0.0152	0.0221

without stacked generalization, while the test error reduces to 0.0152 with stacked generalization. However, for some particular data set such as Sys3, stacked generalization dose not show significant improvement (test error is reduced from 0.0221 to 0.0215), but the computation time is dramatically increased. The results are summarized in Table 6.3.

For a large-scale data set, one of the well-known techniques is *divide-and-conquer* method. That is, partition the data set into subsets, so as to reduce the individual network size. We can also use other methods such as ensemble networks [85] to improve the network performance and then apply weight parameter average to reduce the network size [116]. For example, we can employ k-fold CVPS to train neural networks. As presented in the previous chapter, the ensemble neural network size is reduced by averaging in parameter space.

6.6 Discussions on PIL Features

In this section, we discuss the characteristics of the proposed Pseudoinverse learning algorithm.

On examining the algorithm, it can be seen that we do not need to consider the question of how the weight matrix should be initialized to avoid local minima. We just feed forward examples to get a weight matrix and its solution. The algorithm will not converge to a local minima because the pseudoinverse solution achieves a global minima. This is different

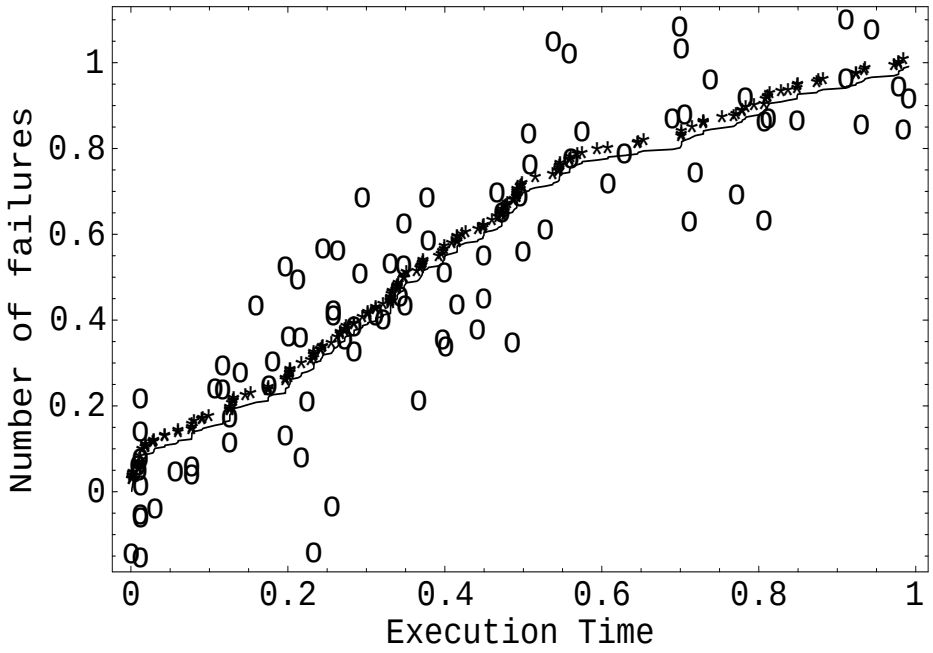


Figure 6.9: The network output for Sys3 data set (normalized).

Solid line is the original data and "o" stands for test data output. Because of overfitting the training samples, the network generalization is poor.

from the BP algorithm.

Furthermore, in PIL, the output layer is a linear layer. The resulting advantages are two-fold. First advantage is that the output values are not restricted to the region between -1 and +1. They can be any finite values. Second, we do not need to calculate the inverse of the activation function. It does not need to be a invertible activation function. In this aspect, the PIL is different from either the BP algorithm or other gradient descent algorithms.

As a comparison, the BP algorithm requires user-selected parameters, such as step size or momentum constant. These parameters have an effect on the learning speed. There is no theoretical basis which guides us how to select these parameters to speed up learning. In PIL, on the other hand, such a problem does not exist.

Another important feature of the algorithm is that desired output matrix $\mathbf{T}$ is embedded in the weight matrix $\mathbf{W}^L$ which connects last hidden layer and output layer. This

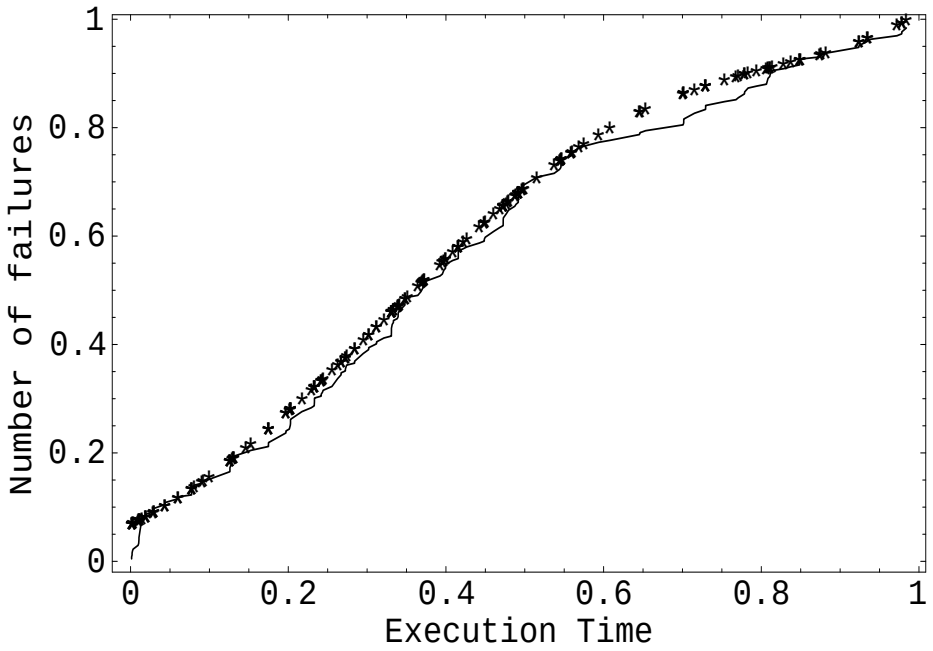


Figure 6.10: The three layer network model trained with software reliability Sys3 data set(normalized).

Solid line is the original data and "$*$" stands for training data samples. Training accuracy is not very high and overfitting is avoided.

gives us a very easy and fast way to get the weight matrix for different target output, as long as input matrix is the same. For example, after we have trained the network to learning $Sine$ function mapping in the region from 0 to 2π, we only need recalculate the $\mathbf{W}^L$, in order to get $Cosine$ function mapping problem in the same region with *Sine* function. On the other hand, for BP algorithm, it is necessary to train whole network again to get all weight matrices for $Cosine$ function mapping though input matrix is the same with $Sine$ function.

It can also be seen that the training procedure is in fact the processing of raising the rank of the weight matrix. When a matrix of some hidden layer output becomes full rank, the right inverse of the matrix can be obtained, thus completing the training procedure. From this learning procedure, it is obvious that no differentiable activation function is needed. We only require that the activation function can perform nonlinear transform to

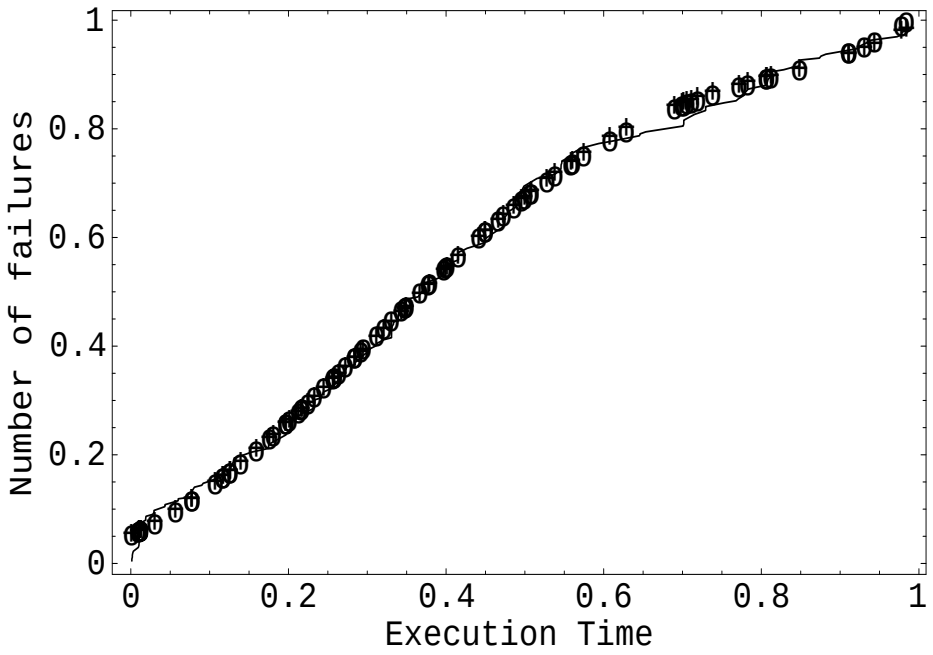


Figure 6.11: The stacked generalization output for Sys3 data set (normalized).

Solid line is the original data, and "o" stands for test data output of level-0 neural network, while "+" stands for test data output of level-1 network output. Generalization is improved at the cost of introducing training bias.

raise the rank of the weight matrix. Nevertheless, a sigmoidal-like nonlinear function is used in this chapter since its transformation has been proven to be capable of raising the rank of a matrix [111].

Because the PIL algorithm is based on the nonlinear function transformation to raise the matrix rank, it will fail if there two or more input vectors are identical in the input matrix. But this case can be eliminated through preprocessing input patterns. The previously proposed algorithm [100] can perform perfect learning with the fast learning speed for some problems. When the rank r of the input matrix is approximately equal to the number of input training patterns N, it is easy to reach perfect learning without further training. When $r \ll N$, it can give good estimated initial weight matrix, but it is still necessary to adopt some training algorithm in order to reach high learning accuracy. The PIL algorithm developed in this chapter improves this drawback.

Another characteristics is that if the input matrix has rank N then a right inverse exists, and we will get a linear network with only two layers. If we give learning error $E = 1.0 \times 10^{-7}$ to Example 1, and when N is greater than 2 and less than 10, it is necessary to add one hidden layer in order to reach the learning accuracy. If N is greater than 10, it is necessary to add two hidden layers. In Example 2, when N is less than 10, we will get a linear network with input and output layer only. The situation of Example 3 is the same as in Example 1 because the input matrices have the same rank. For most problems, with two hidden layers, the network can reach the required high learning accuracy. From the above examples, we see that the network layer number is not only dependent on learning accuracy, but also on the data to be learned. One thing we should address is that after the nonlinear transformation, the degree of rank change is data dependent. It is a more difficult problem to formulate a universal theory to determine how many layers are needed for the perfect learning. To reduce the network complexity, if we add a same dimension gaussian noise matrix to perturb the transformed matrix in step 4 of the PIL algorithm, the perturbed matrix will have the inverse with probability one because the noise is an identical and independent distribution. In such a strategy, we can constrain the hidden layers to at most two to reach the perfect learning. However, the trained network generalization will be degraded with noisy data set. In fact, high learning accuracy is not needed for some real-world tasks.

We have not compared the overall performance of this algorithm with other gradient descent algorithms. Obviously, the number of iterations is not a valid metric considering the fact that the calculation complexity per iteration is not the same for any of the algorithms. However, if we consider the CPU time cost on training network to reach the same high learning accuracy using the same machine, the PIL algorithm is much faster than other gradient descent algorithms in its learning speed. For example, we use the same machine (Sun Ultra 5/270 workstation) and the same software environment (Mathematica software) to train one neural network with the data Sys3 to reach the learning accuracy as

high as 10^{-14}, it takes less than 7.8 seconds (including display time) when using the PIL algorithm. As a contrast, it requires more than 10 hours of computation time when using the BP algorithm to reach the same results.

6.7 Summary

The pseudoinverse learning algorithm was introduced in this chapter. The algorithm is more effective than the standard BP and other gradient descent algorithms for most problems. The algorithm does not contain any user-dependent parameters whose values are crucial for the success of the algorithm. This algorithm is especially suitable for functional mapping and pattern recognition problems. When considering its learning speed and accuracy, the PIL algorithm is most competitive to other gradient descent algorithms in real-time or near real-time applications for practical use. The algorithm is tested on case studies with the stacked generalization applications to software reliability growth modelling data. The fast learning property of the PIL algorithm makes it possible for us to investigate the computation-intensive generalization techniques more efficiently.

Chapter 7

Application: Automatic Image Segmentation

7.1 Introduction

Image segmentation is an important aspect of computer vision. The goal of it is to partition a given image into some regions corresponding to different objects or the background. Also, it is a basic step for high-level image understanding and interpretation. There are a wide variety of image segmentation techniques [117], among which feature space clustering is one of the most popular methods. Pixels of the same segment can usually be characterized by certain features. These features are quantified into feature variables so that pixels of the same segment essentially have similar values of the feature variables, and pixels of different segments have dissimilar values. Then image segmentation can be performed by clustering the feature space and mapping each point back to the spatial pixel.

Among various clustering techniques like the well known k-mean algorithm, competitive learning, etc, the finite mixture of densities, in particular mixture of Gaussian model, has been widely used in many practical situations. The maximum likelihood approach has been utilized extensively to the fitting of finite mixture models [62, 58]. This approach has attracted considerable interest in the image segmentation field in recent years [118, 119, 120].

In paper [118, 119, 120], the authors use color or grey level feature space, or Gauss-Markov random field in image domain. And by assuming data points are generated from a finite mixture distribution, they estimate the probability density using EM algorithm or Generalized EM algorithm with pre-assigned cluster number in feature space. With the learned probability density function, Bayesian pixel classification method was used to produce the image segmentation in paper [120].

In fact, in the feature space clustering method to image segmentation, the number of segment to be yield can be considered as the number of cluster, k, in the feature space. In the clustering methods mentioned above, k has to be specified in advance. If k is correctly selected, good clustering result can be yielded, otherwise, data points cannot be grouped into appropriate clusters and image segmentation cannot be performed appropriately. To determine a reasonable region number is one of difficult things in machine learning. This problem affects the ability to automatically interpret images by a machine, which has been one of the major challenges in computer vision. In the past, most of the work use pre-assigned number of regions or heuristics to determine the number of regions.

In this chapter, we report that apply BYY model selection criterion to determine region number to perform automatic image segmentation. The algorithm we used in clustering is the EM algorithm on finite mixture model.

Apparently, the distribution of clusters depend on the color space selection, therefore determined region number is variable for different color space. The influence of the color space selection on region number determination also experimentally explored in this chapter.

7.2 Background

The application of finite mixture model to image segmentation is based on the assumption that the value of each data point in feature space for given image, e.g., grey-level or color, can be considered as a sample arising from a finite mixture density distribution.

7.2.1 Clustering using Finite Mixture Model

When an image was given, supposing the image has N pixels, we use $\mathbf{x}_i$ to denote the observation at the ith pixel. The total samples in the image form a data set $D = \{\mathbf{x}_i\}_{i=1}^{N}$, and assuming that $\mathbf{x}_i$ is a sample from a finite mixture distribution.

The most used finite mixture distribution is Gaussian, in this work we will adopt Gaussian finite mixture model with EM algorithm without loss generality. The mathematical expression of equations that describes the mixture model joint probability density of system is shown in Chapter 3 Eq.(3.1)-(3.2).

7.2.2 Model Selection Criterion

There exist some information theoretical criteria which can be used to select the number of models, such as AIC [81], AICB [82], CAIC [83], SIC [84]. In this work, we adopt BYY model selection criterion.

The BYY model selection criteria are shown as Eqs. (4.7) and (4.9) in Chapter 4.

With the model selection function $J(k, \Theta)$, we can select the region number k^* simply by $k^* = \arg\min_k J(k, \Theta^*)$ with MLE obtained Θ^*. In practice, we usually start with $k = 1$, estimating parameter Θ^*, computing $J(k = 1, \Theta^*)$. Then $k \to k + 1$, computing $J(k = 2, \Theta^*)$ and so on. After getting a series of $J(k, \Theta^*)$, we choose the minimal one and get corresponding k^*. This k^* is assumed as the region number an image should be segmented.

7.2.3 Bayesian Probabilistic Classification

When an image with N pixels was given, we use $\mathbf{x}_i$ to represent random feature vector in feature space for pixel i. For example, in RGB color space, $\mathbf{x}_i = \{R_i, G_i, B_i\}$ is three dimensional vector, the component of $\mathbf{x}_i$ stands for Red, Green, Blue color value of pixel i of an image respectively, where $i = 1, 2, \cdots, N$. These vectors can be regarded as identical independent distribution. After we got k^* with BYY criterion and the mixture model

parameter Θ^* with EM algorithm, we can calculate *a posterior* probability of sample $\mathbf{x}_i$ belong to region y. From Bayesian rule, The posterior probability is expressed as Eq. (3.3).

For given $\mathbf{x}_i$, we can obtain k probability $P(y = 1|\mathbf{x}_i)$, $P(y = 2|\mathbf{x}_i)$, $\cdots$, $P(y = k|\mathbf{x}_i)$, now use the Bayesian decision to classify pixel i into region y by the solution of:

if $y^* = \text{argmax}_y P(y|\mathbf{x}_i)$, for $y = 1, 2, \cdots, k$, pixel i will be classified to region y^*. Therefore, the finite mixture model image segmentation is actually a pixel classification procedure.

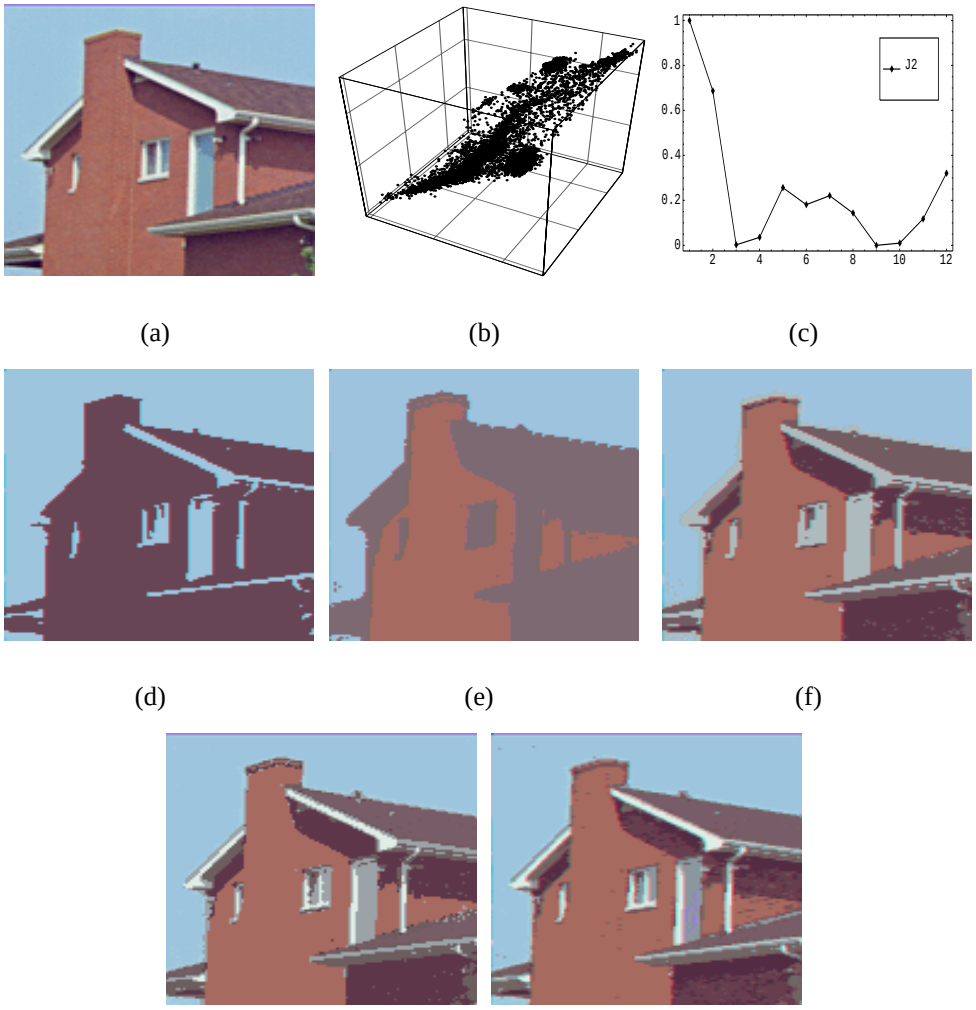
7.3 Application to Image Segmentation

In practice, we have no *prior* knowledge about how many regions should be segmented when an image is presented. If we use a machine to perform automatic image segmentation, this is the first problem we should attack. As mentioned before, at here we assume that the region number in image domain is equal to cluster number in feature space. With this assumption, we can use BYY model selection criterion to determine the region number, it can be considered as how to comprehend the structure for the given image by BYY machine.

Roughly speaking, It takes two mainly steps to perform the automatic image segmentation. First we need to decide how many region there should be, this step is done by searching $J_2(k, \Theta)$ function, find its minima and corresponding k^*. The second step is using Bayesian decision to classify the image pixels into k^* non-overlapped regions.

We can summarize the processing into following steps:

Step (1) Selecting feature space of an image. In the beginning, we choose 3-dimensional RGB color vector of the pixels as feature variables. (If a grey-level image was given, the intensity space will be feature space.) Features on spatial relations of the pixels should be added in further work. For a $m \times n$ pixel image, the $N = m \times n$ RGB vectors of the pixels are input as data points in the feature space for clustering.



(a) (b) (c)

(d) (e) (f)

(g) (h)

Figure 7.1: "House" image.

(a) original image, 128×128 pixels; (b) feature space data distribution; (c) $J_2 - k$ curve; (d) 2-region segmented image. (e) 3-region segmented image. (f) 5-region segmented image. (g) 9-region segmented image. (h) 12-region segmented image. Each region is represented by its mean vector color. 3-region and 9-region segmentation are the possible selections by BYY criterion.

Step (2) Estimating the mixture model parameters with EM algorithm based on feature space data set. Then the parameter learning for each k and the determination of k^* according to the model selection criterion mentioned in Chapter 4 is run.

Step (3) Search the most probable region number k^* based on BYY model selection criterion for Gaussian mixture case.

Step (4) With this obtained k^*, we segment the given image into k^* regions by classifying pixel x_i into one of non-overlapping regions in feature space by Bayesian probabilistic Decision with obtained posterior probability.

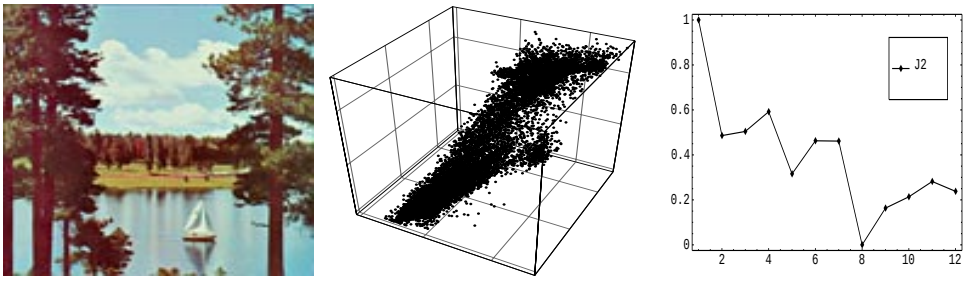
Step (5) Mapping the feature space back to the image domain and labeling the segmented region. In this way, image segmentation with auto-determination of segment number is performed.

In our experiments, we tried two commonly used images - the "house" image and "sailboat" image. Both images are of 128 $\times$ 128 pixels. The image segmentation results are shown in Figure 7.1 and 7.2, respectively, though only grey-scaled rather color pictures are printed out.

From the J_2 - k curve in Figure 7.1c, we can see that there are two local minima. One is at $k = 3$ and the other is at $k = 9$. Intuitively speaking, this illustrates that 3 is an appropriate number of segments for rough segmentation and 9 is another appropriate number of segments which gives more refined segments. Sub-figure (7.1 d to h) show the resulting segmentation with $k = 2, 3, 5, 9, 12$ for comparison. For the 2-region segmentation, we can see that the roof is mixed with the wall. 3-region segmentation can represent the main structure of the house image. While segmentation with more regions increases more details of the image, there is hardly significant increase in quality of segmentation when increasing k from 9 to 12, and 9 can more or less be regarded as an appropriate segment number.

In Figure (7.2), results of 5, 8 and 11-region segmented images are shown. $k = 8$ is determined to be an appropriate segment number by the model selection criterion. The

8-region segmentation can reconstruct the original "sailboat" image well, while the sky and cloud are mixed in the 5-region segmentation and the 11-region segmentation does not show significant increase in segmentation quality.



(a) (b) (c)

(d) (e) (f)

Figure 7.2: "Sailboat" image.

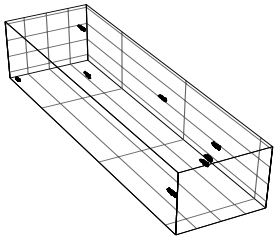
(a) original image, 128×128 pixels; (b) feature space data distribution; (c) $J_2 - k$ curve; (d) 5-region segmented image; (e) 8-region segmented image; (f) 11-region segmented image. Each region is represented by its mean vector color. BYY criterion selected region number is 8.

7.3.1 Color Space

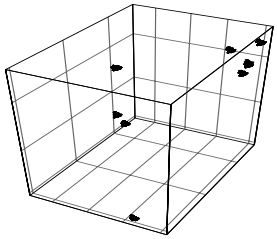
When we segment a color image, apparently the distribution of clusters depend on the color space selection, therefore determined region number is variable for different color space. In this section, we investigate the effect of color space selection on the region number determination problem.

A digital color image is represented by three components, such as RGB, XYZ, YIQ,

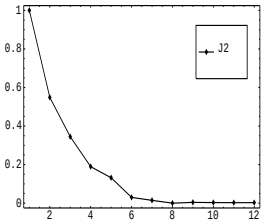
(a) Synthetic image



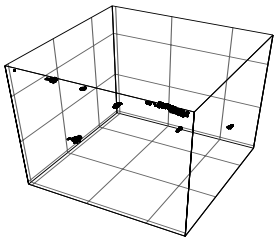
(b) $X_1X_2X_3$ color space



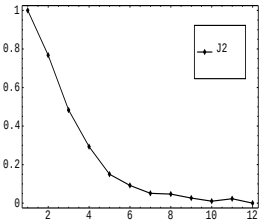
(c) RGB color space



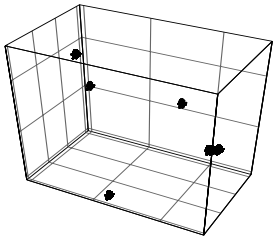
(d) $J_2 - k$ curve for RGB space, $k = 8$ is correct selected.



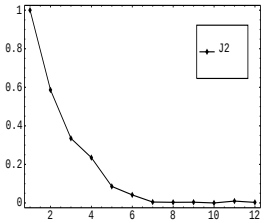
(e) Scaled HSI color space



(f) $J_2 - k$ curve for HSI space, it over estimates region number.



(g) Scaled YIQ color space



(h) $J_2 - k$ curve for YIQ space, it under estimates region number.

Figure 7.3: Synthetic image with 8 classes

In some color spaces, $J_2 - k$ curve for determining segment region number. $J_2 - k$ curves for $X_1X_2X_3$, XYZ and $I_1I_2I_3$ color system are similar to RGB's.

HSI and so on[121]. Which color space is suitable for clustering and how it affects the proper region number determination? There is no theoretical guide for this quite new problem, we believe that the probable answer should be based on experimental testing only.

In this chapter, we concentrate on the following color spaces.

(1) RGB: (Original tristimuli Red, Green, and Blue), this color space is used for display.

(2) YIQ: for color system of TV signal.

(3) XYZ: for C.I.E X-Y-Z color system.

(4) $X_1X_2X_3$: this color feature is obtained by Karhunen Loéve transformation, also called PCA. X_1, X_2, X_3 are uncorrelated each other.

(5) $I_1I_2I_3$: for uncorrelated features.

(6) HSI: (Hue, Saturation and Intensity) for human perception.

Relations of above color system with RGB are as following, the transformation matrices are not the standard ones[121].

(a) YIQ

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ I &= 0.5R - 0.23G - 0.27B \\ Q &= 0.202R - 0.5G + 0.298B \end{aligned} \tag{7.1}$$

(b) XYZ

$$\begin{aligned} X &= 0.618R + 0.177G + 0.205B \\ Y &= 2.299R + 0.587G + 0.114B \\ Z &= 0.056G + 0.944B \end{aligned} \tag{7.2}$$

(c) $I_1I_2I_3$

$$\begin{aligned} I_1 &= (R+G+B)/3 \\ I_2 &= (G-B)/2 \\ I_3 &= (2G-R-B)/4 \end{aligned} \tag{7.3}$$

(d) HSI

$$\begin{aligned} H &= \arctan\left(\frac{\sqrt{3}(G-B)}{2R-G-B}\right) \\ S &= 1-\frac{\min(R,G,B)}{R+G+B} \\ I &= R+G+B \end{aligned} \tag{7.4}$$

(e) $X_1X_2X_3$

$$\begin{aligned} X_1 &= w_{R1}R+w_{G1}G+w_{B1}B \\ X_2 &= w_{R2}R+w_{G2}G+w_{B2}B \\ X_3 &= w_{R3}R+w_{G3}G+w_{B3}B \end{aligned} \tag{7.5}$$

where $W_i = (w_R, w_G, w_B)$, $i = 1, 2, 3$ are three eigenvectors of Σ, $\Sigma W_i = \lambda_i W_i$, λ_i is eigenvalue, and

$$\begin{aligned} \mathbf{m} &= \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i(R,G,B) \\ \Sigma &= \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i-\mathbf{m})(\mathbf{x}_i-\mathbf{m})^T \end{aligned}$$

With these relation equations, other color spaces are transformation of RGB color space.

7.4 Experiments for Color Space Selection

In the experiments, we use one synthetic image and some standard images such as “house” and “sailboat” to test the effect of color space selection on BYY model selection criterion.

Each image is 24 bit color with size of 128×128 pixels.

Choosing different color space will result different shape and distribution of clusters, which leads to estimated parameter variable, though the EM algorithm and classification rule is same for all color spaces. In the experiments, with selected color space, we run EM algorithm to estimate mixture parameters and compute $J(k, \Theta^*)$ curves. In order to eliminate the influence of EM algorithm converge to different local minima, we repeat experiment with same condition but with different initial parameter values several times, then using most probable results.

Several experiments have been done, here only parts of experimental results for the synthetic image are shown in Figure 7.3. Similar results are observed for "house" and "sailboat" images. In the synthetic image, there are 8 colors, each color represents one cluster in color space. If colors are similar for some regions, clusters will overlap in color space, e.g. in HSI or YIQ color space. Overlap has an negative influence on properly clustering, it results in poor region number selection.

In experiments, it is found that using RGB, XYZ, $X_1X_2X_3$ and $I_1I_2I_3$ color spaces yield same reasonable region number. In $X_1X_2X_3$ or $I_1I_2I_3$ color space, it is easily clustering with EM algorithm and computation time is also less than using HSI color space. When using HSI color space, Hue value is unstable when Saturation value is near zero, in this case, it would be very difficult for correctly determining region number and segmentation. On the other side, based on experimental testing, the final choice of color space is $X_1X_2X_3$ or $I_1I_2I_3$ system. This is that $X_1X_2X_3$ or $I_1I_2I_3$ color coordinates are almost uncorrelated, they are effective for region number determination based on BYY model selection criterion.

From experiments, we know that by using BYY model selection criterion, as long as the proper color space was used, in most cases we can select the reasonable region number, and make it possible in automatic segmenting given image without *a priori* knowledge.

7.5 Summary

While most previous works on feature space clustering for image segmentation need manually specifying the number of segments in *a priori*, we apply the model selection criterion to this approach and obtain a method for automatic determination of an appropriate number of segmentation. In other words, it is possible that the BYY model selection criterion give reasonable insight on the structure of the presented image.

In this chapter, we have investigated the effect of color space selection on determining image segmentation region number based on BYY model selection criterion also, six color spaces have been tested and compared experimentally. EM algorithm was used to estimate mixture model parameters and Bayesian decision rule was used to classify pixels into proper regions. Form the experimental results, we can conclude that RGB space is the basic selection while $X_1X_2X_3$ or $I_1I_2I_3$ color space is more appropriate for clustering and BYY criterion application.

Chapter 8

Application: Software Quality Prediction

8.1 Introduction

Software reliability engineering is one of the most important aspect of software quality [47]. The interest of the software community in program testing continues to grow – as does the demand for complex, and predictively reliable programs. It is no longer acceptable to postpone the assurance of software quality until prior to a product's release. Delaying corrections until testing and operational phases may lead to higher costs [122], and it may be too late to improve the system significantly. Recent research in the field of computer program reliability has been directed towards the identification of software modules that are likely to be fault–prone, based on product and/or process–related metrics, prior to the testing phase, so that early identification of fault–prone modules in the life–cycle can help in channeling program testing and verification efforts in the productive direction.

Software metrics represent quantitative description of program attributes and the critical role they play in predicting the quality of the software has been emphasized by Perlis *et al* [123]. That is, there is a direct relationship between some complexity metrics and the number of changes attributed to faults later found in test and validation [124]. Many

[1]This chapter is identical to the paper with the title "Software Quality Prediction using Mixture Model with EM algorithm", *Proceedings of APAQS'2000*, Hong Kong, pp69-78, 2000.

researchers have sought to develop a predictive relationship between complexity metrics and faults. Crawford *et al* [125] suggest that multiple variable models are necessary to find metrics that are important in addition to program size. Consequently, investigating the relationship between the number of faults in programs and the software complexity metrics attracts researchers' interesting.

Several different techniques have been proposed to develop predictive software metrics for the classification of software program modules into fault–prone and non fault–prone categories. These techniques include discriminant analysis [126, 127], factor analysis [128], classification trees [129, 130, 131], pattern recognition (Optimal Set Reduction (OSR)) [126, 132], feedforward neural networks [133], and some other techniques [134]. Most of these techniques are classification models and they partition the modules into two categories, namely, fault–prone and not fault–prone. With these predictive models, the troublesome modules can be identified earlier in the life–cycle of a software product. The advantage of these fault prediction models are multi-fold; however, when building the models, they require to know the number of changes (faults) at the same time. That is, the model parameters need to be estimated with a supervised learning procedure [21]. As we know, to obtain the dependent criterion variable, we will need to a long time for the feedback of test and validation results. For example, for the software of Medical Imaging System (MIS) presented later in this paper, the actual number of changes (faults) in that program is collected during three-year observation period. As software complexity metrics can be obtained relatively early in the software life-cycle, it is worthy to explore new techniques for early prediction of software quality based on software complexity metrics.

In this chapter we present one such new approach – using a finite mixture model with Expectation-Maximum (EM) algorithm [57, 58] to investigate the predictive relationship between software metrics and the classification of the program module. With the mixture model analysis, we can develop a prediction model without the need to know the number of changes (faults) in advance. Namely, it is only based on software complexity metrics

to build the model. The model parameters are estimated by using EM algorithm, which is a procedure of unsupervised learning since the class membership of those metrics is unknown and the metrics are treated as un-labeled vectors.

The mixture model analysis is mainly a probabilistic classification procedure. It is used to assign program modules to classes of modules of similar characteristics without the knowledge of fault rate in advance. By this statistical technique, we can identify a program or a program module as a class of low or high fault rate in the early stage of program development. In addition, we also show that the discriminant analysis is a special case of the mixture model analysis.

8.2 Modeling Methodology

We propose to use the finite mixture model analysis with EM algorithm technique in software quality prediction to classify fault-prone and non fault-prone modules. In the following we will briefly review the mixture model with EM algorithm, and Akaike Information Criterion (AIC) model selection criterion.

The mixture distribution, particular in Gaussian (normal) analysis method, has been used widely in a variety of important practical situations, where the likelihood approach to the fitting of mixture models has been utilized extensively [49, 50, 51, 52]. The application of the finite mixture model to software quality prediction is based on the assumption that the software complexity metrics in a vector space can be considered as a sample arising from two or more models mixed in varying proportions.

8.2.1 Finite Gaussian Mixture Model With EM Algorithm

A mixture model can be of any mixed distribution function, but the mostly-used model is the Gaussian distribution model. Hence, in this paper we only investigate the Gaussian density case. In the software complexity metrics vector space, one module can be considered as one point, and altogether N points consistent of N modules can form a given data

set D. The data set $D = \{\mathbf{x}_i\}_{i=1}^N$ ready for classification is assumed to be samples from a mixture of k Gaussian densities with joint probability density

$$\begin{aligned} p(\mathbf{x}, \Theta) &= \sum_{j=1}^{k} \alpha_j G(\mathbf{x}, \mathbf{m}_j, \Sigma_j), \\ \text{with} \quad \alpha_j &\geq 0, \text{ and } \sum_{j=1}^{k} \alpha_j = 1 \end{aligned} \tag{8.1}$$

where

$$G(\mathbf{x}, \mathbf{m}_j, \Sigma_j) = \frac{\exp[-\frac{1}{2}(\mathbf{x} - \mathbf{m}_j)^T \Sigma_j^{-1} (\mathbf{x} - \mathbf{m}_j)]}{(2\pi)^{d/2} |\Sigma_j|^{\frac{1}{2}}} \tag{8.2}$$

is multivariate Gaussian density function, $\mathbf{x}$ denotes random vector (which integrates a variety of software metrics), d is the dimension of $\mathbf{x}$, and parameter $\Theta = \{\alpha_j, \mathbf{m}_j, \Sigma_j\}_{j=1}^k$ is a set of finite mixture model parameter vectors. Here α_j is the mixing weights, $\mathbf{m}_j$ is the mean vector, and Σ_j is the covariance matrix of the j-th component. In fact, as these parameters are unknown, using how many Gaussian density components can best describe the probability density of the system is also unknown. Usually with a pre-assumed number k, the mixture model parameters are estimated by the maximum likelihood learning (ML) with EM algorithm [57, 58].

The log likelihood function of the system to be explored is

$$l(\Theta|x) = \ln L(\Theta|x) = \sum_{i=1}^{N} \ln(\sum_{j=1}^{k} \alpha_j G(\mathbf{x}_i, \mathbf{m}_j, \Sigma_j)) \tag{8.3}$$

Maximizing this function will re-derive the EM algorithm, which we show in two steps.

1. **E-step:**(Expectation step)

Calculate the *posterior* probability $P(j|\mathbf{x}_i)$ according to

$$P(j|\mathbf{x}) = \frac{\alpha_j G(\mathbf{x}, \mathbf{m}_j, \widehat{\Sigma}_j)}{p(x, \Theta)}, \qquad \text{with } j = 1, 2, \cdots, k, \tag{8.4}$$

2. **M-step:**(Maximum step)

$$\alpha_j^{new} = \frac{1}{N}\sum_{i=1}^{N}\frac{\alpha_j^{old}G(\mathbf{x}_i,\mathbf{m}_j,\Sigma_j)}{\sum_{j=1}^{k}\alpha_j^{old}G(\mathbf{x}_i,\mathbf{m}_j,\Sigma_j)} = \frac{1}{N}\sum_{i=1}^{N}P(j|\mathbf{x}_i) \tag{8.5}$$

$$\mathbf{m}_j = \frac{\sum_{i=1}^{N}P(j|\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^{N}P(j|\mathbf{x}_i)} = \frac{1}{\alpha_j N}\sum\nolimits_{i=1}^{N}P(j|\mathbf{x}_i)\mathbf{x}_i \tag{8.6}$$

$$\widehat{\Sigma}_j = \frac{1}{\alpha_j N}\sum\nolimits_{i=1}^{N}P(j|\mathbf{x}_i)[(\mathbf{x}_i-\mathbf{m}_j)(\mathbf{x}_i-\mathbf{m}_j)^T]. \tag{8.7}$$

The two steps are iterated until convergence to one local minima is obtained.

Unlike supervised learning, the ML with EM algorithm can be used for a totally unlabeled data set; that is, the case of sample class membership is unknown.

In practical implementation, the problem to be handled first is the mixture parameter initialization. It is a common practice that the parameter values are random initialized since no *a priori* information is available. In this paper, we use the following methods to initialize mixture model parameters:

$$\alpha_j^0 = \frac{1}{k}, \tag{8.8}$$

$$\mathbf{m}_j^0 = \min_{1\le i\le N}(\mathbf{x}_i) + j\times\{\max_{1\le i\le N}(\mathbf{x}_i) - \min_{1\le i\le N}(\mathbf{x}_i)\}/(k+1) \tag{8.9}$$

$$\widehat{\Sigma}_j^0 = \frac{||\max(\mathbf{x}_i)-\min(\mathbf{x}_i)||}{20}\mathbf{I}_d. \tag{8.10}$$

where $\mathbf{I}_d$ represents the $d\times d$ dimension identity matrix. This initialization method can guarantee that the mean vectors are within the range of the data set D. The alternative method used is an addition of a small random value on the above equations.

8.2.2 Model Selection Criterion

When the software complexity metric data are to be classified into several classes, each class contain the data samples with similar characteristics. With prior knowledge, we usually divide the modules into two classes: one is fault-prone and the other is non fault-prone. However, by the mixture model approach, how many classes the metric data should be divided is not known. Consequently, the number of Gaussian density components can best describe the probability density of the system is unknown. Nevertheless, we can use some model selection criterion to determine a proper number of model components.

Following Akaike's pioneering work [81] in selecting the number of components in the mixture model analysis, a lot of researchers have developed some modified and newly proposed criteria such as AICB [82], CAIC [83], SIC [84]. These criteria combine the maximum value of the likelihood function with the number of parameters used in achieving that value. Here we list the corresponding AIC formula for a convenient use afterwards, in which $L(k)$ means likelihood function of the number k model with other parameters like Θ has been estimated by using the Eq. (8.3):

$$AIC(k) = -2\ln[\max L(k)] + 2m_k, \tag{8.11}$$

where the $m_k = kd + (k-1) + kd(d+1)/2$ is a penalty term. The other criteria such as AICB, CAIC and SIC are similar to AIC, with the difference at the penalty term.

From the above $AIC(k)$, we can select the model number k^* simply by the solution of $k^* = \arg\min_k AIC(k)$ with ML obtained parameter Θ^*. In practice, we start with $k = 1$, estimate parameter Θ^*, and compute $AIC(k = 1)$. Then by iterating $k \to k + 1$, we compute $AIC(k = 2)$, and so on. After getting a series of $AIC(k)$, we choose the minimal one and get the corresponding k^*. This k^* is assumed as the number of classes of the program modules should be partitioned.

8.2.3 Bayesian Probabilistic Classification

In the mixture model case a Bayesian decision rule is used to classify the vector $\mathbf{x}$ into class j with the largest *posterior* probability. The *posterior* probability $P(j|\mathbf{x})$ represents the probability that sample $\mathbf{x}$ belongs to class j. The probabilities of $P(j|\mathbf{x})$ are usually unknown and have to be estimated from the training samples. With the maximum likelihood estimation, the *posterior* probability can be written in the form of Eq. (8.4).

For a given $\mathbf{x}_i$, we can obtain k probabilities $P(j = 1|\mathbf{x}_i)$, $P(j = 2|\mathbf{x}_i)$, $\cdots$, $P(j = k|\mathbf{x}_i)$. Now we use the Bayesian decision rule to classify $\mathbf{x}_i$ into one of the non-overlapping class j^* by the solution of

$$j^* = \arg\max_j P(j|\mathbf{x}_i), \qquad \text{for } j = 1, 2, \cdots, k. \tag{8.12}$$

If j^* is corresponding to maximum $P(j|\mathbf{x}_i)$, the ith program module will be classified into class j^* with probability $P(j^*|\mathbf{x}_i)$.

When we take the logarithm to Eq. (8.4) and omit the common factors of the classes, such as $\ln p(x, \Theta)$, $d/2 \ln 2\pi$, the classification rule becomes

$$j^* = \arg\min_j d_j(\mathbf{x}), \qquad \text{for } j = 1, 2, \cdots, k \tag{8.13}$$

with

$$d_j(\mathbf{x}) = (\mathbf{x} - \mathbf{m}_j)^T \Sigma_j^{-1} (\mathbf{x} - \mathbf{m}_j) + \ln |\Sigma_j| - 2 \ln \alpha_j \tag{8.14}$$

This equation is often called the discriminant score for the jth class in the literature [135]. Furthermore, if the *prior* density α_j is the same for all classes (an equal sample number in each class), it becomes discriminant function when omitting the term $2 \ln \alpha_j$. If a pooled covariance matrix is used, it is called linear discriminant analysis (LDA), which was used by Munson and Khoshgoftaar for detection of fault-prone programs [127].

If the class membership relation of the sample as well as the number N_j of each class is known, which is assumed in the discriminant analysis application [127], the mean vector

$\mathbf{m}_j$ and the covariance matrix Σ_j can be evaluated based on given samples with maximum likelihood estimation. They take the following forms:

$$\mathbf{m}_j = \frac{1}{N_j} \sum_{i=1}^{N_j} \mathbf{x}_i \tag{8.15}$$

$$\widehat{\Sigma}_j = \frac{1}{N_j - 1} \sum_{i=1}^{N_j} (\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T. \tag{8.16}$$

They are called sample mean and sample covariance matrix, respectively [35]. Here we can see they are different with EM estimate. In a supervised learning case, each sample has determined class membership, while in EM estimate, each sample can belong to every class at the same time with a certain probability value.

8.3 Data Description and Analysis Procedure

In this section, we present a real project to which we apply the finite mixture model with EM algorithm for quality prediction and data analysis. The data used for the application of the mixture model represents the results of an investigation of software for a Medical Imaging System (MIS). The total system consisted of about 4500 modules amounting to about 400,000 lines of code written in Pascal, FORTRAN, assembler and PL/M. A random sample of 390 modules, from the ones written in Pascal and FORTRAN were selected for analysis. These 390 modules consists of approximately 40,000 lines of code. The software was developed over a period of five years, and was in commercial use at several hundred sites for a period of three years[133].

The number of changes made to a module, documented as Change Reports (CRs), was used as an indicator of the number of faults introduced during development[136]. The changes made to the routines were analyzed, and only those that affected the executable code were counted as faults (aesthetic changes such as comments were not counted)[137].

In addition to the change data, the following 11 software complexity metrics were developed for each of the modules:

- Total lines of code (TC) – Total number of lines in the routine including comments, declarations and the main body of the code.
- Number of code lines (CL) – Number of lines of executable code in the routine excluding the declaration and comment lines.
- Number of characters (Cr) – All characters in the routines.
- Number of comments (Cm) – For the Pascal routines, a comment is either a line beginning with test %%, or text in comment brackets, either of the form $\{ <$ comment $> \}$ or (* $<$ comment $>$*). For FORTRAN routines, a comment consists of the text on a line after either $|$, C or *.
- Number of comment characters (CC) – The amount of text found in the routines comments.
- Number of code characters (Co)– The amount of text which makes up the executable code in the routine.
- Halstead's Program Length (N'), where $N' = N'_1 + N'_2$ and N'_1 represents a total operator count and N'_2 represents a total operand count [138]
- Halstead's Estimate of Program Length Metric (Ne), where Ne=$\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$, and η_1 and η_2 represent the unique operator and operand counts, respectively[138].
- Jensen's Estimate of Program Length Metric (JE), where JE=$\log_2 \eta_1! + \log_2 \eta_2!$ [139].
- McCabe's Cyclomatic Complexity Metric (M), where $M = e - n + 2$, and e represents the number of edges in a control flow graph of n nodes [140].

- Belady's bandwidth metric (BW), where:

$$BW = \frac{1}{n}\sum_{i} iL_i \tag{8.17}$$

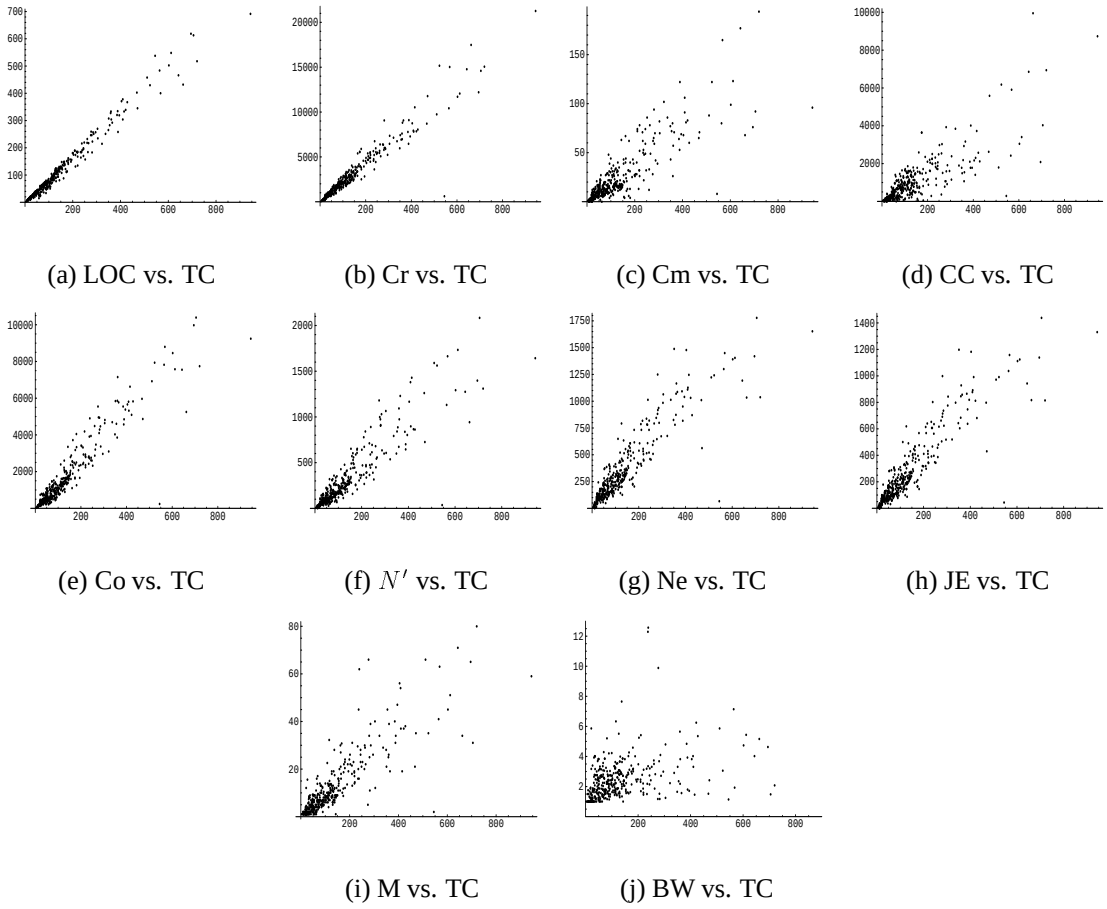
and L_i represents the number of nodes at level i in a nested control flow graph of n nodes [139]. This metric indicates the average level of nesting or width of the control flow graph representation of the program.

By using these independent metrics as integrated complexity metrics, the random vector x is a 11-dimension vector with each metric as one component. Each vector x_i represents one sample point in the metric space, and we can apply the mixture model analysis in this high-dimension vector space to partition data samples into proper classes. When estimating mixture model parameters, we do not need to know the change requests (faults).

Principal Components Analysis(PCA): In a software development application, the independent variables (complexity metrics) may be strongly interrelated as they demonstrate a high degree of multicollinearity. We first examine the relationship of metric TC with other metrics, as shown in Figure 8.1.

It is clearly seen in Figure 8.1 that the metric TC has nearly linear relationship with some metrics such as LOC, Cr and Co. Several independent variables demonstrating a high degree of multicollinearity will have a negative effect on the regression model. One distinct result of multicollinearity in the independent variables is that the statistical models developed from them have highly unstable regression coefficients [127]. To reduce the interrelated effect, we adopt PCA (also called *Karhunen-Loéve transformation*) to transform the original complexity metrics space into an orthogonal vector space. The principle of PCA is simple. Let us assume the data set has a covariance matrix Σ, which is a real symmetric matrix and can be decomposed as follows:

$$\Sigma = \mathbf{U}\Lambda\mathbf{U}^T \tag{8.18}$$



(a) LOC vs. TC (b) Cr vs. TC (c) Cm vs. TC (d) CC vs. TC

(e) Co vs. TC (f) N' vs. TC (g) Ne vs. TC (h) JE vs. TC

(i) M vs. TC (j) BW vs. TC

Figure 8.1: The relationship of metric TC with other metrics.

From (a) to (j): horizontal axis is metric TC , vertical axes are metric LOC, Cr, Cm, CC, Co, N', Ne, JE, M and BW respectively. There are several metrics that exhibit multicollinearity.

where $\mathbf{U}$ is a matrix whose column i is the eigenvector $\mathbf{u}_i$, and Λ is a diagonal matrix of eigenvalues. Note that each of the eigenvectors is called a principal component. The vectors $\mathbf{x}$ are projected onto the eigenvectors to give the components of the transformed vectors $\mathbf{x}'$. That is,

$$\mathbf{x}' = \mathbf{U}^{\mathbf{T}}\mathbf{x}. \tag{8.19}$$

PCA can be used to reduce the dimension of the data space by taking $M < d$ eigen-

Table 8.1: The eigenvalues for the MIS data set

Component	1	2	3	4	5	6
Eigenvalue	1.28×10^7	6.05×10^5	1.71×10^4	1.34×10^4	4.77×10^3	2.41×10^3
Component	7	8	9	10	11	
Eigenvalue	1.78×10^2	47.2	31.5	13.5	0.98	

vectors corresponding to the first M largest eigenvalues to construct the transform matrix. The error introduced by a dimensionality reduction using PCA can be evaluated using

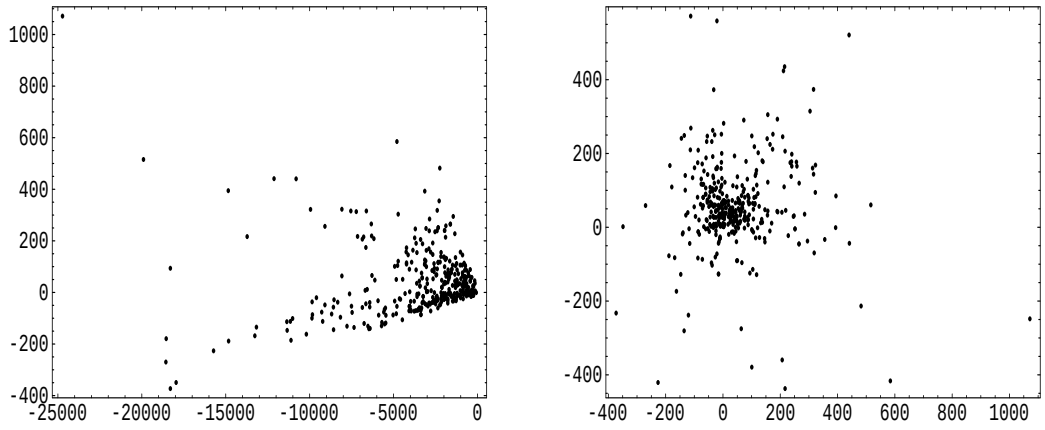
$$E_M = \frac{1}{2}\sum_{i=M+1}^{d} \lambda_i, \tag{8.20}$$

where the smallest $d - M$ eigenvalues λ_i and their corresponding eigenvectors are discarded.

The eigenvalues for the MIS data set are shown in Table 8.1.

When using PCA to reduce the dimension of data space, we know from Table 8.1 that the first 7 components can represent main feature of the data set with a relatively small error (E_M =46.6338). However, some patterns are separable in high dimension space, but they become inseparable when projected into low dimension space. Therefore, we just apply PCA to transform data into an orthogonal set, using all 11-dimension in the data analysis. The results presented in this paper are based on PCA transformed data space, which is a 11-dimensional vector space. Figure 8.2 shows data distribution when projected onto first two principal components space and third-fourth principal components space.

For such a data space, each point represents one program module, which is characterized by its complexity metrics. These points can be assumed as samples arising from two or more models mixed in varying proportions. When the mixture model analysis with EM algorithm was applied to the 390 program modules in the PCA de-correlated 11-dimensional vector space, the most probable results are shown in Figure 8.3 for log



(a) (b)

Figure 8.2: Data distribution in vector space

(a) first two principal components and (b) third-fourth principal components.

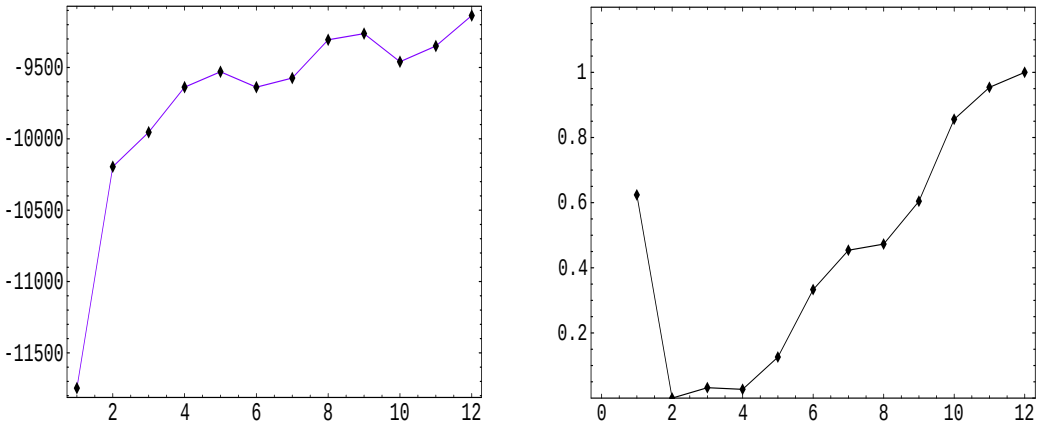
likelihood function vs. model component number k as well as AIC vs. k.

In Figure 8.3a, we can see that the log likelihood function of the system increases as the model number increases. Increasing model number makes finer classification for given software modules, and each model represents a subset of the data in which samples have similar characteristics. The AIC model selection criterion in Figure 8.3b shows that with PCA de-correlated data set, classifying the modules into two groups is a proper selection. This gives us an insight into some intrinsic properties of the PCA de-correlated complexity metrics data set.

With two-class classification, the experimental results as obtained from Eq. (8.12) show that the module number in each group is $N_1 = 264$ and $N_2 = 126$, respectively. Note there are unequal sample numbers for the two-group classification.

The estimated mixture model parameters with EM algorithm for the case $k = 2$ are as the following:

Mixture weights: $\alpha_1 \approx 0.673$, and $\alpha_2 \approx 0.327$. Recall that $\alpha_j = \frac{1}{N} \sum_{i=1}^{N} P(j|\mathbf{x}_i)$, then



(a) (b)

Figure 8.3: The log likelihood function as well as AIC vs. k.

(a) The log likelihood function vs. model number. With the increase of the model number k, the function tends to increase too. (b) Typical results for AIC's vs. model number k for PCA de-correlated data set. The minima occurs at $k^* = 2$.

$N_j = \sum_{i=1}^{N} P(j|\mathbf{x}_i) = \alpha_j N$. This should be the possible module number in class j. The obtained results are $N_1 \approx 0.673 \times 390 = 262$ and $N_2 \approx 0.327 \times 390 = 128$, respectively, which is agreeable with the experimental results obtained by using Eq. (8.12). As the mixture weights are a rough indication of module number distribution, this implies a high confidence in our results.

Mean vector: With two-class partition, the mean vector for each group is shown in Table 8.2 for the original complexity metrics. The maximum and minimum values are also listed in Table 8.2 for reference. Notice that for the sake of readability, the values listed in Table 8.2 are transformed back from the PCA de-correlated space to the original data space.

The positions of the mean for each metric (i.e., m_1 and m_2) show the information to partition modules using single metric. Note that for all the 11 metrics, $m_2 > m_1$. This means class two consistently has a higher value than class one for all the metrics.

Table 8.2: Mean vector component as well as maximum and minimum value for each metric, and the diagonal values of covariance matrices obtained by ML with EM algorithm.

	min	max	m_1	m_2	Σ_1(diag.)	Σ_2(diag.)
TC	3	944	68.04	260.01	1565.7	26771
LOC	2	692	52.28	210.23	1125.9	18132
Cr	59	21266	1458	5620	766272	1.284×10^7
Cm	0	194	12.02	48.54	62.429	1258.87
CC	0	9946	561.52	1825	222703	2.703×10^6
Co	30	10394	761.37	3469	225432	4.573×10^6
$N\prime$	3	2083	137	629	7392.55	158213
Ne	2	1777.3	183.7	669.6	10534	135308
JE	0.8	1437.2	132.8	521.7	6143.7	89333
M	1	80	5.76	24.56	12.507	249.496
BW	1	12.56	2.1	3.13	0.78547	3.774

Covariance matrix: The covariance matrix is a symmetric matrix. Its diagonal element is the variance of each metric, while off-diagonal elements reflect the correlation between the metrics. (Refer to Eq.(8.7).) Here Table 8.2 only shows diagonal elements of the covariance matrices in the last two columns. Some metrics show high variance with two classes partition, implying that two-class partition is not the best choice from the point of view of minimal variance reduction.

The total module number is 390 in the given data set. With the two mixture models approach, the first group has 264 modules, while the second group has 126 modules, and the ratio is about 2/3 and 1/3 respectively. By the mixture model analysis, we now know that there are two classes for the given program modules: class one has more modules than class two for this data set. Furthermore, class two has higher complexity metrics values than class one.

Although at this stage we do not have failure data, we can pretty much determine that class one is non fault-prone while class two is fault-prone. The reason is two-fold. The first reason is that class two has consistently higher values of the complexity metrics,

Table 8.3: The classification for MIS data set by mixture model analysis.

CRs	0,1	2,3	4,5	6,7	8,9	10,11	12,13	14,15	16,17	18-98
Number of group 1	104	66	33	25	11	9	6	1	4	5
Total modules	114	78	49	36	24	19	12	10	9	39
Percent of group 1	91.2	84.6	67.3	69.4	45.8	47.4	50	10	44	12.8

indicating its fault-prone nature. The second reason is that most (80%) of faults are found in a small portion (20%) of the software code, so we can label that the class with larger number of modules as non fault-prone class, and the class with less number modules as fault-prone class. Here we can see that very little prior knowledge about the number of faults is needed to develop this predictive model using mixture model with EM algorithm. This is the major advantage of our approach compared with previous model classification techniques published in the literature.

8.4 Quality Prediction Results and Discussion

8.4.1 Misclassification errors

The above analysis of program metrics with a mixture model can be obtained in early software develop stage. When the change of requests (CRs) become available later, we can use the CRs to assess the merit of the mixture model. The data analysis results are shown in Table 8.3.

There are two types of errors that can be made in the partition. A Type I error is the case where we conclude that a program module is fault-prone when in fact it is not. A Type II error is the case where we believe that a program module is non fault-prone when in fact it is fault-prone. Of the two types of errors, Type II error has more serious implications, since a product would be seem better than it actually is, and testing effort would not be directed where it would be needed the most.

When we consider module with 0 or 1 CRs to be non fault-prone, those with CRs from

Table 8.4: Misclassification rate for randomly drawing 30 samples out of 89 modules without replacement. The mean and standard deviation are computed based on 50 times repeated experiments.

	min.	max.	mean	std.
misclass. rate	0.133	0.40	0.271	0.064

18 to 98 to be fault-prone, then Type I error is 8.8% and Type II error is 12.8%. When modules with CRs from 10 to 98 are considered as fault-prone, then Type II error will rise to 28.1%. It is noted that in supervised learning, the data set is partitioned into two parts: training samples and validation samples. The method of partition data set can have an effect on the prediction accuracy, as shown in the following experiment.

For MIS data set, there are 89 modules with CRs from 10 to 98, which are considered as fault-prone modules. Now let us randomly draw 30 modules (i.e., one third) from this subset of MIS data set. From mixture model analysis results, we can know the Type II error computed from these 30 modules. The Table 8.4 shows the experimental results of randomly drawing 30 samples from 89 modules without replacement, where the experiments are repeated 50 times. It can be known that the best result for Type II error is about 13%, which is the same as that of discriminant analysis method [127]. The statistical mean for Type II error is 27.1%, which is nearly the same as 28.1% obtained by the mixture model analysis based on all 89 modules.

8.4.2 Classification Probability

As stated in Section 8.2.3, assigning a module as either fault-prone or non fault-prone is based on Bayesian classification rule.

In two-model mixed case, the joint density of the system can be written in the form,

$$p(\mathbf{x}, \Theta) = \alpha_1 G(\mathbf{x}, \mathbf{m}_1, \Sigma_1) + (1 - \alpha_1) G(\mathbf{x}, \mathbf{m}_2, \Sigma_2). \tag{8.21}$$

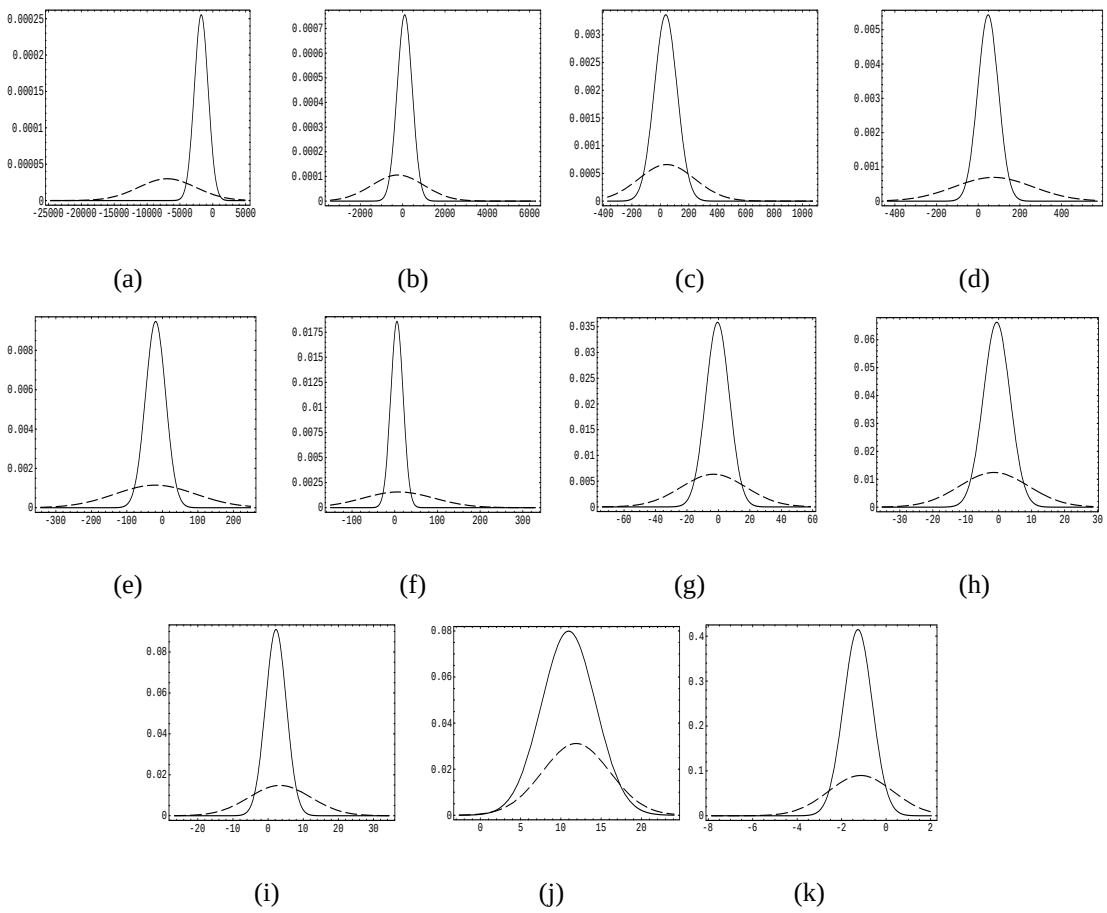


Figure 8.4: The plot for two components of the joint density projected at principal axis.

The figures from (a) to (k) is corresponding to the 11 principal component axes in order.

The posterior probabilities become

$$\begin{aligned} P(1|\mathbf{x}) &= \frac{\alpha_1 G(\mathbf{x}, \mathbf{m}_1, \widehat{\Sigma}_1)}{p(\mathbf{x}, \Theta)}, \\ P(2|\mathbf{x}) &= \frac{(1-\alpha_1) G(\mathbf{x}, \mathbf{m}_2, \widehat{\Sigma}_2)}{p(\mathbf{x}, \Theta)} = 1 - P(1|\mathbf{x}). \end{aligned} \tag{8.22}$$

Figure 8.4 shows the two-component probability distribution of the joint density projected at each principal component axis. The solid line depicts the component $\alpha_1 G(\mathbf{x}, \mathbf{m}_1, \Sigma_1)$, while the dashed line depicts the component $(1-\alpha_1) G(\mathbf{x}, \mathbf{m}_2, \Sigma_2)$. At each point, the value of each probability component is proportional to the value of the posterior probability. When we use Bayesian decision to classify program module i into class j, the

misclassification risk can be obtained with Figure 8.4. If the position of a module is at or near the position at which the values of the two components are nearly equal, (i.e., where the solid line and the dashed line intersect in each figure) the misclassification risk will be high.

Each principal component metric is a linear combination of the original complexity metrics. When we predict that one program module is possible of either fault-prone or non fault-prone, the decision is made by combining all principal components together, not just a single metric. Combining all metrics to predict the software quality is one of the way to reduce the risk of misclassification.

8.4.3 Advantages of Mixture Model Analysis

Building model to support the prediction of software quality based on software complexity metrics can be quite challenging due to various inherent constraints. Sometimes the values of complexity metrics are not complete because it needs a long time collecting them, and building models requires the use of complete data types of variables. The EM algorithm was originally developed for incomplete data set, therefore the approach described above can handle the types of variables with partial missing values. Other methods such as regression tree modeling [131] needs to assign a threshold to split the data set, and requires to know fault number in advance. On the other hand, in the mixture model analysis with EM algorithm, only little prior knowledge is needed to predict the module characteristics based on the complexity metrics.

The mixture model analysis method also does not require an equal class number, so it is a more general model and classification rule used than that discriminant analysis [127]. In the linear discriminant analysis, the covariance matrices are assumed the same for all classes, which is seldom the case in the real world.

Furthermore, if we suppose that the mixture model classification result is correct, from the results shown in Table 8.3, we know that the most non fault-prone modules should

have no more than 3 CRs, which has the percentage greater than 88%. Furthermore, the modules with CRs from 4 to 17 should be mediately fault-prone modules, and the modules with CRs 18 to 98 is the fault-prone group. This shows that the mixture model can help us gain an insight in the relationships between the software complexity metrics and the number of faults in the module.

8.5 Summary

Software metrics can reveal a lot of information about the code at several stages of development. They can identify the routines which need to be redesigned due to higher complexity, routines which may require thorough testing, and features which may require more support. The mixture model with EM algorithm is a novel way to analyze software metrics, to understand the involved relationships among them, to identify the fault-prone modules, and thus to take remedial actions before it is too late. Based on the experimental results, this modeling approach provides an effective way to predict software quality in a very early stage of program development.

Chapter 9

Conclusions

This thesis covers both theoretical and experimental studies on model selection and regularization for generalization in neural networks. One of our main focus is the regularization. Under a general framework, we have shown that one particular case of the system entropy with Gaussian probability density reduces into the first order Tikhonov regularizer for feedforward neural networks in the maximum likelihood learning case, where the regularization parameter is the smoothing parameter h_x in the kernel density function. We derive the formula for approximately estimating the regularization parameter. Experiments show that the estimated regularization parameter is in the same order as that obtained by the validation method.

Under the same framework, we consider the Gaussian mixture model for classification, and the KLIM covariance matrix estimation is derived and investigated. An efficient smoothing parameter approximation formula was provided, and the approximation was found to be accurate for most cases in our experiments.

For model selection in clustering, we perform experiments with bootstrap and data smoothing, and the results indicate that the model selection criterion performance is improved in the small sample number set case.

To select all models in forming ensemble neural networks, we propose an approach to overcome the difficulty in averaging ensemble networks in the parameter space. Experimental results show that the adopted strategy is efficient in improving network per-

formance with finite training samples and the ensemble network architecture is a simple one.

Stacked generalization can be considered as nonlinear combination of trained networks to form ensemble neural networks, which use data set partition to find an overall system with improved generalization performance. In order to efficiently investigate the performance of the stacked generalization, the PIL algorithm is developed for feedforward neural networks. This algorithm is more effective than the standard BP and other gradient descent algorithms for most problems. The fast learning property of the PIL algorithm makes it possible for us to investigate the computation-intensive generalization techniques, such as stacked generalization, more efficiently.

In the application areas, we employ the model selection criterion to image segmentation and obtain a method for automatic determination of the appropriate number of segmentation. We also apply the AIC model selection criterion and the mixture model to analyze software reliability metrics. This approach provides an effective way to predict software quality in an early stage of program development.

The work described in the thesis can be improved or extended in a number of ways, and several interesting problems are worthy of investigation. Our feeling is that some details for generalization in neural networks are still unexplored. In some aspects the Kullback-Leibler distance function may be more general and useful, but one of the open problems in applications is to design reasonable system probability functions for specific problems. Furthermore, the differences in various designed models should be investigated.

On the other hand, the results gained in our techniques should be compared with the results obtained using other techniques with real-world data sets. Different approximations should be analyzed and tested as well. It is considered an important direction to establish an efficient method to estimate the error bound for approximations. Comparing the traditional statistical techniques, such as cross-validation and bootstrap approaches,

to neural network models in their generalization capabilities will also extend the research work broadly.

Applying neural network models to software reliability engineering will be a new and interesting research field, and it may foster significant impact to the software industry.

Appendix A

Formula of Estimating Smoothing Parameter

Here we derive the formula for estimating smoothing parameter in Gaussian mixture model case.

In multi-dimension case,

$$J_r(\mathbf{x}_i, \Theta) = \frac{1}{2Nh} \sum_{i=1}^{N} \int G(\mathbf{x}, \mathbf{x}_i, h\mathbf{I})(\mathbf{x} - \mathbf{x}_i)^T \nabla\nabla g(\mathbf{x}_i)(\mathbf{x} - \mathbf{x}_i) d\mathbf{x}$$

while $\nabla\nabla g(\mathbf{x}_i)$ is

$$\begin{aligned}
\nabla\nabla g(\mathbf{x}_i) &= -\frac{[\nabla\nabla P_{M_2}(\mathbf{x}_i)]P_{M_2}(\mathbf{x}_i) - [\nabla P_{M_2}(\mathbf{x}_i)][\nabla P_{M_2}(\mathbf{x}_i)]}{(P_{M_2}(\mathbf{x}_i))^2} \\
&= \sum_{y=1}^{k} P(y|\mathbf{x}_i)\{\Sigma_y^{-1} - \Sigma_y^{-1}(\mathbf{x}_i - \mathbf{m}_y)(\mathbf{x}_i - \mathbf{m}_y)^T\Sigma_y^{-1}\} \\
&\quad + \{\sum_{y=1}^{k} P(y|\mathbf{x}_i)(\mathbf{x}_i - \mathbf{m}_y)^T\Sigma_y^{-1}\}\{\sum_{y=1}^{k} P(y|\mathbf{x}_i)[(\mathbf{x}_i - \mathbf{m}_y)^T\Sigma_y^{-1}]^T\}
\end{aligned}$$

When integrated out,

$$\begin{aligned}
J_r(\mathbf{x}_i, \Theta) &= \frac{1}{2N} \sum_{i=1}^{N} tr[\nabla\nabla g(\mathbf{x}_i)] \\
&\approx \frac{1}{2N} \sum_{i=1}^{N} || \sum_{y=1}^{k} P(y|\mathbf{x}_i)(\mathbf{x}_i - \mathbf{m}_y)^T\Sigma_y^{-1} ||^2
\end{aligned}$$

From equation (4.26) and (4.30), we have

$$hJ_r - \frac{d}{2}h\int p_{h_x}(\mathbf{x})\ln p_{h_x}(\mathbf{x})d\mathbf{x} + \frac{1}{2N}\sum_i^N \int \ln p_{h_x}(\mathbf{x})G(\mathbf{x},\mathbf{x}_i,h\mathbf{I})||\mathbf{x}-\mathbf{x}_i||^2 d\mathbf{x} = 0 \tag{A.1}$$

For the last term of above equation, we use mean center approximation, that is, let $\ln p_{h_x}(\mathbf{x}) \approx \ln p_{h_x}(\mathbf{x}_i)$

$$\begin{aligned} & \frac{1}{2N}\sum_i^N \int \ln p_{h_x}(\mathbf{x})G(\mathbf{x},\mathbf{x}_i,h\mathbf{I})||\mathbf{x}-\mathbf{x}_i||^2 d\mathbf{x} \\ \approx\ & \frac{h}{2N}\sum_i^N \ln p_{h_x}(\mathbf{x}_i) \end{aligned} \tag{A.2}$$

Combine above equations and with mean field approximation, we can obtain following equation

$$\delta h \approx J_r h^{old} - \frac{1}{2N}\sum_j^N [p_{h_x}(\mathbf{x}_j) - 1]\ln p_{h_x}(\mathbf{x}_j) \tag{A.3}$$

Appendix B

Publication List

Here is the list of my research works which have been published or finished since May 1997, I entered the Chinese University of Hong Kong to begin my Ph.D study.

1. Z.B. Lai, **P. Guo**, T. J. Wang and L. Xu. "Comparison on Bayesian YING-YANG theory based clustering number selection criterion with information theoretical criteria". In *Proceedings of IEEE International Joint Conference on Neural Networks* (IJCNN'98), volume I, pp725-729, Anchorage, USA, 1998.

2. **Ping Guo**, "Averaging ensemble neural networks in parameter space". In *Proceedings of fifth international conference on neural information processing* (ICONIP'98), pp486-489, Kitakyushu, Japan, 1998.

3. **Ping Guo**, *et. al*. "Region number determination in automatic image segmentation based on BKYY model selection criterion". In *Proceedings of 1999 IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing* (NSIP'99), pp743-746, Antalya, Turkey, 1999.

4. **Ping Guo** and Lei Xu. "On the study of BKYY cluster number selection criterion for small sample data set with bootstrap technique". In *Proceedings of 1999 International Joint Conference on Neural Networks* (IJCNN'99), volume I, pp965-968, Washington, DC, USA, 1999.

5. **Ping Guo** and Lei Xu. "Relationship between mixture of experts and ensemble neural networks". In*Proceedings of the 6th International Conference on Neural Information Processing* (ICONIP'99), pp.246-250, Perth, Australia, 1999.

6. **Ping Guo**, *et. al*. "Dynamics of a coupled double-cavity optical interference filter," *Journal of Modern Optics*, vol. 46, no. 1, pp. 167–174, 1999.

7. **Ping Guo** and C.L.Philip Chen, "Regularization parameter estimation based on BKYY data smoothing theory for feedforward nets," in *Proceedings of Artificial Neural Network in Engineering* (ANNIE 2000), pp. 51–56, Missouri, USA, 2000, (Awarded *Second Runner-Up*).

8. **Ping Guo** and C.L.Philip Chen, "A new approach to smoothing parameter estimation in small sample set case," in *Proceedings of Artificial Neural Network in Engineering* (ANNIE 2000), pp. 147–152, Missouri, USA, 2000 .

9. **Ping Guo** and Michael R. Lyu, "A study on color space selection for determining image segmentation region number," in *Proceedings of The 2000 International Conference on Artificial Intelligence*(IC-AI'00), H. R. Arabnia, Ed., vol.III, pp. 1127–1132, CSREA Press, Las Vegas, Nevada, USA, June 26-29, 2000.

10. **Ping Guo** and Michael R. Lyu, "Classification for high-dimension small-sample data sets based on kullback-leibler information measure," in *Proceedings of The 2000 International Conference on Artificial Intelligence* (IC-AI'00), H. R. Arabnia, Ed., vol.III, pp. 1187–1193, CSREA Press, Las Vegas, Nevada, USA, June 26-29, 2000.

11. **Ping Guo** and Michael R. Lyu, "Software quality prediction using mixure model with em algorithm," in *Proceedings of The First Asia-Pacific Conference on Quality Software* (APAQS 2000), Hong Kong, vol. I of *2*, pp. 725–729, IEEE Computer Society Press, October 30-31,2000.

12. **Ping Guo** and Michael R. Lyu, "Pseudoinverse learning algorithm for feedforward neural networks," in *Advances in Neural Networks and Applications* (NNA'01), N. E. Mastorakis Eds., WSES Press (Athens), pp.321–326, 2001.

13. **Ping Guo**, *et. al*, "Pattern Recognition for the Classification of Raman Spectroscopy Signals," Accepted by *Journal of Electron and Information*, 2001, (Chinese).

14. **Ping Guo** and Michael R. Lyu, " A New Approach to Optical Multilayer Learning Neural Network", in the *Proceedings of 2001 International Conference on Artificial Intelligence* (IC-AI'01). vol.I, pp. 426–432, CSREA Press, Las Vegas, Nevada, USA, June 26-29, 2001.

15. **Ping Guo**, *et al*. "Cluster Number Selection for Small Set of Samples Using the Bayesian Ying-Yang Model". Accepted conditionally by the *IEEE trans. Neural Network*, 2001.

16. **Ping Guo** and Michael R. Lyu, " A Case Study on Stacked Generalization with Software Reliability Growth Modeling Data", accepted by *the 8th International Conference on Neural Information Processing* (ICONIP'01), 2001.

Pending Paper

Has submitted the following papers to some proper journals

1. **Ping Guo**, *et. al*, "Study on the Effect of Color Space Selection for Determining Image Segmentation Region Number," Submittted to a journal , 2001, (In Chinese).

2. **Ping Guo**, *et al*. "Regularization Parameter Estimation for Feedforward Neural Networks". Submitted to a journal, 2001.

3. **Ping Guo** and Michael R. Lyu. "A New Approach to Regularized Gaussian Classification for High-Dimension Small Sample Set". Submitted to a journal, 2001.

4. **Ping Guo** and Michael R. Lyu. "A Pseudoinverse Learning Algorithm for Feedforward Neural Networks with Stacked Generalization Application to Software Reliability Growth Data". Submitted to a journal, 2001.

5. **Ping Guo** and Michael R. Lyu. "Software Metrics Classification for Early Detection of Fault-Prone Modules Using an Unsupervised Learning Algorithm". In resubmission, 2001.

Bibliography

[1] V. Vemuri, Ed., *Artifical Neural Networks: Theotetical Concepts*, Computer Society Press of the IEEE, Washington, D.C, 1988.

[2] Jacek M. Zurada, *Introduction to Artificial Neural Systems*, West, St. Paul, 1992.

[3] J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley, Redwood City, CA, 1991.

[4] Vwani Roychowdhury, Kai-Yeung Siu, Alon Orlitsky, Ed., *Theoretical Advances in Neural Computation and Learning*, Kluwer Academic, Boston, 1994.

[5] Philip D. Wasserman, *Neural Computing: Theory and Practice*, Van Nostrand Reinhold, New York, 1989.

[6] M. Vidyasagar, *A Theory of Learning and Generalization: with Applications to Neural Networks and Control Systems*, Springer, London; New York, 1997.

[7] M. Smith, *Neural Networks for Statistical Modeling*, International Thomson Computer Press, Boston, MA, 1996.

[8] S. Geman, E. Bienenstock, and R. Doursat, "Neural Networks and the Bias/Variance Dilemma," *Neural Computation*, vol. 4, pp. 1–58, 1992.

[9] J. E. Moody, "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems," in *Advanced in Neural Infor-*

mation Processing Systems, J.E. Moody, S.J. Hanson, and R.P. Lippmann, Ed., Cambridge, MA, 1992, vol. 4, pp. 847–854, MIT Press.

[10] A. Weigend, "On Overfitting and the Effective Number of Hidden Units," in *Proceedings of the 1993 Connectionist Models Summer School*, M. C. Mozer, Ed., Boulder, CO, 1994, pp. 335–342, LEA/Lawrence Earlbaum Association.

[11] Warren S. Sarle, "Stopped Training and Other Remedies for Overfitting," in *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, Convention Center and Vista Hotel, Pittsburgh, PA, 1995, vol. 27, pp. 352–360.

[12] Amanda J.C. Sharkey, Ed., *Combining Artificial Neural Nets: Ensemble and Modular Multi-net Systems* , Springer, London; New York, 1999.

[13] P. L. Bartlett, "For Valid Generalization, the Size of the Weights is more Important than the Size of the Network," in *Advanced in Neural Information Processing Systems*, M.C. Mozer, M.I. Jordan, and T. Petsche, Ed., Cambridge, MA, 1997, vol. 9, pp. 134–140, MIT Press.

[14] A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth, "Learnability and the Vapnik-Chervonenkis Dimension," *Journal of the Association for Computing Machinery*, vol. 36, no. 4, pp. 929–965, 1989.

[15] Y. S. Abu-Mostafa, "The Vapnik-Chervonenkis Dimension: Information versus Complexity in Learning," *Neural Computation*, vol. 1, no. 3, pp. 312–317, 1989.

[16] Y. Le Cun, J.S. Denker and S.A. Solla, "Optimal Brain Damage," in *Advanced in Neural Information Processing Systems*, D. S. Touretzky, Ed., San mateo, CA, 1990, vol. 2, pp. 598–605, Morgan Kaufmann Publisher.

[17] Lars K. Hansen and Carl E. Rasmussen, "Pruning from Adaptive Regularization," *Neural Computation*, vol. 6, no. 6, pp. 1222–1231, 1994.

[18] F. Girosi, M. Jones and T. Poggio, "Regularization Theory and Neural Networks Architectures," *Neural Computation*, vol. 7, pp. 219–269, 1995.

[19] Lizhong Wu and John Moody, "A Smoothing Regularizer for Feedforward and Recurrent Neural Networks," *Neural Computation*, vol. 8, no. 3, pp. 463–491, 1996.

[20] G. E. Hinton, "Learning Translation Invariant Recognition in Massively Parallel Networks," in *Proceedings PARLE Conference on Parallel Architectures and Languages Europe*, A. J. Nijman J.W. de Bakker and P. C. Treleaven, Eds., Berlin, 1987, pp. 1–13, Springer-Verlag.

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

[22] Yves Grandvalet and Stephane Canu, "Noise Injection: Theoretical Prospects," *Neural Computation*, vol. 9, no. 5, pp. 1093–1108, 1997.

[23] Alan M. Thompson, John C. Brown, Jim W. Kay and D. Michael Titterington, "A Study of Methods of Choosing the Smoothing Paprameter in Image Restoration by Regularization," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 326–339, 1991.

[24] Peter R. Jonston and Ramesh M. Gulrajani, "A New Method for Regularization Parameter Determination in the Inverse Problem of Electrocardiography," *IEEE Trans. on Biomedical Engineering*, vol. 44, no. 1, pp. 19–39, January 1997.

[25] G. Wahba, *Spline Models for Observational Data*, vol. 59 of *CBMS-NSF regional conference series in applied mathematics*, Society for Industria and Applied mathematics, Philadelphia, PA, 1990.

[26] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*, Chaoman and Hall, London, 1993.

[27] D. J. C. MacKay, "Bayesian Interpolation," *Neural Computation*, vol. 4, no. 3, pp. 415–447, 1992.

[28] J. Larsen, L.K. Hansen, C. Svarer and M. Ohlsson, "Design and Regularization of Neural Networks: the Optimal Use of a Validation Set," in *Proceedings of the 1996 IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing*, S. Usui, Y. Tohkura, S. Katagiri and E. Wilson, Ed., 1996, vol. VI, pp. 62–71.

[29] L. Nonboe Andersen, J. Larsen, L.K. Hansen and M. Hintz-Madsen, "Adaptive Regularization of Neural Classifiers," in *Proceedings of the 1997 IEEE Workshop on Neural Networks for Signal Processing*, J. Principe, L. Gile, N. Morgan and E. Wilson, Ed., 1997, vol. VII, pp. 24–33.

[30] Dingding Chen and M. T. Hagan, "Optimal Use of Regularization and Cross-validation in Neural Network Modeling," in *Proceedings of the 1999 International Joint Conference on Neural Networks*, 1999, vol. 2, pp. 1275–1280, (IJCNN'99).

[31] Katsuyuki hagiwara and Kazuhiro Kuno, "Regularization Learning and Early Stoping in Linear Networks," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, S-I, Amari, C.L. Giles, M. Gori and V. Piuri, Ed., 2000, vol. 4, pp. 511–516, (IJCNN 2000).

[32] Isabelle Rivals and Leon Personnaz, "On Cross Validation for Model Selection," *Neural Computation*, vol. 11, pp. 863–870, 1999.

[33] S. Kullback, *Information Theory and Statistics*, Wiley, New York, 1959.

[34] L. Devroye, *A Course in Density Estimation*, Birhhauser Publisher, Boston, 1987.

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

[36] D. Bosq, *Nonparametric Statistics for Stochastic Processes: Estimation and Prediction*, Springer-Verlag Inc., New York, 1996.

[37] J. Rissanen, "Modeling by Shortest Data Description," *Automatica*, vol. 14, pp. 465–471, 1978.

[38] Andrew Barron, Jorma Rissanen and Bin Yu, "The Minimum Description Length Prnciple in Coding and Modeling," *IEEE Trans. on Information Theory*, vol. 44, no. 6, pp. 2743–2760, October 1998.

[39] George S. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications*, Springer-Verlag, New York, 1996.

[40] James E. Gentle, *Random Number Generation and Monte Carlo Methods*, Springer, New York, 1998.

[41] C. M. Bishop, "Training with Noise is Equivalent to Tikhonov Regularization," *Neural Computation*, vol. 7, no. 1, pp. 108–116, 1995.

[42] A. N. Tikhonov and V. Y. Arsenin, *Solutions of Ill-posed Problems*, V. H. Winston and sons, Washington D.C., 1977.

[43] Russell Reed, Robert J. Marks II, and Seho Oh, "Simiarities of Error Regularization, Sigmoid Gain Scaling, Target Smoothing, and Training with Jitter," *IEEE Trans. Neural Networks*, vol. 7, no. 3, pp. 529–538, 1995.

[44] C. M. Bishop, "Regularization and Complexity Control in Feed-forward Networks," Technical Report NCRG/95/022, Aston University, Birmingham, UK, 1995.

[45] R. A. Jacobs, "Increased Rates of Convergence through Learning Rate Adaptation," *Neural Networks*, vol. 1, pp. 295–307, 1988.

[46] Lei Xu, "Bayesian Ying-Yang System and Theory as A Unified Statistical Learning Approach (VII): Data Smoothing," in *Proceedings of Intentional Conference on Neural Information Processing*, Kitakyushu, Japan, 1998, 1, pp. 243–248, (ICONIP'98).

[47] Michael R. Lyu, *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, McGraw Hill, 1996.

[48] D. J. C. MacKay, "A Practical Bayesian Framework for Backpropagation Networks," *Neural Computation*, vol. 4, no. 3, pp. 448–472, 1992.

[49] G. J. McLachlan and K. E. Basford, *Mixture Models: Inference and Applications to Clustering*, Dekker, New York, 1988.

[50] B. S. Everitt and D. Hand, *Finite Mixture Distributions*, Chapman and Hall, London, 1981.

[51] N. E. Day, "Estimating the Component of a Mixture of Normal Distributions," *Biometrika*, vol. 56, pp. 463–474, 1969.

[52] H. H. Bock, "Probability Models and Hypotheses Testing in Partitioning Cluster Analysis," in *Clustering and Classification*, Riverside, California, 1996, pp. 377–453, World Scientific Press.

[53] Stefan Aeberhard, Danny Coomans and Olivier de Vel, "Comparative Analysis of Statistical Pattern Recognition Methods in High Dimensional Settings," *Patt. Recog.*, vol. 27, no. 8, pp. 1065–1077, 1994.

[54] J. H. Friedman, "Regularized Discriminant Analysis," *J. Amer. Statist. Assoc.*, vol. 84, pp. 165–175, 1989.

[55] J.P. Hoffbeck and D.A. Landgrebe, "Covariance Matrix Estimation and Classification With Limited Training Data," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 18, no. 7, pp. 763–767, 1996.

[56] Ping Guo and Michael R. Lyu, "Classification for High-Dimension Small-Sample Data Sets Based on Kullback-Leibler Information Measure," in *Proceedings of The 2000 International Conference on Artificial Intelligence*, H. R. Arabnia, Ed., Monte Carlo Resort, Las Vegas, Nevada, USA, 2000, vol. III, pp. 1187–1193, CSREA Press, (IC-AI'00).

[57] A. P. Dempster, N. M. Laird and D. B. Rubin, "Maximum-likelihood from Incomplete Data via the EM Algorithm," *J. Royal Statist. Society*, vol. B39, pp. 1–38, 1977.

[58] R. A. Redner and H. F. Walker, "Mixture Densities, Maximum Likelihood and the EM Algorithm," *SIAM Review*, vol. 26, pp. 195–239, 1984.

[59] Andrew R. Webb, *Statistical Pattern Recognition* , Oxford University Press, London, 1999.

[60] A. Mkhadri, G. Celeux, A. Nasroallah, "Regularization in Discriminant Analysis: An Overview," *Computational Statistics & Data Analysis*, vol. 23, pp. 403–423, 1997.

[61] I. T. Jolliffe, *Principal Component Analysis*, Springer-Verlag, New York, 1986.

[62] E. W. Forgy, "Cluster Analysis of Multivariate Data: Efficiency Versus Interpretability of Classifications," in *Biometric Soc, Meetings*, Riverside, California, 1965, (Abstract in Biometrics 21, No.3, p768).

[63] J. A. Hartigen, "Distribution Problems in Clustering," in *Classification and Clustering*, J.van Ryzin, Ed., New York, 1977, pp. 45–72, Academic Press.

[64] L. Xu, A. Krzyzak and E. Oja, "Rival Penalized Competitive Learning for Clustering Analysis, RBF net and Curve Detection," *IEEE Trans. on Neural Networks*, vol. 4, no. 4, pp. 636–649, 1993.

[65] K. Matusita and N. Ohsumi, "A Criterion for Choosing the Number of Clusters in Cluster Analysis," in *Recent Development in Statistical Inference and Data Analysis*, Amsterdam, North Holland, 1980, pp. 203–213.

[66] H. Bozdagan, "Mixture-Model Cluster Analysis using Model Selection Criteria and a New Informational measure of Complexity," in *Proceeding of the First US/ Japan Conference on the Frontiers of Statistical Modeling: An Informational Approach*, 1994, vol. 2, pp. 69–113.

[67] E. P. Rosenblum, "A Simulation Study of Information Theoretic Techniques and Classical Hypothesis Tests in One Factor ANOVA," in *Proceeding of the First US/ Japan Conference on the Frontiers of Statistical Modeling: An Informational Approach*, 1994, vol. 2, pp. 319–346.

[68] R. B. Calinski and J. A. Harabasz, "A Dendrite Method for Cluster Analysis," *Communications in Statistics*, vol. 3, pp. 1–27, 1974.

[69] D. M. Titterington, "Some Recent Research in the Analysis of Mixture Distributions," *Statistics*, vol. 21, pp. 619–641, 1990.

[70] J. H. Wolfe, "Pattern Clustering by Multivariate Mixture Analysis," *Multivariate Behavioural Research*, vol. 5, pp. 329–350, 1970.

[71] F. H. C. Marriott, "Separating Mixtures of Normal Distributions," *Biometrics*, vol. 31, pp. 767–769, 1975.

[72] M. P. Windham and A. Culter, "Information Ratios for Validating Mixture Analyses," *Journal of the American Statistical Association*, vol. 87, pp. 1188–1192, 1992.

[73] J. Geweke and R. Meese, "Estimating Regression Models of Finite but Unknown Order," *International Economic Review*, vol. 22, pp. 55–70, 1981.

[74] T. Terasvirta and I. Mellin, "Model Selection Criteria and Model Selection Tests in Regression Models," *Stand. J. Statist.*, vol. 13, pp. 159–171, 1986.

[75] Lei Xu, "How Many Clusters?: A YING-YANG Machine Based Theory for A Classical Open Problem in Pattern Recognition," in *Proceeding of IEEE International Conference on Neural Networks*, 1996, vol. 3, pp. 1546–1551.

[76] Lei Xu, "Bayesian Ying-Yang Machine, Clustering and Number of Clusters," *Pattern Recognition Letters*, vol. 18, no. 11-13, pp. 1167–1178, 1997.

[77] Z.B. Lai, P. Guo, T. J. Wang and L. Xu, "Comparison on Bayesian YING-YANG Theory based Clustering Number Selection Criterion with Information Theoretical Criteria," in *Proceedings of IEEE International Joint Conference on Neural Networks*, Anchorage, USA, 1998, vol. I, pp. 725–729, IEEE Press, (IJCNN'98).

[78] Ping Guo and Lei Xu, "On the Study of BKYY Cluster Number Selection Criterion for Small Sample Data Set with Bootstrap Technique," in *Proceedings of 1999 International Joint Conference on Neural Networks* , Washington, DC, USA, 1999, vol. I, pp. 965–968, IEEE Press, (IJCNN'99).

[79] Lei Xu, "Bayesian YING-YANG System and Theory as A Unified Statistical Learning Approach(I): for Unsupervised and Semi-Unsupervised Learning," in *Brain-like Computing and Intelligent Information Systems*, S. Amari and N. Kassabov, Eds. 1997, pp. 241–247, Springer-Verlag.

[80] Lei Xu, "Bayesian YING-YANG System and Theory as A Unified Statistical Learning Approach (II): From Unsupervised Learning to Supervised Learning and Temporal Modeling," in *Theoretical Aspects of Neural Computation: A Multidisciplinary Perspective*, I. King K.W.Wong and D.Y.Yeung, Eds. 1997, pp. 25–42, Springer-Verlag, (TANC97).

[81] H. Akaike, "A New Look at the Statistical Model Identification," *IEEE Transactions on Automatic Control*, vol. AC-19, pp. 716–723, 1974.

[82] H. Bozdogan, *Multiple Sample Cluster Analysis and Approaches to Validity Studies in Clustering Individuals*, Doctoral dissertation, University of Illinois ar Chicago Circle, Chicago,IL, 1981.

[83] H. Bozdogan, "Modle Selection and Akaike's Information Criterion: The General Theory and its Analytical Extensions," *Psychometrika*, vol. 52, no. 3, pp. 345–370, 1987.

[84] G. Schwarz, "Estimating the Dimension of a Model," *The Annals of Statisticals*, vol. 6, no. 2, pp. 461–464, 1978.

[85] M. P. Perrone and L. N. Cooper, "When Networks Disagree: Ensemble Methods for Hybrid Neural Networks," in *Artificial Neural Networks for Speech and Vision*, R. J. Mammone, Ed., London, 1993, pp. 126–142, Chapman & Hall.

[86] A. J. C. Sharkey, "On Combining Artificial Neural Nets," *Connection Science*, vol. 8, pp. 299–313, 1996.

[87] L. Xu, A. Krzyzak, and C.Y. Suen, "Methods of Combining Multiple Classifiers and Their Application to Handwriting Recognition," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 22, no. 3, pp. 418–435, 1992.

[88] L. Breiman, "Bagging Predictors," Tech. Rep. 421, Department of Statistics, University of California at Berkely, 1994.

[89] A. Krogh and J. Vedelsby, "Neual Network Ensemble, Cross Validation, and Active Learning," in *Advanced in Neural Information Processing Systems*, D.S.Touretzky G. Tesauro and T.K.Leen, Eds., Cambridge, 1995, vol. 7, pp. 231–238, MIT Press.

[90] K. Tumer and J. Ghosh, "Error Correlation and Error Reduction in Ensemble Clasifiers," *Connection Science*, vol. 8, no. 3-4, pp. 385–404, 1996.

[91] J. Ghosh, L. Deuser, and S. Beck, "A Neural Network based Hybrid System for Detection, Characterization and Classification of Short-duration Oceanic Signals," *IEEE Journal of Ocean Engineering*, vol. 17, no. 4, pp. 351–363, October 1992.

[92] T.K.Ho, J.J. Hull, and S.N.Srihari, "Decision Combination in Multiple Classifier System," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 1, pp. 66–76, 1994.

[93] Jianchang Mao, "A Case Study on Bagging, Boosting, and Basic Ensembles of Neural Networks for OCR," in *Proceedings of IEEE International Joint Conference on Neural Networks*, Anchorage, USA, 1998, vol. III, pp. 1828–1833, (IJCNN'98).

[94] L. Xu and M.I. Jordan, "EM Learning on a Generalized Finite Mixture Model for Combining Multiple Classifiers," in *Proceedings of World Congress on Neural Networks*, Portland, OR, 1993, vol. IV, pp. 227–230.

[95] U. Naftaly, N. Intrator and D. Horn, "Optimal Ensemble Averaging of Neural Networks," *Network: Comput. Neural Syst.*, vol. 8, pp. 283–296, 1997.

[96] M. Taniguchi and V. Tresp, "Averaging Regularized Estimators," *Neural Computation*, vol. 9, pp. 1163–1178, 1997.

[97] R.A. Jacobs, M.I.Jordan, S.J. Nowlan and G.E. Hinton, "Adaptive Mixtures of Local Experts," *Neural Computation*, vol. 3, pp. 79–97, 1991.

[98] L. Xu, M. I. Jordan and G. E. Hinton, "An Alternative Model for Mixtures of Experts," in *Advances in Neural Information Processing Systems*, G. Tesauro J.D. Cowan and J. Alspector, Eds., Cambridge,MA, 1995, vol. 7, pp. 633–640, MIT Press.

[99] Daniel Jimènez and Nicolas Walsh, "Dynamically Weighted Ensemble Neural Networks for Classification," in *Proceedings of IEEE International Joint Conference on Neural Networks*, Anchorage, USA, 1998, vol. I, pp. 753–756, (IJCNN'98).

[100] P. Guo, C.L.P. Chen and Y.G.Sun, "An Exact Supervised Learning for a Three-Layer Supervised Neural Network," in *Proceedings of International Conference on Neural Information Processing*, Beijing, China, 1998, pp. 1041–1044, (ICONIP'95).

[101] Ping Guo and Michael R. Lyu, "Pseudoinverse Learning Algorithm for Feedforward Neural Networks," in *Advances in Neural Networks and Applications*, N. E. Mastorakis, Ed., Puerto De La Cruz, Tenerife, Canary Islands, Spain, February 2001, World Scientific and Engineering Society, pp. 321–326, WSES Press, (NNA'01).

[102] S. Haykin and C. Deng, "Classification of Radar Clutter Using Neural Networks," *IEEE Transactions on Neural Networks*, vol. 2, pp. 589–600, 1991.

[103] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard and W. Hubbard, "Handwritten Digit Recognition with a Back-propagation Network," in *Advanced*

in Neural Information Processing Systems, D. S. Touretsky, Ed., San Mateo, CA, 1990, pp. 396–404.

[104] D. E. Rumelhurt, G. E. Hinton and R. J. Williams, "Learning Internal Representations by Error Propagating," in *Parallel Distributed Processing*, D.E.Rumelhart and J.L.McClelland, Eds., Cambridge, MA, 1986, vol. 1, pp. 318–362, MIT Press.

[105] P. Patrick and Van Der Smagt, "Minimization Methods for Training Feedforward Neural Networks," *Neural Networks*, vol. 7, pp. 1–11, 1994.

[106] E. Barnard, "Optimization for Training Neural Nets," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 232–240, 1992.

[107] F. A. Zodewyk, Wesswls and B. Etienne, "Avoid False Local Minima by proper Initializations of Connections," *IEEE Transaction on Neural Networks*, vol. 3, no. 6, pp. 899–905, 1992.

[108] S. Kollias and D. Anastassiou, "An Adaptive Least Squares Algorithm for the Efficient Training of Artificial Neural Networks," *IEEE Transaction on Circuit and System*, vol. CAS-36, pp. 1092–1101, 1989.

[109] D. H. Wolpert, "Stacked Generalization," *Neural Networks*, vol. 5, pp. 241–259, 1992.

[110] Thomas L. Boullion and Patrick L. Odell, *Generalized Inverse Matrices*, John Wiley and Sons, Inc., New York, 1971.

[111] S. Tamura, "Capabilities of a Tree Layer Feedforward Neural Network," in *Proceedings of International Joint Conference on Neural Networks*, Seattle, USA, 1991, pp. 2757–2762, (IJCNN'91).

[112] C. R. Rao and S. K. Mitra, *Generalized Inverse of Matrices and Its Applications*, Wiley, New York, 1971.

[113] Jon F. Claerbout, *Fundamentals of Geophysical Dada Processing with Applications to Petroleum Prospecting*, McGraw-Hill Inc, USA, 1976, (TN 271, P4C6).

[114] F. Biegler-K önig and F. Bärmann, "A Learning Algorithm for Multilayered Neural Networks Based on Linear Least Squares Problems," *Neural Networks*, vol. 6, pp. 127–131, 1993.

[115] N. Karunanithi, D. Whitley and Y. K. Malaiya, "Prediction of Software Reliability Uisng Connectionist Models," *IEEE Transaction on Software Engineering*, vol. 18, pp. 563–574, 1992.

[116] Ping Guo, "Averaging Ensemble Neural Networks in Parameter Space," in *Proceedings of fifth International Conference on Neural Information Processing*, Takashi Omori Shiro Usui, Ed., Kitakyushu, Japan, 1998, pp. 486–489, IOS Press, (ICONIP'98).

[117] K. S. Fu and J. K. Mui, "A Survey on Image Segmentation," *Pattern Recognition*, vol. 13, pp. 3–16, 1981.

[118] H. S. Choi, D. R. Haynor and Kam, "Partial Volume Tissue Classification of Multichannel Magnetic Resonance Images–Amixel Model," *IEEE Trans. Med. Image.*, vol. 10, pp. 395–407, 1991.

[119] P. Santago and H. D. Gage, "Statistical Models of Partial Volume Effect," *IEEE Trans. Image Processing*, vol. 4, no. 11, pp. 1531–1540, 1995.

[120] S. Sanjay-Gopal and Thomas J. Hebert, "Bayesian Pixel Classification Using Spatially Variant Finite Mixtures and the Generalized EM Algorithm," *IEEE Transaction on Image Processing*, vol. 7, no. 7, pp. 1014–1028, 1998.

[121] Yu-TCHi, Takeo Kanade, and Toshiyuki Sakai, "Color Information for Region Segmentation," *Computer Graphics and Processing*, vol. 13, pp. 222–241, 1980.

[122] B. W. Boehm and P. N. Papaccio, "Understanding and Controlling Software Costs," *IEEE Trans. on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, October 1988.

[123] A. J. Perlis, F. G. Sayward and M. Shaw, *Software Metrics: An Analysis and Evaluation*, MIT Press, Cambridge, MA, 1981.

[124] V. Y. Shen, T. Yu, S. M. Thebaut, and L. R. Paulsen, "Identifying Error-prone Software — An Empirical Study," *IEEE Trans. on Software Engineering*, vol. SE-11, pp. 317–323, April 1985.

[125] S. G. Crawford, A. A. McIntosh, and D. Pregibon, "An Analysis of Static Metrics and Faults in C Software," *J. Syst. Sofyware*, vol. 5, pp. 27–48, 1985.

[126] L. C. Briand, V. R. Basili, and C. Hetmanski, "Developing Interpretable Models for Optimized Set Reduction for Identifying High-Risk Software Components," *IEEE Trans. on Software Engineering*, vol. SE-19, no. 11, pp. 1028–1034, November 1993.

[127] J. Munson and T. Khoshgoftaar, "The Detection of Fault-prone Programs," *IEEE Trans. on Software Engineering*, vol. SE-18, no. 5, pp. 423–433, May 1992.

[128] T. Khoshgoftaar and J. Munson, "Predicting Software Development Error Using Software Complexity metrics," *IEEE Trans. on Software Engineering*, vol. 8, no. 2, pp. 253–261, February 1990.

[129] A. A. Porter and R. W. Selby, "Empirically Guided Software Development Using Metric-based Classification Trees," *IEEE Software*, vol. 7, no. 2, pp. 46–54, March 1990.

[130] R. W. Selby and A. A. Porter, "Learning from Examples: Generation and Evolution of Decision Trees for Software Resource Analysis," *IEEE Trans. on Software Engineering*, vol. 14, no. 12, pp. 1743–1756, December 1988.

[131] S. S. Gokhale and M. R. Lyu, "Regression Tree Modeling for the Prediction of Software Quality," in *Proceedings of Third ISSAT International Conference:Reliability & Quality in Design*, Hoang Pham, Ed., Anaheim, CA, 1997, pp. 31–36.

[132] L. C. Briand, V. R. Basili, and W. M. Thomas, "A Pattern Recognition Approach for Software Engineering Data Analysis," *IEEE Transaction on Software Engineering*, vol. SE-18, no. 11, pp. 931–942, November 1992.

[133] T. Khoshgoftaar, D. L. lanning and A. S. Pandya, "A Comparative Study of Pattern Recognition Techniques for Quality Evaluation of Telecommunications Software," *IEEE J. Selected Areas in Communication*, vol. 12, no. 2, pp. 279–291, February 1994.

[134] L. M. Ottenstein, "Quantitative Estimates of Debugging Requirements," *IEEE Trans. on Software Engineering*, vol. SE-5, no. 2, pp. 504–514, September 1979.

[135] W. R. Dillon and M. Goldstein, *Multivariate Analysis*, Wiley, New York, 1984.

[136] V. R. Basili and D. H. Hutchens, "An Empirical Study of a Syntactic Complexity Family," *IEEE Trans. on Software Engineering*, vol. SE-9, no. 6, pp. 664–672, November 1983.

[137] R. K. Lind, "An Experimental Study of Software Metrics and their Relationship to Software Error," M.S. thesis, University of Wisconsin-Milwaukee, Milwaukee, December 1986, Master's thesis.

[138] M. Halstead, *Elements of Software Science*, New York Elsevier, North-Holland, 1977.

[139] H. Jensen and K. Vairavan, "An Experimental Study of Software Metrics for Real-Time Software," *IEEE Trans. on Software Engineering*, vol. SE-11, no. 2, pp. 231–234, February 1994.

[140] T. J. McCabe, "A Complexity Measure," *IEEE Trans. on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.